A Sampling Schema for Rapidly Exploring Random Trees using a Guiding Path

Vojtěch Vonásek Jan Faigl* Tomáš Krajník Libor Přeučil
Department of Cybernetics, *Center for Applied Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague
{vonasek,xfaigl,tkrajnik,preucil}@labe.felk.cvut.cz

***Abstract*— In this paper, a novel sampling schema for Rapidly Exploring Random Trees (RRT) is proposed to address the narrow passage issue. The introduced method employs a guiding path to steer the tree growth towards a given goal. The main idea of the proposed approach stands in a preference of the sampling of the configuration space $\mathcal{C}$ along a given guiding path instead of sampling of the whole space. While for a low-dimensional $\mathcal{C}$ the guiding path can be found as a geometric path in the robot workspace, such a path does not provide useful information for efficient sampling of a high-dimensional $\mathcal{C}$. We propose an iterative scaling approach to find a guiding path in such high-dimensional configuration spaces. The approach starts with a scaled geometric model of the robot to a fraction of its original size for which a guiding path is found using the RRT algorithm. Then, such a path is iteratively used in the proposed RRT-Path algorithm for a larger robot up to its original size. The experimental results indicate benefit of the proposed technique in significantly higher ratio of the successfully found feasible paths in comparison to the state-of-the-art RRT algorithms.**

***Index Terms*— motion planning, RRT**

I. Introduction

The motion planning is a classical robotic problem, which can be described using a notion of the configuration space $\mathcal{C}$, where $q \in \mathcal{C}$ is a configuration of a robot. The obstacles in the workspace correspond to a set $\mathcal{C}_{obs} \subseteq \mathcal{C}$, while $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$ is a set of feasible configurations. The motion of a robot in a workspace equals to a feasible path in $\mathcal{C}_{free}$.

Several complete methods have been proposed to solve the planning problem, e.g., Voronoi diagrams or Visibility Graphs. These approaches require an explicit representation of the configuration space. However, such a representation cannot be easily computed for systems with many degrees of freedom.

To overcome this problem, sampling-based methods as Probabilistic Roadmaps (PRM) [13], Rapidly Exploring Random Trees (RRT) [15] or Expansive Spaces Trees (EST) [9] were suggested to obtain an approximation of the configuration space. The common idea of the sampling-based methods is to build a roadmap of free configurations $q \in \mathcal{C}_{free}$. These methods randomly sample the configuration space $\mathcal{C}$ and classify the samples as free or non-free using a collision detection method. The free samples are stored and connected to make a roadmap, in which a solution is found. The collision detection is applied as a "black-box", which allows to cope with robots of arbitrary shapes.

The sampling-based methods are intensively studied, and their advantages and issues have been described in many papers. Here, we refer to summary [17]. The well known issue of the sampling-based methods is the narrow passage problem. A narrow passage can be defined as a part of $\mathcal{C}$, which removal changes the topology of $\mathcal{C}$ [14]. The narrow passage becomes important if the result path in $\mathcal{C}$ has to pass it. The issue comes from a less number of the samples covering the narrow passage, which disallows to construct a feasible path through it. The sparse coverage of the passage is due to the uniform distribution, which is usually used in sampling-based motion planning methods. Even though the methods are probabilistic complete, they do not provide a feasible path within a given computational time.

In this paper, a novel sampling schema for the RRT algorithm is proposed to cope with the narrow passage problem, and to increase success ratio of finding feasible paths. The approach is based on a modification of the RRT–Path [2] algorithm, which uses a guiding path to steer the tree growth in $\mathcal{C}$. Although a guiding path can be computed directly in workspace corresponding to a low dimensional $\mathcal{C}$, it is difficult to construct a useful path in a general $\mathcal{C}$. Therefore, we propose to iteratively employ the RRT–Path algorithm to find a path for a robot with downscaled size, but preserving the same motion model. The path found for a smaller robot is then used as a guiding path for a larger one, which increases performance of the motion planning.

