Path planning of 3D solid objects using approximate solutions

Vojtěch Vonásek and Robert Pěnička
Faculty of Electrical Engineering, Czech Technical University in Prague,
Technická 2, 166 27 Prague, Czech Republic,

Abstract—Path planning of 3D solid objects has many practical applications. Sampling-based planners like Rapidly-exploring Random Tree (RRT) tackle the problem by the randomized sampling of the configuration space. The well-known issue of the sampling-based planners is the narrow passage problem. Narrow passages are small collision-free regions in the configuration space that are, due to their low volume, difficult to cover by the random samples, which prevents the sampling-based planners from finding a path leading through the passages. To increase the success rate of the planners, the search in the configuration space can be guided using an approximate solution. An approximate solution is found, for example, considering a smaller object (robot). The performance of this planning concept depends on the way of reducing the size of the object and the ability to find an initial solution. In this paper, we propose a novel technique to reduce the geometry of the object by a combination of triangulation and iterative removal of surface triangles. This technique is suitable for both convex and non-convex objects. In many real-world applications, narrow passages can also contain the initial or goal configurations. We thus propose the extension of the goal region to increase the probability of finding the initial approximate solution. Experiments have shown that the proposed modification outperforms the simple guiding using approximate solutions, as well as other related state-of-the-art planners.

I. INTRODUCTION

Path planning of 3D solid objects, where the task is to find a collision-free trajectory for an object between two positions, has many practical applications in robotics and industry, e.g., in bin-picking and data collection [26]. In virtual prototyping, path planning helps to determine a sequence of actions to assemble a product from its parts [11], [12]. Computational biochemistry utilizes path planning to detect exit pathways from proteins [6] or to identify binding sites [3].

Sampling-based planners like Rapidly Exploring Random Tree (RRT) [21] and Probabilistic Roadmaps (PRM) [19] are widely used to solve this problem [22], [10]. These methods generate random samples in the configuration space and classify the samples as free or non-free using collision detection. The free samples are stored in a graph structure, which approximates the free region of the configuration space. A path in the graph relates to a motion in the workspace.

The well-known bottleneck of sampling-based planners is the narrow passage problem [20], [17], [16]. Narrow passages are small collision-free regions whose removal changes the connectivity of the space [14]. The planners need to place enough samples into the passages to find a path leading through it. Due to low volume of the narrow passages (relatively to the whole volume of the configuration space), the probability of placing random samples inside a narrow passage is low. Consequently, the probability of finding a path through the passages is also low.

Several approaches use scaled-down version of the robot to search the configuration space [14], [34], [4]. This results in approximate solutions, i.e., solutions feasible for a scaled-down robot (object). The search for the full solution is then realized by dense sampling along the approximate solution. The success rate of these guiding planners depends on the ability to find an initial solution. Typically, this is difficult in the presence of narrow passages and even more challenging if start or goal are located in the passage. A situation where both start and goal are located in a narrow passage is depicted in Fig. 1, where the motion of the cylinder is limited by the shaft.

This paper deals with the guided sampling of the configuration space of 3D solid objects using approximate solutions. To cope with the problem of start/goal configurations buried in narrow passages, we propose to extend the goal region by building an auxiliary tree rooted in the goal configuration. The nodes of this tree then represent the extended goal region and the subsequent planning finishes if the path to any of these nodes is found. This is easier than reaching the single

---

Fig. 1: Motivation scenario where the task is to pickup an object (cylinder) for the inspection (green area) and deliver it to the target zone. Both start and goal are located in a narrow passage caused by the shaft. In our approach, the goal region is extended by the auxiliary tree (A). The object is decomposed to several levels of volume (thinned red cylinders). A plan is computed for the smallest robot (B). The plan is then repeatedly improved while using larger robot (C).
hardly reachable, goal configuration. We further propose an alternative technique to reduce the volume of the robot for the purpose of approximate solution computation. The proposed reduction technique relies on a fine triangulation of the robot and iterative removal of triangles (or tetrahedrons) forming the border of the object.

