Computing multiple guiding paths for sampling-based motion planning

Vojtěch Vonásek a and Robert Pěnička a and Barbora Kozlíková b

Abstract—Path planning of 3D solid objects leads to search in a six-dimensional configuration space, which can be solved by sampling-based motion planning. The well-known issue of sampling-based planners is the narrow passage problem, which is caused by the presence of small regions of the configuration space that are difficult to cover by random samples. Guided-based planners cope with this issue by increasing the probability of sampling along an estimated solution (a guiding path). If the approximate solution is too far from the desired solution, the approximate solutions are computed using a proposed iterative process: after a path (solution) is found, it forms a region where the subsequent search is inhibited, which boosts the search of new solutions. The performance of the proposed approach is verified in scenarios with multiple narrow passages and compared with the state-of-the-art planners.

I. INTRODUCTION

Robotic motion planning, where the task is to find collision-free paths between two configurations through an environment, has many practical applications in robotics and other fields. In computer-aided design, path planners are the part of assembly/disassembly planners [1], where the task is to find a sequence of actions in order to assemble/disassemble a given product. In computational biology, path planning is used, e.g., to detect exit pathways from proteins [7], to detect tunnels [31], to identify binding sites [4], or even in protein folding [25].

Path planning of 3D objects requires to search the related six-dimensional configuration space (C-space), which is often achieved by sampling-based planners [20]. Sampling-based planners draw uniformly distributed random samples in the C-space and classify them as free or non-free using collision detection. The free samples are stored and connected, if possible, with their close neighbors, which results in a graph approximating the free region of C-space. Path in the graph describes a collision-free motion in the workspace.

The well-known issue of the sampling-based planners is the narrow passage problem [21], [15], [18]. Narrow passages are small collision-free regions with a low volume, therefore, the probability of placing the random samples into them is low. Consequently, the presence of narrow passages requires to significantly increase the runtime in order to find a solution.

Guided-based planners sample the C-space using non-uniform samples that are generated along a guiding path (an estimation of a possible solution). A na"ıve approach is to compute the guiding paths in the workspace, but this is useful only for low (two- or three-) dimensional C-space. Guided-sampling in a high-dimensional C-space requires the guiding paths to be computed in the C-space. To reduce the computation time required to find such a guiding path, the problem constraints can be relaxed, e.g., by reducing the volume of the robot [3], [30], [14]. By decreasing the volume of the robot, the narrow passages are widened. From the runtime point of view, finding an approximate solution is faster than solving the original problem. Sampling the space around an approximate solution may, however, fail, if the desired solution (the solution of the original problem) is located far away from the approximate solution.

A possible approach is to use multiple guiding paths, ideally with different homotopy, to maximize the probability that at least one of them can be traversable by the original robot. We propose to find multiple paths in the C-space iteratively: after a path is found, it forms an inhibited region...
around which the subsequent search is suppressed, so the search for a new alternative path is boosted (Fig. 1h–c). The probability of entering the inhibited region is maintained according to the progress of the C-space exploration. If no new path is found, the probability of entering the inhibited region is increased. This enables to find distinct paths that may pass the same region, e.g., the same narrow passage (Fig. 1h–f). The multiple guiding paths are computed considering a scaled-down version of the robot. These paths are then used to sample the C-space again to find the solution of the original problem.

II. RELATED WORK

Two widely used sampling-based planners are Rapidly-Exploring Random Trees (RRT) [19] and Probabilistic Roadmaps (PRM) [17]. Both planners generate random samples in the configuration space and store the collision-free ones into a graph structure (roadmap) which approximates the collision-free region of the configuration space. RRT simultaneously samples the C-space and expands the tree (roadmap) rooted in the starting configuration. The tree is expanded towards the random samples from nodes that are the nearest ones to a currently generated sample. PRM first samples the configuration space and stores the collision-free samples. Then, the stored samples are connected, if possible, with their near neighbors.

The original sampling-based planners [17], [19], [20] draw the random samples from the uniform distribution, which leads to the well-known narrow passage problem [18], [21], [15]. Removing these narrow passages leads to changes of connectivity of the configuration space. Due to their low volume (in comparison to the volume of the whole configuration space), the probability of placing uniformly-distributed samples into them is low. The presence of a narrow passage practically means that the number of samples has to be significantly increased, which consequently increases the computational time.

Many modifications of RRT and PRM have been introduced to cope with the narrow passage problem [10]. In scenarios with few DOFs, the difficult regions can be estimated from the properties of the workspace [18], [28], [2], [26]. For example, MA-PRM [33] generates the random samples around the medial axis of the workspace. In [8], new samples are pushed towards the medial axis. These modifications are designed for the PRM-based planners, where the random samples are generated prior their connection into the roadmap.