The paper is organized as follows. Related work is described in the next section. In Section III, the modified RRT–Path algorithm is described. It is then used in Section IV in the iterative method to find a guiding path in a high-dimensional $\mathcal{C}$. Experimental verifications are described in Section V.

II. Related work

A crucial part of the sampling-based methods is the sampling of the configuration space. The sampling process can be characterized by a sampling source and a sampling measure [11]. The sampling source denotes how the samples are constructed, e.g., using pseudorandom number generators or by deterministic approaches like Halton sequences. The sampling measure is the distribution of the samples in $\mathcal{C}$ and it is more important for the planning process itself than the sampling source [11].

In the original PRM and RRT, the configuration space is sampled uniformly. To cope with the narrow passage problem, the sampling measure can be modified to generate more samples in difficult regions. A simple method that modifies the sampling measure in the RRT is the goal-bias [16], where

random configuration is replaced by the goal configuration with a probability p_g. The goal-bias suppresses the exploration of $\mathcal{C}$, while the tree is expanded more preferably towards the goal. Although the goal-bias speeds up the growth towards the goal, it may cause a congestion problem of the tree growth due to existence of an obstacle in the pathway.

A location of narrow passage can be estimated using a shape of the workspace prior the planning. This approach is used in the Gaussian PRM [18] and Bridge-Test [8], where random configurations are generated close to obstacles. If a roadmap with a higher clearance is preferred, the samples can be generated on the medial axis [23]. Other approaches that use knowledge about the workspace to determine regions for dense sampling are [14, 22, 1, 10].

The sampling measure can be adapted in the planning phase, which allows to generate more samples in sparsely covered regions. The adaptive sampling can be based on the level of information gain brought by the samples [4, 5]. The coverage of the configuration space by the samples can be estimated using KPIECE [20] algorithm. The adaptive methods improve the sampling in high-dimensional configuration spaces, where the sampling cannot be modified respecting purely the workspace knowledge [14].

The sampling measure can be easily modified in a PRM algorithm, because such algorithm first samples the configuration space, and after that, the roadmap is constructed using the samples located in $\mathcal{C}_{free}$. The modifications originally suggested for the PRM cannot be directly used in the RRT, because the RRT algorithm simultaneously builds the tree and samples the configuration space. To modify the sampling measure in RRT, the tree growth needs to be considered, otherwise the tree can get stuck due to an obstacle, like in the situation depicted in Fig. 1a. More samples in the narrow passage np do not guarantee the tree will escape the "u" obstacle, because the tree tends to growth towards the samples in the np, but in that directions an obstacle is located. Therefore, a sampling measure should be boosted around np after the tree approaches the narrow passage (Fig. 1b).

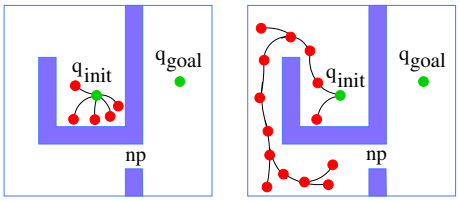


Fig. 1. Environment with a narrow passage (np). If the sampling is boosted around np before the tree approaches it, the tree attempts to grow to the obstacles (left). To help the tree to pass np the sampling measure should be boosted in np when the tree approaches it (right).

Several approaches to adjust the sampling measure considering the tree growth in the RRT have been presented so far. The tree nodes can be classified into two groups: (1) frontier nodes, whose Voronoi cells grow together with the growth of the environment; and (2) boundary nodes, which are close to obstacles [24]. The tree should be expanded from the frontier nodes, as these are found at the boundary of the tree; thus, these can be extended towards unexplored regions. In narrow passages, the nodes are mainly of both the frontier and the boundary types. These are frequently chosen for tree expansion, because they are frontier nodes. Nevertheless, the expansion of the nodes may not be successful as these are located close to obstacles. The authors of [24] proposed the RRT-DD algorithm to deactivate nodes from which the tree cannot be expanded. Each node has assigned a radius defining maximum distance of a random sample in $\mathcal{C}$, which can activate the node for the expansion. The radius is initialized to ∞; hence, the nodes can be chosen for the expansion by an arbitrary random sample. If the expansion of a node fails, its radius is decreased to a predefined value R. This suppresses frequent selection of the boundary nodes, and increases probability of selecting frontier nodes. However, the performance of the RRT-DD strongly depends on the value of R. A strategy for adapting the activation radius has been proposed in [12]. If the expansion of a node succeeds, its activation radius is increased by a given percentage α, otherwise it is decreased by the same percentage.

