Hierarchical Path Planner using Workspace Decomposition and Parallel Task-Space RRTs

George Mesesan[1], Máximo A. Roa[1], Esra Icer[2], and Matthias Althoff[2]

Abstract— **This paper presents a hierarchical path planner consisting of two stages: a global planner that uses workspace information to create collision-free paths for the robot end-effector to follow, and multiple local planners running in parallel that verify the paths in the configuration space by expanding a task-space rapidly-exploring random tree (RRT). We demonstrate the practicality of our approach by comparing it with state-of-the-art planners in several challenging path planning problems. While using a single tree, our planner outperforms other single tree approaches in task-space or configuration space (C-space), while its performance and robustness are comparable or better than that of parallelized bidirectional C-space planners.**

I. Introduction

Finding a collision-free path for a robot is a decades-old problem in robotics. Solutions have been proposed using mainly two approaches: sampling- and optimization-based planners. Sampling-based approaches work by randomly producing different robot configurations, verifying which ones are collision-free, and creating graph structures that can be queried for a trajectory between the initial and final robot configuration. For static scenarios where multiple queries are performed, probabilistic roadmaps (PRMs) [1] offer a good solution strategy. For single queries or non-static environments, rapidly-exploring random trees (RRTs) [2] have been proposed. These techniques are probabilistically complete, but often require a post-processing step to smooth the resulting jerky trajectories. On the other hand, optimization-based planners define a cost function to be minimized while traveling from an initial to a goal configuration [3]. These approaches suffer in general from problems such as non-convergence or local minima, typically associated with optimization methods. Also, on difficult path scenarios that require more exploration than exploitation, these algorithms might fail to find the solution.

Sampling-based path planners can search for a solution either in the configuration space (C-space) or in the task-space. While C-space planners use a simplified representation of the robot as a point in a $\mathbb{R}^{n_q}$ space, where n_q is the number of degrees of freedom (DoF) of the robot, the goal also needs to be specified as a point (sometimes as a region) in C-space. When a C-space goal point is directly available, or can be easily computed, a bidirectional RRT approach [4] is an efficient method of finding the solution. However, in many real-world path planning problems, a C-space goal point is not available. As an example, consider a redundant robotic manipulator whose task is to grasp an object. The task may be specified simply as the position of the end-effector, which corresponds to an infinite number of goal configurations. Some of these configurations may not be reachable from the starting configuration of the robot, while others may lead to convoluted paths in configuration space. Based on these considerations, task-space approaches [5], [6] try to find a path for the robot by applying probabilistic techniques (like RRT) directly in the lower-dimensional task-space $\mathbb{R}^{n_r}$ (in the example mentioned above, $n_r = 3$). A task-space tree can be grown using a Jacobian transpose [7] or a Jacobian pseudoinverse method [5]. An interesting bidirectional approach using both C-space and task-space trees was proposed in [8].

A common method of improving the efficiency of path planning algorithms consists in using information extracted from the robot workspace. One approach used by C-space RRT planners is to bias the tree exploration using the obstacle geometry [9], the medial axis of the free workspace [10], or spherical free-space regions computed on a generalized Voronoi graph [11]. An alternative approach is to partition the free workspace into cells, and use their connectivity information. Tunnels of free workspace are constructed in [12] by a sphere expansion algorithm. An approximate cell decomposition of the workspace is used in [13] with a method from image processing called watershed segmentation to identify narrow passages. Path planning problems involving robotic manipulators typically have simpler workspace topologies, which make it possible to use exact cell decompositions based on convex shapes [14], intersecting convex shapes [15], or a combination of convex volumes and generalized cones [16]. A hierarchical approach using workspace decomposition and sampling-based motion planning was proposed in [17] in the context of nonholonomic mobile robots navigating through maze-like environments. However, this method is not directly applicable to redundant manipulators with a fixed base, where collision-free C-space paths may require the end-effector to pass through previously visited regions of the workspace.

An alternative method of improving a path planner's performance is parallelization. Several parallel C-space RRT implementations have been proposed [18]–[20]. However, growing a C-space tree concurrently encounters two fundamental problems: the C-space tree is a global tree that requires synchronization of the data structure via semaphores

*This work has received funding from the People Programme (Marie Curie Actions) of the European Union Seventh Framework Programme FP7/2007- 2013/ under REA grant agreement number 608022.

[1]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Wessling, Germany. Contact: `george.mesesan@dlr.de`

[2]Department of Informatics, Technische Universität München, 85748 Garching, Germany

or message passing; and the C-space has no natural decomposition, leading to arbitrary partitioning schemes subject to heuristics. In contrast, task-space approaches can usually use a workspace decomposition method as a natural partitioning scheme for concurrent exploration. Based on this insight, we propose in this paper to decompose the workspace into disjoint convex shapes. Multiple task-space RRT planners can thus grow the search tree in parallel with no synchronization, each planner extending the tree within its own convex free-space. Furthermore, this approach makes it possible to explore the same task-space region for different configuration subspaces, which greatly increases the robustness of the overall planner compared to global task-space tree approaches, such as [5], as we show in the comparison section (Section IV).

Our goal is to obtain a planner that solves real-world path planning problems efficiently even in the case of robots with very high dimensionality, such as humanoid robots or serial-chain approximations to soft, deformable manipulators. To this end, we introduce a hierarchical path planner consisting of multiple local task-space RRT planners running in parallel, and a global planner that uses the free-space connectivity information for coordinating the local planners.

The rest of this paper is organized as follows: we formalize the path planning problem in Section II, and describe our path planning algorithm in detail in Section III. A comparison with state-of-the-art planners for several examples is presented in Section IV. Conclusions are drawn in Section V.

II. Problem Formulation

In this section, we formalize the path planning problem for serially connected manipulators in two- or three-dimensional workspaces. Let d denote the dimensionality of the physical space $\mathbb{R}^d$ containing the workspace, $d \in \{2, 3\}$. In this space, a *convex polytope* V is a set of points

$$V = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{x} \leqslant \mathbf{b}\}, \mathbf{A} \in \mathbb{R}^{n \times d}, \mathbf{b} \in \mathbb{R}^n, \qquad (1)$$

which is called *bounded* if it is contained in a ball of finite radius (i.e., $\exists \mathbf{c} \in \mathbb{R}^d, r \in \mathbb{R}_{>0}, \forall \mathbf{x} \in V : \|\mathbf{x} - \mathbf{c}\| < r$). Convex bounded polytopes are also called convex polygons in 2D, and convex polyhedra in 3D. This paper only uses convex and bounded polytopes, for brevity they will be simply referred to as polytopes hereafter.