II. RELATED WORK

Rapidly-Exploring Random Trees (RRT) [21] and Probabilistic Roadmap (PRM) [19] are widely used sampling-based planners. PRM first samples the configuration space, classifies the samples using collision-detection and stores the free-ones. The stored samples are then connected, if possible, by a collision-free edge with their close neighbors, which results in a graph structure (roadmap). RRT iteratively builds a tree rooted at the initial configuration. In each iteration, a random sample is generated in the configuration space and its nearest node in the tree is found. This node is expanded towards the random sample using a local planner (e.g., using the straight-line expansion). The algorithm terminates if the goal configuration is approached by the tree close enough or after a predefined number of planning iterations.

The RRT and PRM planners sample the configuration space uniformly, which leads to the well known narrow passage problem [20], [17], [24], [16]. The presence of narrow passage practically means that the number of samples has to be significantly increased, which consequently increases the computational time. In problems with two or three DOFs, the difficult regions can be estimated from the shape of the 2D/3D workspace and PRM can sample more dense around these regions [20], [33]. Similarly, works [2], [28] sample more dense around obstacles. The PRM-based method [36] samples along the medial axis of the workspace. Using workspace knowledge for the sampling however does not bring advantage in the case of high-dimensional configuration spaces [20], as the high-dimensional paths are less correlated with the low-dimensional paths in the workspace.

Increasing the probability of sampling around given regions (e.g., around obstacles) is used mainly in PRM-based planners, that first sample the space and then connect the samples into the roadmap. As RRT samples the space and builds the tree simultaneously, the naïve increase of probability of sampling in a region does not ensure that RRT builds the tree through the region, as the growth of the tree may be blocked by the obstacles. Therefore, different approaches to cope with the narrow passage problem have been studied for RRT-based planners.

In the Retraction-RRT [39], [23], the tree is retracted along the boundary of the obstacles in the configuration space. The boundary is estimated by a local sampling. In RRT-DD [37], an activation radius is assigned to each node of the tree. The node is selected for the expansion only if the random sample is located within this radius. The radius of new nodes is set to ∞ and it is decreased to a predefined (default) value if the node cannot be expanded. Therefore, the radius is small for nodes located near to the boundary of obstacles of the configuration space. An automatic adaptation of this radius was proposed in [18].

Another way to attract the tree towards the narrow passages is to place key-configurations close to the narrow passage and increase the probability of sampling around the key configurations [32]. The work [32] however does not specify how to find the key-configurations. The generalization of the key-configuration bias is the guided sampling, where a sequence of waypoints (a guiding path) is used to generate samples in the configuration space [35], [9], [8], [27]. In the guided sampling, the probability of drawing the random samples around the waypoints is adapted according to the progress of the tree. This is the main difference from the PRM-based extensions, that increase the probability of sampling in all regions simultaneously. A simple guiding path can be constructed in a polygonal workspace assuming a point robot [35]. In [21], workspace is represented by a grid in which a sequence of regions that has to be explored by a sampling-based planner. The method [25] steers the sampling process using information from Any-Angle path planner. Multiple workspace-based guiding paths are used simultaneously in [9]. Medial axis of the workspace can also be used as a guiding path. The approach [7] pushes newly generated samples towards the medial axis. Guiding-based planners are used for many-DOF robots operating on rough terrain, e.g. for walking robots [5].

The narrow passages are located in the configuration space, and therefore, their properties (e.g., shape and volume) depend both on the geometry of the robot and the obstacles. Ideally, the sampling of high-dimensional configuration space should be guided by a path located in the configuration space. Computing such an ideal guiding path would however require to solve the motion planning problem of the sample complexity as the original problem, which would be time consuming.

