Enhancing Dexterity in Robotic Manipulation via Hierarchical Contact Exploration

Xianyi Cheng, *Graduate Student Member, IEEE*, Sarvesh Patil, *Graduate Student Member, IEEE*, Zeynep Temel, *Member, IEEE*, Oliver Kroemer, *Member, IEEE*, and Matthew T. Mason, *Life Fellow, IEEE*

***Abstract*—Planning robot dexterity is challenging due to the non-smoothness introduced by contacts, intricate fine motions, and ever-changing scenarios. We present a hierarchical planning framework for dexterous robotic manipulation (HiDex). This framework explores in-hand and extrinsic dexterity by leveraging 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. 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 provides 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* tasks on two robot platforms.**

***Index Terms*—Dexterous manipulation, manipulation planning, in-hand manipulation, contact modeling.**

I. Introduction

ROBOTS need dexterity for complex manipulation 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 needs to not only use its own fingers dexterously, but also be smart about exploiting its surroundings, as "external" fingers to support the movements of the object.

Planning for dexterity remains challenging. First, planning through contacts, which involves changes in system dynamics and non-smoothness, is particularly difficult [1], especially considering both robot and environment contacts. Second, due to the diverse nature of manipulation, robots need to discover various fine motions, making it hard to simplify problems with predefined primitives. Third, current manipulation planners are tailored for specific tasks. An in-hand manipulation policy [2] cannot solve object reorientation in [3], and neither of them can be directly applied to planar pushing [4]. As real-world manipulation is a mix of various manipulation problems, it is important for a general manipulation planner to cover different tasks.

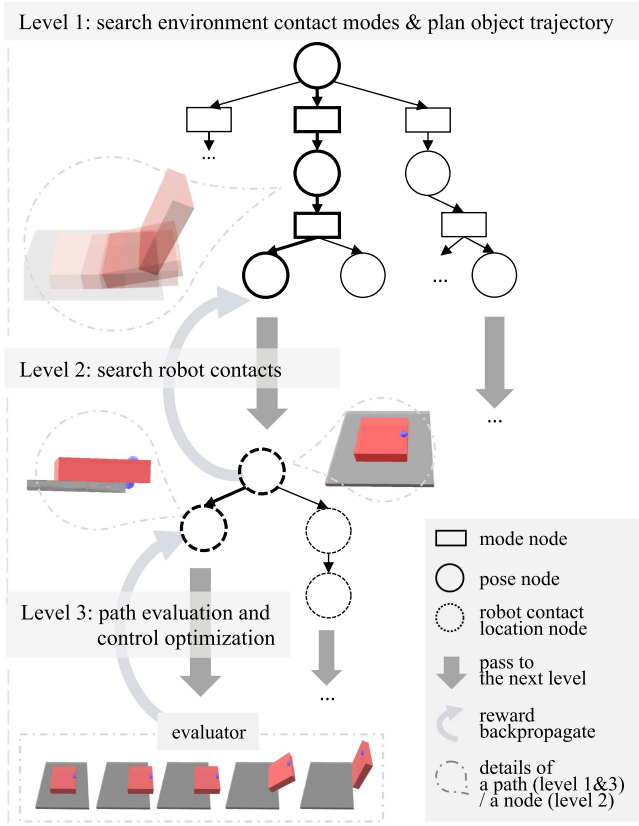

Fig. 1. Overview of our framework, with an example of picking up a card. The following processes run iteratively. Level 1 plans object trajectories, interleaving searches over contact modes (▭ nodes) and continuous object poses (◯ nodes). An object trajectory is passed to Level 2 (⇩) to plan robot contacts on the object (◌ 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 to the edge of the table and then grasps it.

Manuscript received 4 July 2023; accepted 1 November 2023. Date of publication 16 November 2023; date of current version 28 November 2023. This letter was recommended for publication by Associate Editor H. Wang and Editor H. Liu upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Grant CMMI-2024794. *(Corresponding author: Xianyi Cheng.)*

The authors are with the Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: xianyic@andrew.cmu.edu; sarveshp@andrew.cmu.edu; ztemel@andrew.cmu.edu; okroemer@andrew.cmu.edu; mm3x@andrew.cmu.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3333699, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3333699

© 2023 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

We propose a *hierarchical* framework, as shown in Fig. 1, aiming to address the challenges mentioned above. In Level 1, we perform object trajectory and environment contact mode planning. Contact modes of object-environment contacts guide the generation of object motions — 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 of the plans are computed and rewards are backpropagated.