For the path planning problem, the robot workspace W is considered to be a polytope. The workspace contains a set of obstacles $\{O_k\}_{k=1}^{n_O}$, which are themselves polytopes. Obstacles that are not polytopes (e.g., spheres) can be modeled as polytope overapproximations (e.g., a dodecahedron), and non-convex shapes can be modeled as a set of polytopes through convex partitioning. The union of all obstacles is denoted by $O = \bigcup_{k=1}^{n_O} O_k$, and the free workspace by $F_W = W \setminus O$.

This paper considers serially connected manipulators, whose kinematics are uniquely determined by a vector $\mathbf{q} \in \mathbb{R}^{n_q}$ of joint positions (angles for rotational joints and displacements for prismatic joints). The subset of the space occupied by the robot for a specific configuration $\mathbf{q}$ is indicated by $B(\mathbf{q}) \subset \mathbb{R}^d$. Let $f_p : \mathbb{R}^{n_q} \to \mathbb{R}^d$ be the forward kinematics function for computing the end-effector position $\mathbf{p} \in \mathbb{R}^d$, i.e., $\mathbf{p} = f_p(\mathbf{q})$.

We denote by $\mathbb{R}^{n_r}$ the task-space of the path planning problem, and by $f_r : \mathbb{R}^{n_q} \to \mathbb{R}^{n_r}$ a function mapping a configuration $\mathbf{q}$ to a task-space point $\mathbf{r} \in \mathbb{R}^{n_r}$, i.e., $\mathbf{r} = f_r(\mathbf{q})$. In this work, the path planning problem is constrained only by the kinematic model of the manipulator and the static obstacles in the environment. We further assume that we can always find the end-effector position $\mathbf{p}$ for a task-space point $\mathbf{r}$ via a function $g_p : \mathbb{R}^{n_r} \to \mathbb{R}^d$, i.e., $\mathbf{p} = g_p(\mathbf{r})$. As an example, the task-space consisting of the position and orientation of the end-effector fulfills the criteria introduced above. Formally, we define the path planning problem as a tuple $(\mathbf{q}_b, \mathbf{r}_g)$, where $\mathbf{q}_b \in \mathbb{R}^{n_q}$ is the initial configuration of the robot, and $\mathbf{r}_g \in \mathbb{R}^{n_r}$ represents the goal point in the task-space. A solution to the path planning problem $(\mathbf{q}_b, \mathbf{r}_g)$ is a continuous function $\mathbf{q}(s) : [0, 1] \to \mathbb{R}^{n_q}$ that fulfills the following conditions:

$$\begin{aligned} \mathbf{q}(0) &= \mathbf{q}_b \ , \\ f_r(\mathbf{q}(1)) &= \mathbf{r}_g \ , \\ \forall s \in [0,1] : B(\mathbf{q}(s)) &\subset F_W \ . \end{aligned} \qquad (2)$$

For a solution $\mathbf{q}(s)$, $\mathbf{p}(s) = f_p(\mathbf{q}(s))$ denotes the end-effector trajectory. The total length of the solution path $\mathbf{q}(s)$ is denoted by l_q, the total length of the end-effector trajectory $\mathbf{p}(s)$ by l_p, and t is the duration of the planner's execution. Intuitively, the planner should find solutions with low values for t, l_q, and l_p; these values are later used to compare the proposed planner with other state-of-the-art approaches.

III. Path Planning Algorithm

The path planner proposed in this work consists of a global planner guiding the search by using information about the workspace topology, and multiple parallel local planners constructing collision-free paths in the task-space. The *global planner* partitions the free workspace F_W into a set of disjoint polytopes $\mathcal{F} = \{F_i\}_{i=1}^{n_F}$, such that $\cup_{i=1}^{n_F} F_i = F_W$, and constructs a polytope adjacency graph with the polytopes $\mathcal{F}$ as graph vertices. Using this information, the planner creates and maintains two trees:

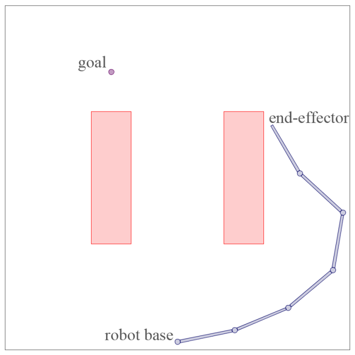
- a task-space tree $\mathcal{T}$, in which each node $\mathbf{t} = (\mathbf{r}, \mathcal{Q})$ consists of a task-space point $\mathbf{r} \in \mathbb{R}^{n_r}$, and a set of configurations $\mathcal{Q} = \{\mathbf{q} \in \mathbb{R}^{n_q} \mid f_r(\mathbf{q}) = \mathbf{r}\}$,[1] i.e., all configurations in $\mathcal{Q}$ map to $\mathbf{r}$;
- a polytope tree $\mathcal{N}$, in which each node $\mathbf{n} = (F, \mathcal{T}_F)$ consists of a free-space polytope $F \in \mathcal{F}$, and a subset of the task-space tree $\mathcal{T}_F \subset \mathcal{T}$ with the property that all nodes in $\mathcal{T}_F$ correspond to end-effector points in F ($\forall \mathbf{t} \in \mathcal{T}_F$, $g_p(\mathbf{t}.\mathbf{r}) \in F$).[2]

The *local planners* are used by the global planner to find collision-free paths between adjacent free-space polytopes.

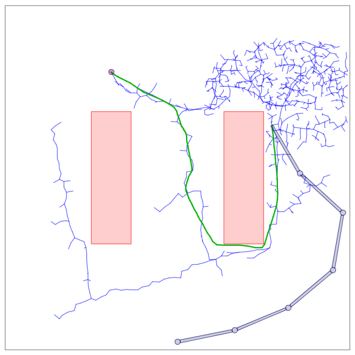
Figure 1 presents a 2D path planning problem for a 6 DoF robot, which we use hereafter to explain the algorithm.