To attract the tree into a narrow passage, the Retraction-RRT [26] computes contact configurations q'_c that are located on a boundary of $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$. Such configurations are found using retraction procedure, which works as follows. A random configuration $q_{rand} \in \mathcal{C}_{free}$ is generated and a close non-free configuration $q_{obs} \in \mathcal{C}_{obs}$ is found. A contact configuration q_c on a segment (q_{rand}, q_{obs}) is found and its neighborhood is searched for q'_c minimizing the distance between q_{obs} and q'_c. The configuration q'_c is then added to the tree. It was shown that this approach can deal with narrow passages efficiently, because the generated contact configurations penetrate into the narrow passages. However, to find the contact configurations, the collision detection algorithm is called frequently, which can decrease the performance of the algorithm.

In [2], the performance of the RRT for problems with a low-dimensional $\mathcal{C}$ has been increased by the proposed RRT–Path algorithm. The approach is based on a precomputed geometric path in the workspace that is used to guide randomized construction of the tree through the configuration space. Therein, the guiding path is computed using well known approaches like Visibility graph, Voronoi diagram, or the Visibility-Voronoi combination. A similar idea for attracting the tree was proposed in [21]. In this approach, the tree is grown towards multiple precomputed key configurations. In the RRT–Path, only one attraction configuration is used. Moreover, this configuration is moved along the path considering the growth of the tree. It allows the tree to reach the goal configuration even in an environment with obstacles.

A significant performance improvement of the RRT–Path in problems with many narrow passages (like office like building plans) allows to consider the motion planning in the watchman route problem [6] where many trajectories have to be determined to find a final watchman route trajectory from which the whole environment is covered. However, a guiding path found in the workspace cannot be used to guide the RRT in a high-dimensional $\mathcal{C}$, because the workspace importance decreases as the dimension of the configuration space increases [14]. Therefore, in this paper, we extended the

former RRT–Path algorithm to efficiently deal with problems in a high-dimensional $\mathcal{C}$.

III. Guiding a tree using a precomputed path

Although the RRT–Path has been proposed in [2], the algorithm has been further improved to cope with high-dimensional spaces. That is why a detailed description of the modified algorithm is presented in this section. The basic idea of the algorithm is to steer the growth of the tree through the environment without unnecessary exploration steps.

To steer a growing process of the tree a precomputed guiding path is employed, and new samples are generated along this path using a temporal goal g_t, which follows the goal-bias principle. The temporal goal g_t is maintained considering the growth of the tree; thus, g_t slides along the guiding path during the tree construction. To determine the temporal goal g_t, the nearest node in the tree should be projected to a line segment of the guiding path. To avoid computing of this projection, we represent the guiding path by a set of sample points. Then, the temporal goal is searched among these samples. To bias the tree growth towards g_t a new random sample q_{rand} is randomly drawn from w neighbor sample points of g_t. So, q_{rand} is sampled around g_t with the probability p_g, otherwise q_{rand} is sampled from $\mathcal{C}$. The principle of the sampling is depicted in Fig. 2. Similarly to the original RRT, the nearest configuration q_{near} in the tree is found and the tree is extended towards q_{rand}. If the tree approaches the current goal g_t in a distance less than δ_t, the temporal goal is updated to the most distant point of the guiding path that has not yet been reached by the tree. The algorithm terminates if the tree approaches q_{goal}. The RRT–Path algorithm is listed in Algorithm 1.