Our contributions are in three aspects: framework, methodology, and experiment. *Framework:* Our framework adapts with minor adjustments to various tasks. It easily encodes expert knowledge and priors through the MCTS action policies, value estimations, and rewards. This is the first framework that achieves manipulation tasks of such complexity and variety. For future development, it can directly integrate new components like trajectory optimization and learning. *Methodology:* We demonstrate the efficacy of three novel ideas with potential benefits for contact-rich robotic research: 1) an efficient hierarchical structure that decomposes the search of environment contact modes, object trajectory, and robot contacts; 2) replacing Monte-Carlo rollout with a rapidly exploring random tree (RRT) based method, leveraging its effectiveness in goal-oriented exploration without losing the Monte-Carlo spirit; 3) a novel representation that plans contact changes rather than contacts for each step, which greatly speeds up robot contact planning. *Experiments:* We instantiate and demonstrate a variety of dexterous manipulation tasks, including *pick up a card*, *book-out-of-bookshelf*, *peg-out-of-hole*, *block flipping*, *muti-robot planar manipulation*, and *in-hand reorientation*. Some tasks were never explored in previous works, including *occluded grasp*, *upward peg-in-hole*, and *sideway peg-in-hole*. As a contribution to the community, we open-sourced our code for these tasks. It is also easy to configure new scenarios and adjust parameters with one setup.yaml file.

II. Related Work

A. Dexterous Manipulation Planning

In dexterous manipulation, potential contact changes introduce a high dimensional, non-convex, discrete and continuous space to plan through. Contact-implicit trajectory optimization [1] performs local optimization in the complex space, which can be intractable without good initialization, and currently only works for 2D problems with shape simplifications [5], [6]. Alternatively, contact kinematics [7], [8], [9], [10] can be leveraged to generate motions between two rigid bodies [11], within a robot hand [12], dexterous pregrasps [13], under environment contacts [14], [15], [16], [17], and with trajectory optimization [18]. We build upon [16], which incorporates contact modes into an RRT to guide node expansion. We introduce a more optimized search structure, addressing three key issues: no mechanism to optimize for any reward, random sampling for robot contact planning, and an entangled configuration space of the object and the robot that results in many redundant searches. We introduce hierarchical search to decompose the space of object motion and robot contacts [19], [20]. Unlike previous approaches using contact states, which can have the combinatoric explosion for 3D cases, our method leverages contact modes and efficient robot contact search, extending the application from 2D polyhedrons to 3D meshes.

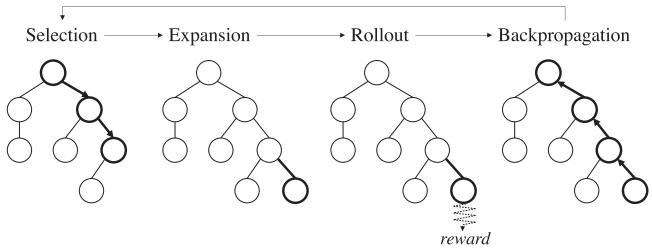

Fig. 2. Four steps run iteratively in GROW-TREE in MCTS [26]. Selection: start from the root node and select successive nodes. Expansion: Create a new node from unexplored children of the last selected node. Rollout: evaluate the new node by simulating to the end with random sampling. Backpropagation: Update the tree using rollout rewards.

Reinforcement learning (RL) can discover skills like in-hand manipulation [2], dexterous grasping [21], and object reorientation [22]. RL faces the same challenges from high-dimensional complex spaces, leading to sample efficiency problems. Our framework's advantage lies in adapting to new tasks and discovering new skills through contact exploration, without the need for training or reward shaping.

B. Monte-Carlo Tree Search

MCTS is a heuristic search algorithm that uses random sampling for efficient exploration. AlphaGo [23] combines MCTS with deep neural networks, achieving superhuman performance in board games like Go. MCTS has shown potential in contact planning, including gait planning for legged robots [24] and robot contact sequence planning given object trajectories [25]. Our work plans not only robot contacts but also object motions and interactions with environment contacts. MCTS offers benefits like efficient search through vast complex space, continuous improvement through learning and self-exploration, and parallelizability.

III. Preliminaries

A. 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 expands the tree by iteratively running four steps in Fig. 2: selection, expansion, rollout, and backpropagation. We employ the idea in AlphaGo [23] — 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 an adaptive parameter $\lambda \in [0, 1]$ balances ratios of the obtained value and the value estimation. λ increases as the search goes on.

