HDRM: A Resolution Complete Dynamic Roadmap for Real-Time Motion Planning in Complex Scenes

Yiming Yang, Wolfgang Merkt, Vladimir Ivan, Zhibin Li, and Sethu Vijayakumar

***Abstract*—In this letter, we first theoretically prove the conditions and boundaries of resolution completeness for deterministic roadmap methods with a discretized workspace. A novel variant of such methods, the hierarchical dynamic roadmap (HDRM), is then proposed for solving complex planning problems. A unique hierarchical structure to efficiently encode the configuration-to-workspace occupation information is introduced and allows the robot to check the collision state of tens of millions of samples on-the-fly—the number of which was previously strictly limited by available memory. The hierarchical structure also significantly reduces the time for path searching, hence, the robot is able to find feasible motion plans in real-time in extremely constrained environments. A rigorous benchmarking shows that HDRM is robust and computationally fast compared with classical dynamic roadmap methods and other state-of-the-art planning algorithms. Experiments on the seven degree-of-freedom KUKA LWR robotic arm integrated with live perception further validate the effectiveness of HDRM in complex environments.**

***Index Terms*—Motion planning, dynamic roadmap, realtime planning, collision avoidance.**

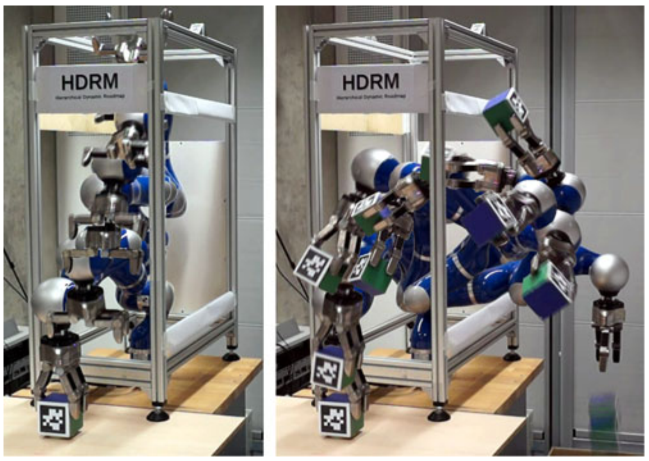

Fig. 1. The 7-DoF KUKA LWR robot with a SCHUNK Dexterous Hand operating inside a cage. Left: grasping the target from upright posture; right: dropping the object to the side.

I. Introduction

MOTION planning is one of the fundamental problems in robotics and involves automatically finding a sequence of configurations that take the robot from a start to a goal pose. Generally, motion planning can be categorized into optimization- and sampling-based methods. Optimization-based approaches [1], [2] generate optimal trajectories with respect to cost functions, but may get stuck in local minima and fail to produce a valid solution when the problem is non-convex or ill-defined. On the other hand, sampling-based algorithms [3]–[6] promise to solve complex problems by sampling globally in the configuration space.

In sampling-based algorithms, collision checking is usually the most expensive operation and reportedly consumes up to 90–95% of the planning time [7]. Lazy collision checking is used to delay the collision checking until it is needed or limit it to particular regions [8]. However, these techniques only reduce the collision checking time indirectly by reducing the number of calls rather than the actual computation time of the collision checking function. Parallel implementations for collision checking and motion planning have been proposed [9], but these approaches focus on parallelization and system implementation based on existing algorithms.

In contrast, the Dynamic Roadmap (DRM) [10], an extension to the probabilistic roadmap (PRM) [4], algorithmically reduces the collision checking time by encoding configuration-to-workspace occupation information. Given different environments, the DRM can efficiently remove invalid edges and form a valid subset of the full roadmap. Subsequently, search algorithms can proceed without considering collision checking since the remaining vertices and edges are all collision-free. However, encoding the occupation information requires to store a significant amount of data which needs to be loaded into memory during run-time. In the early work [10], [11], the low amount of available memory allowed storing only small roadmaps with limited number of vertices and edges. Without enough vertices and edges to densely cover the configuration space, the DRM achieves very low planning success rates [11], [12]. The success rate is closely tied with the term *Completeness*. An algorithm is considered complete if for any input it correctly reports whether there is a solution or not. If a solution exists, it must return one in finite time [13]. Optimization-based methods are incomplete due to local minima. Unfortunately, sampling-based algorithms, such as PRM and Rapidly-exploring Random Tree (RRT), are also incomplete. A weaker notion of completeness called *probabilistic completeness* is used to describe random sampling-based

Manuscript received June 2, 2017; accepted October 22, 2017. Date of publication November 15, 2017; date of current version December 11, 2017. This letter was recommended for publication by Associate Editor M. Morales and Editor N. Amato upon evaluation of the reviewers' comments. This work was supported in part by the Engineering and Physical Sciences Research Council under Grant EP/L016834/1 and in part by the Centre for Doctoral Training in Robotics and Autonomous Systems program. *(Corresponding author: Yiming Yang.)*

The authors are with the Institute of Perception, Action and Behaviour, School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, U.K. (e-mail: yiming.yang@ed.ac.uk; wolfgang.merkt@ed.ac.uk; v.ivan@ed.ac.uk; zhibin.li@ed.ac.uk; sethu.vijayakumar@ed.ac.uk).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LRA.2017.2773669

algorithms. This means that with enough samples (possibly infinite number of samples), the probability of finding a solution asymptotically converges to one. Another term, *resolution completeness*, is used if an algorithm guarantees to find a solution in finite time; however, if a solution does not exist, the algorithm may run forever by incrementally increasing the sampling resolution. Alternatively, the algorithm may terminate in finite time by reporting no solution at a certain resolution, although one may exist at a finer resolution. In more recent work [14], a DRM consisting of up to a million vertices can be stored and updated efficiently by using more memory and powerful GPUs. However, even one million vertices are not enough for dense coverage of 6–7 dimensional configuration spaces. Instead of claiming completeness, a small size DRM was built on customized hardware for solving very specific tasks [15]. In contrast to a roadmap, a Dynamic Reachability Map only stores vertices but not edges and is able to find valid reaching poses for high-dimensional floating-base robots [16], [17]. No matter whether we store a small roadmap with limited vertices and edges, or only the vertices, these methods eventually need to sacrifice completeness for a manageable storage size.