Practically, guiding path in the configuration space has to be computed while relaxing the original problem. By reducing the geometry of the robot, e.g., by scaling it down, shrinking or thinning, the narrow passages became larger. A similar effect can be achieved by allowing a little penetration between the robot and the obstacles. This was first discussed in [4], where the PRM roadmap computed for a relaxed version of the problem is reused for planning for a less relaxed version of the problem. The paper [4] proposes the general concept of using the approximate solutions, but without specifying technical details for reusing the roadmap or for fixing an invalid solution. Another PRM-based planner utilizing approximate solutions was proposed in [14]. The core of the method is the PRM-based SBL planner [29], which constructs the roadmap without checking collisions. After a path is found in the planner, its edges are checked for collision and new path is calculated if a collision is detected; this repeats until a solution is found or the roadmap does not contain any collision-free path. In our previous work [34], we proposed the RRT-IS (RRT with Iterative Scaling) planner that incrementally improves the approximate solution. Contrary to work presented in this paper, the work [34] assumes naïve simple scaling of the objects,
which is suitable only for objects without holes (and mostly, co convex objects).

III. PRELIMINARIES

Let \( C \) denote the configuration space. The collision-free configurations form the subset \( C_{\text{free}} \subseteq C \), where the object (robot) can move. The task is to compute a collision-free path from an initial configuration \( q_{\text{start}} \in C_{\text{free}} \) to the goal configuration \( q_{\text{goal}} \in C_{\text{free}} \) for a 3D solid object.

We utilize the concept of guided sampling using approximate solutions \([4], [34], [14]\). First, an approximate plan for a relaxed problem is found considering a robot with decreased volume. This effectively widens the narrow passages. The probability of placing the random samples into the widened narrow passages increases, which also increases the probability of finding a path through the passages. The vicinity of the approximate solution is then sampled densely to find the solution for the robot with a larger volume. This process repeats until the solution of the original problem is found.

The performance of the planning using the approximate solutions depends on the way of reducing the volume of the robot as well as the ability to find the initial approximate plan. This paper proposes a novel method to reduce the volume of the robot and a new method to enhance the search of the initial solution. These contributions are described in the next subsections.

A. Reducing volume of the robot by “peeling”

The shape, location, and volume of narrow passages depend on the geometry of the robot as well as the obstacles. The volume of the narrow passages can be increased, e.g., by allowing a certain penetration between the robot and the obstacles [15] or by decreasing the volume of the robot [14], [4], [34], which is considered in this paper. Decreasing the volume of the robot is achieved by modifying the robot’s geometry. To widen the narrow passages, it has to be ensured that the geometry of the robot with lower volume is always inside the geometry of the robot with a larger volume. This is especially important in the case of objects with holes, as decreasing of object’s volume should result in the increase of the holes. This cannot be achieved by the naïve scaling-down of the object’s geometry (Fig. 3).

To cope with such situations, we propose a different method to reduce the size (volume) of the robot. For the sake of simplicity, we describe the idea considering 2D polygonal robots and we later extend this for 3D robots. In the case of 2D workspaces, the robot and the obstacles can be described by polygons. Alternatively, the polygons can be triangulated. Triangulated objects are usually used for fast collision detection [13], [11].

In the classic triangulation, the corners of the triangles are the points of the polygon (Fig. 2). A finer triangulation can be achieved by Constrained Delaunay Triangulation (CDT) [30], where first Delaunay Triangulation is computed and if necessary, certain segments or faces are further triangulated to satisfy a given criterion (Fig. 2b,c). The quality criterion is, e.g., the number of elements, their area or the size of the angles. Let \( T_0 \) be the a fine CDT triangulation of the object. A triangle \( t \in T_0 \) has \( n(t) \) neighbors. Two triangles are neighbors if they have the common edge.

The robot’s size is reduced by iterative removal of the border triangles, i.e., those triangles that have less than three neighbors. The set of border triangles in \( i \)-th iteration, \( i > 0 \) is \( T_i^* = \{ t \in T_{i-1} | n(t) < 3 \} \). After removal, \( T_i = T_{i-1} \setminus T_i^* \). This process repeats until \( T_i \) is empty. We denote this process as peeling, as the shape of the robot is reduced layer by layer. Example of the object peeling is depicted in Fig. 3.

