# RRT–Path: a guided Rapidly exploring Random Tree

Vojtěch Vonásek, Jan Faigl, Tomáš Krajník, and Libor Přeučil

The Gerstner Laboratory for Intelligent Decision Making and Control
Department of Cybernetics, Faculty of Electrical Engineering
Czech Technical University in Prague
{vonasek,xfaigl,tkrajnik,preucil}@labe.felk.cvut.cz

During last decade rapidly exploring random trees (RRT) became widely used for solving a motion planning problem in various areas. Poor performance of these algorithms has been noticed in environments with narrow passages. Several variants of the RRT have been developed to address this issue.

This paper presents a new variation of the RRT designed for sampling environments with narrow passages. Performance of the proposed method has been experimentally verified and results are compared with the original RRT, RRT-Bidirectional and RRT–Blossom algorithm.

## 1 Introduction

Motion planning is one of the most studied problems in robotics. Various methods for solving this problem has been introduced in last two decades. Applications beyond robotics including 3D object manipulation, computational biology, computational graphics or drug folding are presented [9].

During last decade the RRT algorithm [11] has became widely used for solving motion planning problem. The algorithm is based on random sampling of a configuration space. The sampled configurations are connected to a tree structure in which the result path can be found. The algorithm can be divided into three main parts: selection of a vertex for expansion, expansion and terminating condition. The original RRT algorithm is outlined in Alg. 1.

The performance of the RRT algorithm can be poor in environments containing narrow passages. The narrow passages stunt the tree growing process which slows down searching of the result path. Example of environment with narrow passages is depicted on Fig. 1.

In this paper the new way of growing the tree through an environment is introduced. The proposed method called RRT–Path employs an auxiliary path that guides the tree through the environment from a start configuration to a goal configuration.

The rest of the paper is organized as follows. In next section a brief summary of related works is given. The main part of the paper is dedicated to description of the new extension called RRT–Path, which introduces new approach for selection of a vertex for the expansion. The proposed algorithm and several methods for computing an auxiliary path are described in section 3. Experimental results are presented in section 4.

**Algorithm 1**: Original RRT

```
1  T.add(q_start)
2  while iteration < K do
3  |   q_rand = random configuration
4  |   q_near = nearest neighbor in tree T to q_rand
5  |   q_new = extend q_near toward q_rand
6  |   if q_new can connect to q_near then
7  |   |   T.addVertex(q_new);
8  |   |   T.addEdge(q_near,q_new);
9  |   end
10 |   if ϱ(q_new, q_goal) < distanceToGoalTh then
11 |   |   break;
12 |   end
13 end
```

## 2 Related works

Several methods have been proposed to improve performance of the RRT algorithm. One of the first method to speed up process of tree growing toward a goal configuration is based on replacement of $q_{\text{rand}}$ by the $q_{\text{goal}}$ in certain number of iterations [9]. This extension is known as the *goal bias* and it is number of iterations between repeated replacement of the $q_{\text{rand}}$ by the $q_{\text{goal}}$. In an environment with concave obstacles higher goal bias can however worse efficiency of the algorithm. Another early extension is given by the RRTConnect [8] where the expansion step is iterated until no new vertex can be added.

To improve performance of the algorithm in environments with narrow passages, the bidirectional version of the RRT algorithm called RRT–Bidirectional [10] can be used. The algorithm uses two trees. One is growing from the start configuration and the other from the goal configuration. If the trees meet close enough the result path can be found.

The Dynamic–Domain RRT [15] changes way how a vertex is selected for the expansion. Each vertex has its action radius $r$ and it is selected for the expansion only if its radius is larger than the distance to $q_{\text{rand}}$. If the expansion

is successful, the radius of the expanded vertex is set to infinity otherwise the radius is set to value of parameter $R$.

In [3] authors present modification of sampling schema in which configurations closer to obstacles are sampled. An approach based on identification of narrow passages has been introduced in [2]. Knowledge of location of narrow passages enables more density sampling around them which can help the algorithm to find solution.