Algorithm 1: MCTS Skeleton.

procedure Search
  $\mathcal{T} \leftarrow$ new-tree()
  $n \leftarrow$ root-node($\mathcal{T}$)
  while n is not a terminal node **do**
    grow-tree($\mathcal{T}, n$)
    $n \leftarrow$ best-child(n)
  end while
end procedure

Selection determines search directions by balancing exploration and exploitation. 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:

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

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

B. Contact Modes

Collision detection obtains the contact points between the object and the environment. Contact modes describe the possible evolution of these contact points. In this letter, for N contacts, a contact mode is $\boldsymbol{m} = [sign(\boldsymbol{v}^{\boldsymbol{i}}_{\boldsymbol{c,n}})] \in \{0,+\}^N$, where $\boldsymbol{v}^{\boldsymbol{i}}_{\boldsymbol{c,n}}$ is the contact normal velocity for the ith contact. 0 means a contact maintained. $+$ means a contact separate. A more comprehensive description can be found in [9].

Contact modes offer an efficient way to generate object motions in lower-dimensional manifolds that have zero probability to be generated through random sampling. In addition, each contact mode corresponds to one set of continuous contact dynamics. By choosing among contact modes, discreteness in dynamics is efficiently captured.

IV. Hierarchical Planning Framework

A. Task Description

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

A planner designed under this framework takes in:

1) The object: a rigid body $\mathcal{O}$ with known center of mass and inertia matrix, and friction coefficients with environment μ_{env} and with the manipulator μ_{mnp}. The object geometry, used for collision check and surface point sampling, should be provided as a mesh model or a primitive shape like a cuboid.
2) Environment with known geometries, as primitive shapes or mesh models.
3) Robot model: The robot manipulates the object through N_{mnp} predefined fingertip contacts. The collision models, forward and inverse kinematics, and finger contact models are known. We assume the robot makes non-sliding and non-rolling contacts.
4) Start specification: object start pose $\boldsymbol{x}_{\text{start}} \in SE(3)$.
5) Goal specification: object goal region $X_{\text{goal}} \subset SE(3)$.

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

Algorithm 2: Level 1: Search Object Motion.

procedure Grow-tree-level-1($\mathcal{T}$, startnode)
  while resources left **do**
    $n \leftarrow$ startnode
    ▷ Interleaved selection over *pose* and *mode*
    while n is not terminal **do**
      if *nodetype*(n) is *pose* **then**
        ▷ [Selection] next contact mode
        $\mathcal{A}(n) \leftarrow$ feasible contact modes of n
        $a \leftarrow$ select($\mathcal{A}(n)$)
        $n \leftarrow mode - node(a)$
      end if
      if *nodetype*(n) is *mode* **then**
        ▷ [Selection] next pose
        $\mathcal{A}(n) \leftarrow children - of(n) \cup explore - new$
        $a \leftarrow$ select($\mathcal{A}(n)$)
        if a is *explore-new* **then** ▷[Expansion]
          path $\leftarrow$ RRT-explore(n) ▷[Rollout]
          attach($\mathcal{T}$, path)
          $r \leftarrow$ evaluate-reward(path) ▷To Level 2
          back-propagate($r, \mathcal{T}$) ▷[Backpropagation]
          break loop
        else
          $n \leftarrow to - next - node(a)$
        end if
      end if
    end while
    $r \leftarrow$ evaluate-reward(n) ▷To Level 2
    back-propagate($r, \mathcal{T}$) ▷[Backpropagation]
  end while
end procedure

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

Level 1 is summarized in Algorithm 2, and visualized in Fig. 3. It plans trajectories of environment contact modes and object configurations. The search process is as follows. In the existing tree, it first performs the selection phase to choose a node to grow a new branch. We interleave the selection over environment contact modes and object configurations. When a node is selected for expansion and rollout, an RRT-based rollout replaces random rollouts to improve exploration efficiency. The RRT rollout grows a new branch for the MCTS and then passes to Level 2. During the process, environment contact modes are generated through a contact mode enumeration algorithm [9]. Object configurations are generated through the projected forward integration in the RRT rollout (details in Section IV-B3).