At each level \( i \) of the shape reduction, \( T_i \) is the set of inner triangles and \( T_i^* \) is the set of border triangles. As the shape reduction removes all border triangles, the process is suitable for convex as well as non-convex objects and it is even suitable for objects with holes. The difference between the peeling and naïve scaling is depicted in Fig. 3.

This type of volume reduction can be easily extended to 3D objects. In this case, the triangulation \( T_i \) contains a set of 3D tetrahedra and a tetrahedron \( t \in T_i \) is considered as a border if it has less than four neighbors \( n(t) < 4 \) (in 3D case, inner tetrahedra have four neighbors). The removal of the border tetrahedrons is same as in the described 2D case.

B. Extending the goal region

While the above described technique aims to cope with any narrow passage problem, this section presents a method to cope with a special case of narrow passage: when the goal is...
RRT, despite the goal to few (tens), in order to enable fast planning. The size of the map is the narrow passage if the tree is not located inside already. Located in a narrow passage. More difficult is, however, to enter the narrow passage. The reason is that RRT can grow a tree that is already located in the narrow passage. The local planner relies on collision detection and testing the connection of the two trees can be computationally demanding. The root is longer than a given threshold $d_{\text{goal}}$. In the subsequent planning process, the goal region is constructed from the goal (green circles) (b). In the subsequent planning, approaching any of these nodes ensures that the real goal (black) is reached.

Fig. 5: Example of scenario with start/goal buried in a narrow region (gray areas) (a). Example of the auxiliary tree $T_g$ constructed from the goal (green circles) (b). In the subsequent planning, approaching any of these nodes ensures that the real goal (black) is reached.

depicted in (Fig. 5a), which is motivated by the task depicted in Fig. 1.

To increase the performance in these situations (e.g., Fig. 5O5), we propose to extend the goal region by an auxiliary tree $T_g$ (Fig. 5b). The tree is grown from the goal configuration $q_{\text{goal}}$ until the path in the tree from a node to the root is longer than a given threshold $d_{\text{boost}}$.

In the subsequent planning process, the goal region is defined by all nodes of the tree $T_g$. As each of the nodes in $T_g$ is already connected to the main goal $q_{\text{goal}}$, the planning can terminate if the actual tree approaches any of the nodes of $T_g$ to the distance $d_{\text{goal}}$ or less and if the two trees can be connected. The distance of an actually built tree to the extended goal region can be evaluated fast in $O(\log |T_g|)$ time assuming that KD-tree data structure is used for the nearest-neighbor search. If the tree approaches the goal region tree $T_g$, the straight-line planner attempts to connect both trees. The local planner relies on collision detection and testing the connection of the two trees can be computationally demanding. Therefore, it is necessary to limit the number of nodes of $T_g$ to few (tens), in order to enable fast planning.

This auxiliary tree $T_g$ can be built using the standard RRT [21], despite the goal $q_{\text{goal}}$ is located in the narrow passage. The reason is that RRT can grow a tree that is already located in a narrow passage. More difficult is, however, to enter the narrow passage if the tree is not located inside already.

We demonstrate this behavior of the basic RRT [21] and a map with long narrow passage (Fig. 6). The size of the map is 1000 × 700 units, the robot is spherical with radius 40 units, and the width of the narrow passage is 50 units. Five maps are used, each with different position of the narrow passage. In the map $O_1$, the start is inside the narrow passage, in $O_5$, the goal is located in the passage. The basic RRT was run 100 times and the cumulative probability of reaching the goal in less than the given number of planning iterations was measured (Fig. 7). The number of planning iterations with the maximal probability was detected and summarized in Tab. I.

The best performance is achieved in the scenario $O_1$, where the tree starts to grow from the narrow passage. In this case, the goal was reached in less than 5100 iterations. Contrary, other cases $O_2$–$O_5$ are more difficult, which is indicated by a lower probability of success and especially the required size of the tree, which is more than one order of magnitude larger than in $O_1$ map. Tab. I also shows the percentage of the planning iterations that were needed to enter the left part of the narrow passage. For example, in the map $O_1$, the tree is already located in the narrow passage, therefore the passage was entered in 0 % of the iterations. In $O_2$, the algorithm spends 96.7 % of iterations by growing the tree in the left room before entering the narrow passage. After the tree enters the narrow passage, the goal is reached within 100−96.7 = 3.3 % of iterations (out of 98100) which corresponds to ~ 3237 iterations. Therefore, the basic RRT algorithm spent most of the time by finding the entrance to the narrow passage.