The same strategies cannot be directly used in RRT-based planners, as RRT expands the tree incrementally by little steps. By increasing the probability of sampling in a given region (e.g., on the medial axis), it is not ensured that the RRT tree will reach and cover the region, as its growth can be blocked by the obstacles. To increase the rate of connecting a random sample to the existing tree, Retraction-RRT [34] generates a sequence of configurations along the estimated boundary of the obstacles in the C-space. A different approach is used in ADD-RRT [16]: an activation radius is assigned to each node of the tree. A node is selected for the expansion only if the random sample is located within its radius, which helps the tree to penetrate the narrow passages.

Other RRT-based techniques to cope with the narrow passage problem utilize the concept of guided-sampling: random samples are generated with the increased probability at a desired region near the tree until the tree reaches the region. Then, the region is shifted along a guiding path [9], [30], [29]. Guiding paths are often computed in the workspace, e.g., based on Voronoi diagram [29], Reeb graph [9], or using a path computed by A* on the discretized workspace [24], [23], [5].

Using workspace-based paths, as in [29], [9], [24], [23], [6], however, does not help to sample high-dimensional C-space, because a path in the workspace is not tightly correlated with the path in the high-dimensional C-space. Steering the RRT tree growth in such space requires a guiding path in the configuration space. Constructing an ideal guiding path would require to solve the path planning problem of the same complexity (NP-hard [20]). The purpose of a guiding path is to focus the sampling towards promising parts of the configuration space that may contain a solution, but the guiding path itself does not have to be a solution for the problem.

Suitable guiding paths in the C-space can be constructed as an approximate solution of the original problem, e.g., by relaxing collision constraints [3], [14], [30]. By reducing the volume of the robot, e.g., by scaling-down [3], [30] or thinning [14], the relative volume of the narrow passages increases, which also increases the probability of sampling the passage. Practically, finding the approximate solutions is faster than solving the original problem. In [3], the PRM roadmap is computed for a relaxed version of the problem and the found solution is iteratively repaired until it is feasible for the original, non-relaxed problem. The repair process depends on the planning problem being solved, therefore, the work [3] proposes the general idea without technical details. Another PRM-based method utilizing approximate solutions is presented in [14], where both the robots and the obstacles are thinned by an adaptive factor. The level of thinning is determined using a binary search in each step of the algorithm. Guiding RRT along a single approximate solution was presented in [30]. In this case, the initial solution is found using RRT considering the scaled-down robot.

The works [3], [14], [30] utilize only a single approximate solution. The solution is iteratively improved while increasing the complexity of the problem, e.g., while increasing the scale of the robot. The improvement of the path is performed only in its vicinity. This local improvement may fail if the initial approximate solution is not homotopic to the solution of the original problem, i.e., if one path cannot be continuously deformed to the other without intersecting any obstacle. A better approach is to use multiple guiding paths, ideally with different homotopic classes. Classification of the homotopy classes has been studied only in 2D workspaces [12], [13], or on graphs [27], [32], but it is generally difficult in the case of
high-dimensional configuration space. We propose a method
to compute multiple guiding paths in the configuration space.
The new method is described in the next sections.

III. USED NOTATION

Let $C$ denote the configuration space. The collision-free
configurations form the subset $C_{\text{free}} \subseteq C$, where the robot
can move. The distance between two configurations is de-
noted $d(q,q')$. A path $P_i$ is a sequence of $n_i$ configurations
 waypoints) $P_i = (q_1, \ldots, q_{n_i}), q_i \in C, i = 1, \ldots, n_i$.
The inhibited region $\mathcal{R}$ is formed by $m$ paths $P_1, \ldots, P_m$:
$\mathcal{R} = \{ q \in C | d(q,q') \leq d_{\text{inhib}}, q' \in P_i, i = 1, \ldots, m \}$, where $d_{\text{inhib}}$ is the radius around each waypoint of the paths
forming the region. Let $q_{i,j} \in P_i, P_i \in \mathcal{R}$ is the $j$-th point
in the path $P_i$.

IV. GUIDED-BASED PLANNING WITH APPROXIMATE
SOLUTIONS

The task is to compute several collision-free paths from
an initial configuration $q_{\text{start}} \in C_{\text{free}}$ to a goal configuration
$q_{\text{goal}} \in C_{\text{free}}$, which we approach using guided-based
sampling along approximate solutions. In the first stage,
several approximate solutions are found for the small robot.
Then, the configuration space is searched again while using
the approximate solutions to guide the search.