[1]Note that in [5], the tree nodes have the form $\mathbf{t} = (\mathbf{r}, \mathbf{q})$

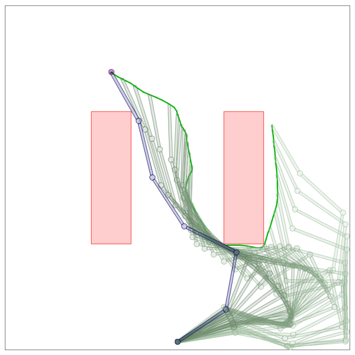
[2]The . (dot) operator is used to identify individual members of a tree node. Here, $\mathbf{t}.\mathbf{r}$ denotes the task-space point $\mathbf{r}$ corresponding to the node $\mathbf{t}$.



(a) Example path planning problem

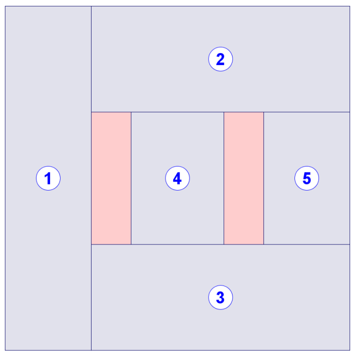


(b) End-effector path (green) and expanded task-space RRT (blue)

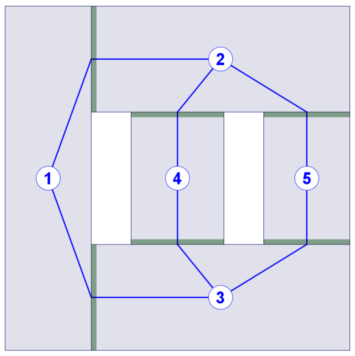


(c) Robot path

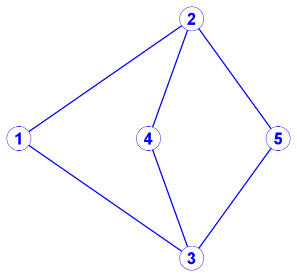
Fig. 1: Hierachical Planner solving a 2D path planning problem for a 6 DoF robot



(a) Free-space partitioning (gray: free space, red: obstacle)



(b) Boundaries between adjacent polytopes (green)



(c) Adjacency graph

Fig. 2: Data structure for the global planner: Convex decomposition and adjacency graph

A. Global Planner

The global planner (Algorithm 1) starts by partitioning the free workspace F_W into a set of disjoint polytopes $\mathcal{F}$ (line 1) using the binary space partitioning algorithm, introduced initially for computer graphics in [21], and used in [14] in the context of motion planning for a mobile robot in a 2D environment. The execution time for the partitioning algorithm is $O(|\mathcal{L}|^2)$ [22], where $\mathcal{L}$ is a set containing all facets of all workspace obstacles. The result of the partitioning algorithm for the example workspace is shown in Fig. 2a, where the free-space polytopes are labeled using the numerals 1 to 5. Using the convex decomposition $\mathcal{F}$, the planner constructs a polytope adjacency graph A_F (line 2), with the elements of $\mathcal{F}$ as vertices (Fig. 2c). The planner checks all pairs of polytopes and connects the corresponding graph vertices by an edge if the polytopes have co-planar facets whose intersection is not empty (Fig. 2b shows the boundaries for the example workspace). After finding the initial end-effector point $\mathbf{p}_b$ (line 3) and the corresponding polytope F_b (line 4), the global planner initializes the root nodes of the trees $\mathcal{T}$ and $\mathcal{N}$ (lines 5 and 6, respectively). In our example, F_b is polytope 5, and the goal point is inside polytope 2. The global planner starts the search at the root node of the polytope tree (line 7).

Algorithm 1 Global planner algorithm

```
Input: $F_W, (\mathbf{q}_b, \mathbf{r}_g)$
Output: $\mathbf{q}(s)$
1: $\mathcal{F} \leftarrow$ PARTITION($F_W$)
2: $A_F \leftarrow$ ADJACENCYGRAPH($\mathcal{F}$)
3: $\mathbf{p}_b \leftarrow f_p(\mathbf{q}_b)$
4: $F_b \leftarrow$ FINDPOLYTOPE($\mathcal{F}, \mathbf{p}_b$)
5: $\mathbf{t}_{root} \leftarrow (\mathbf{p}_b, \{\mathbf{q}_b\})$          ▷ task-space tree root node
6: $\mathbf{n}_{root} \leftarrow (F_b, \{\mathbf{t}_{root}\})$          ▷ polytope tree root node
7: $\mathbf{q}(s) \leftarrow$ FINDPATH($\mathbf{n}_{root}$)          ▷ start search
```

Given a polytope tree node $\mathbf{n}$, the FINDPATH function (Algorithm 2) starts parallel local planners for each adjoining polytope F_n of the polytope $\mathbf{n}.F$, attempting to find a collision-free path moving the end-effector from $\mathbf{n}.F$ towards F_n (lines 2 and 3). In our example, the global planner starts two local planners, one for connecting polytope 5 with polytope 2, and, in parallel, one for connecting polytope 5 to polytope 3. When a polytope F_n is reached, a new node $\mathbf{n}_{new}$ is created and added to the polytope tree $\mathcal{N}$, and FINDPATH is called for the new node (lines 4 to 7). If the current polytope $\mathbf{n}.F$ contains the goal point $\mathbf{r}_g$, a local planner is started asynchronously with a ball of radius r_{goal}