The comparison between $O_1$ and other cases ($O_2$–$O_5$) shows that RRT can easily grow the tree if the tree is already located inside the narrow passage. Therefore, the auxiliary tree $T_g$ extending the goal region can be constructed using the basic RRT [21], starting from $q_{\text{goal}}$. Table I: Performance of the basic RRT depending on the position of the start/goal in the map with the narrow passage.

<table>
<thead>
<tr>
<th>Map</th>
<th>Cum. prob.</th>
<th>Tree size</th>
<th>Runtime [s]</th>
<th>Gap-enter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>100.0 %</td>
<td>5100</td>
<td>0.26</td>
<td>0.0 %</td>
</tr>
<tr>
<td>$O_2$</td>
<td>92.5 %</td>
<td>98100</td>
<td>3.54</td>
<td>96.7 %</td>
</tr>
<tr>
<td>$O_3$</td>
<td>92.0 %</td>
<td>94100</td>
<td>3.44</td>
<td>98.3 %</td>
</tr>
<tr>
<td>$O_4$</td>
<td>96.0 %</td>
<td>90100</td>
<td>3.09</td>
<td>95.6 %</td>
</tr>
<tr>
<td>$O_5$</td>
<td>95.5 %</td>
<td>98100</td>
<td>2.71</td>
<td>98.2 %</td>
</tr>
</tbody>
</table>

Fig. 4: Reduction of the robot shape by removing the border triangles. (a) shows a fine CDT of the robot shape, $T_0$, the border triangles of each reduction level are shown in red (b)–(g). The peeling is terminated when the number of triangles reaches zero; practically, such a simplification does not resemble the original object, and therefore, the peeling is finished at the level $T_6$ (g). The numbers denote the number of the border triangles.

Fig. 6, in (Fig. 5a), which is motivated by the task depicted in Fig. 1. The numbers denote the number of the border triangles.
Fig. 6: Examples of solutions in the scenario with changing narrow passage. The green circle is the start configuration, the black sphere is the goal. In $O_1$, the start configuration is already inside the narrow passage, and the tree can easily grow out, which is indicated by a relatively sparse tree in the right part of $O_1$. Contrary, in $O_2$–$O_5$, the trees are more dense on the left from the narrow passage, which indicates that the tree spent more iterations before the narrow passage was penetrated.

Fig. 7: The cumulative probability of reaching the goal configuration in scenarios depicted in Fig. 6 with increasing planning iterations.

IV. PATH PLANNING USING APPROXIMATE SOLUTIONS

This section briefly describes our approach for planning using approximate solutions \cite{34} and its extension by the methods described in the previous sections. The proposed method is referred to as RRT-IIS (RRT with Improved Iterative Search).

The RRT-IIS planner consists of two main procedures: guided sampling along an approximate solution and the procedure to reduce the geometry of the robot. $L$ levels of object volume (and geometry) are created using the peeling procedure (Section III-A), where 0–th level represents the full object and $L$–th level denotes the low-volume representation of the object. The geometry of the robot at the level $i$ is represented by the triangles $T_i$. A guiding path $P = (q_1, \ldots, q_m), q_i \in C$ is a sequence of $m$ waypoints in the configuration space. The process of sampling along the guiding path is described in the next section.

A. Guided sampling of the configuration space

The task of the guided sampling is to find a path from $q_{\text{start}} \in C_{\text{free}}$ to $q_{\text{goal}} \in C_{\text{free}}$, assuming that a guiding path $P$ is given. Instead of sampling the whole configuration space uniformly, the random samples are generated along the guiding path with the probability $p_{\text{guide}}$.