In order for robust motion planning in complex environments to be practical, a tremendous number of vertices and edges are required; yet storing such a roadmap is infeasible on commodity computers with current technology. However, we observe that the memory consumption for storing a DRM can be greatly reduced by exploiting the topology of the robot. On this basis, in this letter we propose a new resolution complete planning algorithm, the *Hierarchical Dynamic Roadmap* (HDRM), with the following contributions:

1) Theoretical proof of resolution completeness of any deterministic roadmap with a discretized workspace;
2) A novel formulation for encoding the occupancy information of roadmap vertices and eliminating the necessity of computing/storing edges, which enables efficient storage of roadmaps with tens of millions of vertices;
3) A novel hierarchical structure that allows the roadmap to efficiently remove colliding samples, which in turn enables real-time motion planning in complex scenes.

Extensive benchmarking shows that the HDRM is able to find valid solutions in extremely constrained conditions within a few milliseconds or less, which could not be achieved previously by classical DRM and other state-of-the-art algorithms. Experiments on a KUKA LWR arm further demonstrate HDRM's capability to solve real-world problems.

II. Dynamic Roadmap

A. Preliminaries

Let $\mathcal{C} \in \mathbb{R}^N$ be the configuration space of a N-DoF robot and $\mathbf{q} \in \mathcal{C}$ be a state in configuration space. Let $\mathcal{C}_{obs}$ represent the obstacles and $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$ the collision-free region. A classical PRM contains a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \in \mathcal{C}_{free}$ are the vertices and $\mathcal{E} \subset \mathcal{C}_{free}$ are the edges that connect two neighboring vertices, as highlighted in Fig. 2 (left). These vertices and edges are generated during off-line pre-processing. During the on-line planning phase, given start and goal states $\mathbf{q}_{start}, \mathbf{q}_{goal}$, we first find these two vertices $\mathcal{V}_{start}$ and $\mathcal{V}_{goal}$ which are closest to the start and goal states respectively. Then a graph search algorithm, such as A* [18], is deployed to find a path in the roadmap connecting $\mathcal{V}_{start}$ and $\mathcal{V}_{goal}$. However, the

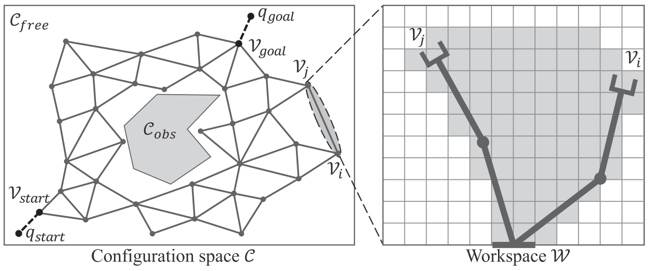

Fig. 2. Preliminaries of DRM. Left: probabilistic roadmap in configuration space; right: workspace swept volume of an edge.

pre-generated vertices and edges may not be valid in unknown and non-static environments. The validity of the stored vertices and edges must be checked, and in many cases we need to sample new collision-free configurations during the on-line phase which is very time consuming.

The dynamic roadmap (DRM) is a variation of the PRM proposed by Leven and Hutchinson [10]. The DRM is *dynamic* in the sense that the graph $\mathcal{G}$ can be dynamically updated in different environments. The invalid vertices and edges can be efficiently identified and removed, with the remaining ones forming a new graph of only valid vertices and edges. This reduced graph is ready for path searching algorithms without considering collision checking. The key feature of DRM is a configuration-to-workspace mapping, as highlighted in Fig. 2 (right). One can find the list of discretized workspace voxels which an edge occupies, referred to as the swept volume. If one or more of the voxels in the swept volume are in collision, then the corresponding edge becomes invalid. In practice, it is inefficient to check the swept volume of all edges exhaustively. Instead, the occupation information is stored per each workspace voxel, i.e., each voxel stores a list of edges that sweep through this voxel. In a new environment, we first find all the voxels that are occupied by the obstacles in the environment. Then, by iterating through the occupation lists of these invalid voxels, all the invalid edges can be found and then removed accordingly.

The main observed limitation of the existing DRM method is its low success rate [11], [12]. Although the success rate can be improved for particular tasks by carefully selecting the samples [15], those approaches can not solve generic problems, therefore they are not (resolution) complete.

B. Resolution Completeness of a Deterministic Roadmap With Discretized Workspace

In this section, we provide theoretical proof of the conditions and boundaries of resolution completeness for deterministic roadmap methods with a discretized workspace. Note that the proposed HDRM is a special case of deterministic roadmap with discretized workspace, therefore the following general proof also applies to HDRM.

The work in [19] has proven that a deterministic roadmap, such as a uniform Sukharev grid, is resolution complete. Let Ψ be the subset of the power set of $\mathcal{C}$ corresponding to all open subsets that can be constructed with algebraic constraints (see [20]), and $\Psi(x)$ for $x \in (0, \infty)$ be the set of all $\mathcal{C}_{free}$ with the *width* of $\mathcal{C}_{free}$, $w(\mathcal{C}_{free}) \geq x$. The width x can be viewed as the minimum width of a passable corridor in the collision-free portion of the configuration space, as illustrated in Fig. 3(a).

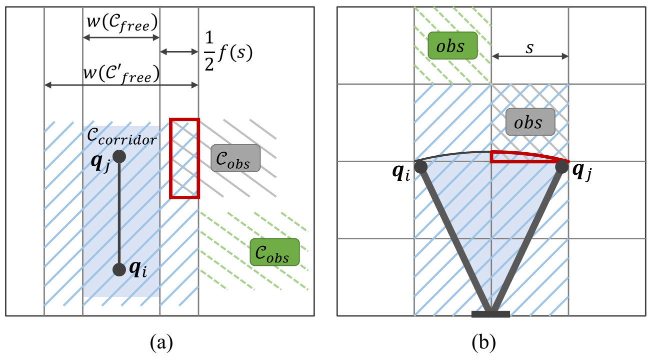

Fig. 3. Illustration of additional volume an obstacle in a discretized workspace occupies in the configuration space. An algorithm is resolution complete if it accounts for the additional increase corridor width $f(s)$ due to discretization. (a) Configuration Space. (b) Workspace.