Algorithm 2 FINDPATH algorithm

```
1:  function FINDPATH($\mathbf{n}$)
2:      for $F_n \in A_F$.neighbors($\mathbf{n}.F$) do parallel
3:          $\mathbf{t}_{new} \leftarrow$ CONNECTREGION($\mathbf{n}.\mathcal{T}_F, \mathbf{n}.F, F_n, \rho_F$)
4:          $\mathbf{n}_{new} \leftarrow (F_n, \{\mathbf{t}_{new}\})$
5:          $\mathbf{n}$.children $\leftarrow \mathbf{n}$.children $\cup \{\mathbf{n}_{new}\}$
6:          FINDPATH($\mathbf{n}_{new}$)
7:      end for
8:      if $g_p(\mathbf{r}_g) \in \mathbf{n}.F$ then parallel
9:          $G \leftarrow$ BALL($\mathbf{r}_g, r_{goal}$)          ▷ define goal region
10:         $\mathbf{t}_g \leftarrow$ CONNECTREGION($\mathbf{n}.\mathcal{T}_F, \mathbf{n}.F, G, \rho_{goal}$)
11:         $\mathbf{q}_g \leftarrow$ SAMPLE($\mathbf{t}_g.\mathcal{Q}$)
12:         $\mathbf{q}(s) \leftarrow$ BACKTRACK($\mathbf{q}_g$)        ▷ create solution
13:         return $\mathbf{q}(s)$
14:     end if
15: end function
```

around the goal point as the target region (lines 8 to 10). In our example, the local planner can easily find a path from polytope 5 to polytope 2, for which a node is created in the polytope tree. As polytope 2 contains the goal point $\mathbf{r}_g$, the global planner starts a local planner inside polytope 2 to find the path to the goal. However, the robot cannot reach the goal point moving directly from polytope 5 to polytope 2 due to the obstacles, so the algorithm continues expanding the polytope tree until it constructs the sequence $(5, 3, 4, 2)$, which the end-effector can follow to the goal (Fig. 1b and 1c). When the goal is reached, the solution is obtained by generating a continuous function $\mathbf{q}(s)$ from the sequence of configurations connecting the start $\mathbf{q}_b$ with goal $\mathbf{q}_g$ (lines 11 to 13). The solution is returned immediately by the global planner, and the planner terminates. A simplified view of the resulting trees $\mathcal{N}$ and $\mathcal{T}$ is shown in Fig. 3; the polytope nodes $\mathbf{n}$ are shown as rectangles labeled with the numeral corresponding to the free-space polytope $\mathbf{n}.F$, while the task-space nodes $\mathbf{t}$ are shown as dots within the polytope nodes. Note that there are two nodes corresponding to the polytope 2, one for the sequence $(5, 2)$, the other for the sequence $(5, 3, 4, 2)$.

The global planner is equivalent to a breadth-first search algorithm in the adjacency graph. Due to this behavior, the number of local planners running in parallel might grow exponentially, which can become a critical factor in expansive environments. This problem is mitigated by several factors. First, we consider workspaces for robotic manipulators with a fixed base, which tend to have simpler topologies than workspaces for mobile robots. Second, our workspace decomposition implementation tries to reduce the number of resulting polytopes, thereby simplifying the topology of the adjacency graph. Third, we prevent the obvious case when the end-effector returns to the previous polytope. For example, in Fig. 2, after finding the path from polytope 5 to polytope 3, the global planner starts two local planners, one for connecting polytope 3 to polytope 1, the other for connecting polytope 3 to polytope 4 (no planner is started for

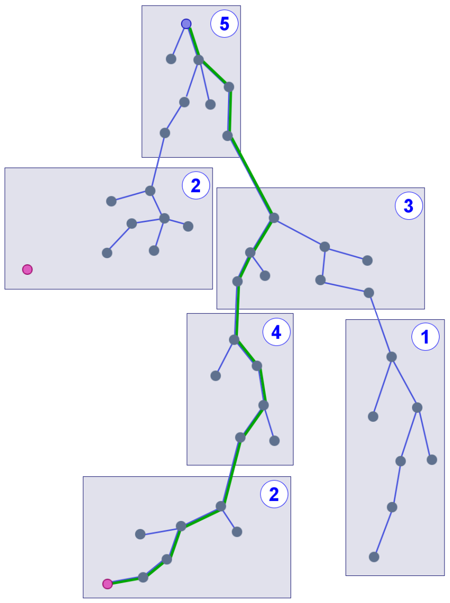


Fig. 3: Polytope and task-space trees after finding the path to the goal (blue: start point, magenta: goal, green: solution)

connecting polytope 3 to polytope 5, as these polytopes are already connected). Fourth, we schedule the execution of the local planners explicitly, giving priority to more successful local planners, and to planners exploring shorter paths in the adjacency graph. We give additional details about the scheduling process below.

Let $\mathcal{J}$ be the set of active local planners at a given moment in time, $\mathcal{J}_s \subset \mathcal{J}$ the set of planners that have been scheduled at least once, and $\mathcal{J}_u \subset \mathcal{J}$ the set of planners that have not been scheduled. As an example, assume that $\mathcal{J}_s$ contains the local planner connecting the goal via the sequence $(5, 2)$ and a local planner exploring the sequence $(5, 3, 4)$. The set $\mathcal{J}_u$ contains two planners, one for the sequence $(5, 3, 1)$, the other for $(5, 2, 4)$. For each planner in $\mathcal{J}_s$, we count the total number of times the planner failed to expand the task-space tree $\mathcal{T}$ towards the target region (exploitation failures), which we denote by n_{fail}. We choose planners in $\mathcal{J}_s$ randomly with a probability proportional to $1/n_{fail}$ so that more successful planners have a higher probability of being selected. In our example, the local planner for the sequence $(5, 3, 4)$ has a higher probability of being selected because the other planner cannot make progress towards the goal, which increases its n_{fail} value. For each planner in $\mathcal{J}_u$, we estimate the end-effector path to the goal point $\mathbf{p}_g$ using the information from the adjacency graph. The scheduler selects with probability $\rho_{newpath}$ the planner from $\mathcal{J}_u$ with the shortest estimated path length (in our example, this is $(5, 3, 1)$), or with probability $1 - \rho_{newpath}$ a planner from $\mathcal{J}_s$. The selected planner is assigned to one of the available execution threads for a limited amount of time. If the local planner cannot solve its task within the allocated time, it is paused, and the scheduler selects another planner. This approach balances the breadth-first search in the adjacency graph with depth-first scheduling of the local planners.