To ensure that the tree can grow along the path from start to end, we use the principle of the moving sampling region \cite{9, 35}. Let $v, 1 \leq v \leq m$ is an index of a point on the guiding path $P$; we refer to this point as the active waypoint in the rest of the paper. The sampling region is centered at the active waypoint $q_v \in P$. The random sample $q_{\text{rand}}$ is generated around $q_v$ as follows. The 3D positions of $q_{\text{rand}}$ are generated using the Normal distribution $N(q_v, \Sigma^2)$, where $\Sigma^2$ is a diagonal matrix with entries $d_{\text{goal}}$. The random part of $q_{\text{rand}}$ is generated from the Uniform distribution $U(0, \pi)$.

In each iteration, a random sample $q_{\text{rand}}$ is generated with probability $p_{\text{guide}}$ around an active waypoint $v$ and with probability $(1 - p_{\text{guide}})$ from the whole configuration space. At the beginning of the algorithm, $v = 1$ (the first guiding waypoint). After $q_{\text{rand}}$ is generated, its neighbor $q_{\text{near}} \in T$ in the tree is found. The node is expanded towards $q_{\text{rand}}$ using the straight-line planner which results in a new configuration $q_{\text{new}}$. If the new configuration $q_{\text{new}}$ is collision-free, it is added to the tree.

After a successful expansion, the distance to the active waypoint is measured. The waypoint is considered as visited if this distance is smaller than $d_{\text{goal}}$. In such a case, the active waypoint $v$ is moved to the next guiding waypoint (lines 19–20 in Alg. 1). As the active waypoint is moved forward on the path as soon as the tree approaches it, the configuration space is sampled along the guiding path.

The algorithm terminates if the newly added node $q_{\text{new}}$ can be connected by a collision-free line to any of the nodes of the auxiliary tree $T_g$. This is verified by finding nearest neighbor $q' \in T_g$. If the connection is possible, the path from $q'$ to $q_{\text{goal}}$ is retrieved from $T_g$ and joined with the path from $q_{\text{start}}$ to $q_{\text{new}}$ that is found in the tree $T$. Alternatively, the planner is terminated after the predefined number of planning iterations $I_{\text{max}}$.

The algorithm (Alg. 1) can be used both for the guided search (in such a case, $P$ is not empty and $p_{\text{guide}} > 0$) as well as for a non-biased search (if $P$ is empty or $p_{\text{guide}} = 0$). In the latter case, it behaves the same as basic RRT. The result of the sampling process described in Alg. 1 is a trajectory from $q_{\text{start}}$ to $q_{\text{goal}}$. The next section describes how to repeatedly use this sampling process to find approximate solutions and improve them.
could perform up to five retraction steps, the retraction was α the domain radius 0.4 MU and realized using the Rapid library \[13\]. RRT-ADD was used with C++ and all used the same data structures for nearest-neighbor map units (MU). All algorithms have been implemented in \(T_\text{g} \) auxiliary tree \(B\). Iterative guiding using approximate solutions with ex-

path. This process repeats until the original scale is achieved.

while using the previous approximate solution as the guiding

RRT (Alg. 1). Solution of this planning is an approximate

The latter version is referred to as RRT-IIS'.

The task was to find a path in environments where the start and/or the goal positions are buried in the narrow passageway. The test maps are depicted in Fig. 8; their size is 11 × 11 × 7 map units (MU). All algorithms have been implemented in C++ and all used the same data structures for nearest-neighbor search using the MPNN library \[33\]. Collision detection was realized using the Rapid library \[13\]. RRT-ADD was used with the domain radius 0.4 MU and \(\alpha = 0.01\), RRT-Retraction could perform up to five retraction steps, the retraction was performed in the distance 0.4 MU. RRT-IIS was run with \(I_{\text{repeat}} = 5\), \(p_{\text{guide}} = 0.9\), \(d_{\text{boost}} = 3\) MU. The objects were triangulated using the Tetgen tool \[31\] for the purpose of peeling and for thinning in MLDP. Tetgen was used with the default settings, the maximal tetrahedron volume constraint was set to 0.001.