1) Selection — Interleaved Search Over Discrete and Continuous Space: We define Level 1 state as $s1 = (x \in$

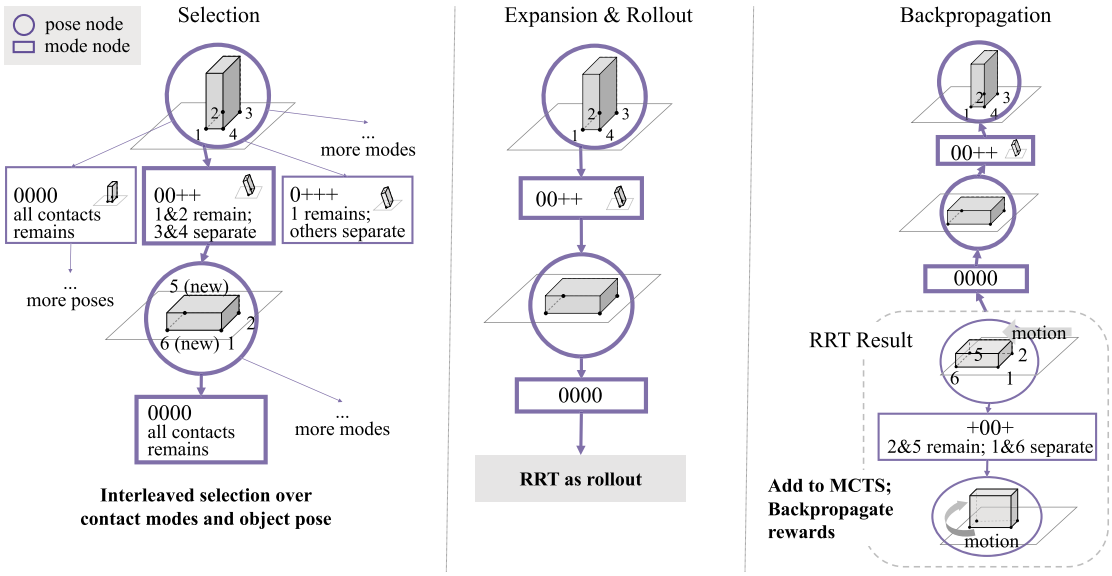

Fig. 3. Level 1 search visualized with a block reorientation example. In the selection phase, the block is selected to follow contact mode 00++, leading to a 90-degree rotation at contact 1 and 2 and the separation of 3 and 4. In the new node, contact mode 0000, indicating contact 1, 2, 5, 6 are maintained, is then selected for the RRT rollout. The RRT generates an object configuration trajectory where the block is first pushed and then rotated 90 on the edge of contact 2 and 5. New nodes from the RRT solution path are added to the MCTS after reward evaluation.

$\mathrm{SE}(3), nodetype \in \{mode, pose\}$), where $\boldsymbol{x}$ 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 Fig. 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 $\boldsymbol{x}$ using the algorithm in [9]. A *mode* node is a *pose* node assigned with a contact mode. 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 *explore-new*, which triggers the rollout phase to explore new object poses.

The selection policy follows (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 [27]), we could have high probabilities for modes that maintain more 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 [28], where we control the expansion rate with the action probability of *explore-new*.

3) RRT as Rollout: Contact-rich solutions live sparsely on lower-dimensional manifolds of the search space. It is unlikely to get any useful results from random trajectory rollouts as most traditional MCTS do. We replace the random rollout with an RRT search guided by contact modes (line 17, Algorithm 2), modified based on [16].

The RRT begins with an object pose $\boldsymbol{x}_{\text{current}}$ and a selected contact mode $\boldsymbol{m}_{\text{selected}}$. The RRT tries to reach $\boldsymbol{x}_{\text{goal}}$. 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 $\boldsymbol{x}_{\text{extend}} \in \mathrm{SE}(3)$ and find its nearest neighbor $\boldsymbol{x}_{\text{near}}$. We then extend $\boldsymbol{x}_{\text{near}}$ towards $\boldsymbol{x}_{\text{extend}}$. Each extension is guided by a contact mode. If $\boldsymbol{x}_{\text{near}}$ is $\boldsymbol{x}_{\text{current}}$, the contact mode should be $\boldsymbol{m}_{\text{selected}}$. Otherwise, we enumerate all contact modes and filter them using feasibility checks. New object poses are generated by projected forward integration that follows each contact mode towards $\boldsymbol{x}_{\text{extend}}$ as close as possible. If the RRT finds a solution to $\boldsymbol{x}_{\text{goal}}$ 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 a zero reward and no new node is added. The RRT is reused throughout the entire lifespan of the MCTS. More details about the RRT-based rollout can be found in appendices.