Lemma 1: After M iterations, a deterministic DRM is resolution complete for all $\mathcal{C}_{free} \in \Psi(4b(N)M^{-\frac{1}{N}} + f(s))$, where M is the number of samples, N is the dimension of the configuration space, s is the resolution of the workspace, $b(N)$ is a factor that depends on the sampling method ($b(N) = 1$ for HDRM) and $f(s)$ is a robot-dependent function.

Proof: It has been proven in [19] that, after M iterations, a deterministic roadmap planner is resolution complete for all $\mathcal{C}_{free} \in \Psi(4b(N)M^{-\frac{1}{N}})$, without workspace discretization. However, as shown in Fig. 3, with a discretized workspace with voxel size $s > 0$, the corresponding $\mathcal{C}_{corridor}$ and $\mathcal{C}_{obs}$ are both inflated by maximum $\frac{1}{2}f(s)$ due to the workspace discretization, where $\mathcal{C}_{corridor}$ is the narrowest corridor in the configuration space. The inflated $\mathcal{C}_{corridor}$ and $\mathcal{C}_{obs}$ must not intersect. Thus, after discretizing the workspace, the algorithm is able to solve problems for $\mathcal{C}'_{free}$ where $w(\mathcal{C}'_{free}) \geq w(\mathcal{C}_{free}) + f(s)$. Thus, the algorithm is resolution complete for all $\mathcal{C}_{free} \in \Psi(4b(N)M^{-\frac{1}{N}} + f(s))$. ∎

To calculate $f(s)$, let $V(e)$ denote the voxelized swept volume of an edge e, and $\mathcal{C}_{V(e)}$ be the $\mathcal{C}$ space region occupied by $V(e)$, then the *width* of $\mathcal{C}'_{free}$ can be defined as

$$w(\mathcal{C}'_{free}) = 4b(N)M^{-\frac{1}{N}} + f(s) = \sup_{e\in\mathcal{E}}\{w(\mathcal{C}_{V(e)})\}, \quad (1)$$

which yields

$$f(s) = \sup_{e\in\mathcal{E}}\{w(\mathcal{C}_{V(e)})\} - 4b(N)M^{-\frac{1}{N}}. \quad (2)$$

It is practically difficult to pre-determine $f(s)$ before sampling as it depends not only on the number of samples M and resolution s, but also on the robot's geometric shape.

C. Limitations: Curse-of-Dimensionality

According to *Lemma 1*, although certain level of resolution completeness can be guaranteed for any given workspace and configuration space resolution, the algorithm would not be practically useful if $w(\mathcal{C}'_{free})$ is too wide. To achieve a smaller $w(\mathcal{C}'_{free})$, one could either increase the number of samples or the workspace resolution. However, both options are restricted by the available memory size.

For example, consider a 6-DoF robotic manipulator where $K = 15$ different discretization values are chosen for each joint. In this case we would generate $M = 15^6 \approx 11.4$ million vertices to build a DRM. If each vertex occupies around 400 workspace voxels, that means the occupation lists may contain billions of indices (`unsigned int`[1]), which is too expensive to store on commodity hardware. Note that this is only a 6-DoF case and the problem scales exponentially with the dimension of the configuration space (*curse-of-dimensionality*).

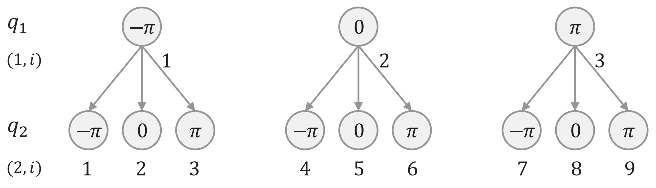

Fig. 4. A 2-DoF example of HDRM with $K_1 = K_2 = 3$, the number of roadmap vertices stored in the structure is $K_1 \times K_2 = 9$.

III. Hierarchical Dynamic Roadmap

To obtain a resolution complete method with a smaller $w(\mathcal{C}'_{free})$, we introduce the Hierarchical Dynamic Roadmap by exploiting the inherent hierarchical structure of the robot, which in turn enables us to store a roadmap with tens of millions of vertices within a limited amount of memory.

A. Hierarchical Configuration Structure

Let $[b_{n,l}, b_{n,u}]$ be the lower and upper bounds of joint $n \in N$ of a N-DoF robot. An even discretization of the n-th joint to $K_n \in \mathbb{N}$ values results in configurations

$$q_n(k_n) = b_{n,l} + (k_n - 1) \times \frac{b_{n,u} - b_{n,l}}{K_n - 1} \quad (3)$$

where $k_n \in \{1, \ldots, K_n\}$. Let

$$\mathbf{k}(n) = [k_1, \ldots, k_n] \quad (4)$$

be a n-dimensional vector containing the joint value indices for the first n joints, and

$$\mathbf{q}(\mathbf{k}(n)) = [q_1(k_1), \ldots, q_n(k_n)] \quad (5)$$

be a n-dimensional vector contains the actual joint values corresponding to $\mathbf{k}(n)$. The full N-dimensional robot configuration can be retrieved given $\mathbf{k}(N)$. For example, as shown in Fig. 4, consider a 2-DoF robot, where the range of motion of each joint is $[-\pi, \pi]$. Given $K_1 = K_2 = 3$, we have $q_1(1) = q_2(1) = -\pi$, $q_1(2) = q_2(2) = 0$, and $q_1(3) = q_2(3) = \pi$. Then, $\mathbf{k}(2) = [1,1]$ gives the robot configuration $\mathbf{q} = [-\pi, -\pi]$, $\mathbf{k}(2) = [2,3]$ gives another robot configuration $\mathbf{q} = [0, \pi]$, and $\mathbf{k}(1) = [2]$ gives the first joint value $q_1 = 0$.

The data structure stores the equivalent of $M = \prod_1^N K_n$ vertices. The robot configurations can be accessed with $\mathbf{k}(n)$, however, the vertices in the roadmap are indexed with one integer index $i \in M$. Let $\mathcal{H} : (n,i) \mapsto \mathbf{k}(n)$ be the map from pair (n,i) to $\mathbf{k}(n)$, and $\mathcal{H}^{-1} : \mathbf{k}(n) \mapsto (n,i)$ be the corresponding inverse map. Given an index i and level n, the first n indices $\mathbf{k}(n) = \mathcal{H}(n,i)$ can be efficiently calculated using Algorithm 1. Similarly, given hierarchical indices $\mathbf{k}(n)$, the corresponding (n,i) can also be found by Algorithm 2.