Each method was run 200 times and the number of planning iterations was set to \(I_{\text{max}} = 500 \times 10^3\). Alternatively, planners were terminated as soon as they approached the goal to distance \(d_{\text{goal}} = 0.4\) MU. The performance of the planners is described by the probability of finding a solution with a given tree size (Fig. 9). The highest probability and the corresponding tree size was detected in this graph and related measures (e.g., runtime) were calculated. Examples of motion plans together with animations of the results are available at https://youtu.be/4XJYc0CKZG8.

The results are summarized in Tab. II. Both RRT-IIS' and RRT-IIS use the same objects obtained by the proposed peeling technique, and they differ in the usage of the extended goal region (RRT-IIS' does not use the extended goal region tree \(T_\text{g}\), while RRT-IIS does). The performance of RRT-IIS is better than RRT-IIS', which shows the advantage of extending the goal region. The runtime of RRT-IIS is lower than the runtime of RRT-IIS', despite the additional effort to connect to the actual tree \(T\) with the auxiliary tree \(T_\text{g}\). The average number of the nodes of the goal region tree \(T_\text{g}\) was 25 (measured over all the maps and all planning instances). Therefore, testing the proximity to the goal region, which is realized using KD-trees, is very fast.

The performance of RRT-DD and RRT-Retraction varies between the environments. For example, RRT-DD provides solutions with high probability in the case of C-robots (probability

\begin{algorithm}
\caption{Guided-CSpace-Sampling}
\begin{algorithmic}[1]
\State \(T\).addNode\((q_{\text{start}})\);
\State \(v = 1; \) // index of guiding waypoint \((1 \leq v \leq m)\)
\State \(\text{iter} = 0;\)
\While {\(\text{iter} < I_{\text{max}}\) and \(v \leq m\)}
\If{\(\text{random}(0,1) \leq p_{\text{guide}}\)}
\State \(q_{\text{rand}} = \) random configuration around \(q_{c}\);
\Else
\State \(q_{\text{rand}} = \) random configuration from \(C\);
\EndIf
\State \(q_{\text{near}} = T\).nearestNode\((q_{\text{rand}})\);
\State \(q_{\text{new}} = \) straight-line-expansion\((q_{\text{near}}, q_{\text{rand}})\);
\If {\(\text{noCollision}(q_{\text{new}})\)}
\State \(T\).addNode\((q_{\text{new}})\);
\State \(T\).addEdge\((q_{\text{near}}, q_{\text{new}})\);
\State \(q' = T_{\text{g}}\).nearestNode\((q_{\text{rand}})\);
\If {\(\varrho(q_{\text{new}}, q') < d_{\text{goal}}\) and \(\text{canBeConnected}(q_{\text{new}}, q')\)}
\State \(P_1 = T\).path\((q_{\text{start}}, q_{\text{new}})\);
\State \(P_2 = T_{\text{g}}\).path\((q', q_{\text{goal}})\);
\State \(\text{return} P_1 \cup P_2;\)
\EndIf
\EndIf
\State \(v = v + 1; \) // next guiding waypoint
\State \(\text{iter} = \text{iter} + 1;\)
\EndWhile
\State \(\text{return} \) failure;
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\caption{RRT–IIS: Improved Iterative Scaling}
\begin{algorithmic}[1]
\State \(\text{Input: Start/goal } q_{\text{start}}, q_{\text{goal}} \in C_{\text{free}}, \) planning iterations \(I_{\text{max}}\), stopping criterion \(d_{\text{goal}}\), guiding bias \(p_{\text{guide}}\), guiding path \(P = (q_1, \ldots, q_m)\), \(q \in C\), goal region tree \(T_{\text{g}}\)
\State \(\text{Global params.: distance to goal } d_{\text{goal}}\)
\State \(\text{Output: trajectory or failure}\)
\State \(T_{\text{g}}.\text{addNode}(q_{\text{start}});\)
\State \(v = 1; \) // index of guiding waypoint \((1 \leq v \leq m)\)
\State \(\text{iter} = 0;\)
\While {\(\text{iter} < I_{\text{max}}\) and \(v \leq m\)}
\If {\(\text{random}(0,1) < p_{\text{guide}}\)}
\State \(q_{\text{rand}} = \) random configuration around \(q_c;\)
\Else
\State \(q_{\text{rand}} = \) random configuration from \(C;\)
\EndIf
\State \(q_{\text{near}} = T_{\text{g}}.\text{nearestNode}(q_{\text{rand}});\)
\State \(q_{\text{new}} = \) straight-line-expansion\((q_{\text{near}}, q_{\text{rand}})\);
\If {\(\text{noCollision}(q_{\text{new}})\)}
\State \(T_{\text{g}}.\text{addNode}(q_{\text{new}});\)
\State \(T_{\text{g}}.\text{addEdge}(q_{\text{near}}, q_{\text{new}});\)
\State \(q' = T_{\text{g}}.\text{nearestNode}(q_{\text{rand}});\)
\If {\(\varrho(q_{\text{new}}, q') < d_{\text{goal}}\) and \(\text{canBeConnected}(q_{\text{new}}, q')\)}
\State \(P_1 = T_{\text{g}}.\text{path}(q_{\text{start}}, q_{\text{new}})\);
\State \(P_2 = T_{\text{g}}.\text{path}(q', q_{\text{goal}})\);
\State \(\text{return} P_1 \cup P_2;\)
\EndIf
\EndIf
\EndIf
\State \(v = v + 1; \) // next guiding waypoint
\State \(\text{iter} = \text{iter} + 1;\)
\EndWhile
\State \(\text{return} \) failure; // no solution
\end{algorithmic}
\end{algorithm}
100 % and 86.6 % in the case of tight C-robot), its performance decreases in Bottle-box and Cylinder environments. Despite the same size of all workspace as comparable sizes of the robots, RRT-DD is sensitive to its parameters, especially in the case where the goal configuration is in the narrow passage (e.g. C2, goal on the shaft).

