ISSN 2658-5014 (Print)
2020. Т. 2, № 2 (4). С. 16–22
СИИТ
СИСТЕМНАЯ ИНЖЕНЕРИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
ISSN 2686-7044 (Online)
http://siit.ugatu.su

УДК 004.65

THE RTR PATH PLANNER FOR DIFFERENTIAL DRIVE ROBOTS

D. KISS [1], G. TEVESZ[2]

[1] domokos.kiss@aut.bme.hu, [2] tevesz@aut.bme.hu

Budapest University of Technology and Economics, Budapest, Hungary

Поступила в редакцию 21 октября 2020 г.

Abstract. Path planning among obstacles is a challenging task for nonholonomic mobile robots. One class of the most widespread approaches for solving this are sampling-based roadmap methods. These take samples from the configuration space and use a local planner to connect them and to build a roadmap. The resulting path is obtained by a graph search algorithm in this roadmap. This paper presents the RTR-Planner, a sampling-based global planning algorithm for differential drive mobile robots, based on the idea of Rapidly exploring Random Trees (RRT). The RTR-planner builds two search trees consisting of rotation (R) and translation (T) primitives, starting from the initial and the goal configuration. Samples are taken randomly or biased towards passages in the free workspace. If the two trees reach each other, the resulting path can be obtained easily. Simulation results are presented which show the effectiveness of the method even in the presence of narrow corridors and passages.

Key words: robots; locomotion systems; algorithm; construction process, method.

INTRODUCTION

Motion planning for robots is a widely studied field in the last decades [1]. The application of autonomous mobile robots is increasingly widespread thus planning and control of their motion in environments cluttered with obstacles is a very important area of research. From the perspective of motion planning, any robot can be described by its *configuration*. For example, the configuration of a rigid mobile robot moving in a planar workspace $W \subset R^2$ can be given by $q = (x, y, \theta)$, a vector of its position and orientation in the configuration space C. The set of not allowed configurations (e.g. because of collision with obstacles) is called the configuration space obstacle C_{obs}, its complement is the free space $C_{free} = C \setminus C_{obs}$.

The most popular locomotion systems for mobile robots moving on a planar surface are based on rolling wheels, according to their mechanical simplicity and popularity among everyday vehicles. However, the rolling without slipping constraint of wheels induce nonholonomic kinematic constraints which cause remarkable difficulties in the control of these systems [2] (this can be acknowledged by anyone who ever tried to parallel park a car). Although the motion control of a nonholonomic vehicle (like a car or a differential drive robot) is a control theoretic challenge, it is worth incorporating the knowledge about the specific system in the path planning task as well.

The RTR (rotate–translate–rotate) planning algorithm described in this paper was motivated by the differential drive, which is one of the most popular wheel arrangements among mobile robots. Its kinematic motion equation looks like as follows:

$$\begin{aligned} \dot{x} &= \cos(\theta) \cdot \frac{v_l + v_r}{2} \\ \dot{y} &= \sin(\theta) \cdot \frac{v_l + v_r}{2} \\ \dot{\theta} &= \frac{v_l + v_r}{w_b} \end{aligned} \tag{1}$$

where v_l and v_r are the velocities of the left and right wheels, respectively, treated as input variables of the system, and w_b is the wheelbase of the robot. This arrangement induces a nonholonomic kinematic constraint, namely the robot can move only in the direction of its actual heading, however, it allows the robot to turn in place. This ensures a great mobility e.g. compared to a steered, car-like robot.

Based on this property, a differential drive robot can move in any direction provided that it has enough space to turn to that direction. It follows that if the robot can be virtually replaced by a disk of its turning radius, then any geometric path planned for a holonomic (non-constrained), circle-shaped robot of the same size can be followed using differential drive as well. Unfortunately this simplification is useless if the environment contains narrow corridors and passages (see Fig. 1). Our currently proposed RTR planning method does not need the above mentioned simplification, because it takes the exact shape of the robot into account. The RTR-planner can be treated as an adaptation and modification of the Rapidly exploring Random Trees (RRT) approach [3] to the case of polygonal-shaped differential drive robots moving among polygonal obstacles.