One difference to [16] is that we can choose whether to plan robot contacts or not in the RRT. We can turn on the option to relax a contact mode to be feasible if there "exists any feasible robot contact", while previously the RRT needs to plan and store robot contacts, searching in a higher-dimensional space. 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 configuration trajectory, and outputs the best robot contact sequence. Algorithm 3 summarizes the GROW-TREE process in Level 2 MCTS.

1) State and Action Representation: Each node is associated with a robot contact state $s2 = \big(t, \{(i, \boldsymbol{p_i}) | i \in \text{active fingers at } t\}\big)$. A robot contact state specifies active

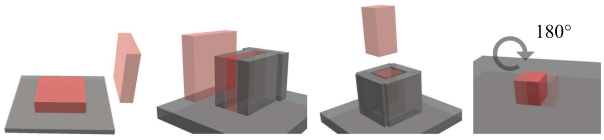

Fig. 4. Manipulation with extrinsic dexterity. From left to right: Scenario 1, *pick up a card:* pick up a thin card that cannot be directly grasped using two available fingertips. Scenario 2, *bookshelf:* get a book among other books out of a bookshelf using three available fingertips. Scenario 3, *peg-out-of-hole:*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

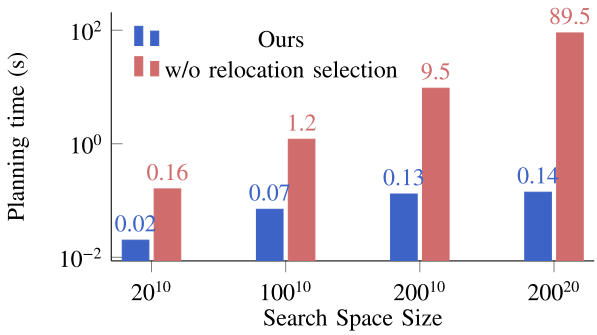

Fig. 5. Planning time (in log scale) with respect to search space size for planning robot contacts for cube sliding. We have $search\ space\ size = candidate\ contacts^{trajectory\ size}$.

fingers, fingers that are in contact with the object, and their corresponding contact locations $\{\boldsymbol{p_i} \in \mathbb{R}^3\}$ on the object surface at timestep t. An action $a = \big(t_c, \{(j, \boldsymbol{p_j}) | j \in$ relocating fingers at $t_c\}\big)$ represents robot contact relocations, specified by relocating timestep t_c, relocating fingers, and the object surface points they are relocating to $\{\boldsymbol{p_j}\}$.

For example, consider 5 available robot fingertips. Grasping an object at timestep 0 with the first and third finger at locations (1, 0, 0) and $(-1, 0, 0)$ corresponds to the state $\big(t = 0, (\mathit{finger1}, \boldsymbol{p_1} = (1, 0, 0)), (\mathit{finger3}, \boldsymbol{p_3} = (-1, 0, 0))\big)$. 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 $\big(t_c = 4, (\mathit{finger3}, \boldsymbol{p_3} = (0, 0, 1)), (\mathit{finger5}, \boldsymbol{p_5} = (-1, 0, 0.5))\big)$. The new state is $\big(t = 4, (\mathit{finger1}, \boldsymbol{p_1} = (1, 0, 0)), (\mathit{finger3}, \boldsymbol{p_3} = (0, 0, 1)), (\mathit{finger5}, \boldsymbol{p_5} = (-1, 0, 0.5))\big)$.

Compared to the common practice of planning contacts for every timestep [6], [25], we plan 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, Fig. 5).

2) Sampling and Pruning for Action Selection: Consider 100 object surface points, 4 fingers, and 10 steps. The action space is $(100^4)^{10} = 1e80$. It is not practical to evaluate all actions. To mitigate this, we adopt action sampling techniques for non-enumerable action space [29]. In Equation (1), we use a subset of all actions $\mathcal{A}_{\text{sp}}(s2) \subset \mathcal{A}(s2)$. $\mathcal{A}_{\text{sp}}(s2)$ includes previously explored actions and newly sampled actions. Newly sampled actions are generated as follows: 1) 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-C3. 2) After t_c is sampled, we sample robot contact relocation through rejection sampling. We first find relocatable robot contacts by checking if the remaining fingers satisfy the force conditions. Then we sample new feasible contact locations on the object surface.