Algorithm 3 Local planner algorithm

```
1: function ConnectRegion($\mathcal{T}_R, R, R_{new}, \rho$)
2:     repeat
3:         if Sample([0,1]) $< \rho$ then
4:             $\mathbf{r}_{new} \leftarrow$ Sample($R_{new}$)          ▷ exploitation
5:         else
6:             $\mathbf{r}_{new} \leftarrow$ Sample($R$)          ▷ exploration
7:         end if
8:         $\mathbf{t}_{near} \leftarrow \mathrm{argmin}_{\mathbf{t} \in \mathcal{T}_R} \|\mathbf{t}.\mathbf{r} - \mathbf{r}_{new}\|$
9:         $\mathbf{t}_{new} \leftarrow$ Extend($\mathbf{t}_{near}, \mathbf{r}_{new}$)
10:        if $\mathbf{t}_{new}.\mathbf{r} \in R$ then
11:            $\mathcal{T}_R \leftarrow \mathcal{T}_R \cup \{\mathbf{t}_{new}\}$
12:        end if
13:    until $\mathbf{t}_{new}.\mathbf{r} \in R_{new}$          ▷ reached target region
14:    return $\mathbf{t}_{new}$
15: end function
```

B. Local Planner

The local planner algorithm (Algorithm 3) extends the Task-Space RRT algorithm introduced in [5] with obstacle avoidance and explicit null-space exploration. Given a task-space region $R \subset \mathbb{R}^{n_r}$, and a set of tree nodes $\mathcal{T}_R \subset \mathcal{T}$ with the property that all nodes in $\mathcal{T}_R$ correspond to points in R ($\forall \mathbf{t} \in \mathcal{T}_R$, $\mathbf{t}.\mathbf{r} \in R$), the local planner extends the tree (lines 8 to 12) until it reaches the target region $R_{new} \subset \mathbb{R}^{n_r}$ (line 13). We assume that R and R_{new} are adjacent regions in $\mathbb{R}^{n_r}$, i.e., $R \cap R_{new} \neq \varnothing$; the global planner ensures that this property holds for all calls to ConnectRegion. The parameter ρ denotes the probability of sampling the next point from the target region R_{new}.

Given the node $\mathbf{t}_{near}$ and the new sample $\mathbf{r}_{new}$, the Extend function (Algorithm 4) selects randomly a configuration $\mathbf{q}_{near}$ from $\mathbf{t}_{near}.\mathcal{Q}$ (line 2), and computes a step towards $\mathbf{r}_{new}$ of maximum length δ_r (lines 3 to 6). The function AvoidObstacles (called in line 8) finds the closest robot joint to the workspace obstacles, and uses a Jacobian pseudoinverse method to create a vector $\Delta\mathbf{q}_{avoid}$ that moves the joint away from the closest obstacle. As AvoidObstacles is a potentially expensive function, we only call it with probability ρ_{avoid} (line 7). Finally, we compute the C-space step $\Delta\mathbf{q}$ using the task Jacobian pseudoinverse $\mathbf{J}^\dagger$, and projecting $\Delta\mathbf{q}_{avoid}$ in the task null-space (line 12). If the C-space step is unsuccessful (verified by collision checking in Algorithm 5, line 6), a random self-motion step is attempted (lines 15 to 17). Exploring the null-space explicitly through self-motion provides more possibilities for finding collision-free paths when compared to approaches where a single configuration is used for each tree node (e.g., Task-Space RRT in [5]).

C. Properties of the proposed path planner

The proposed path planner has several important characteristics. First, we define the local planner to run on a subset $\mathcal{T}_R$ of the complete tree $\mathcal{T}$. This enables running multiple local planners in parallel, with little synchronization needed, which greatly improves the performance of the overall planner. Second, partitioning the free-space into convex polytopes implies that the line connecting any two points within a polytope does not intersect any obstacles. Thus, because the end-effector cannot collide with the workspace obstacles, the local planner can use higher probabilities ρ of sampling in the target region compared to typical values used by global RRT planners to sample the goal point. This means that our path planer focuses more on exploitation than exploration, and can potentially find the solution in a shorter time. Third, the global planner creates different

Algorithm 4 Extend algorithm

```
1: function Extend($\mathbf{t}_{near}, \mathbf{r}_{new}$)
2:     $\mathbf{q}_{near} \leftarrow$ Sample($\mathbf{t}_{near}.\mathcal{Q}$)
3:     $\Delta\mathbf{r} \leftarrow \mathbf{r}_{new} - \mathbf{t}_{near}.\mathbf{r}$
4:     if $\|\Delta\mathbf{r}\| > \delta_r$ then
5:         $\Delta\mathbf{r} \leftarrow \delta_r \frac{\Delta\mathbf{r}}{\|\Delta\mathbf{r}\|}$          ▷ limit task-space step size
6:     end if
7:     if Sample([0,1]) $< \rho_{avoid}$ then
8:         $\Delta\mathbf{q}_{avoid} \leftarrow$ AvoidObstacles($\mathbf{q}_{near}$)
9:     end if
10:    $\mathbf{J} \leftarrow$ Jacobian($\mathbf{q}_{near}$)
11:    $\mathbf{J}^\dagger \leftarrow \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}$
12:    $\Delta\mathbf{q} \leftarrow \mathbf{J}^\dagger\Delta\mathbf{r} + (\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})\Delta\mathbf{q}_{avoid}$
13:    $\mathbf{q}_{new} \leftarrow$ QStep($\mathbf{q}_{near}, \Delta\mathbf{q}$)
14:    if $\mathbf{q}_{new} = \varnothing$ then          ▷ explore null space
15:        $\mathbf{q}_{random} \leftarrow$ Sample($\mathbb{R}^{n_q}$)
16:        $\Delta\mathbf{q} \leftarrow (\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})(\mathbf{q}_{random} - \mathbf{q}_{near})$
17:        $\mathbf{q}_{new} \leftarrow$ QStep($\mathbf{q}_{near}, \Delta\mathbf{q}$)
18:        if $\mathbf{q}_{new} \neq \varnothing$ then
19:            $\mathbf{q}_{near}$.children $\leftarrow \mathbf{q}_{near}$.children $\cup \{\mathbf{q}_{new}\}$
20:            $\mathbf{t}_{near}.\mathcal{Q} \leftarrow \mathbf{t}_{near}.\mathcal{Q} \cup \{\mathbf{q}_{new}\}$
21:        end if
22:        return $\varnothing$
23:    else
24:        $\mathbf{q}_{near}$.children $\leftarrow \mathbf{q}_{near}$.children $\cup \{\mathbf{q}_{new}\}$
25:        $\mathbf{t}_{new} \leftarrow (f_r(\mathbf{q}_{new}), \{\mathbf{q}_{new}\})$
26:        $\mathbf{t}_{near}$.children $\leftarrow \mathbf{t}_{near}$.children $\cup \{\mathbf{t}_{new}\}$
27:        return $\mathbf{t}_{new}$
28:    end if
29: end function
```