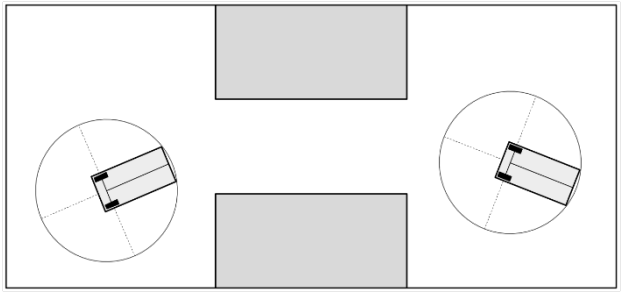


Fig. 1. A differential drive robot can be replaced by a holonomic disk of its turning radius for path planning. Although, this leads to problems when narrow corridors have to be passed

The remainder of the paper is organized as follows. An overview of planning approaches for nonholonomic robots is given in Section 2, including the RRT approach which served as inspiration for the invention of our method. In Section 3 we explain our RTR planning algorithm in detail. To show the effectiveness of the planner, simulation results are presented in Section 4. Conclusions and directions of future work are summarized in Section 5.

NONHOLONOMIC PLANNING AMONG OBSTACLES

Generating feasible paths for nonholonomic robots is not trivial even in the absence of obstacles. Algorithms that can solve this are called local planners or steering methods. In case of specific wheeled robots exact methods exist for computing optimal (e.g. shortest length) local paths. These include car-like robots moving forward (Dubins-car [4]) or both forward and backward (Reeds–Shepp-car [5, 6]), and robots equipped with differential drive [7].

However, a useful planning algorithm has to generate paths in the presence of obstacles while taking into account the kinematic constraints of the vehicle as well. To the best of our knowledge, there is no general *optimal* solution available for this problem. For differential drive, an optimal approach is presented in [8], but only for disk-shaped robots. Thus generally, if obstacles are present, one should be satisfied with a *feasible* solution which is not necessarily optimal. The majority of planning algorithms delivering a feasible solution can be grouped into two main categories. The first category consists of techniques that approximate a not necessarily feasible but collision-free initial geometric path by a sequence of feasible local paths obtained by a steering method [9, 10]. The second category involves sampling-based roadmap methods, which build a graph in order to capture the topology of C_{free} and use local steering methods to connect the graph nodes.

SAMPLING-BASED ROADMAP METHODS

A good survey of sampling-based planning methods can be found in [11]. The majority of these are based on random sampling of the configuration space. Their popularity arise from the fact that they do not require an explicit representation of C_{obs}, only a black-box collision detector module which can tell whether a given configuration is in C_{obs} or not. These methods proved to be successful in many planning problems, including high-dimensional configuration spaces. For example, the Probabilistic Roadmap Method (PRM) [12] samples the configuration space in advance and tries to connect the samples using collision-free local paths in order to obtain a roadmap (preprocessing phase). In the next step the initial and goal configurations are connected to the roadmap and a solution path is obtained by a graph search algorithm (query phase). As the preprocessing phase usually requires great computational effort, this approach is well-suited to multiple query problems in a static environment.

For single-query problems, the Rapidly exploring Random Trees (RRT) approach is better suited [3]. The main idea of it is to incrementally build a search tree starting from the initial configuration in a way that the tree covers the free space rapidly and with gradually increasing resolution.

RAPIDLY EXPLORING RANDOM TREES

The basic RRT construction process can be seen in Algorithm 1. The building of the tree T starts from the initial configuration q_{init}. RANDOM_CONFIG returns a random configuration q_{rand} from C (sampling step).