The guiding path can be found easily for 2D or 3D workspaces using well know approaches like Visibility graphs or Voronoi diagrams. Here, we do not focus on the guiding path computation itself, rather we assume to have one and focus to the sampling along such a path. To measure distance between a configuration $q \in C$ and a path point p, only the corresponding variables are used in Euclidean metric. For example, for a car-like robot with $q = \{x, y, \varphi\}$ and 2D guiding path point $p = \{x, y\}$, only x and y variables are used to determine the distance between q and p.

Maintenance of the temporal goal (lines 4–12 in Algorithm 1) is based on distances between the tree and points on the guiding path that have a higher index than g_t. If the distance is lower than δ_t for a point p_i, then the next point p_{i+1} is labeled as a new temporal goal.

The considered distance δ_t implies that the tree may not follow the path exactly. A higher δ_t leads to a less accurate tracking of the guide path. In such a situation, a new path

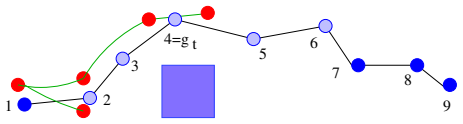


Fig. 2. An example of a guiding path with nine points. The g_t is set to point 4. If $w = 5$ neighbors of the temporal goal is used to attract the tree, q_{rand} is drawn from points $\{2, 3, 4, 5, 6\}$.

Algorithm 1: RRT–Path

Input: Configurations q_{init}, q_{goal}, the maximum number of iterations K, the temporal goal bias p_g, the number of the temporal goal neighbors w, a guiding path $P = \{p_1, \ldots, p_n\}$

Output: A trajectory between q_{init} and q_{goal} or failure

1 $T.add(q_{init})$
2 $g_t = 1$ // index of the temporal goal point
3 **while** *iteration*$< K$ and q_{goal} *not reached* **do**
4 $i = n$
5 **while** $i > g_t$ **do**
6 $q_n = \text{nearestNeighbor}(T, p_i)$
7 **if** $\text{distance}(q_n, p_i) < \delta_t$ **then**
8 $g_t = i + 1$ // new temporal goal
9 **break**
10 **end**
11 $i = i - 1$
12 **end**
13 **if** $p_g > \text{rand}(0, 1)$ **then**
14 q_{rand} = random point among w neighbors of g_t
15 **else**
16 q_{rand} = random configuration in $\mathcal{C}$
17 **end**
18 $q_{near} = \text{nearestNeighbor}(T, q_{rand})$
19 q_{new} = extend q_{near} towards q_{rand}
20 **if** q_{new} *can be connected to* q_{near} **then**
21 $T.add(q_{new})$
22 **end**
23 $iteration = iteration + 1$
24 **end**
25 **if** q_{goal} *was reached by the tree* T **then**
26 **return** trajectory between q_{init} and q_{goal}
27 **else**
28 **return** failure
29 **end**

through the environment may be found and even it can be interconnected with an existing path behind the temporal goal. So, the tree can "skip" a part of the guiding path, which is useful in situations, where a part of the guiding path is not able to steer the tree properly, e.g., due to a low clearance. An example of such a situation is depicted in Fig. 3.

A balance between path following and exploration of $\mathcal{C}$ depends on the number of neighbors of g_t and the distance between the consecutive points on the guiding path. If the number of the neighbors w is high, or the consecutive path points are too far, the algorithm prefers exploration of $\mathcal{C}$ rather than following the guiding path. The reason is that the tree is more attracted by widespread points than by the guiding path itself. If the distance between the consecutive path sample points is small or only several g_t neighbors are used to select the random points, the tree is attracted by points located in a close area; thus, the tree does not explore, but attempts to reach these points. As the distance between consecutive path points decreases, the number of path points increases, which also increases the computational burden of the temporal goal

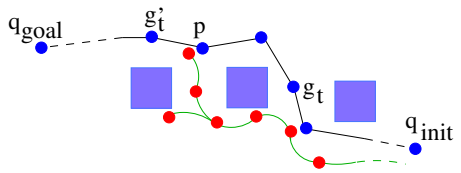


Fig. 3. Example of skipping part of the guiding path by the tree. The temporal goal is set to g_t but the tree approaches the point p. The new temporal goal is thus set to g'_t.

selection. To overcome these situations, the distance between consecutive sample points on the guiding path should not be higher than δ_t.