[1]The size of one `unsigned int` is 4 Byte on 64-bit operating systems. Hence storing indices for the occupation lists for these vertices in this example requires $4 \times 400 \times 11.4 \times 10^6$ Byte $\approx$ 17 GB of memory. 400 is the average number of voxels occupied by one sample of the LWR robot at $s = 5$ cm.

Algorithm 1: Get Hierarchical Indices From Integer Index.

```
Require: Dimension level $n$, vertex index $i$
Ensure: Hierarchical indices $\mathbf{k}(n)$
1: Quotient= $i$
2: While $n > 1$ do
3:    Quotient, Remainder=Division(Quotient,$\prod_1^n K_n$)
4:    $k_n$ =Remainder
5: $k_1$ =Quotient
   return $\mathbf{k}(n) = [k_1, \ldots, k_n]$
```

Algorithm 2: Get Integer Index From Hierarchical Indices.

```
Require: Hierarchical indices $\mathbf{k}(n) = [k_1, \ldots, k_n]$
Ensure: Dimension level $n$, vertex index $i$
1: $i = 0$
2: for $l \in \{1, \ldots, n-1\}$ do
3:    Counter $= 1$
4:    for $j \in \{l+1, \ldots, n-1\}$ do
5:       Counter $=$ Counter $\times K_j$
6:    $i = i +$ Counter $\times k_l$
7: $i = i + k_n$
   return $n, i$
```

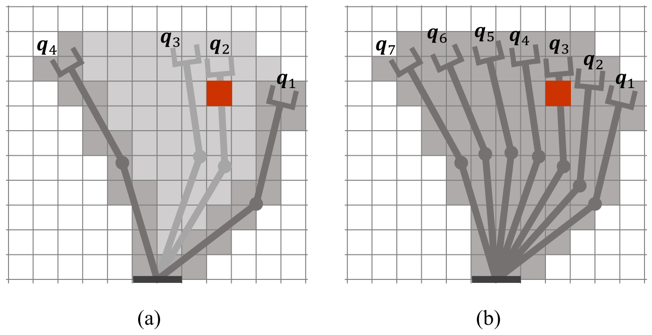

Fig. 5. Comparison of classical and hierarchical DRM. (a) A long edge $\mathcal{E}(\mathbf{q}_1, \mathbf{q}_4)$ in classical DRM sweeps through a large number of workspace voxels. (b) Dense vertices and short edges in HDRM. (a) Classical DRM, (b) Hierarchical DRM.

The hierarchical structure is a multi-resolution Sukharev grid [19] with $b(N) \equiv 1$, thus the hierarchical roadmap is resolution complete for $\mathcal{C}_{free} \in \Psi(4M^{-\frac{1}{N}} + f(s))$. As we will show in Section III-C, such a structure enables a hierarchical way for storing the configuration-to-workspace occupation information that dramatically reduces the memory consumption.

B. Removal of Swept Volumes

There are two types of occupation information: the occupation voxels of a vertex and the swept volume of an edge (dark and light grey voxels in Fig. 5 respectively). In the classical DRM algorithm, the edge is invalidated if one or more voxels in the swept volume are in collision. However, there will be many sub-edges still valid in the cases where only very few of the voxels are in collision. For example, in Fig. 5(a), if only the red voxel is in collision, the long edge $\mathcal{E}(\mathbf{q}_1, \mathbf{q}_4)$ is invalid while the sub-edge $\mathcal{E}(\mathbf{q}_3, \mathbf{q}_4)$ is still valid. Yet, the whole edge is considered invalid as these sub-edges are not stored in the roadmap. This is the underlying reason for planning failures and the low success

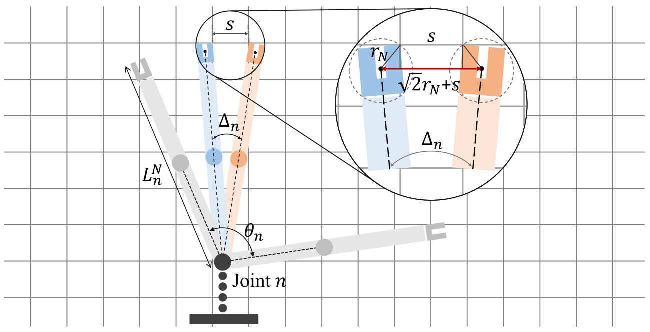

Fig. 6. Illustration of the maximum discretization step Δ_n the n-th joint can take without violating (6).

rate of the classical DRM method, i.e., the $f(s)$ in (2) is too large to make it practically useful. In fact, we argue that storing swept volumes does not utilize available resources well as this information is used only for collision checking and not the actual path planning. Therefore, we propose to use the memory for storing more vertices and edges instead.

In our method, we store only the occupation voxels of the vertices excluding the swept volumes of the edges, while still being able to check the collision status of both vertices and edges. Let O_a be the occupation voxels of a vertex a, and $O_{a,b}$ be the swept volume of edge $\mathcal{E}(a,b)$. If two vertices a, b are very close and the edge is so short that

$$O_{a,b} = O_a \cup O_b, \tag{6}$$

then we do not need to store the swept volume of the edge since it can be represented by the occupation voxels of these two end-point vertices, as illustrated in Fig. 5(b). The edge $\mathcal{E}(a,b)$ is collision-free if vertices a, b are collision-free, and vice versa. This ensures that a colliding workspace voxel only affects those corresponding short edges without invalidating others. A lower bound of K_n needs to be met in order to achieve such roadmap density.