Algorithm 1.
The basic RRT construction algorithm [13]

1. $T.\text{init}(q_{init})$
2. **for all** $k = 1$ **to** K **do**
3. $q_{rand} \leftarrow$ RANDOM_CONFIG()
4. $q_{near} \leftarrow$ NEAREST_NEIGHBOR(q_{rand}, T)
5. **if** CONNECT$(T, q_{rand}, q_{near}, q_{new})$ **then**
6. $T.\text{add\_vertex}(q_{new})$
7. $T.\text{add\_edge}(q_{near}, q_{new})$
8. **end if**
9. **end for**
10. **return** T

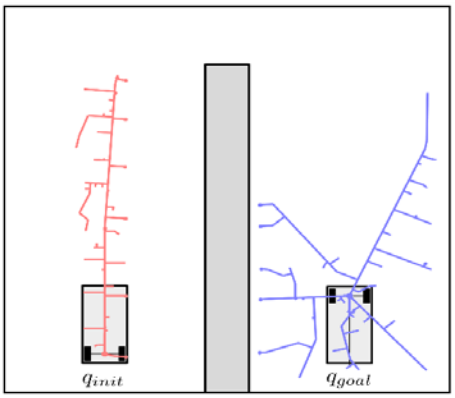
NEAREST_NEIGHBOR determines the nearest configuration q_{near} in the tree, according to a metric defined on the configuration space (vertex selection step). It depends on the implementation if this function can return only graph vertices or inner configurations of edges as well. CONNECT tries to connect q_{near} to q_{rand} by interpolating between them (tree extension step). More versions of this function are proposed by the authors of RRT. The first version extends q_{near} only by a fixed Δq amount towards q_{rand} to obtain q_{new} (let us call this version EXTEND instead of CONNECT. The second version extends q_{near} until it is connected to q_{rand} or a collision is detected. In this case q_{new} will be the farthest collision-free configuration towards q_{rand}. In order to reach the goal configuration, the random sampling can be biased to include q_{goal} sometimes in the random sequence, or a bidirectional search can be performed by growing two trees from both the initial and goal configurations [14].

The RRT method can be applied to nonholonomic systems as well. Instead of a simple interpolation towards q_{rand} (which assumes free mobility in any directions) a system-specific action should be applied in the CONNECT step. In case of its EXTEND version, an input has to be chosen and applied for a given time quantum Δt to obtain a Δq which brings q_{near} closer to q_{rand} (actually this was the original version proposed by the authors of RRT in [3]). On the other hand, the CONNECT version can be applied for systems where an explicit steering method is available to connect two configurations.

THE RTR-PLANNER

In case of the differential drive, both versions can be implemented. In the EXTEND version, a finite set of discrete $u = (v_l, v_r)$ input realizations has to be simulated for Δt time, choosing the one which brings q_{near} closest to q_{rand}. This version causes curved edges between tree vertices, which makes NEAREST_NEIGHBOR difficult if inner configurations of edges should be taken into account in the vertex selection step. This issue can be resolved by using a small Δt to obtain very short edges and considering only tree vertices for vertex selection. Of course, this will highly increase the number of vertices and distance measurement steps between q_{rand} and the tree.[1]

If one would like to use the CONNECT version, a steering method is needed. For differential drive, the simplest steering method is the following sequence of straight motion and turning-in-place primitives: (1) Turn to head to the next position, (2) move straight until it is reached and (3) turn to the desired orientation. This simple method has the advantage that it delivers an exact solution between two configurations (no sampling of the input set is needed) and additionally, the edges of the tree will be straight thus making the nearest neighbor search easy, even if inner configurations of edges are involved in the search. This latter property helps to keep the number of tree vertices as low as possible. Although these are attractive properties, this version of the algorithm has problems if narrow passages are present in the environment. An example is depicted in Fig. 2, where the trees are shown projected to the two-dimensional workspace after 1000 iterations.



[1] Note that the efficiency of nearest neighbor search can be increased in this case by using the KD-tree data structure [11].