\begin{algorithm}
\caption{Main loop of the planner}
\begin{algorithmic}[1]
\Require $q_{\text{start}}, q_{\text{goal}} \in C_{\text{free}}$, maximal number of initial
\normalfont solutions: $m$
\Ensure feasible path or $\emptyset$ if no path is found
\State $\mathcal{R} = \emptyset$ // paths defining inhibited regions
\State $\mathcal{G} = \emptyset$ // no guiding paths at the beginning
\State $\text{robot.setLowScale}()$
\For{$i = 1, \ldots, m$}
\State $P = \text{sampleConfigurationSpace}(q_{\text{start}}, q_{\text{goal}}, \mathcal{G}, \mathcal{R})$
\If{$P \neq \emptyset$}
\State $\mathcal{R} = \mathcal{R} \cup \{P\}$
\State $\mathcal{G} = \mathcal{G} \setminus \{q \in \mathcal{G} | d(q, q_{\text{goal}}) < d_{\text{safe}}\}$
\State $\mathcal{R} = \mathcal{R} \setminus \{q \in \mathcal{R} | d(q, q_{\text{goal}}) < d_{\text{safe}}\}$
\State $\mathcal{G} = \mathcal{G}$ // all paths are becoming guiding paths
\State $\mathcal{R} = \emptyset$ // no inhibited regions
\EndIf
\State $\text{robot.setOriginalScale}()$
\State $P = \text{sampleConfigurationSpace}(q_{\text{start}}, q_{\text{goal}}, \mathcal{G}, \mathcal{R})$
\State \textbf{return} $P$ // solution for the full robot
\end{algorithmic}
\end{algorithm}

The main loop of the planner is listed in Alg. 1. First,
the task is to find up to $m$ initial approximate solutions
(lines 4,9 in Alg. 1) considering the reduced robot (line 5 in Alg. 1).
After a path is found, it is added to the set of inhibited
regions $\mathcal{R}$. The inhibited region is formed by the
found paths, which start at $q_{\text{start}}$ and end in $q_{\text{goal}}$.
Inhibiting the tree growth around these start and goal configurations
would be counterproductive, therefore, the waypoints around
$q_{\text{start}}$ and $q_{\text{goal}}$ in the distance $d_{\text{safe}}$ are not considered in
the inhibited region $\mathcal{R}$ (lines 8,9 in Alg. 1). The parameter
$d_{\text{safe}}$ is illustrated in Fig. 2.

The search for the initial approximate solutions is repeated
up to $m$ times while considering the already found paths as
inhibited regions, which forces the planner to find alternative
paths (line 5 in Alg. 1). After the initial solutions are found,
they are considered as guiding paths $\mathcal{G}$ (Alg. 10 line 10) and
the configuration space is searched again (line 13 in Alg. 1)
for the full, non-reduced robot. The method to sample the
configuration space is described in the next section.

A. C-space sampling with inhibited regions

The core of the proposed planner is the sampling of the
configuration space, which is realized by a modified RRT
principle. The planner builds a single tree $T$ of collision-free
configurations rooted at $q_{\text{start}} \in C_{\text{free}}$. In each iteration, ran-
don sample $q_{\text{rand}}$ is generated, its nearest neighbor $q_{\text{near}} \in T$
in the tree is found and expanded towards $q_{\text{rand}}$. The
node is expanded using the straight-line expansion [19]. This
process repeats until the goal is reached to the predefined
distance $d_{\text{goal}}$, or until the number of planning iterations
$I_{\text{max}}$ is reached. The RRT sampling principle is extended by
two modifications: a) the sampling is guided along multiple
guiding paths $\mathcal{G}$, and b) the growth of the tree towards the
inhibited region $\mathcal{R}$ is suppressed. These modifications are
described in the following text, and the algorithm is listed in
Alg. 2. We show a single algorithm that may, depending on
the inputs, serve as a guided-based planner (if $|\mathcal{G}| \neq 0$ and
$p_{bias} > 0$) or as the planner avoiding the inhibited regions
(if $|\mathcal{R}| \neq 0$).

The purpose of guided sampling is to generate the
random samples $q_{\text{rand}}$ around the given guiding paths.
This is achieved by maintaining so-called active waypoints of each
path. The active waypoint $1 \leq v_i \leq n_i, i = 1, \ldots, m$ is an
index of a point in the path $P_i \in \mathcal{G}$. These active waypoints
define a region in which the random samples are generated.
At the beginning of the process, active waypoints are set
to the first point of each path, i.e., $v_i = 1, i = 1, \ldots, m$.
The random sample $q_{\text{rand}}$ is selected along a guiding path
with probability $p_{bias}$, otherwise, it is generated uniformly
from $C$. To generate $q_{\text{rand}}$ along a guiding path, first a path
$i$ is selected randomly and $q_{\text{rand}}$ is generated along the
active waypoint of the selected path, $q_{\text{rand}} \sim N(q_{i,v_i}, \Sigma)$,
where $q_{i,v_i}$ is the active waypoint of the path $i$ and $\Sigma$ is the
diagonal matrix with values \( d_{\text{goal}} \). Then, the nearest node \( q_{\text{near}} \) is found in the tree towards \( q_{\text{rand}} \) and it is expanded by standard straight-line expansion which results in a new candidate configuration \( q_{\text{new}} \).