IV. RRT–Path with Iterative Scaling of Robot Geometry

The RRT–Path is able to grow the tree along a predefined geometric path. The guiding path can be computed easily for a point robot in 2D or 3D workspaces, while its computation is PSPACE-Hard [19] for a non-point robot. In this section, we proposed an iterative scaling algorithm denoted as RRT-IS to find a guiding path in a general $\mathcal{C}$.

The approach is based on an iterative refinement of the guiding path using a scaled model of the robot. First, the original RRT is used to find an initial guiding path with a downscaled model of the robot. This path is then used as the guiding path in the modified RRT–Path algorithm that is executed several times using an enlarged robot in a stepwise manner unless the original size of the robot is achieved. The RRT–Path attempts to follow the given guiding path at most m-times. Here, we assume that the robot is a single body robot and it is scaled equally in all dimensions. The algorithm is listed in Algorithm 2.

The narrow passages in $\mathcal{C}$ of a smaller robot are relatively wider than the passages in $\mathcal{C}$ of the original (full-sized) robot; thus, it is easier to find a path through the passages. The iterative scaling approach assumes that the path found for a smaller robot is topologically equivalent to the path for the robot with the original size. It is clear that this assumption is not always valid. For example a path in $\mathcal{C}$ found for a small robot can become infeasible for a larger robot due to existence of an obstacle, see for an example Fig. 4. However, the modified RRT–Path is able to skip a part of the guiding path, and therefore, it may find a solution also in these cases.

Performance of the proposed RRT-IS depends on the used scaling strategy. The simplest strategy is to start the planning with a low scale and increase the scale at each iteration by a predefined value. Although the initial robot scale can be chosen as extremely low to increase a chance to find an admissible guiding path, such a path may become infeasible for a larger robot. An improved scaling strategy can be used, e.g., employing a binary search, to speed up the RRT-IS algorithm.

The idea of iterative scaling approach for finding the guiding path in $\mathcal{C}$ is similar to the Iterative Relaxation of Constraints (IRC) [3], which was introduced for PRM. The IRC method iteratively scales the robot and for each scale a roadmap is

Algorithm 2: RRT–IS

Input: q_{init}, q_{goal}, the number of RRT–Path trials m, the maximum number of allowed iterations K

Output: A path for the robot with the scale 1.0 or failure

```
guidingPath = ∅
while q_goal is not reached by the original robot do
    scale = getNextScale()        // scaling strategy
    robot.scaleGeometry(scale)
    if guidingPath.size = 0 then
        p=rrtOriginal.findPath(q_init,q_goal,robot, K)
    else
        for t = 1 : m do
            p=rrtpath.findPath(guidingPath, q_init,q_goal,K)
            if goal reached by p then
                break
            end
        end
    end
    if goal reached by p then
        guidingPath = p
    else
        return failure
    end
end
return guidingPath   // a solution for scale 1.0
```

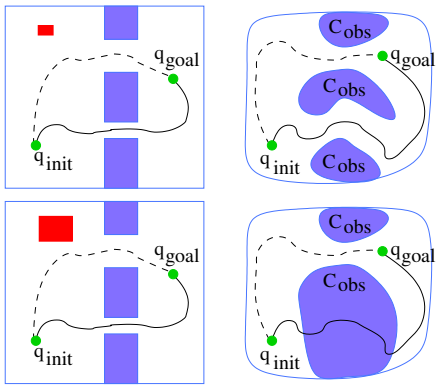


Fig. 4. An example of guiding paths found for two different scales of a robot. The workspaces with the robot are depicted in the left column, and their configuration spaces are on the right. Two paths can be found in the configuration space of a smaller robot (b); however, only the dashed path is feasible for a larger robot.

built. In the next step, the scale of the robot is increased and the previous roadmap is adapted. The benefit of our approach designed for the RRT is that it provides solution in situations where: (a) differential constraints must be considered, or (b) in a case of changing environment, where finding one path between start and goal is faster than building a roadmap of the whole configuration space. Moreover, the proposed RRT-IS uses a guiding path that considers the motion model of the robot, and therefore, it can be used to find a path for a non-holonomic robot.