Extension proposed in [6] called RRT–Blossom adds constraint to a vertex being added to the tree: vertex $v$ is added to the tree only if the distance from $v$ to its parent vertex is not greater then distance from $v$ to another vertex in the tree. This constraint speeds up growing toward unexplored regions. The proposed approach works well for simple kinematic systems, but it becomes problematic for kinodynamic systems.

However several approaches to improve the RRT technique by a usage of prior knowledge of the environment in more or less explicit form has been published, the explicit combination of other techniques with the RRT has not been found in literature. That is why this paper presents combination of several simple path planning techniques with the RRT algorithm.

The real time performance of the RRT algorithm is also influenced by particular implementation of its parts. The most time consuming parts of the RRT schema are searching for the nearest neighbors in the tree and testing if a new configuration being added to the tree is collision free. To support fast searching the KD–tree structure can be used. Implementation of the KD–tree data structure appropriate for the RRT is described in [16]. Several methods to test collision of the robot configuration with the environment can be used. One of the fastest method called RAPID [5] has been used in presented experimental results.

## 3 RRT–Path

To speed up configuration sampling near narrow passages the RRT–Path algorithm employs an auxiliary path. The auxiliary path represents knowledge of the environment, it is a rough estimation how to get to the goal position. The RRT samples configurations along this path. The algorithm is outlined in Alg. 2. The auxiliary path is assumed to be traversed by simple turn–move movement and no differential constraints are considered during its construction hence the path can be generated only in 2D ($x$ and $y$ robot's position). Various methods can be used for computing the auxiliary path. In the following section four methods to compute an auxiliary path are discussed. These methods represent simple path planning techniques.

### 3.1 Generating auxiliary path

**Visibility graph** - The simplest approach to generate an auxiliary path is based on computation of a visibility graph of the environment. The visibility

**Algorithm 2**: RRT–Path

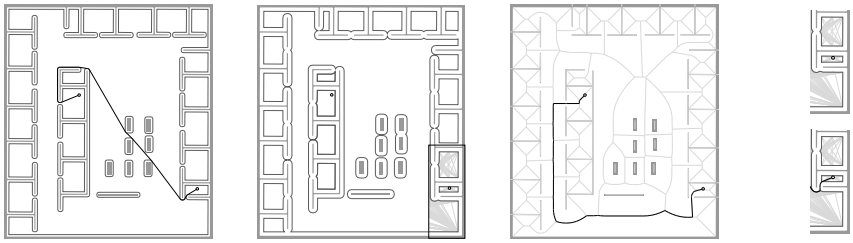
```
1  T.add(q_start)
2  P[] = prepare auxiliary path from q_start to q_goal
3  D[] = distances between points in P to the nearest point in tree T
4  temporalGoal = first point in P[];
5  while iteration < K do
6  |  if temporal goal is reached then
7  |  |  choose next temporalGoal in P[] not reached yet according to D[];
8  |  end
9  |  if iteration % tempGoalBias != 0 then
10 |  |  q_rand = temporalGoal
11 |  else
12 |  |  q_rand = random config from whole configuration space
13 |  end
14 |  for every point i in path update its D[i]
15 |  extends tree in same way as in basic RRT
16 |  stopping configuration same as in basic RRT
17 end
```

graph of the polygonal representation of the environment can be obtained in $O(n^2)$ [13]. However a path from the start position to the goal position can be easily found by the Dijkstra's algorithm, such path is not well suited to support the RRT. Segments of the path can lie very close to obstacles or even they can be part of an obstacle edge, hence sampling in the RRT algorithm will likely to fail. To avoid such possible collisions, a visibility graph is computed on grown polygonal representation. The Minkowski addition [12] of a polygon with a disk can be used in case of a differential robot. Found auxiliary path for the map jari-huge and offset parameter 20 is depicted in Fig. 1 (a).