After the tree is expanded by \( q_{\text{new}} \), the active waypoints \( v_i \) of the guiding paths are recalculated. If the tree approaches the active waypoint \( q_{i, v_i} \), closer than \( d_{\text{goal}} \), the active waypoint is moved to the next waypoint of the \( i \)-th path (line 27 in Alg. 2). By drawing the random samples \( q_{\text{rand}} \) around the guiding path of a randomly chosen path, the tree attempts to grow along all guiding paths simultaneously. Yet, if some of the paths cannot be followed, e.g., due to an obstacle, the tree can still follow the other guiding paths. The update of the active waypoints \( v_i \) is depicted in Fig. 3.

The purpose of the inhibited regions is to suppress the exploration of the configuration space in areas that have been already identified to contain a solution. By suppressing the search in such regions, the tree is forced to find new, alternative path.

The collision-free candidate \( q_{\text{new}} \) is added to the tree if it is not located inside the inhibited region \( \mathcal{R} \), otherwise it is added to the tree with the probability \( p_{i,j} \). In the latter case, the nearest waypoint in \( \mathcal{R} \) towards the candidate \( q_{\text{new}} \) is found; let denote this waypoint as \( q_{i,j} \); therefore, the \( j \)-th point of the path \( P_i \in \mathcal{R} \). The probability \( p_{i,j} \) of entering the region defined by the waypoint \( q_{i,j} \) is calculated as:

\[
p_{i,j} = \begin{cases} 
\exp(-\frac{B(i,j)}{a_{\text{sum}}}) & \text{if } A(i,j) = 0 \\
0 & \text{otherwise}
\end{cases}
\]  

(1)

where \( A(i,j) \) and \( B(i,j) \) are defined as:

\[
A(i,j) = \max_{k \geq j} (a_{i,k}) \quad \text{“after” } j
\]

(2)
\[
B(i,j) = \max_{k \leq j} (a_{i,k}) \quad \text{“before” } j.
\]

(3)

Therefore, \( B(i,j) \) refers to the number of (maximal) attempts to enter the inhibited region defined by the path \( P_i \) “before” the point \( j \). Similarly, \( A(i,j) \) is the maximal number of attempts to reach the inhibited region of path \( P_i \) “after” the point \( j \). Entering the inhibited region at point \( j \) (of a path \( i \)) is allowed only if the tree does not visit any of the waypoints after the waypoint \( j \) (which is indicated by zero \( A(i,j) \)). This happens if the growth of the tree into the rest of the environment is blocked by the inhibited region.

\[\text{Algorithm 2: sampleConfigurationSpace}(q_{\text{start}}, q_{\text{goal}}, \mathcal{G}, \mathcal{R})\]

**Input:** start/goal: \( q_{\text{start}}, q_{\text{goal}} \), guiding paths \( \mathcal{G} = (P_1, \ldots, P_m), \) inhibited region \( \mathcal{R} \)

**Global params.:** radius of inhibited region \( d_{\text{inhib}} \), number of sampling iterations \( I_{\text{max}} \), guiding bias \( p_{\text{bias}} \)

**Output:** path from \( q_{\text{start}} \) to \( q_{\text{goal}} \) or Failure

1. create new configuration tree \( \mathcal{T} \)
2. \( \mathcal{T}.\text{addNode}(q_{\text{start}}) \) // initialization of the tree
3. \( v_i = 1, i = 1, \ldots, m \) // active waypoints
4. \( a_{i,j} = 0, i = 1, \ldots, m; j = 1, \ldots, n_i \)
5. \( a_{\text{sum}} = 0 \)
6. for iteration = 1, \ldots, \( I_{\text{max}} \) do
7. \( \text{if random}(0,1) < p_{\text{bias}} \) and \( |\mathcal{G}| > 0 \) then
8. \( i = \text{random } 1 \leq i \leq m \) // select random path
9. \( q_{v_i} = v_i \)-th point of path \( P_i \)
10. \( q_{\text{rand}} = \text{random configuration around } q_{v_i} \)
11. else
12. \( q_{\text{rand}} = \text{random configuration from } \mathcal{C} \)
13. \( q_{\text{new}} = \text{straightLineExpansion}(q_{\text{rand}}, \mathcal{G}_{\text{rand}}) \)
14. \( \text{if isFree}(q_{\text{new}}) \) then
15. \( p = 0 \)
16. if \( |\mathcal{R}| > 0 \) then
17. \( i, j = \mathcal{R}.\text{nearestNeighbor}(q_{\text{new}}) \)
18. \( d = g(q_{\text{new}}, q_{i,j}) \) // distance towards \( q_{i,j} \)
19. if \( d < d_{\text{inhib}} \) then
20. \( a_{i,j} = a_{i,j} + 1 \) // num. attempts to enter \( \mathcal{R} \)
21. \( a_{\text{sum}} = a_{\text{sum}} + 1 \) // all attempts
22. \( p = p_{i,j} \) // Eq. (1)
23. if \( |\mathcal{R}| = 0 \) or random(0,1) < \( p \) then
24. \( \mathcal{T}.\text{addNode}(q_{\text{new}}) \)
25. \( \mathcal{T}.\text{addEdge}(q_{\text{new}}, q_{i,j}) \)
26. if \( g(q_{v_i}, q_{\text{new}}) < d_{\text{goal}} \) then
27. \( v_i = v_i + 1 \) // next guiding waypoint
28. if \( g(q_{\text{new}}, q_{\text{goal}}) < d_{\text{goal}} \) then // goal reached
29. return path in \( \mathcal{T} \) from \( q_{\text{start}} \) to \( q_{\text{goal}} \)
30. return \( \emptyset \)