Algorithm 5 QStep algorithm

```
1: function QStep($\mathbf{q}, \Delta\mathbf{q}$)
2:     if $\|\Delta\mathbf{q}\| > \delta_q$ then
3:         $\Delta\mathbf{q} \leftarrow \delta_q \frac{\Delta\mathbf{q}}{\|\Delta\mathbf{q}\|}$          ▷ limit c-space step size
4:     end if
5:     $\mathbf{q}_{new} \leftarrow \mathbf{q} + \Delta\mathbf{q}$
6:     if $B(\mathbf{q}_{new}) \subset F_W$ then          ▷ collision checking
7:         return $\mathbf{q}_{new}$
8:     else
9:         return $\varnothing$
10:    end if
11: end function
```

polytope nodes $\mathbf{n}$ for a polytope found through different sequences (Algorithm 2, line 4), which means that multiple local planners can grow the task-space tree $\mathcal{T}$ within the same polytope without interfering with each other. The local planners are thus exploring different subspaces of the configuration space. This approach ensures that our planner's robustness is significantly better compared to global task-space RRT, and comparable or better than that of C-space planners, as shown in the next section. Fourth, polytope sequences containing cycles are permitted, which enables the exploration of different configuration subspaces in previously visited regions of the workspace. As an example, consider the path planning problem shown in Fig. 4a, where the planner needs to retract the end-effector on a path around the bottom-right obstacle, then pass again through the central region to reach the goal in the top-left corner. This characteristic is a significant advantage both over hierarchical path planners that use cycle-free paths in the adjacency graph (found via shortest path algorithms), such as the approach in [17], and global task-space planners like [5], where the tree expansion can prevent the planner from finding a solution (see Fig. 4d).

An important topic when introducing new sampling-based path planning algorithms is the notion of *probabilistic completeness*. A path planning algorithm is probabilistically complete if, assuming a feasible path exists, the probability of finding the solution approaches one as the computation time goes to infinity. While being mathematically useful, it has been argued [12] that the definition has little practical value for real-world path planning problems, where the computation time *going to infinity* is not an option. A more useful notion would be the probability of finding a solution given a finite amount of time, although this might be difficult to estimate. Similar to the approach in [12], our path planning algorithm values computational efficiency over probabilistic completeness.

IV. Application examples and comparisons

The performance and robustness of the proposed hierarchical planner is tested in several different scenarios, and compared to different state-of-the-art path planning algorithms. The following examples were selected to illustrate the advantages and drawbacks of the proposed planner:[3]

- A 2D workspace containing 4 square obstacles, with a 100 DoF manipulator that approximates a flexible robot (Fig. 4).
- A 3D workspace containing a wall with 4 square holes, and an 8 DoF manipulator. The robot starts with the end-effector inside the top-left hole, and needs to enter one of the bottom holes to reach the goal position (Fig. 5).
- A 3D workspace where a humanoid robot needs to reach with the right hand the content of a box placed on top of a table. The kinematics is modeled after TORO [23], a 27 DoF humanoid robot developed at DLR. For this example, the torso is not connected to the waist, and can rotate freely in three dimensions. Together with the 6 DoF arm, it creates a 9 DoF robotic manipulator (Fig. 6).

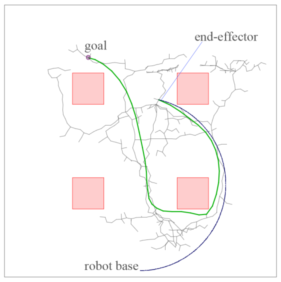
The planner proposed here (*Hierachical Planner*) is compared with four other planners: RRT-Connect with one tree (*RRT-Connect*) [4], RRT-Connect with two trees (*BiRRT-Connect*) [4], an adaptive dynamic-domain RRT with two trees (*ADD-RRT*) [24], and a task-space RRT with one global tree (*TS-RRT*) [5], which we modified to use the same task-space RRT structure as our local planner, and the Extend function presented in Algorithm 4. For these planners, the nearest neighbor search was implemented using kd-trees [25], with the parameter $k = 30$. In order to ensure a fair comparison, as our hierarchical planner was designed for a multi-threaded execution, we implemented multi-threaded variants of the mentioned algorithms, and configured them to use the same number of execution threads as our planner. For all planners, the solution returned by the planner is first simplified using an iterative approach that finds shortcuts in the configuration sequence, and then smoothed to create a quadratic Bézier curve.

All planners are implemented in Java, and the examples were executed on a standard Windows PC with 3.5 GHz Intel Xeon processor and 8 GB of RAM. Collision detection and distance queries were performed using a proprietary implementation. The results are averaged over 100 trials; if a planner does not solve the problem within 30 seconds, the trial is considered a failure and the planner is stopped. The tables in Fig. 4 to 6 show the mean value and the standard deviation of the following values (only considering successful trials): execution time t, number of collision checks, C-space path length l_q (using Euclidean norm), and end-effector path length l_p. The last column shows the percentage of successful trials. Accompanying images show one trial result for the four different planners; BiRRT-Connect was omitted as the results are very similar to ADD-RRT.