Fig. 2. A naive application of RRT with a rotate–translate–rotate steering method usually fails to solve problems containing narrow passages

The tree starting from q_{init} preferred forward motion during the translation primitive, while the other preferred backward motion. It can be seen that the trees have not so many branches as would be expected after this number of iterations. This means that the CONNECT operation failed frequently to extend the tree. The problem is caused by the naive application of the rotate–translate–rotate steering method. As defined above, CONNECT applies the steering method until it reaches q_{rand} or a collision is detected. In the vicinity of a wall or a narrowing, the collision will occur most likely during the first rotation primitive, hence no translation, i.e. no effective extension of the tree will be achieved.

The advantageous properties of the RRT and the rotate–translate–rotate steering method, together with the above mentioned problem served as the main motivation for developing the proposed RTR-planner algorithm.

RT-TREES AND PRIMITIVES

The RTR-planner is similar to a bidirectional RRT algorithm. However, it has differences in the sampling, the vertex selection and the extension steps as well. It uses translation (T) and rotation (R) primitives for building the trees. In the extension steps, only RT (rotate–translate) sequences are used instead of the exact RTR steering method. For this reason, we denote the constructed trees as RT-trees. The vertices of the trees are configurations as usual and the edges are continua of configurations. According to T and R motion primitives, these are called Translational Configuration Intervals (TCIs) and Rotational Configuration Intervals (RCIs). The process of an RT-tree construction can be seen in Algorithm 2 and is detailed in the sequel.

Algorithm 2.
Construction of an RT-tree

```
1.  function RT_CONSTRUCT(q_init)
2.      T.init(q_init)
3.      for all k = 1 to K do
4.          p_G ← SAMPLE_POS()
5.          q_near ← NEAREST_NEIGHBOR
    (p_G, T)
6.          σ_min ← MIN_TURN_DIR(q_near, p_G)
7.          collision ← EXTEND(T, q_near, σ_min)
8.          if collision then
9.              EXTEND(T, q_near, −σ_min)
10.         end if
11.     end for
12.     return T
13. end function

14. function T.init(q)
15.     T.add_vertex(q)
16.     TCI ← TCI_EXTEND(q, 'forward')
17.     T.add_vertex(TCI.q_end)
18.     T.add_edge(q, TCI)
19.     TCI ← TCI_EXTEND(q, 'backward')
20.     T.add_vertex(TCI.q_end)
21.     T.add_edge(q, TCI)
22. end function

23. function EXTEND(T, q, turn_dir)
24.     [RCI, collision] ← RCI_EXTEND(q, turn_dir)
25.     T.add_vertex(RCI.q_end)
26.     T.add_edge(q, RCI)
27.     TCI ← TCI_EXTEND(RCI.q_end, 'forward')
28.     T.add_vertex(TCI.q_end)
29.     T.add_edge(RCI.q_end, TCI)
30.     TCI ← TCI_EXTEND(RCI.q_end, 'backward')
31.     T.add_vertex(TCI.q_end)
32.     T.add_edge(RCI.q_end, TCI)
33.     return collision
34. end function
```

SAMPLING, VERTEX SELECTION AND EXTENSION

The first difference to RRT can be found in the sampling step. SAMPLE_POS returns a position p_G – denoted as the guiding position in the sequel – instead of a configuration. It can be treated as a one-dimensional continuous set of configurations, from which any element can serve as local goal in the EXTEND step. The guiding position can be chosen randomly from the workspace, or the random sampling can be biased as seen in case of the RRT method. We use a bias towards positions which are expected to guide the growth of trees through narrowings. These positions are obtained by a triangular cell decomposition of the free workspace, where the free triangle edge midpoints are saved as possible guiding positions (see Fig. 3). The midpoints are mostly located far from obstacles but are present around narrow passages as well. A similar result could be obtained by determinig the Voronoi regions of the free space, but that would require more computational effort.