In such a situation, the tree will more likely be expanded towards the inhibited region, which increases the number of entering attempts \( a_{i,j} \) (Fig. 2b). If there exists a waypoint \( j \) on a path \( i \), after which the number of attempts is zero, i.e., \( A(i,j) = 0 \), then the probability of expanding into the inhibited region around the point \( j \) is increased. Contrary, if the tree already reaches some part behind the point \( j \), then \( A(i,j) > 0 \), and entering the inhibited region at point \( j \) is not allowed anymore (Fig. 2b, c). This situation is depicted in Fig. 2b, c.

**B. Discussion**

The proposed planner utilizes the RRT principle extended by the concept of inhibited regions and by using multiple guiding paths. The complexity of each iteration of the basic RRT is given by two main parts: collision detection and nearest-neighbor search. This leads to the complexity \( O(\log n + \log k) \) for \( n \) nodes in the tree, and \( k \) geometric objects in collision detection assuming KD-tree is used for the nearest-neighbor search and hierarchical approach is used.
for collision detection. The complexity of one iteration of the proposed method is increased by the two extensions mentioned above, yielding the complexity $O(\log n + \log k + \log |R|)$, where the first two terms stand for the nearest-neighbor search in the tree and collision detection, respectively. The third term relates to the nearest-neighbor search for computing the distance to the inhibited region, assuming that KD-tree is also used to retrieve the nearest neighbor from $R$. Practically, the inhibited region is formed by the previously found paths, which typically have hundreds or thousands of nodes. Even in the situation when the inhibited region is formed by multiple paths and the total number of waypoints in $R$ is a few thousands, the nearest-neighbor search in $R$ realized by KD-tree is very fast.

Similarly to many other planners, the behavior of the proposed planner is controlled by several parameters. On the highest level, the main parameter is the initial size of the robot used to find the approximate solution. In our work, we start with the initial scale of 0.6 (60 % of the robot), but one can reduce the object even more. The work [14] suggests factor 0.5 for thinning. However, the actual value depends on the geometry of the robot. We recommend starting with the smallest scale/thinning factor that still preserves the shape and appearance of the object. For example, substantial thinning of the Hedgehog object (Fig. 7b) removes the spikes, so the maximal thinning which still preserves the spikes is suitable for finding the approximate solutions.

The number of trials to find the first approximate solutions (parameter $m$ in Alg. 1) depends on the prior knowledge about the instance being solved. We suggest setting this value according to the number of human-identified narrow passages in the workspace. In our tested case scenarios (Fig. 4), we set $m = 3$, as the three windows in the walls can be considered as narrow passages.

The guided sampling of the configuration space depends on the guiding bias $p_{bias}$, we suggest to use values in the range $0.8 \leq p_{bias} < 1$. The volume of the inhibited region, besides the number of waypoints forming it, is determined by the parameter $d_{inhib}$. The larger value of $d_{inhib}$ results in more diverse paths. However, if the paths have to pass the same narrow passage, setting $d_{inhib}$ too high would suppress the exploration of the passage. We suggest to set $d_{inhib}$ according to the average thickness of the object. The inhibited regions are not defined around $q_{start}$ and $q_{goal}$, which is controlled by the parameter $d_{safe}$. We suggest to use $d_{safe} \geq 2d_{goal}$, where $d_{goal}$ is the user-specified distance of reaching the goal.

V. EXPERIMENTAL VERIFICATION

The performance of the proposed planner, denoted as RRT-IR (RRT with inhibited regions), was verified in several scenarios and compared with other RRT-based approaches. The animation of the results are available at [http://mrs.felk.cvut.cz/icar-2019-ir](http://mrs.felk.cvut.cz/icar-2019-ir).

![Fig. 4: Scenarios with multiple passages S1–S4 (a–d) and the robots (b). The windows have different size: blue (smallest), red (middle size) and green (largest). All robots (b) can pass only through the green window. Robots reduced to 60 % of their size can pass through all windows. The green cube in the bottom left corner shows one map unit.](image)

A. Square objects

In this scenario, the task is to find a collision-free path for various objects (Fig. 4) in environments with several rooms. Three robots are considered: Cube, Stick, and L-shape (Fig. 4b). The rooms are connected through several windows forming the narrow passages of various widths. The size of the environments is $20 \times 20 \times 7$ map units (one map unit is depicted by the green box in Fig. 4). The method was compared to basic RRT [19], Retraction-RRT [34], ADD-RRT [16], RRT-IS [30], and PRM-based method MLDP [14].