The polygon growing approach has two drawbacks. If just a visibility graph is used to find auxiliary path, it is not guaranteed such path exists in the visibility graph, because after polygon growing, a free space can be divided into several disconnected regions, see Fig. 1 (b) and (c). The second drawback is required computation time, which depends on number of vertices of the polygonal map. This number is increased after polygon growing.

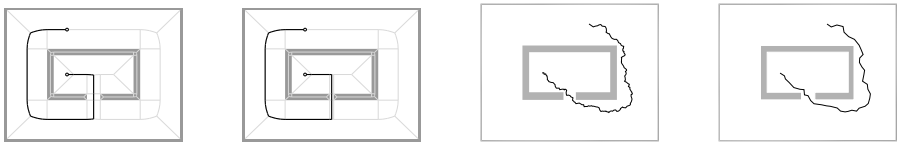
**Voronoi diagram** - Instead of visibility graph the segment Voronoi diagram can be used. The auxiliary path is also found by the Dijkstra's algorithm in a graph with added edges. The edges represent segments from the start (resp. goal) position to the closest vertex in the Voronoi diagram. As segments of Voronoi diagram are "in the middle" of the free space, polygon growing is not necessary. An example of found path is shown in Fig. 1 (c).

Figures 2 (a) and (b) show almost no difference in auxiliary path for environment BT1 and BT4. The BT4 environment has just narrower entrance to the room than in the BT1. This approach does not require polygon growing, but auxiliary path are typically longer.



(a) (b) (c) (d)

**Fig. 1.** Growed polygon as a segment approximation of Minkowski addition: (**a**) offset 20, (**b**) offset 40 with part of disconnected visibility graphs; (**c**) Segment Voronoi diagram of jari-huge environment with added edges and found auxiliary path; (**d**) Detail how Visibility-Voronoi diagram connects visibility graphs - disconnected visibility graphs for environment jari-huge after polygon growing by 40 (upper) and found auxiliary path in Visibility-Voronoi diagram (below)



(a) (b) (c) (d)

**Fig. 2.** Found auxiliary path on Segment Voronoi diagram, (**a**) BT1 environment, (**b**) BT4 environment; Example of auxiliary path computed by PRM planner (**c**) and its simplification (**d**) in BT1 environment

**Visibility-Voronoi diagram** - The Visibility-Voronoi diagram [14] combines two previous algorithms. The main idea of the combination is a connection of disconnected visibility graphs by edges from Segment Voronoi diagrams, see Fig. 1 (c). As this method combines both previous approaches it is also more time consuming than previous. The Visibility-Voronoi is suitable in cases where the path should not be in the middle of free space corridors and pure visibility graph does not provide an auxiliary path.

**PRM planner** - An auxiliary path can be also constructed by simple PRM planning method [7] that works in 2D. An environment is randomly sampled by $m$ points with uniform distribution. For each point it is determined whether a robot in that position collides with any obstacle. To ensure turn–move motion of the robot through the auxiliary path each point on the path should be tested to be collision free for several orientation of the robot.

A graph is constructed from sampled collision free points. Vertices represent random collision free points and an edge $(i, j)$ exists only if the point $j$ is one of $k$ nearest neighbors to the point $i$ and the robot can move without collision on straight line from the point $i$ to the point $j$. Start and goal states are added to this auxiliary graph and connected in the same way with the

rest of the graph. The path between the start position and the goal position is found by the Dijkstra's algorithm as in the previous methods.

The PRM approach provides an auxiliary path which seems to be too rugged, see Fig. 2 (c). One should ask whether the auxiliary path can be simplified. As the path must preserve ability to track a robot with turn–move behaviour, the algorithm inspired by [4] has been used to simplify the path. The algorithm has been extended by additional condition that point $i$ can be connected to point $j$ only if their distance is less than predefined parameter $s$. An example of simplified path is shown in Fig. 2 (d).