Let $\theta_n = b_{n,u} - b_{n,l}$ be the range of motion of joint n, l_n and r_n be the approximate length and radius of n-th robot link, as illustrated in Fig. 6. For joint n, set all the subsequent/child joints such that the rest of the robot kinematic chain is fully extended. We assume that only one joint moves at a time, in which case, the distance between the n-th link of two neighboring configurations must not be greater than $s + \sqrt{2}r_n$, so that two end-effector links occupy the same or neighboring workspace voxels in order to satisfy (6). Hence, the inequality constraint shall be

$$\sup_{n \le k \le N} \left\{ \frac{s + \sqrt{2}r_k}{2\pi L_n^k} \right\} \ge \frac{\Delta_n}{2\pi}, \tag{7}$$

where $L_n^k = \sum_{j=n}^{j=k} l_j$ is the fully extended length from link n to link $k, k \ge n$. Rearranging terms yields

$$\Delta_n \le \sup_{n \le k \le N} \left\{ \frac{s + \sqrt{2}r_k}{L_n^k} \right\}. \tag{8}$$

So joint n should have evenly distributed values of

$$K_n = \left\lceil \frac{\theta_n}{\Delta_n} + 1 \right\rceil = \left\lceil \theta_n \left(\sup_{n \le k \le N} \left\{ \frac{s + \sqrt{2}r_k}{L_n^k} \right\} \right)^{-1} + 1 \right\rceil \tag{9}$$

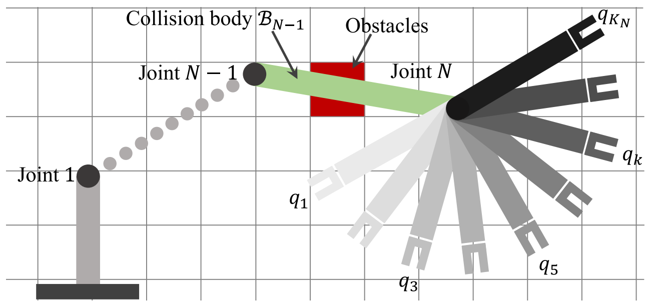

Fig. 7. Illustration of hierarchical occupation lists.

within its range of motion. We choose the minimum valid value for each K_n which already guarantees resolution completeness for a certain workspace voxel resolution s. Greater K_n only introduce more vertices and edges that increase memory consumption and slow down the search process.

The swept volumes can be removed if (9) is true for all joints. Furthermore, the information of the hundreds of millions of edges itself can be removed as well, because all the edges can be calculated analytically from the hierarchical structure. A N-dimensional configuration $\mathbf{k}(N)$ has $2 \times N$ neighbors (apart from the ones at the boundary of the range of motion), each of which forms an edge with $\mathbf{k}(N)$. As the edges have no direction, a N-dimensional HDRM with M vertices contains roughly $N \times M$ edges. It is also possible to allow full connectivity to all neighbors and thus the ability to move multiple joints at a time, but it will require a higher sampling density to achieve resolution completeness with the same $\Psi(x)$. Alternatively, one can also apply smoothing or trajectory optimization techniques to further optimize the trajectory to obtain multi-joint movement.

C. Hierarchical Occupation Lists

Section III-A described how to create a hierarchical structure to efficiently store tens of millions of configurations, and Section III-B explained why and how to remove the swept volumes and edges. The final step involves processing and storing the occupation lists of all vertices. However, when the roadmap contains tens or potentially hundreds of millions of vertices, their occupation lists are too expensive to store using classical methods. Next, we discuss how to exploit the hierarchical structure to resolve this problem.

Let $\mathcal{B}_n$ be the collision body between joint n and $n+1$. Consider K_N configurations with identical values for the first $N-1$ joints but only differing at the last joint, as illustrated in Fig. 7. These K_N configurations are invalid if $\mathcal{B}_{N-1}$ is in collision at the red voxel. In the classical DRM method, the red voxel's occupation list needs to store K_N indices to encode this information where each index corresponds to a particular configuration—which is very inefficient.

Instead of storing integer indices $i \in M$ for each configuration, we store a list of pairs (n, i), where $i \in \prod_1^n K_n$ and $n \in N$. A pair (n, i) is added to a workspace voxel v's occupation list if $\mathcal{B}_n$ at configuration $\mathbf{k}(n) = \mathcal{H}(n, i)$ occupies this voxel. In Fig. 7, when the red voxel is in collision with the environment, based on the pair $(N-1, i)$, we can invalidate the i-th vertex of level $N-1$ of the hierarchical structure. It is clear that all these K_N configurations are invalid since the first $N-1$ joints have already caused body $\mathcal{B}_{N-1}$ to be in collision. Hence, we can encode the occupation information of K_N configurations using only two rather than K_N indices. Consider another case with $K_2 \times \cdots \times K_N$ vertices, which could be millions, that have same value k_1 for the first joint but differ at all other joints. If k_1 puts $\mathcal{B}_1$ to a colliding position with the environment, then the millions of vertices with same k_1 index are all invalid. In such case, we can more efficiently use only a pair $(1, k_1)$ instead of millions of indices to encode the occupation information of all these vertices. As we will show later in Section IV-B, using the hierarchical structure and this novel indexing technique, we can significantly reduce the memory required for storing the occupation information.

Algorithm 3: Generate Hierarchical Occupation Lists.

```
**Require:** Robot model $\mathcal{R}$, voxelized workspace $\mathbb{V}$
**Ensure:** Hierarchical occupation lists $\mathcal{O}_v, v \in \mathbb{V}$
1:  for $v \in \mathbb{V}$ do
2:      Occupation list $\mathcal{O}_v = \emptyset$
3:  for $n \in \{1, \ldots, N\}$ do
4:      for $i \in \{1, \ldots, \prod_1^n K_n\}$ do
5:          $\mathbf{k}(n) = \mathcal{H}(n, i)$
6:          Set first $n$ joints of $\mathcal{R}$ to $\mathbf{q}(\mathbf{k}(n))$
7:          if $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ are NOT in self-collision then
8:              $V$ = findBodyOccupiedVoxels($\mathbb{V}, \mathcal{R}, \mathcal{B}_n$)
9:              for $v \in V$ do
10:                 $\mathcal{O}_v = \mathcal{O}_v \cup \{(n, i)\}$
11:         else
12:             Set vertex $(n, i)$ as default invalid
13: for $v \in \mathbb{V}$ do
14:     for $n = N$ to 1 do
15:         $\mathbf{O}$ = extractListOfDimension($\mathcal{O}_v, n$)
16:         Remove duplicated indices and sort $\mathbf{O}$
17:         for $O_i \in \mathbf{O}$ do
18:             if $O_i$ mod $K_n = 0$ & $O_{i+K_n} = O_i + K_n$ then
19:                 $\mathcal{O}_v = \mathcal{O}_v \backslash \{(n, p) | p \in [O_i, \ldots, O_{i+K_n}]\}$
20:                 if $n > 1$ then
21:                     $\mathbf{k}(n) = [k_1, \ldots, k_n] = \mathcal{H}(n, O_i)$
22:                     $\mathcal{O}_v = \mathcal{O}_v \cup \{(n-1, \prod_{p=1}^{p=n-1} k_p)\}$
23:                 $i = i + K_n - 1$
```