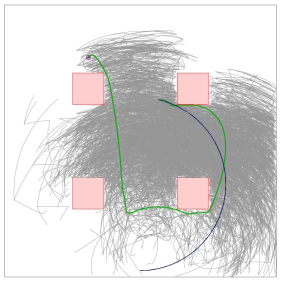
For all planners, we used the following common parameters: number of execution threads $n_{threads} = 6$, goal radius $r_{goal} = 0.005$ m, maximum C-space step $\delta_q = 0.1$, and maximum task-space step $\delta_p = 0.025$ m. For our hierarchical planner, we used the following parameters: $\rho_{goal} = 0.5$ (probability of sampling the goal region), $\rho_F = 0.9$ (probability of sampling in the neighboring polytope), $\rho_{avoid} = 0.5$ (probability of adding collision avoidance in the Extend function), $\rho_{newpath} = 0.3$ (probability of selecting a local planner which explores a new path). These values were empirically chosen to maximize the performance using Fig. 6 as an archetypal example. For the other single tree planners (RRT-Connect and TS-RRT), the probability of sampling the goal was set to $\rho = 0.25$. Note that the free-space partitioning in the proposed planner makes it possible to use higher probabilities for sampling the local goal (ρ_{goal} and ρ_F compared to ρ), which means that the planner focuses more on exploitation than exploration.

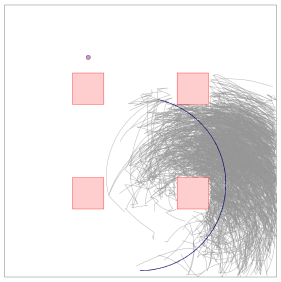
The first example from Fig. 4 shows the advantage of the task-space over the C-space RRT in the case of a high-DoF manipulator (100 DoF here). The bidirectional planners only manage to solve the path planning problem in 49% of the

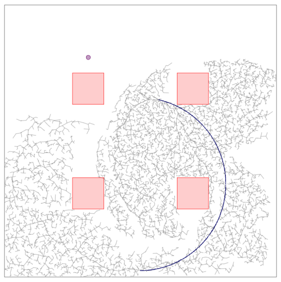
[3]the attached video shows more application examples.



(a) Hierarchical Planner

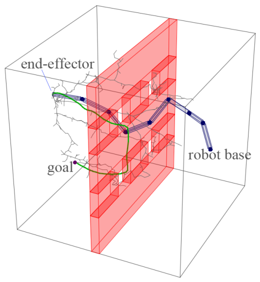

(b) ADD-RRT


(c) RRT-Connect

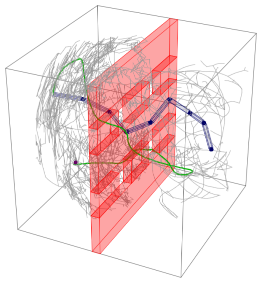

(d) TS-RRT

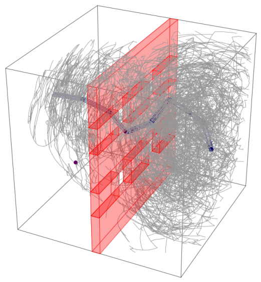
Planner	Time (s)	Collision Checks	l_q	l_p (m)	Success (%)
Hierachical Planner	4.70 ± 1.25	34281 ± 11926	3.498 ± 0.221	1.830 ± 0.064	100
ADD-RRT	18.31 ± 7.37	62625 ± 18305	14.197 ± 1.551	2.197 ± 0.256	49
BiRRT-Connect	18.69 ± 8.04	62874 ± 21273	14.710 ± 1.943	2.296 ± 0.297	37
RRT-Connect	N/A	N/A	N/A	N/A	0
TS-RRT	N/A	N/A	N/A	N/A	0

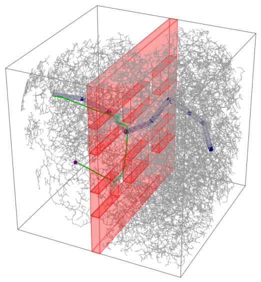
Fig. 4: 2D workspace example with 100 DoF robot



(a) Hierarchical Planner

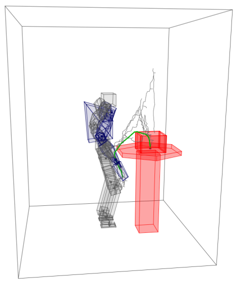

(b) ADD-RRT

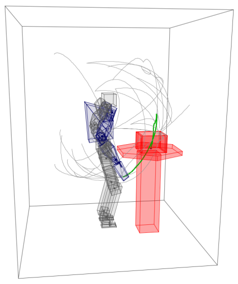

(c) RRT-Connect

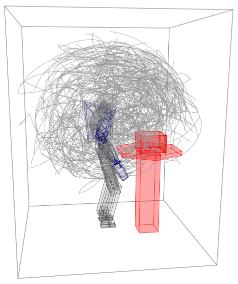

(d) TS-RRT

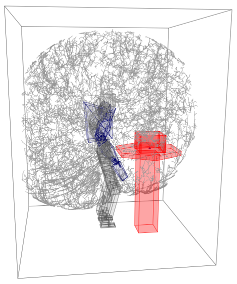
Planner	Time (s)	Collision Checks	l_q	l_p (m)	Success (%)
Hierachical Planner	1.39 ± 0.40	7675 ± 5049	5.147 ± 0.181	1.304 ± 0.036	100
ADD-RRT	2.42 ± 0.79	18809 ± 10117	13.986 ± 3.631	2.699 ± 0.760	100
BiRRT-Connect	2.64 ± 0.94	21637 ± 14349	13.937 ± 3.981	2.664 ± 0.805	100
RRT-Connect	N/A	N/A	N/A	N/A	0
TS-RRT	6.33 ± 4.84	147262 ± 121840	4.706 ± 0.125	1.637 ± 0.079	41

Fig. 5: Stationary 8 DoF manipulator reaching through openings in a wall


(a) Hierarchical Planner


(b) ADD-RRT


(c) RRT-Connect


(d) TS-RRT

Planner	Time (s)	Collision Checks	l_q	l_p (m)	Success (%)
Hierachical Planner	0.74 ± 0.06	1304 ± 199	2.732 ± 0.185	1.090 ± 0.068	100
ADD-RRT	0.71 ± 0.21	1801 ± 1281	3.411 ± 0.676	1.207 ± 0.173	100
BiRRT-Connect	0.70 ± 0.19	1838 ± 1064	3.359 ± 0.589	1.261 ± 0.190	100
RRT-Connect	2.72 ± 4.41	21517 ± 42611	3.249 ± 0.453	1.334 ± 0.213	9
TS-RRT	N/A	N/A	N/A	N/A	0