This approach is suitable for planning during mobile robot mapping mission. On-line created maps usually consist of hundreds of vertices and robust geometric operations does not have sufficient performance.

**Parametrization of auxiliary path**

The methods described above find an auxiliary path that is used in the RRT–Path as a guide in which direction sampling of random configurations should be performed. Properties of the found auxiliary path depend on parameters of particular method. An auxiliary path found by Segment-Voronoi Diagram approach is a property of the environment hence it is unique for the environment. A path found by algorithms based on visibility graph is parametrized by the offset value used to grow polygon. The offset should be at least radius of circle in which a robot is able to turn to any orientation. Higher values lead to longer paths and possibly disjoint regions of free space. For a certain point of view such path is more similar to path found in the Voronoi Diagram. The main objective to select appropriate offset value is to provide enough free space around an auxiliary path to let the RRT expand. The PRM approach provides an auxiliary path which can be simplified by a value of parameter $s$.

Results of initial experiments with found auxiliary paths have shown negative influence of the simplification parameter to number of samples in the RRT algorithm. If a path is simplified too much the algorithm looses its ability to sample configuration along the path, because vertices on the path are too far. In perspective of this observation each found auxiliary path must be sampled to have vertices sufficiently close. That is why the $s$ parameter is used for all auxiliary paths regardless by which method has been found. A number of vertices of the auxiliary path is appropriately increased or decreased.

### 3.2 Sampling configurations along auxiliary path

The main contribution of the proposed RRT–Path algorithm is novel approach of sampling configurations in the environment. The algorithm uses an auxiliary path which attract the tree to grow along the path from the start configuration to the goal configuration.

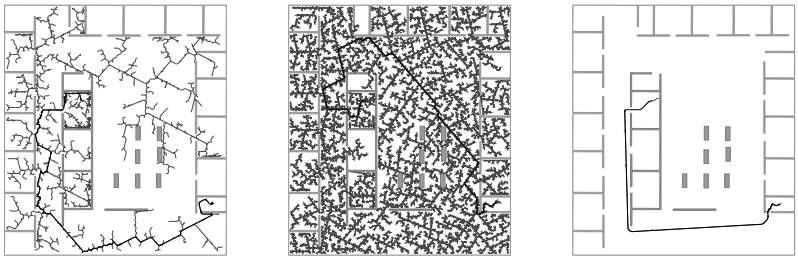
To ensure proper sampling near an auxiliary path, each point in the path stores its distance to the nearest configuration in the tree. Proposed methods

for computing auxiliary path work in 2D so the distance between point in the auxiliary path and configuration in the tree is computed as 2D Euclidean distance.

To grow the tree along an auxiliary path a temporal goal moving on the auxiliary path is used. As the algorithm samples configurations along the path, the temporal goal is dynamically changed and it moves toward the goal state $q_{\text{goal}}$. If the tree approaches temporal goal close enough, the partial goal is moved on the auxiliary path toward the goal state.

Expansion of the tree is done in the same way as in the original RRT algorithm. To permit sampling configurations in the whole environment, the goal bias is used. In every $k$-th iteration the random configuration $q_{\text{rand}}$ is sampled from whole configuration space, otherwise $q_{\text{rand}}$ is sampled around actual temporal goal. The parameter $k$ is called the *temporary goal bias* in rest of this paper. The algorithm terminates if the tree is close enough to the target configuration $q_{\text{goal}}$ or number of iterations exceeds predefined parameter $K$.

It is worth to mention that updating distance from a point of the auxiliary path to its nearest neighbor in the tree can be done in logarithmic time with the KD–tree supporting structure.



(a) (b) (c)

**Fig. 3.** Examples of generated tree in map jari–huge with final path: (**a**) original RRT, (**b**) RRT–Blossom, (**c**) RRT–Path

## 4 Experiments

The proposed method has been experimentally verified in environments jari–huge and BT[1].These environments have been chosen for their narrow passages. The original RRT algorithm, RRT-Bidirectional and RRT–Blossom have been also implemented to compare performance of the proposed algorithm. The KD-tree structure and the RAPID library for collision tests has been used in all tested algorithms.