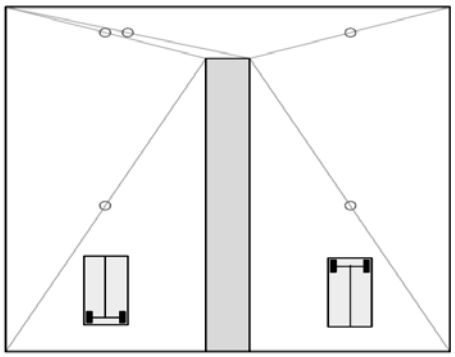


Fig. 3. The random sampling is biased towards fixed guiding positions obtained by a triangular cell decomposition

NEAREST_NEIGHBOR returns the configuration in the existing tree which has the smallest *position* distance to p_G. This step uses a simple Euclidean metric, hence no special configuration space metrics are needed.[2]

The main difference to the RRT method can be found in the tree extension step. On the one hand, translation is always performed in both forward and backward directions. Additionally, the translation is not stopped when p_G is reached, but continued until the first collision in both directions (this is the functionality of TCI_EXTEND, called by the T.init and the EXTEND functions as well). On the other hand, an important difference is the fact that an RT sequence will always be planned, even if a collision occurred in the rotation phase. The RTR-planner balances between reaching the guiding positions and extending the tree. If p_G cannot be reached because of a collision, then two things can be done:

If the collision occured during the T primitive, then the iteration is finished (because the tree has been extended).

If the collision occured during the R primitive, then TCI_EXTEND is performed in both forward and backward directions at the colliding orientation (first extension), and the rotation is tried again in the other turning direction as well. After the second rotation, independently from its success or collision, TCI_EXTEND is called again (second extension). An example of this procedure can be seen in Fig. 4.

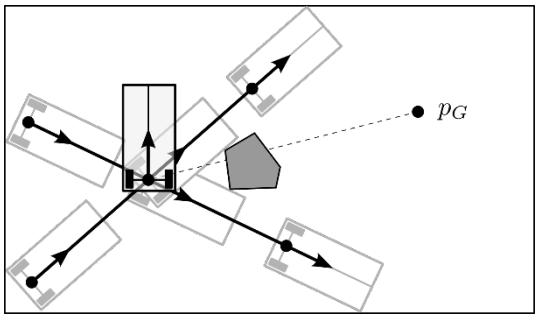


Fig. 4. Illustration of the tree extension procedure if the direction to p_G is blocked

OBTAINING THE RESULT

As already mentioned, the RTR-planner builds two RT-trees, one from the initial and one from the goal configuration. In order to obtain a final path, the two trees have to be connected. This is attempted in every iteration as follows. At the end of an iteration, the newly added TCIs are checked against every TCI in the other tree, starting from its root. If an intersection is found and if a collision-free RCI can be put between the intersecting TCIs to connect them, then the two trees can be merged and the path determined easily by tracing the trees back to their roots.

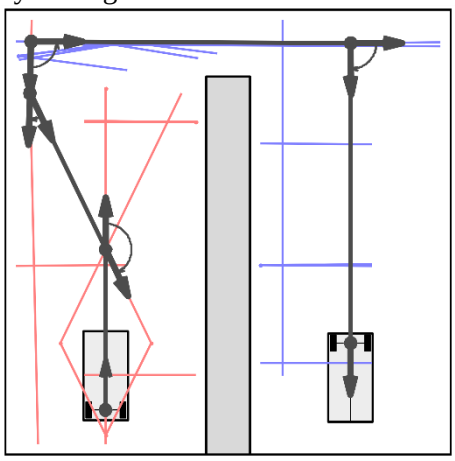


Fig. 5. Result of the RTR-planner (after 14 iterations)

Fig. 5 shows the result of the RTR-planner for the same scenario as seen in fig. 2. In this case the path was obtained after 14 iterations (remember that the naive application of the RRT method failed to return a path in 1000 iterations). The average iteration count of the RTR-planner was 18.82 for this scenario (averaged over 50 runs).

[2] Several approaches are available for obtaining metrics in configuration spaces, but these always have the problem of mixing different quantities (e.g. length and angle). For more information see [11].