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
```

We mix the selection, expansion, and rollout using the same heuristic function. If the selected action is explored before, it is the selection phase. If the selected action is new, it is the expansion and rollout phase.

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, like quasistatic, quasidynamic, force balance, or force closure conditions.
- Other task-specific requirements may also be added.

D. Level 3: Path Evaluation and Control Optimization

Level 3 (line 9, Algorithm 3) computes feasibility, robot controls $\boldsymbol{u}(t)$, rewards for full trajectories from Level 2.

If the task mechanics are quasi-static or force-closure, we individually solve for each step t to check whether quasi-static or force-closure solutions exist and output the robot positions and optimal contact forces as the control $\boldsymbol{u}(t)$. If full dynamics is required, the framework can potentially use the path as a warm-start for trajectory optimization.

For the control trajectory $\boldsymbol{u}(t)$, 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 reward values 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

A. Implementation

We implemented two task types: manipulation with extrinsic dexterity and in-hand manipulation.[1] We use Dart [30] for visualization and Bullet [31] for collision detection. New scenario setup requirements and more implementation details can be found in appendices.

1) Robot Model: We implemented two robot types.

Sphere fingertips: Each fingertip is a sphere with workspace limits as kinematic feasibility checks. Collision is checked with 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 [32], [33]. We provide an analytical inverse kinematics model and use the line contact model as the fingertip contact model.

2) Task Mechanics: quasi-static, quasi-dynamic, and force closure models (details in appendices).

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.

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 [34]. 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:

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

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 $t_{\max}$:

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

[1]Supplementary materials: video (https://youtu.be/fScfat1Ys6U), code (https://github.com/XianyiCheng/HiDex), and website (https://xianyicheng.github.io/HiDex-Website).

TABLE I
Planning Statistics for Manipulation With Extrinsic Dexterity: Our Method (Left, Bold), CMGMP (right)

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

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 Fig. 4. Each scenario is implemented with sphere fingertips 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 i9-10900 K 3.70 GHz CPU (also for all other statistics in this letter). As our algorithm is anytime, we let the planner stop after 10 seconds and collect the results. The search process is fast. Similar to CMGMP [16], about 80% of the computation is on the projected forward integration of the RRT.

1) Ablation of Hierarchical Structure and MCTS: We compare with CMGMP [16], which uses an RRT in searching object motions and robot contacts. For all scenarios, our method has a faster "solution found time". The hierarchical structure speeds up solution discovery by decoupling the object pose and robot contact space. Our method continuously improves the solution by searching more space, whereas CMGMP can only stop upon initial solution discovery, resulting in our method generating more "nodes in the tree". Guided by the MCTS rewards, our method also finds solutions with a smaller "travel distance ratio", less "finger relocation" and "environment contact changes", and smaller "grasp centroid distance", while CMGMP cannot optimize for any reward.

2) Efficient Robot Contact Planning: We compare the planning of contact relocations in Level 2 with the common practice that explicitly plans contacts for each timestep [6], [25] (w/o relocation selection). We run the robot contact planning for a straight-line cube sliding trajectory with one allowable robot contact, with varying numbers of total timesteps in the trajectory and object surface points. As shown in Fig. 5, the search space size grows exponentially for both methods. The planning time of the common practice also grows exponentially. Our modification uses drastically less time. Our assumption is that this modification aligns better with the fact that contact relocations are sparse compared to discretization of the entire trajectory.

TABLE II
PLANNING STATISTICS IN THE FORMAT OF (AVERAGE ± STANDARD DEVIATION) FOR IN-HAND MANIPULATION FOR DIFFERENT FINGER ARRANGEMENTS (WORKSPACES SHOWN BY COLORED BOXES) AND OBJECTS (IMAGES FROM YCB DATASET [35], EXCEPT FOR THE CYLINDER)