The planning resolution was $\varepsilon = 0.2$ map units (resolution of the expansion step of RRT-based planners, in the case of MLDP, collision detection of edges was performed using this resolution). The approximate solutions of RRT-IR and MLDP were found for the scale 0.6 (60 % of the original robot). The parameters of RRT-IR were: $d_{goal} = 1$ map unit, $d_{safe} = 2$ map units, $d_{inhib} = 1$ map unit, $p_{bias} = 0.8$, $m = 3$. The number of sampling iterations was $I_{max} = 50 \times 10^3$ for all planners.

The empirical cumulative probability of finding a solution with a given number of planning iterations was computed. We denote this probability as the success rate in the rest of the text. The lowest number of iterations where the maximal success rate was reached was identified for each planner and other related measures, e.g., the runtime, were calculated.

The results are summarized in Tab. 1. Example of the cumulative probability for the scenario S1 and Cube robot is depicted in Fig. 6. Fig. 5 shows the progress of finding the initial approximate solutions using the inhibited regions. Despite the simplicity of the used robots, the scenarios are difficult to solve, as the windows are quite tight for the objects. Moreover, the movements of the Cube robot are further constrained in the S3 and S4 scenario, because the Cube robot, when located between the walls, cannot
rotate around the vertical axis. The best success rate was achieved by the guided-based planners RRT-IS, which uses a single guiding path, and the proposed RRT-IR, that uses multiple guiding paths. The RRT-IR was superior to RRT-IS in scenarios S3 and S4, where the initial solution (that was found for the scaled-down robot) could lead using passages, that are not traversable by the full robots. The success rate of the MLDP was lower than the success rate of RRT-IS, which was caused by the difficulty to construct the initial roadmap for the most scaled-down robots. Contrary, the initial solution for RRT-IR was built in a few thousands of iterations (Tab. I). The performance of ADD-RRT and Ret-RRT varies with the instance being solved. We explain this by the sensitivity of these planners to their parameters; in our case, we tuned the parameters of ADD-RRT and Ret-RRT using the Cube robot.

### B. Complex objects

In the second scenario, non-square robots Cylinder, Cross and Gear (Fig. 7b) are considered. Similarly to the previous case, the start and goal configurations are separated by three walls with windows. The large green window is traversable by all robots, while the smaller windows can be traversed only by the robot of the same shape. When considering the robots reduced to 0.6 of their size, all windows are traversable.

**TABLE I: Comparison of the planners in scenarios S1–S4.** Runtime is shown in format: mean/std.dev. Number of iterations of RRT-IR is shown in format: total iterations/iterations needed to find the first approximate solution.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Robot</th>
<th>RRT</th>
<th>RRT-IS</th>
<th>Ret-RRT</th>
<th>ADD-RRT</th>
<th>MLDP</th>
<th>RRT-IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1, robot Cube</td>
<td>Runtime [s]</td>
<td>0.7/1.7</td>
<td>6.57/6</td>
<td>16.6/4.2</td>
<td>0.9/1.1</td>
<td>45.5/10.3</td>
<td>1.4/1.5</td>
</tr>
<tr>
<td></td>
<td>Cum. prob.</td>
<td>86.1 %</td>
<td>99.6 %</td>
<td>96.8 %</td>
<td>57.6 %</td>
<td>97.0 %</td>
<td>100.0 %</td>
</tr>
<tr>
<td></td>
<td>Iters.×10^3</td>
<td>44</td>
<td>46/1</td>
<td>48</td>
<td>9</td>
<td>47</td>
<td>18/1</td>
</tr>
<tr>
<td>S2, robot Cube</td>
<td>Runtime [s]</td>
<td>1.5/3.5</td>
<td>8.89/8.7</td>
<td>45.7/46.9</td>
<td>2.5/7.7</td>
<td>37/0.8</td>
<td>2.1/2.6</td>
</tr>
<tr>
<td></td>
<td>Cum. prob.</td>
<td>85.7 %</td>
<td>100.0 %</td>
<td>98.4 %</td>
<td>87.6 %</td>
<td>95.0 %</td>
<td>100.0 %</td>
</tr>
<tr>
<td></td>
<td>Iters.×10^3</td>
<td>46</td>
<td>49/1</td>
<td>49</td>
<td>48</td>
<td>49</td>
<td>25/1</td>
</tr>
<tr>
<td>S3, robot Cube</td>
<td>Runtime [s]</td>
<td>2.1/1.7</td>
<td>16.4/5.8</td>
<td>45.8/66.8</td>
<td>4.1/3.3</td>
<td>47/5.7</td>
<td>10.0/6.4</td>
</tr>
<tr>
<td></td>
<td>Cum. prob.</td>
<td>87.6 %</td>
<td>63.3 %</td>
<td>92.0 %</td>
<td>88.0 %</td>
<td>99.5 %</td>
<td>91.6 %</td>
</tr>
<tr>
<td></td>
<td>Iters.×10^3</td>
<td>28</td>
<td>49/1</td>
<td>46</td>
<td>28</td>
<td>40</td>
<td>48/15</td>
</tr>
<tr>
<td>S4, robot Cube</td>
<td>Runtime [s]</td>
<td>4.0/4.0</td>
<td>32/12.8</td>
<td>52.4/65.5</td>
<td>3.7/2.8</td>
<td>43/6.0</td>
<td>10.5/7.2</td>
</tr>
<tr>
<td></td>
<td>Cum. prob.</td>
<td>85.7 %</td>
<td>100.0 %</td>
<td>91.6 %</td>
<td>86.1 %</td>
<td>100.0 %</td>
<td>100.0 %</td>
</tr>
<tr>
<td></td>
<td>Iters.×10^3</td>
<td>45</td>
<td>49/1</td>
<td>46</td>
<td>5</td>
<td>47</td>
<td>36/9</td>
</tr>
</tbody>
</table>