SIMULATION RESULTS

Two example scenarios are depicted in Fig. 6. Scenario A contains relative large free areas and one narrow passage between two obstacles. Scenario B illustrates a more difficult situation: a narrow M-shaped corridor has to be passed, where the width of the corridor is comparable with the dimensions of the robot. The figure shows the paths resulting from the RTR planning process for each scenario, together with the RT-trees projected to the workspace. Both the naive RRT-based planner (using bidirectional search and the rotate–translate–rotate steering method) and the RTR-planner algorithm was tested on these scenarios. The algorithms have been run 50 times, the maximum allowed number of iterations was 1000. The success ratio (percentage of successful runs) and the average iteration count of the successful runs are collected in Table 1. It can be seen that the RTR-planner is more successful and significantly faster. The first scenario was mostly solvable for the naive RRT as well but with an approx. 4 times higher iteration count. This is not surprising because the RRT extends the tree with maximum one rotate-translate sequence in one iteration, while the RTR-planner can insert up to four of these (as seen in Fig. 4).

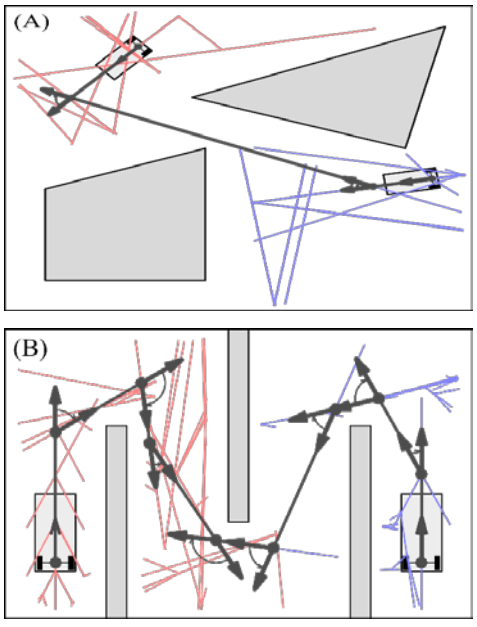


Fig 6. Example scenarios

Table 1. Comparing the efficiency of the naive RRT and the RTR-planner methods

Scenario A	**Naive RRT**	**RTR-planner**
Success ratio	96%	100%
Avg. iteration count	53.38	12.9
Scenario B	**Naive RRT**	**RTR-planner**
Success ratio	16%	100%
Avg. iteration count	683.5	67.2

The advantages of the RTR-planner are stronger highlighted in the second example. Here the naive RRT method mostly failed to return a path. In contrast with that, the RTR planner was 100% successful with about 10 times lower iteration count.

CONCLUSIONS AND FUTURE WORK

A new path planning method was proposed for differential drive mobile robots, based on the idea of Rapidly exploring Random Trees. The main results are:

Different sampling, vertex selection and tree extension steps

Flexible free space exploration using only rotation and translation primitives

Good results in scenarios containing narrow passages

Despite the convincing properties of the RTR planning method, it has weaknesses as well. It mostly fails in more complicated environments, like a multiple bug trap or a maze. These are challenging for any sampling-based method in general. Further work has to be done to improve the performance in such situations, e.g. by exploiting some system-specific properties of polygonal differential drive robots.

ACKNOWLEDGMENTS

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) organized by VIKING Zrt. Balatonfüred. This work was partially supported by the Hungarian Government, managed by the National Development Agency, and financed by the Research and Technology Innovation Fond through project eAutoTech (grant no.: KMR_12-1-2012-0188).