Algorithm 3 shows the details of generating the full occupation lists for all workspace voxels. First, given the size of the workspace and grid resolution s, a set of workspace voxels $\mathbb{V}$ can be generated. Each voxel $v \in \mathbb{V}$ is associated with an empty occupation list $\mathcal{O}_v$. Lines 3–12 generate the initial hierarchical occupation lists, but we can compress these to further reduce memory storage (lines 13–23). The compression is based on the fact that some robots, or part of the robots, are axially symmetric, which means that rotating a joint will not change the occupation list of the subsequent links. More generally, if the collision body $\mathcal{B}_n$ of K_n vertices ($xK_n + 1$ to $xK_n + K_n$, $x \in \mathbb{N}$) from the same sub-tree of level n occupies a voxel v, then the occupation list of v needs to store only one pair of $(n-1,)$ rather than K_n pairs of $(n, \cdot)$, because the first $n-1$ joints already make $\mathcal{B}_n$ unavoidably occupy voxel v. We *"promote"* the occupation list from level n to $n-1$ if such axial symmetry occurs.

D. Motion Planning Using HDRM

With the HDRM created and loaded, our goal is to efficiently solve motion planning queries online in different environments. The three main steps are as follows.

TABLE I
ROBOT KINEMATIC ANALYSIS FOR CREATING HDRM.

Robot	s (m)	K_n
UR5	0.1	$\{37, 36, 21, 9, 7, 1\}$
	0.05	$\{52, 51, 30, 12, 9, 1\}$
LWR	0.1	$\{35, 20, 21, 10, 7, 2, 1\}$
	0.05	$\{49, 27, 29, 14, 10, 2, 1\}$

1) Collision Update: First, we create a voxelized environment to represent the discretized workspace, and then apply conventional collision checking on this voxelized environment against the real collision environment to find the list of voxels that are occupied by the obstacles. For each occupied voxel, we iterate though its occupation lists and invalidate vertices in the hierarchical structure accordingly.

2) Connecting Start/Goal to Roadmap: The start and goal vertices $\mathcal{V}_{start}, \mathcal{V}_{goal}$ are required for the graph search algorithm, which are the closest valid vertices to the start and goal configurations $\mathbf{q}_{start}, \mathbf{q}_{goal}$. Traditionally, this involves comparing the distance between a given configuration $\mathbf{q}$ and all vertices in the roadmap and finding the one with the shortest distance. As we will show later, such process could be very slow for a roadmap with a large number of vertices. In our approach, instead of searching though all vertices, we can analytically compute the closest one. Given a configuration $\mathbf{q}$, we can easily get the closest hierarchical configuration $\mathbf{k}_{closest}(N)$. Then, the index of closest vertex $\mathcal{V}_{closest}$ can be found using Algorithm 2.

3) Shortest Path Searching: The last step is to find a valid path on the roadmap connecting $\mathcal{V}_{start}$ and $\mathcal{V}_{goal}$. The A* shortest path searching algorithm is used. We implemented the sequential version of A* using a single CPU-thread. Parallelization is not the main focus of this letter, however, we believe that parallel version of Dijkstra or A* algorithms would be more efficient [14].

IV. EXPERIMENTS

The proposed HDRM method is benchmarked against the classical DRM approach and standard sampling-based planners (SBP) in various scenarios using two different fixed-base manipulators: a 6-DoF Universal Robot UR5 and a 7-DoF KUKA LWR. The evaluation was performed using an Intel Core i7-6700 K 4.0 GHz CPU with 32 GB 2133 MHz RAM, and the KUKA LWR for hardware experiments.

A. Experimental Setup

Given the robot model, first K_n is calculated using (7)–(9), as shown in Table I. Two different workspace voxel resolutions are used, $s = 0.1$ m and $s = 0.05$ m. Smaller s leads to greater K_n and more samples are required to densely cover the space. For the KUKA LWR robot, the off-line construction time is roughly 30 minutes with $s = 0.1$ m and 5–6 hours with $s = 0.05$ m. We have also implemented classical DRM methods for comparison. To achieve resolution completeness, we generate the vertices by uniformly sampling in the configuration space and apply no roadmap compression technique. Three classical DRM datasets are created with different number of vertices: 1,000 (DRM$_a$),

TABLE II
COMPARISON BETWEEN CLASSICAL DRM AND HDRM

Robot & Method	No. Vertices	No. Edges	s (m)	Memory size (MB)
UR5 DRM	1,000	6,336	0.1	2.8
			0.05	13.6
	10,000	61,274	0.1	22.3
			0.05	104
	200,000	1,200,956	0.1	356
			0.05	1593
UR5	$\mathbf{1,762,236}$	$\mathbf{10,573,416}$	0.1	$\mathbf{8.5}$
HDRM	$\mathbf{8,592,480}$	$\mathbf{51,554,880}$	0.05	$\mathbf{145}$
LWR DRM	1,000	6,369	0.1	7.9
			0.05	33.4
	10,000	62,031	0.1	70
			0.05	280
	200,000	1,216,755	0.1	1239
			0.05	4793
LWR	$\mathbf{2,058,000}$	$\mathbf{14,406,000}$	0.1	$\mathbf{16.7}$
HDRM	$\mathbf{10,742,760}$	$\mathbf{75,199,320}$	0.05	$\mathbf{266}$

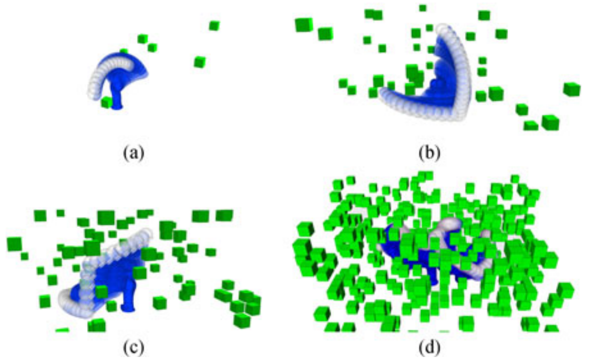

Fig. 8. Random problems in environments with different workspace obstacle densities. (a) Obstacle density 0.1%. (b) Obstacle density 0.5%. (c) Obstacle density 1%. (d) Obstacle density 5%. The highlighted trajectories are valid solutions found by HDRM.