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

C. In-Hand Manipulation

In-hand manipulation demonstrates intrinsic dexterity. For inherently safer motions, we require every motion to have force balance or force closure solutions.

1) Different Hand Configurations: Table II shows the statistics for YCB dataset objects [35] with 100 runs of randomized start and goal poses on three workspace configurations. Without any training or tuning, our framework achieves a high planning success rate within seconds. Point sampling on object surfaces ensures consistent performance for complex shapes, enabling planning contacts inside concave objects, like the mug and the power drill in the video.

2) Add Auxiliary References: While our framework is designed to achieve object goal poses, we can incorporate auxiliary references, like goal fingertip locations, by modifying the reward function and the action probability. We define d_c, the average robot contact distance to the reference fingertip locations divided by an empirical characteristic length (like the object length). We fit a new reward function that prefers small d_c. We bias the action probability to sample contact locations that are closer to the goal through $w(s2, a)$:

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

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 reference fingertip locations. As Table III shows, the new planner results in smaller "Final finger distance" to the reference fingertip locations, but more finger relocations are needed. Note that due to potential conflicts from the primary goal of object pose 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 the DDHand (12 kHz bandwidth) and a configurable array of soft delta robots (delta array, 500 Hz control frequency) [36]. We perform open-loop trajectory execution (no object pose or contact estimation). 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 within 1 mm for the DDHand and 2 cm for the delta array.

1) DDhand: The planner enables the DDHand to use intrinsic and extrinsic dexterity, as shown in Fig, 6 and the video. 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; and move the other finger to the other side to form a grasp. In *upward peg-in-hole*, the hole prevents the fingers from getting in while

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

	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**

The bold values highlight the planning statistics from the method.

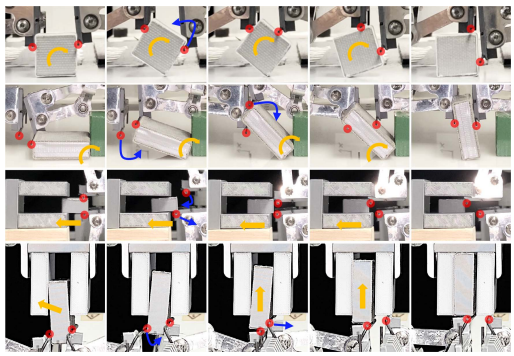

Fig. 6. Keyframes of DDHand experiments. From top to bottom: *cube reorientation*, *occluded grasp*, *sideway peg-in-hole*, *upward peg-in-hole*. Object motions, fingertip locations, and fingertip relocations are respectively marked in yellow, red, and blue.

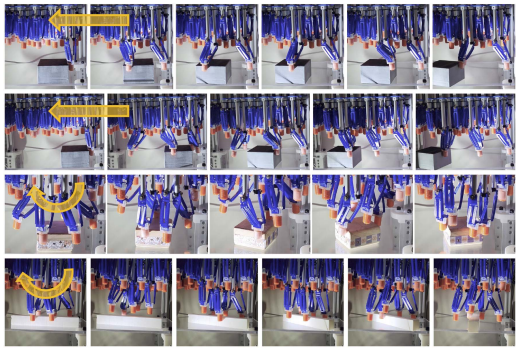

Fig. 7. Keyframes of the delta array experiments. Object motions are marked with yellow arrows. From top to bottom: 2-finger and 6-finger *planar block passing*, 6-finger and 5-finger *planar reorientation*.

grasping the object, but without a grasp, gravity will cause the peg to fall. 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 relocates and pushes the peg from the bottom.

2) Delta Array: As shown in Fig. 7 and the video, due to the small workspace of a delta robot (a cylinder with a 2 cm radius and 6 cm height), many contact changes are required to accomplish the tasks.

VI. Discussion

This letter proposes a hierarchical framework for planning dexterous robotic manipulation. It facilitates efficient searches across complex spaces, the generation of diverse manipulation skills, and adaptability for various scenarios. This method can potentially automate wide-ranging manipulation applications, such as functional grasps, caging, forceful manipulation, and mobile and aerial manipulation. For future development, this framework is compatible with direct integration of trajectory optimization, learning, complex robot contact strategies like sliding and rolling.

References

[1] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 69–81, 2014.

[2] O. M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, 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 *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 1631–1638.

[4] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *Int. J. Robot. Res.*, vol. 15, no. 6, pp. 533–556, 1996.

[5] N. Doshi, F. R. Hogan, and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitives," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 6759–6765.

[6] B. Aceituno-Cabezas and A. Rodriguez, "A global quasi-dynamic model for contact-trajectory optimization in manipulation," *Robot.: Sci. Syst. Found.*, 2020.

[7] J. Xiao and X. Ji, "Automatic generation of high-level contact state space," *Int. J. Robot. Res.*, vol. 20, no. 7, pp. 584–606, 2001.