V. EXPERIMENTS

The performance of the proposed modification of the RRT with iterative scaling (RRT-IS) has been experimentally verified and compared with state-of-the-arts RRT approaches in two sets of experiments. All experiments have been performed within the same computational environment using the Intel Core2Duo 3.0 GHz CPU with 2 GB RAM. The nearest neighbors in RRT algorithms were searched using the MPNN library [25] and the collisions were detected using the RAPID library [7].

In the first set of experiments, the RRT-IS was verified in 2D environments[1] BT_{100} and BT_{30} with a car-like robot. The subscript in the map name denotes the size of the narrow passage in centimeters. The size of the robot is 100×20 cm.

Each algorithm was executed 100 times for each map. A trial in which the distance between the resulting trajectory and the goal state was higher than 30 cm is considered as planning failure and the trial is discarded. The ratio of the number of planning failures to the total number of the trials performed denotes the failure ratio. Results from the successful trials are averaged.

In the RRT-IS, the scale was initialized to 0.9, and it was increased by 0.02 at each iteration. The number of the inner RRT–Path trials was set to $m = 3$. The parameters of RRT–Path were: $w = 15$, $p_g = 95\%$, and $\delta_t = 20$ cm.

The results are shown in Table I and examples of found solutions in Fig. 5. The maximum number of iterations for the RRT-Original is in row *No. iterations*. For RRT-IS, this denotes maximum number of iterations of one run of RRT and RRT–Path on a certain scale (parameter K in Alg. 2). The row *Time* is the required computational time, *Tree size* is the size of tree created by the method, *No. Collisions* is the number of collision queries. The tree size of the RRT-IS can be higher than the number of iterations as the RRT–Path algorithm can be performed up to m-times.

A passage that is a more than four times wider than the robot does not represent a significant issue for the RRT-Original algorithm in the BT_{100} environment. However, the failure ratio of RRT-Original is higher in the BT_{30} environment with a narrow passage. The failure ratio of RRT-Original depends on the maximum number of iterations. However, even if the number was increased (to $50,000$), the failure ratio of the RRT-Original was still higher than in RRT-IS with only $5,000$ of allowed iterations.

Although the RRT-IS is more computationally intensive than the RRT-Original, it provides significantly better performance regarding the failure ratio. The high computational requirements of the RRT-IS are caused by the determination of the temporal goals as RRT–Path has to find the nearest neighbors to all points on the guiding path in each iteration. The initial construction of the first guiding path for the most downscaled robot is time consuming, whilst the guidance of larger robots is faster as can be seen in Fig. 7.

The second experimental verification concerns 3D environments in scenarios Bugtrap[2] and Room. The task is to find a path for a stick-shaped robot. Both environments contain a narrow passage. The iterative scaling was started at a scale 0.05 for the Bugtrap environment and it was increased by 0.05 after each iteration. The initial scale for the Room environment was 0.7 and it was increased by 0.02 after each iteration. The parameters of RRT–Path were: $w = 15$, $p_g = 95\%$, and $\delta_t = 0.5$ map unit. A trial was accepted if the distance between found trajectory and the goal state was less than 1 map unit. The results are shown in Table II. Beside the RRT-Original, the RRT-Retraction [26] and RRT-DynamicDomain [24] with adaptive tuning were implemented for a comparison. The parameters $\alpha = 0.1$ and $R = 100\epsilon$ of RRT-DD were used, where ϵ is the length of the expansion step in the RRT.

Although the state-of-the-art methods RRT-DD and RRT-Retraction are faster and need less number of collision detection queries, their failure ratios are significantly higher than for the RRT-IS. Regarding the experimental results the proposed RRT-IS provides feasible path with a higher frequency, which indicates benefit of proposed method of guided sampling.

TABLE I

RESULTS FOR BT_{100} AND BT_{30} MAPS WITH CAR-LIKE ROBOT.