Three planning problems were defined in the map C2: a) with start only the shaft, b) with the goal on the shaft, c) both start and goal have to be on the shaft. All the planners (except MLDP) found plans in 100 % of cases if the start was on the shaft and the goal not. Contrary, reaching the goal if it is on the shaft is difficult for all the planners (except RRT-IIS‘ and RRT-IIS)’. This is in accordance with the experiment described in Section III-B.

The most similar method to RRT-IIS is the PRM-based MLDP planner [14]. In our experiments, MLDP found solutions with a lower probability than RRT-IIS. We observed, that MLDP often struggled to find the initial solution (for the object with the lower volume). A typical solution of MLDP finished more than 1.0 MU far away from the goal, which indicates, that MLDP was not able to sample the narrow passages dense enough. As the narrow passages in C2 maps are caused by the shaft, they are thin and long, therefore, they are difficult to sample by the PRM planner. Contrary, RRT-IIS does not have to reach the goal configuration, as it used the extended goal region.

**TABLE II: Comparison of the planners.** Prob. is the best cumulative probability of finding a solution; Iter. denotes the number of planning iterations that are necessary to reach the given probability; Runtime is shown in format: avg/std.dev. N/A denotes the instances where no plan was found

![Fig. 8: Test environments with the start (yellow) and the goal (red) positions. The green box illustrates the size of 1 × 1 map unit.](image)

![Fig. 9: Example of the cumulative probability of finding a solution for the Box scenario.](image)
VI. CONCLUSION

This paper deals with sampling-based motion planning using approximate solutions. Sampling-based planners like RRT suffer from the narrow passage problem. The configuration space can be searched repeatedly while using various levels of object simplification. The solution from the previous level is then used to guide the search on the next level until the solution of the original problem is found. We propose a novel technique to obtain various levels of object simplification by removing border triangles (or tetrahedrons in the case of 3D objects) of the object’s triangulation. Besides, we propose to extend the goal region to increase the probability of finding the initial solution. The experiments have shown that the proposed method outperforms several state-of-the-art planners.

REFERENCES