REFERENCES

1. **Latombe J.-C**. "Robot Motion Planning", Kluwer Academic Publishers, Boston, MA, 1991.
2. **Laumond J.-P., Sekhavat S., Lamiraux F.** "Guidelines in nonholonomic motion planning for mobile robots". In: Laumond J.-P. (ed) *Robot Motion Planning and Control*. Springer, 1998, (*Lecture Notes in Control and Information Sciences* No. 229).
3. **LaValle S. M.** "Rapidly-exploring random trees: A new tool for path planning", tech. rep., Computer Science Dept., Iowa State University, 1998.
4. **Dubins L. E.** "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". *Amer. J. of Math*. 1957; 79:497-516.
5. **Reeds J. A., Shepp L. A.** "Optimal paths for a car that goes both forwards and backwards". *Pacif. J. of Math.* 1990; 145:367-393.
6. **Giordano P. R., Vendittelli M., Laumond J.-P**., Soueres P. "Nonholonomic distance to polygonal obstacles for a car-like robot of polygonal shape". *IEEE Trans. Robot.* 2006; 22:1040-1047.
7. **Chitsaz H., O'Kane J. M., Balkcom D. J., Mason M. T.** "Minimum wheel-rotation paths for differential-drive mobile robots". In: *Proc. of the IEEE International Conference on Robotics and Automation*, 2006, pp. 1616-1623.
8. **Chitsaz H., LaValle S. M.** "Minimum wheel-rotation paths for differential drive mobile robots among piecewise smooth obstacles". In: *Proc. of the IEEE International Conference on Robotics and Automation*, 2007, pp. 2718-2723.
9. **Laumond J.-P., Jacobs P. E., Taix M., Murray R. M.** "A motion planner for nonholonomic mobile robots". *IEEE Trans. Robot. Autom.* 1994; 10:577-593.
10. **Sekhavat S., Chyba M.** "Nonholonomic deformation of a potential field for motion planning". In: *Proc. of the IEEE International Conference on Robotics and Automation*, 1999, pp. 817-822.
11. **LaValle S. M.** "Sampling-based motion planning". In: *Planning Algorithms*. Cambridge University Press, Cambridge, U. K., 2006, Available online at http://planning.cs.uiuc.edu/
12. **Kavraki L. E., Svetska P., Latombe J.-C.**, Overmars M. H. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". *IEEE Trans. Robot. Autom.* 1996; 12:566-580.
13. **Yershowa A., Jaillet L., Simeon T., LaValle S. M.** "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain". In: *Proc. of the IEEE International Conference on Robotics and Automation*, 2005, pp. 3856-3861.
14. **LaValle S. M., Kuffner J. J.** "Randomized kinodynamic planning". *Int. J. Robot. Res.* 2001; 20:378-400.

METADATA

Title: The RTR Path Planner for Differential Drive Robots

Authors: Domokos Kiss [1], Gábor Tevesz[2]

Affiliation:

[1, 2] Budapest University of Technology and Economics Budapest, Hungary

Email: domokos.kiss@aut.bme.hu, tevesz@aut.bme.hu

Language: English

Source: SIIT, vol. 2, no. 2 (4), pp. 16-22, 2020. ISSN 2686-7044 (Online), ISSN 2658-5014 (Print).

Abstract: Path planning among obstacles is a challenging task for nonholonomic mobile robots. One class of the most widespread approaches for solving this are sampling-based roadmap methods. These take samples from the configuration space and use a local planner to connect them and to build a roadmap. The resulting path is obtained by a graph search algorithm in this roadmap. This paper presents the RTR-Planner, a sampling-based global planning algorithm for differential drive mobile robots, based on the idea of Rapidly exploring Random Trees (RRT). The RTR-planner builds two search trees consisting of rotation (R) and translation (T) primitives, starting from the initial and the goal configuration. Samples are taken randomly or biased towards passages in the free workspace. If the two trees reach each other, the resulting path can be obtained easily. Simulation results are presented which show the effectiveness of the method even in the presence of narrow corridors and passages.

Keywords: robots; locomotion systems; algorithm; construction process, method

About authors:

Domokos Kiss, Assistant lecturer. Department of Automation and Applied Informatics.
Budapest University of Technology and Economics
Budapest,Hungary

Gábor Tevesz, PhD. Department of Automation and Applied Informatics.
Budapest University of Technology and Economics
Budapest, Hungary