	RRT-Original		RRT-IS
BT_{100}			
No. iterations	15,000	35,000	5,000
Time [s]	1.45	3.18	5.0
Tree size	10,738	24,485	8,279
No. collisions	180,557	384,971	257,796
Failures	8 %	1 %	1 %
BT_{30}			
No. iterations	30,000	50,000	5,000
Time [s]	2.72	4.2	28.81
Tree size	19,335	26,734	18,834
No. collisions	331,085	510,084	860,929
Failures	82 %	78 %	1 %

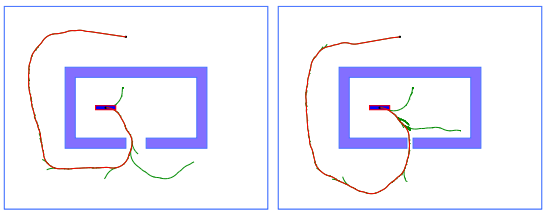


Fig. 5. Results of RRT-IS on maps BT_{100} (left) and BT_{30} (right). The rectangle represents the robot.

VI. CONCLUSION

A novel sampling schema for RRT planning algorithms has been presented. The proposed method employs a guiding path to steer the growth of the tree in the configuration space. The guiding path can be computed in the workspace by geometric path-planning methods, or by the RRT approach with the herein proposed iterative scaling. Whilst the geometric methods are more suitable for a low-dimensional configuration space, the RRT-IS can even deal with high-dimensional

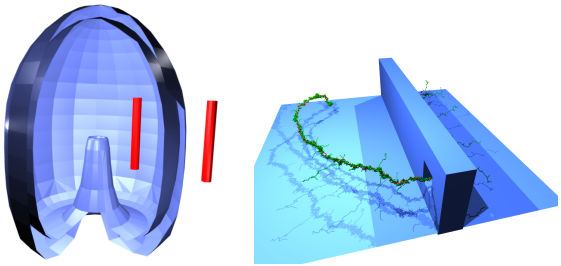
[1] Maps are available at http://imr.felk.cvut.cz/planning/maps.xml

[2] http://parasol.tamu.edu/groups/amatogroup/benchmarks/mp/bug-trap/

TABLE II

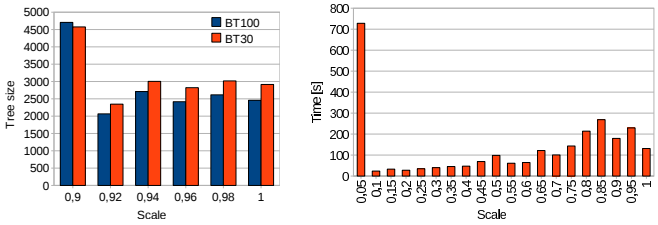
RESULTS FOR 3D ENVIRONMENTS.

	RRT-Original	RRT-IS	RRT-Retr	RRT-DD
Room				
No. iterations	200,000	40,000	40,000	100,000
Time [s]	66.84	58.2	9.9	19.2
Tree size	156,874	209,031	69,886	100,000
No. collisions ($\times 10^6$)	4.38	4.57	1.23	1.2
Failures	59%	0%	75%	80%
Bugtrap				
No. iterations	$4 \cdot 10^6$	$3.5 \cdot 10^6$	10^6	10^6
Time [s]	2,692	3,000	257	394
Tree size ($\times 10^6$)	3.8	2.2	1.63	0.99
No. collisions ($\times 10^6$)	133	120	16	12
Failures	86 %	1 %	46 %	86 %



a b

Fig. 6. 3D environments Bugtrap (a) and Room (b) with a tree along a path.



a b

Fig. 7. Tree size needed to find a solution with different scales of a car-like robot in BT_{100} and BT_{30} (a); running time of the RRT–Path for various scales of robot in the Bugtrap problem (b).

problems; thus, it can find a path through a narrow passage in the $\mathcal{C}$ even if the corresponding part of the workspace does not indicate the presence of a narrow passage.