**Fig. 5:** Visualization of the search for initial approximate solutions for the Cube robot (yellow) in the scenario S3 (top view). After the first solution is found (a), it defines the inhibited region and the C-space is searched again. The inhibited region can be entered with the probability (cyan=zero, magenta=non-zero) depending on the progress of the tree (b,c). The results of the second search is another tree and a new path (d). The results are two different paths that are further used for the guided sampling.

**Fig. 6:** Cumulative probability in the S1 scenario and the Cube robot.
The results are summarized in Tab. II. All tested planners
their shape, rotation, translation, and internal conformational
changes need to be considered.

Although the proper analysis of tunnel’s traversability by
small ligands is beyond the scope of this paper, we show that
the proposed method can also be applied in these planning
scenarios. We simplified the problem by considering a “rigid”
ligand, i.e., we fixed its internal degrees of freedom (dihedral
angles between the ligand’s atoms). Both the ligand and
protein were represented by the hard-sphere models, i.e.,
leads to a high-dimensional motion planning problem, as
Analyzing the traversability of these tunnels by the ligands
leads to a high-dimensional motion planning problem, as
analyzing protein tunnels, assuming that a spherical
probe (one atom) moves through the tunnels [11], [22].
This scenario is motivated by the needs of biochemists to
estimate possible interactions between a protein and other
small molecule (ligand), which typically takes place in a
deply buried active site. Nowadays, this analysis is typically
based on analyzing protein tunnels, assuming that a spherical
probe (one atom) moves through the tunnels [11], [22].

C. Protein structures

This scenario is motivated by the needs of biochemists to
estimate possible interactions between a protein and other
small molecule (ligand), which typically takes place in a
deply buried active site. Nowadays, this analysis is typically
based on analyzing protein tunnels, assuming that a spherical
probe (one atom) moves through the tunnels [11], [22].

The comparison of the planners is summarized in Tab. I.
In this case, the runtime of MLDP exceeded the time limit
(10 minutes) without providing any solution, therefore the
performance of MLDP is not shown. The best performance
was achieved with the RRT-IR planner. The guided planning
with one approximate solution, that is realized by RRT-IS,
found paths with a significantly lower success rate. The
reason is that the initial solution (found for the 60 % scale of
the robot) more likely leads through the window that is not
traversable by the non-scaled robot. Similarly to the previous
experiment, the performance of Ret-RRT and ADD-RRT was
the best for the Cylinder robot that is geometrically similar
to Cube robot, for which the parameters of Ret-RRT and
ADD-RRT were tuned. This shows the sensitivity of these
methods to the proper selection of the parameters.

In the Hedgehog benchmark, the task is to remove the
spiky object from the cage. Though there are five windows,
their width is just about 10 % wider than the object’s sphere.
The results are summarized in Tab. II. All tested planners
are able to solve the simplified version of the problem (with
scaled-down spiky object), but as the scale of the Hedgehog
increases (from 0.7 to 0.85), the performance of the non-
guided planners RRT, ADD-RRT, and Ret-RRT decreases.

The Hedgehog benchmark,[1] the task is to remove the
spiky object from the cage. Though there are five windows,
their width is just about 10 % wider than the object’s sphere.
The results are summarized in Tab. II. All tested planners
are able to solve the simplified version of the problem (with
scaled-down spiky object), but as the scale of the Hedgehog
increases (from 0.7 to 0.85), the performance of the non-
guided planners RRT, ADD-RRT, and Ret-RRT decreases.