10,000 (DRM$_b$) and 200,000 (DRM$_c$). A K-nearest neighbor search based on the configuration space Euclidean distance with $K = 10$ is then applied to find the edges in the roadmap.

B. Memory Requirements

As highlighted in Table II, the HDRM scales exponentially with roadmap size while the memory requirement is much lower compared to classical DRM. In the case of the UR5 robot with 0.1 m voxel size, HDRM can store over 1.7 million vertices and 10 million edges using only 8.5 MB of memory, which is even less than the memory required for classical DRM to store only 10,000 vertices. In the LWR scenario with 0.05 m voxel size, the HDRM stores over 10 million vertices and up to 75 million edges using only 266 MB of memory whereas the memory size for classical DRM to store 200,000 vertices is over 4.7 GB.

C. Evaluation of Motion Planning

We benchmarked the performance of eight candidate methods: three classical DRM methods, DRM$_a$, DRM$_b$ and DRM$_c$; four standard sampling-based planners, i.e., RRT, PRM, Single-

TABLE III
EVALUATION OF 8 CANDIDATE MOTION PLANNERS, SHOWING THE SUCCESS RATE OF SOLVING 1000 PROBLEMS IN RANDOM VALID ENVIRONMENTS, FOLLOWED BY THE AVERAGE SOLVING TIME OVER THE SUCCESSFUL CASES (IN MILLISECONDS)

Method		Obstacle density 0%		Obstacle density 0.1%		Obstacle density 0.5%		Obstacle density 1%		Obstacle density 5%	
		Success	Time (ms)	Success	Time (ms)	Success	Time (ms)	Success	Time (ms)	Success	Time (ms)
Standard SBP	RRT	100%	13.392	99%	22.708	92%	325.21	82%	1036.1	36%	1893.6
	PRM	100%	5.7416	100%	4.5041	100%	322.86	99%	656.09	34%	3386.4
	SBL	100%	6.8909	100%	14.775	100%	82.473	100%	273.96	31%	4439.4
	RRTConnect	100%	1.1930	100%	2.1220	100%	10.117	100%	48.926	74%	1723.7
Classical DRM	DRM$_a$	100%	0.1564	92.2%	0.7108	65.6%	0.7911	39.0%	1.2403	1.6%	-
	DRM$_b$	100%	0.3123	93.9%	0.8779	69.5%	1.0203	48.9%	1.5221	3.6%	-
	DRM$_c$	100%	3.4152	95.7%	4.3431	74.7%	6.5206	53.0%	9.2316	3.3%	-
Hierarchical DRM		100%	0.2759	100%	0.8574	100%	1.5813	100%	3.7152	100%	15.506

All algorithms are given 10 seconds to solve each problem.

Query Bi-directional Planning with Lazy Collision Checking (SBL) [8], and RRTConnect; and finally the proposed HDRM. For DRM/HDRM, the datasets of the LWR robot with 0.1 m voxel resolution are used. For standard sampling planners, we use the standard implementations from the OMPL library [21] and the FCL library [22] for explicit online collision checking with all parameters set to library defaults.

Five different categories of environments with random obstacles were created. From simple to complex, these environments have 0%, 0.1%, 0.5%, 1% and 5% of the whole workspace occupied by obstacles, where the latter four are illustrated in Fig. 8. The environments with 1% and 5% obstacle densities are extremely complicated for any kind of motion planning algorithm. We have created 1000 random valid problems for each category. Each problem has a valid start and goal states, and at least one trajectory connecting these two. To guarantee this, a random self-collision-free trajectory is generated in an empty space, which is then populated with obstacles that are not colliding with the trajectory. All algorithms are given 10 seconds to solve each problem. Since all problems are solvable, reporting no solution or exceeding the time limit is considered a failure.

Evaluation results are detailed in Table III. As a baseline, all algorithms achieve 100% success rate in free space. The success rate of the classical DRMs falls below 100% when the environment is populated with only a few obstacles (0.1% obstacle density). The SBP methods are generally slower due to explicit collision-checking for every sample which is very time consuming. In more complicated environments (0.05% and 1% obstacle densities), the success rate of classical DRM methods decreases significantly. SBP methods still achieve reasonable success rates, however, the planning time increases considerably. HDRM performs better compared to all other methods in complicated scenarios in terms of both success rate and planning time. In the extreme cases with 5% obstacle density, we do not show the average planning time for the classical DRM methods as the success rate is too low. All SBP methods also report lower success rates and much longer planning times. On the contrary, HDRM constantly achieves 100% success rate in these extremely constrained environments.

It is interesting that DRM$_c$ has a much smaller roadmap than HDRM, but takes longer time to find a solution even in free space. We break down the DRM/HDRM planning time into

TABLE IV
BREAKDOWN OF COMPUTATIONAL TIME (IN MICROSECONDS)

Obstacle density	Method	Roadmap update		Planning		Total
		Coll. check	Remove invalids	Connect roadmap	A* search	
0%	DRM$_a$	141.6	0	14.17	0.581	156.4
	DRM$_b$			170.1	0.626	312.3
	DRM$_c$			3273	0.698	3415
	HDRM			0.229	134.1	275.9
0.1%	DRM$_a$	694.8	1.665	13.77	0.557	710.8
	DRM$_b$		18.71	163.8	0.601	877.9
	DRM$_c$		435.0	3212	0.711	4343
	HDRM		17.48	0.267	144.9	857.4
1%	DRM$_a$	1212	13.92	13.32	0.607	1240
	DRM$_b$		145.5	163.5	0.745	1522
	DRM$_c$		4728	3290	1.021	9231
	HDRM		177.9	0.233	2325	3715

separate components, as in Table IV, and the time is given in microseconds. The collision check takes 141.6 microseconds in free space, which is basically the overhead of communication and function calls. We use FCL for explicit collision checking where the time increases as expected in more complicated environments. Classical DRMs with more vertices and edges require much longer time to remove invalid roadmap parts, whereas the HDRM is able to do so relatively faster, considering the enormous number of vertices and edges. Another expensive step of classical DRMs is connecting the roadmap, which increases exponentially with the number of vertices. After connecting the roadmap, running A* search is actually very fast since the roadmap size is relatively small. On the other hand, the time for connecting the roadmap is negligible for HDRM since the closest vertices can be analytically computed as elaborated in Section III-D2. However, the searching takes longer due to the enormous roadmap size.