The experimental verifications have shown that the proposed RRT-IS algorithm is able to cope with the narrow passage problem in both 2D and 3D workspaces with significantly higher success rate than other methods. Although the proposed method is computationally demanding, it may be combined with the RRT-DD or RRT-Retraction to decrease the computational burden, which is a subject of our future work.

VII. ACKNOWLEDGMENTS

This work has been supported by the Grant Agency of the Czech Technical University in Prague under grant No. SGS10/185 and SGS10/195, by the Ministry of Education of the Czech Republic under Projects No. 7E08006, No. 1M0567, and No. LH11053, and by the EU under Project No. 216342.

REFERENCES

[1] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. OBPRM: an obstacle-based PRM for 3D workspaces. In *WAFR*, pages 155–168. A. K. Peters, Ltd., 1998.

[2] Vonásek Vojtěch, Jan Faigl, Tomáš Krajník, and Libor Přeučil. RRT-Path: a guided Rapidly Exploring Random tree. In *Robot motion and control*, Poznan, Poland, June 2009.

[3] O.B. Bayazit, Dawen Xie, and N.M. Amato. Iterative relaxation of constraints: a framework for improving automated motion planning. In *IROS 2005*, pages 3433 – 3440, aug. 2005.

[4] B. Burns and O. Brock. Information theoretic construction of probabilistic roadmaps. In *IROS*, volume 1, pages 650–655, Oct. 2003.

[5] Brendan Burns and Oliver Brock. Toward optimal configuration space sampling. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[6] Jan Faigl. *Multi-Goal Path Planning for Cooperative Sensing*. PhD thesis, Czech Technical University in Prague, 2010.

[7] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, NY, USA, 1996. ACM.

[8] David Hsu. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE ICRA*, 2003.

[9] David Hsu, Jean claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *International Journal of Computational Geometry and Applications*, pages 2719–2726, 1997.

[10] David Hsu, Lydia E. Kavraki, Jean-Claude Latombe, and Rajeev Motwani. On finding narrow passages with probabilistic roadmap planners. In *WAFR*, 1998.

[11] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *International Journal of Robotics Research*, 25(7):627–643, 2006.

[12] L. Jaillet, A. Yershova, S.M. La Valle, and T. Simeon. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *IEEE/RSJ IROS*, pages 2851 – 2856, 2-6 2005.

[13] Lydia E. Kavraki, Petr Svestka, Jean claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[14] Hanna Kurniawati and David Hsu. Workspace importance sampling for probabilistic roadmap planning. In *IROS*, September 2004.

[15] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning, 1998. TR 98-11.

[16] Steven M. Lavalle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[17] Stephen R. Lindemann and Steven M. LaValle. Current issues in sampling-based motion planning. In *Robotics Research: The Eleventh International Symposium*, pages 36–54, Berlin, Germany, 2005. Springer-Verlag.

[18] Mark H. Overmars. The Gaussian Sampling strategy for probabilistic roadmap planners. In *ICRA*, pages 1018–1023, 1999.

[19] John H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of SFCS '79*, pages 421–427, Washington, DC, USA, 1979. IEEE Computer Society.

[20] Ioan Alexandru Sucan and Lydia E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *WAFR*, 2008.

[21] E. Szadeczky-Kardoss and B. Kiss. Extension of the rapidly exploring random tree algorithm with key configurations for nonholonomic motion planning. In *IEEE International Conference on Mechatronics*, 2006.

[22] J.P. van den Berg and M.H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In *IEEE ICRA*, volume 1, pages 453–460 Vol.1, April-1 May 2004.

[23] Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE ICRA*, pages 1024–1031, 1999.

[24] A. Yershova, L. Jaillet, T. Simeon, and S.M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *ICRA*, pages 3856–3861, April 2005.

[25] A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *Robotics, IEEE Transactions on*, 23(1):151–157, Feb. 2007. http://msl.cs.uiuc.edu/ yershova/MPNN/MPNN.htm.

[26] Liangjun Zhang and D. Manocha. An efficient retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation*, pages 3743 –3750, 19-23 2008.