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

| Map | Method | Preparation time [ms] | Tree size | Nbr. iter. | Time [ms] | Path length |
|---|---|---|---|---|---|---|
| BT1 | Voronoi | 20 | 220 | 385 | 600 | 73 |
| BT4 | Voronoi | 20 | 250 | 781 | 720 | 78 |
| BT1 | PRM | 1,590 | 189 | 274 | 1,580 | 74 |
| BT4 | PRM | 1,630 | 297 | 568 | 1,860 | 68 |
| jari–huge | Visibility off. 20 | 1,700 | 753 | 7,539 | 3,090 | 109 |
| jari–huge | Voronoi | 2,600 | 760 | 153 | 300 | 96 |
| jari–huge | Vis–Voronoi off. 20 | 3,420 | 595 | 7,525 | 3,700 | 111 |
| jari–huge | Vis–Voronoi off. 30 | 3,380 | 592 | 4,632 | 4,400 | 110 |
| jari–huge | Vis–Voronoi off. 60 | 2,900 | 96 | 171 | 200 | 83 |
| jari–huge | PRM | 6,000 | 591 | 5,625 | 3,040 | 100 |

**Table 1.** Performance of the RRT–Path for several auxiliary paths, *Preparation time* is creation of the auxiliary path, the tree construction phase is characterize by number of iterations and *Time*, *Path length* represents quality of founded path

Maximum number of iterations of all algorithms has been set to $5,000$ for the BT map and $15,000$ for the jari–huge map. All algorithms terminate if they had reached goal state to distance less then 30 cm. Planned trajectories have been found for a differential robot with dimensions 20x20 cm. The Euclidean distance has been used as metric for distance between configurations. All experiments have been performed on the Core 2 Duo 2.8 GHz computer with 4 GB of RAM, all algorithms have been implemented in C++.

Comparison of presented methods to generate an auxiliary path has been studied for following environments and methods with particular parameters. The Voronoi Diagram and the PRM planner methods have been tested in all environments: jari-huge, BT1, BT2, BT3, BT4. Visibility based methods have been applied only for the jari-huge environment with following particular parameter values: the Visibility Graph with the polygon offset 20 cm and the Visibility-Voronoi Diagram with offsets 20 and 30 cm. The auxiliary path simplification parameter (resp. sampling parameter) has been chosen to be twice longer than size of the robot, so the consecutive vertices on the path are closer than 40 cm. A value of the temporary goal bias parameter $k$ has been experimentally identified and $k = 15$ has been used during experiments. The Visibility Graph and the Voronoi Diagram algorithms have been implemented in CGAL [1] with the exact kernel to ensure robust geometrical computation.

Performance characteristics of the RRT–Path algorithm with different auxiliary path are shown in Table 1. Presented values are averages from 400 runs.

The comparison between the original RRT, RRT-Bidirect, RRT–Blossom and RRT–Path has been tested in both environments. For each environment and algorithm $5,000$ runs have been performed and average values are shown in Table 2. The size of the tree is lowest for the proposed RRT–Path algorithm, as this number is related to number of needed collision tests, the RRT-Path is also the fastest algorithm.