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

[9] E. Huang, X. Cheng, and M. T. Mason, "Efficient contact mode enumeration in 3D," in *Proc. Int. Workshop Algorithmic Found. Robot.*, 2020, pp. 485–501.

[10] E. Huang, X. Cheng, Y. Mao, A. Gupta, and M. T. Mason, "Auto-generated manipulation primitives," *Int. J. Robot. Res.*, vol. 42, 2023, Art. no. 02783649231170897.

[11] P. Tang and J. Xiao, "Automatic generation of high-level contact state space between 3D curved objects," *Int. J. Robot. Res.*, vol. 27, no. 7, pp. 832–854, 2008.

[12] J. C. Trinkle and J. J. Hunter, "A framework for planning dexterous manipulation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1991, pp. 1245–1251.

[13] S. Chen, A. Wu, and C. K. Liu, "Synthesize dexterous nonprehensile pregrasp for ungraspable objects," in *Proc. ACM SIGGRAPH Conf.*, Los Angeles, CA, USA, 2023.

[14] X. Cheng, Y. Hou, and M. T. Mason, "Manipulation with suction cups using external contacts," in *Proc. Robot. Res.: 19th Int. Symp.*, 2022, pp. 692–708.

[15] X. Cheng, E. Huang, Y. Hou, and M. T. Mason, "Contact mode guided sampling-based planning for quasistatic dexterous manipulation in 2D," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 6520–6526.

[16] X. Cheng, E. Huang, Y. Hou, and M. T. Mason, "Contact mode guided motion planning for quasidynamic dexterous manipulation in 3D," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 2730–2736.

[17] J. Liang, X. Cheng, and O. Kroemer, "Learning preconditions of hybrid force-velocity controllers for contact-rich manipulation," in *Proc. Conf. Robot Learn.*, 2022.

[18] T. Pang, H. T. Suh, L. Yang, and R. Tedrake, "Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models," *IEEE Trans. Robot.*, 2023.

[19] G. Lee, T. Lozano-Pérez, and L. P. Kaelbling, "Hierarchical planning for multi-contact non-prehensile manipulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 264–271.

[20] B. Aceituno and A. Rodriguez, "A hierarchical framework for long horizon planning of object-contact trajectories," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 189–196.

[21] W. Zhou and D. Held, "Learning to grasp the ungraspable with emergent extrinsic dexterity," in *Proc. IEEE Int. Conf. Robot. Autom. Workshop: Reinforcement Learn. Contact-Rich Manipulation*, 2022.

[22] W. Zhou, B. Jiang, F. Yang, C. Paxton, and D. Held, "HACMan: Learning hybrid actor-critic maps for 6D non-prehensile manipulation," in *Proc. 7th Annu. Conf. Robot Learn.*, 2023.

[23] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," in *Nature*, vol. 529, pp. 484–489, 2016.

[24] L. Amatucci, J.-H. Kim, J. Hwangbo, and H.-W. Park, "Monte Carlo tree search gait planner for non-gaited legged system control," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 4701–4707.

[25] H. Zhu, A. Meduri, and L. Righetti, "Efficient object manipulation planning with Monte Carlo tree search," *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2023.

[26] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo tree search: A review of recent modifications and applications," *Artif. Intell. Rev.*, vol. 56, pp. 2497–2562, 2023.

[27] C. Eppner, R. Deimel, J. Alvarez-Ruiz, M. Maertens, and O. Brock, "Exploitation of environmental constraints in human and robotic grasping," *Int. J. Robot. Res.*, vol. 34, no. 7, pp. 1021–1038, 2015.

[28] J. Lee, W. Jeon, G.-H. Kim, and K.-E. Kim, "Monte-Carlo tree search in continuous action spaces with value gradients," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 4561–4568.

[29] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver, "Learning and planning in complex action spaces," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 4476–4486.

[30] J. Lee et al., "DART: Dynamic animation and robotics toolkit," *J. Open Source Softw.*, vol. 3, no. 22, 2018, Art. no. 500.

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

[32] A. Gupta et al., "Extrinsic dexterous manipulation with a direct-drive hand: A case study," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 4660–4667.

[33] A. Bhatia, A. M. Johnson, and M. T. Mason, "Direct drive hands: Force-motion transparency in gripper design," in *Proc. Robot.: Sci. Syst.*, 2019.

[34] M. A. Roa and R. Suárez, "Grasp quality measures: Review and performance," *Auton. Robots*, vol. 38, no. 1, pp. 65–88, 2015.

[35] B. Calli et al., "Yale-CMU-Berkeley dataset for robotic manipulation research," *Int. J. Robot. Res.*, vol. 36, no. 3, pp. 261–268, 2017.

[36] S. Patil, T. Tao, T. Hellebrekers, O. Kroemer, and F. Z. Temel, "Linear delta arrays for dexterous distributed manipulation," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2023, pp. 10324–10330, doi: 10.1109/ICRA48891.2023.10160578.