D. Experimental Validation on Robot Hardware

We further validate the HDRM method on a 7-DoF KUKA LWR manipulator fitted with the SCHUNK Dexterous Hand

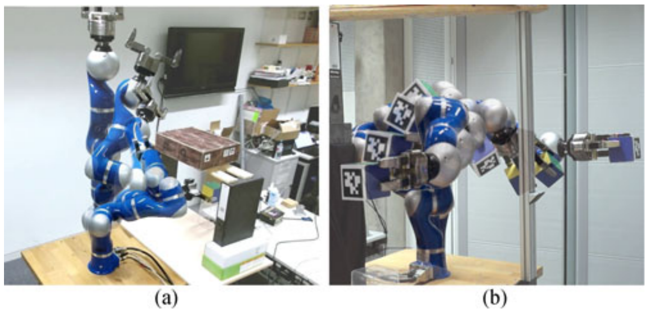

Fig. 9. Experiments on a 7-DoF KUKA LWR robotic arm fitted with a SCHUNK Dexterous Hand. (a) Reaching into a confined shelf. (b) Fetching object in distance.

2.0. The dataset was generated with $s = 0.05\,\text{m}$ voxel resolution. Four Microsoft Kinect One RGB-D sensors were fused to sense the environment and create an OcTree representation for collision checking. It shall be noted that due to the limitation of the out-of-the-box sensing devices and algorithms, the real experiment setup was limited to simpler environments compared with the simulation benchmark. In our supplementary video (`https://youtu.be/4AzbmiTI1iE`), we demonstrate challenging motions in three different, highly constrained environments: reaching into a confined shelf space and grasping a target object [see Fig. 9(a)]; retrieving an object through a frame [see Fig. 9(b)]; and moving an object from within a cage (see Fig. 1).

V. Conclusion

This letter first provided a theoretical proof for the resolution completeness of a deterministic roadmap with a discretized workspace. We then presented a novel such method, the Hierarchical Dynamic Roadmap (HDRM), for real-time motion planning in complex environments. The HDRM is able to encode large numbers of vertices and edges in a memory efficient manner that allows the algorithm to be resolution complete. An extensive benchmarking shows that HDRM can find valid motion plans in extremely complicated environments in real-time and empirically validates that the algorithm is resolution complete. Experiments on the KUKA LWR robot further demonstrate that our method is capable of incorporating live sensing information and providing collision-free trajectories suitable for tackling practical problems.

Both DRM and HDRM compute solutions in static environments, which can be different between planning queries but need to be static during execution—they inherently do not adapt to runtime changes. Since HDRM guarantees resolution completeness and is able to plan in real-time (few milliseconds or less), the future work will focus on the implementation of online adaptation/re-planning framework with feedback for applications such as real-time interaction between human and robot in a shared workspace similar to the work in [23].

References

[1] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1251–1270, 2014.

[2] V. Ivan, D. Zarubin, M. Toussaint, T. Komura, and S. Vijayakumar, "Topology-based representations for motion planning and generalization in dynamic environments with interactions," *Int. J. Robot. Res.*, vol. 32, no. 9-10, pp. 1151–1163, 2013.

[3] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, vol. 2, pp. 995–1001.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[5] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[6] Y. Yang, V. Ivan, W. Merkt, and S. Vijayakumar, "Scaling sampling-based motion planning to humanoid robots," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, 2016, pp. 1448–1454.

[7] D. Hsu and Z. Sun, "Adaptively combining multiple sampling strategies for probabilistic roadmap planning," in *Proc. IEEE Conf. Robot., Autom. Mechatronics*, 2004, vol. 2, pp. 774–779.

[8] G. Sánchez and J.-C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *Intl. J. Robot. Res.*, vol. 21, no. 1, pp. 5–26, 2002.

[9] J. Pan and D. Manocha, "GPU-based parallel collision detection for fast motion planning," *Intl. J. Robot. Res.*, vol. 31, no. 2, pp. 187–200, 2012.

[10] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. J. Robot. Res.*, vol. 21, no. 12, pp. 999–1030, 2002.

[11] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, vol. 5, pp. 4399–4404.

[12] A. Voelz and K. Graichen, "Distance metrics for path planning with dynamic roadmaps," in *Proc. Int. Symp. Robot.*, 2016, pp. 1–7.

[13] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[14] H. Schumann-Olsen, M. Bakken, Ø. H. Holhjem, and P. Risholm, "Parallel dynamic roadmaps for real-time motion planning in complex dynamic scenes," in *Proc. 3rd Workshop Robots Clutter*, 2014.

[15] S. Murray, W. Floyd-Jones, Y. Qi, D. Sorin, and G. Konidaris, "Robot motion planning on a chip," in *Proc. Robot., Sci. Syst.*, 2016.

[16] Y. Yang, V. Ivan, Z. Li, M. Fallon, and S. Vijayakumar, "iDRM: Humanoid motion planning with realtime end-pose selection in complex environments," in *Proc. IEEE Int. Conf. Humanoid Robots*, 2016, pp. 271–278.

[17] Y. Yang, W. Merkt, H. Ferrolho, V. Ivan, and S. Vijayakumar, "Efficient humanoid motion planning on uneven terrain using paired forward-inverse dynamic reachability maps," *IEEE Robot. Autom. Letters*, vol. 2, no. 4, pp. 2279–2286, Oct. 2017.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[19] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Intl. J. Robot. Res.*, vol. 23, no. 7–8, pp. 673–692, 2004.

[20] J.-C. Latombe, *Robot Motion Planning*. New York, NY, USA: Springer, 1991.

[21] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[22] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3859–3866.

[23] W. Merkt, Y. Yang, T. Stouraitis, C. E. Mower, M. Fallon, and S. Vijayakumar, "Robust shared autonomy for mobile manipulation with continuous scene monitoring," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2017, pp. 130–137.