Fig. 6: Humanoid robot reaching into a box placed on a table

cases for ADD-RRT, and 37% for BiRRT-Connect, while the single tree RRT-Connect cannot find a solution in the allotted time. The TS-RRT fails to find a solution as it starts by expanding the tree in the center of the workspace, and after finding the path around the bottom-right obstacle, it can no longer search for a solution towards the goal through the center (Fig. 4d). Our planner does not share this inherent weakness of task-space RRT planners using global trees, as it uses sequences of free-space polytopes as the main path finding construct. This means that when the center of the workspace is reached through two different sequences, there is no interference, as the sequences lead to different branches of the polytope tree, and the local planner expands two independent subtrees. In the second example illustrated in Fig. 5, our planner can find shorter paths faster than the other planners, as it can use the relatively simple graph of adjacent free-space polytopes consisting of only 6 vertices (consisting of the 2 halves of the workspace and the 4 windows in the wall). Finally, in the third example illustrated in Fig. 6, our planner's performance and robustness are comparable to the bidirectional C-space planners. The TS-RRT planner fails to find the box top opening within the allocated time, as the tree expansion tends to approach the goal point from the side of the box.

For all examples, our planner achieves a success rate of 100%, and finds consistently shorter paths than the other planners, both in configuration-space (l_q) and in task-space (l_p). In general, our planner tends to have the most consistent behavior, which is visible in the lower standard deviations of the investigated values compared to the other planners. RRT-Connect performs poorly in all examples, as it uses only one tree rooted at the start configuration to search for the solution. Our planner also uses only one tree, however it matches or outperforms bidirectional planners, which shows the practical utility of our approach.

V. Conclusions

This paper presented a fast and robust hierarchical task-space path planning algorithm that can solve a variety of real-world path planning problems with excellent success rate. Our planner achieves a good performance by partitioning the free-space, and thus decomposing the global problem into a set of easier-to-solve local problems. The additional benefit of this decomposition is the ability to solve local path planning problems in parallel with little or no synchronization needed, which is a significant advantage over planners using global data structures, like bidirectional RRT. Our planner uses a single tree, but implementing a bidirectional variant is part of our current research efforts. Also, as future work, we consider applications in the challenging problem of multi-contact locomotion for legged robots in cluttered environments.

References

[1] L. Kavraki, P. Švestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[3] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *Int. J. Robotics Research*, vol. 32, no. 9, pp. 1164–1193, 2013.

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

[5] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space Voronoi bias," in *IEEE Int. Conf. Robotics and Automation*, 2009, pp. 2061–2067.

[6] L. Keselman, E. Verriest, and P. Vela, "Forage RRT-an efficient approach to task-space goal planning for high dimensional systems," in *IEEE Int. Conf. Robotics and Automation*, 2014, pp. 1572–1577.

[7] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *IEEE Int. Conf. Robotics and Automation*, 2006, pp. 1874–1879.

[8] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," *Robotics: Science and Systems*, 2008.

[9] S. Rodríguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *IEEE Int. Conf. Robotics and Automation*, 2006, pp. 895–900.

[10] J. Denny, E. Greco, S. Thomas, and N. M. Amato, "MARRT: Medial axis biased rapidly-exploring random trees," in *IEEE Int. Conf. Robotics and Automation*, 2014, pp. 90–97.

[11] D. Kim, J. Lee, and S.-e. Yoon, "Cloud RRT*: Sampling cloud based RRT*," in *IEEE Int. Conf. Robotics and Automation*, 2014, pp. 2519–2526.

[12] M. Rickert, A. Sieverling, and O. Brock, "Balancing exploration and exploitation in sampling-based motion planning," *IEEE Trans. Robotics*, vol. 30, no. 6, pp. 1305–1317, 2014.

[13] J. P. Van den Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Int. J. Robotics Research*, vol. 24, no. 12, pp. 1055–1071, 2005.

[14] A. Tokuta, "Motion planning using binary space partitioning," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1991, pp. 86–90.

[15] J. Singh and M. Wagh, "Robot path planning using intersecting convex shapes: Analysis and simulation," *IEEE J. Robotics and Automation*, vol. 3, no. 2, pp. 101–108, 1987.

[16] D. Kuan, J. Zamiska, and R. Brooks, "Natural decomposition of free space for path planning," in *IEEE Int. Conf. Robotics and Automation*, 1985, pp. 168–173.

[17] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[18] D. Devaurs, T. Siméon, and J. Cortés, "Parallelizing RRT on distributed-memory architectures," in *IEEE Int. Conf. Robotics and Automation*, 2011, pp. 2261–2266.

[19] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup." in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2012, pp. 1206–1212.

[20] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed RRT for motion planning," in *IEEE Int. Conf. Robotics and Automation*, 2013, pp. 5088–5095.

[21] H. Fuchs, Z. Kedem, and B. Naylor, "On visible surface generation by a priori tree structures," in *ACM Siggraph Computer Graphics*, vol. 14, no. 3, 1980, pp. 124–133.

[22] M. Paterson and F. Yao, "Efficient binary space partitions for hidden-surface removal and solid modeling," *Discrete & Computational Geometry*, vol. 5, no. 5, pp. 485–503, 1990.

[23] J. Englsberger, A. Werner, C. Ott, B. Henze, M. A. Roa, G. Garofalo, R. Burger, A. Beyer, O. Eiberger, K. Schmid, and A. Albu-Schäffer, "Overview of the torque-controlled humanoid robot TORO," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2014, pp. 916–923.

[24] L. Jaillet, A. Yershova, S. La Valle, and T. Siméon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2005, pp. 2851–2856.

[25] A. Yershova and S. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Trans. Robotics*, vol. 23, no. 1, pp. 151–157, 2007.