| | **Map** | **BT1** | **BT2** | **BT3** | **BT4** | **jari–huge** |
|---|---|---|---|---|---|---|
| | *passage width* | *100* | *75* | *50* | *30* | *100* |
| **Original RRT** | time | 250 | 260 | 270 | 350 | 1,790 |
| | treeSize | 1,215 | 1,251 | 1,470 | 1,792 | 8,138 |
| | nbr. iter. | 1,587 | 1,671 | 2,812 | 2,284 | 10,518 |
| | pathLength | 82 | 83 | 80 | 87 | 149 |
| **RRT–Bidirect** | time | 3,710 | 3,270 | 3,570 | 4,320 | 7,510 |
| | treeSize | 2,200 | 2,260 | 2,308 | 2,536 | 3,410 |
| | nbr. iter. | 2,624 | 2,685 | 2,776 | 3,283 | 5,626 |
| | pathLength | 87 | 88 | 88 | 89 | 140 |
| **RRT–Blossom** | time | 570 | 970 | 1,000 | 1,050 | 640 |
| | treeSize | 5,475 | 6,213 | 6,841 | 6,982 | 8,183 |
| | nbr. iter. | 1,991 | 2,049 | 2,119 | 2,149 | 2,400 |
| | pathLength | 67 | 67 | 68 | 68 | 73 |
| **RRT–Path** | time | 40 | 40 | 120 | 120 | 210 |
| | treeSize | 58 | 56 | 84 | 82 | 95 |
| | nbr. iter. | 93 | 61 | 307 | 255 | 170 |
| | pathLength | 53 | 53 | 57 | 59 | 97 |

**Table 2.** Final comparison between algorithms on various maps

## 5 Conclusion

New variation of the RRT algorithm has been introduced. The proposed algorithm is called RRT–Path and has been designed to solve motion planning problem in the environments with narrow passages. Prior to building the RRT tree, an auxiliary path from the start to the goal position is constructed and then it is used as a guide to support sampling of the environment in the RRT algorithm. Several methods for auxiliary path construction has been discussed and tested. From these preliminary results the auxiliary path found by the Visibility-Voronoi outperforms other approaches for the final path length and real time performance of the RRT algorithm aspects.

The fundamental question about proposed algorithm with guided sampling is whether it is not in contrary with primary principle of randomized sampling. The auxiliary path brings information about the environment which should be gained by randomized sampling. Despite of this question the results indicate significant reduction of tree size without affect to final path length and total computation time. At least each vertex in the tree must be tested to be collision free. Less number of required collision tests allows to use more complicated detection of collisions.

The next step to verify proposed approach is experimental verification in problems which require consideration additional DOFs in case of more complex robot.

**Acknowledgments.** The presented work has been supported by the Ministry of Education of the Czech Republic under program "National research program II" by the project 2C06005 and by the research grant MSMT No. 1M0567. The work has been partially supported by the FP7-ICT project 'REPLICATOR' No. 216240.

## References

1. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.
2. J. Berg and M. H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners, 2003.
3. V. Boor, M.H. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. *Robotics and Automation*, 2:1018 – 1023, 1999.
4. Z. Deak and R. Jarvis. Robotic path planning using Rapidly Exploring Random Trees. *Proceedings of the Australian Conference on Robotics & Automation*, 2003.
5. S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. pages 171–180, 1996.
6. Maciej Kalisiak. RRT-blossom: RRT with a local flood-fill behavior. *International Conference on Robotics and Automation*, 2006.
7. Lydia E. Kavraki, P. Svestka, J.-C. 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.
8. James J. Kuffner, Jr. Steven, and M. Lavalle. Rrt-connect: An efficient approach to single-query path planning. In *In Proc. IEEE Int. Conf. on Robotics and Automation*, pages 995–1001, 2000.
9. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.
10. Steven M. Lavalle. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.
11. LaValle S. M. Rapidly–exploring random trees: A new tools for path planning, 1998.
12. Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998. Hardback ISBN: 0521640105; Paperback: ISBN 0521649765.
13. M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 164–171, New York, NY, USA, 1988. ACM.
14. Ron Wein, Jur P. van den Berg, and Dan Halperin. The visibility–voronoi complex and its applications. In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, pages 63–72, New York, NY, USA, 2005. ACM.
15. Anna Yershova and Léonard Jaillet. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *in Proceedings IEEE International Conference on Robotics and Automation*, pages 3867–3872, 2005.
16. Anna Yershova and Steven M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.