C. Protein structures

This scenario is motivated by the needs of biochemists to
estimate possible interactions between a protein and other
small molecule (ligand), which typically takes place in a
deply buried active site. Nowadays, this analysis is typically
based on analyzing protein tunnels, assuming that a spherical
probe (one atom) moves through the tunnels [11], [22].

Analyzing the traversability of these tunnels by the ligands
leads to a high-dimensional motion planning problem, as
their shape, rotation, translation, and internal conformational
changes need to be considered.

Although the proper analysis of tunnel’s traversability by
small ligands is beyond the scope of this paper, we show that
the proposed method can also be applied in these planning
scenarios. We simplified the problem by considering a “rigid”
ligand, i.e., we fixed its internal degrees of freedom (dihedral
angles between the ligand’s atoms). Both the ligand and
protein were represented by the hard-sphere models, i.e.,
each atom is represented by a sphere with radius given by
the Van der Waals radii of the atoms. The task was to
find paths for the ligand 1,2-Dichloroethane heading
from the surface of a protein Haloalkane Dehalogenase
(protein 1CQW) towards its active site. The paths were
computed in 100 protein conformations (the conformations
were obtained using Molecular Dynamics simulation). In
each conformation, the planning was realized 40 times. In
this case, the most important performance measure is the
number of protein conformations, in which a path connecting
the surface with the active site was found. The higher
number of found paths is better, as the subsequent analysis of
protein’s behavior can be performed on more data samples.

The traversability of the protein was tested with various
scales of the ligand. The results are shown in Tab. III
and an example of a found solution is shown in Fig. 8.
The table shows the runtime of finding one trajectory and
the average number of planning iterations needed to find a
solution (the values are computed over 100 frames × 40
trials, i.e., over 4000 cases). For all tested ligand’s scales,
the proposed method identified more protein conformations
where the active site can be reached.

<table>
<thead>
<tr>
<th>RRT</th>
<th>RRT-IS</th>
<th>Ret-RRT</th>
<th>ADD-RRT</th>
<th>RRT-IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shapes, Robot Cross</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime [s]</td>
<td>8.8/8.6</td>
<td>8.5/5.2</td>
<td>249.7/178.0</td>
<td>11.1/17.1</td>
</tr>
<tr>
<td>Iters. × 10^3</td>
<td>48</td>
<td>30/1</td>
<td>41</td>
<td>49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shapes, Robot Cylinder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
</tr>
<tr>
<td>Iters. × 10^3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shapes, Robot Gear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
</tr>
<tr>
<td>Cum. prob.</td>
</tr>
<tr>
<td>Iters. × 10^3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hedgehog, scale 0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
</tr>
<tr>
<td>Cum. prob.</td>
</tr>
<tr>
<td>Iters. × 10^3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hedgehog, scale 0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
</tr>
<tr>
<td>Cum. prob.</td>
</tr>
<tr>
<td>Iters. × 10^3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hedgehog, scale 0.85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
</tr>
<tr>
<td>Cum. prob.</td>
</tr>
<tr>
<td>Iters. × 10^3</td>
</tr>
</tbody>
</table>

Table II: Comparison of the planners on scenarios with
complex robots. Runtime is shown in format: mean/std.dev.
Number of iterations/iterations needed to find the first approximate solution.

Fig. 7: Scenarios with complex robots.

Fig. 8: Detail of the protein 1CQW (hard-sphere model, grey) with the ligand (colored) (a), example of a found path (blue) in protein cartoon representation (b) and with showing a part of the protein’s surface (c).
TABLE III: Planning in protein data. Runtime is shown in format mean/std. dev.

<table>
<thead>
<tr>
<th>Ligand scale</th>
<th>RRT</th>
<th>RRT-IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
<td>0.3/1.3</td>
<td>1.8/5.1</td>
</tr>
<tr>
<td>Conformations</td>
<td>198</td>
<td>179</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ligand scale</th>
<th>RRT</th>
<th>RRT-IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [s]</td>
<td>0.3/1.3</td>
<td>2.2/6.3</td>
</tr>
<tr>
<td>ITERS. × 10^5</td>
<td>198</td>
<td>194</td>
</tr>
<tr>
<td>Conformations</td>
<td>67</td>
<td>72</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

Sampling-based planners solve motion planning by randomized search of the configuration space, which is challenging if narrow passages are present. Their shape and exact location is not known in advance. In this paper, we propose to search the configuration space repeatedly. First, several approximate solutions (paths) are found for a simplified problem with the scaled-down robot. The concept of inhibited regions is introduced to iteratively find different paths in the configuration space: after a path is found, it defines an inhibited region that is suppressed in a subsequent search. Practically, finding an approximate solution is faster than solving the original problem. All approximate solutions are then used to guide the sampling of the configuration space considering the original problem. The proposed method thus increases the success rate of finding the solution of the original problem, compared to the state-of-the-art methods, as shown in several scenarios of the experimental verification.

REFERENCES


