RTR+C*CS: An Effective Geometric Planner for Car-like Robots

Gábor Csorvási, Ákos Nagy and Domokos Kiss
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
gabor.csorvasi@aut.bme.hu, akos.nagy@aut.bme.hu, domokos.kiss@aut.bme.hu

***Abstract*—The need for intelligent autonomous vehicles is increasing in industrial and everyday life as well. Path planning among obstacles is one of the challenging problems to be solved to achieve autonomous navigation. In this paper we present a global geometric path planning method for car-like robots, which proved to be effective especially in cluttered environments, containing narrow passages. Navigation in such scenarios usually requires non-obvious manoeuvring with many reversals, which is challenging even for a human driver. We also present a comparative analysis of our method with possible alternatives from the literature to illustrate its effectiveness regarding computation time and path quality.**

I. Introduction

Driver assist systems in cars are becoming more and more widespread and affordable nowadays. These include functions that help the human driver (e.g. lane change assistance, automatic parking, collision avoidance), but researchers strive to reach a higher integrated autonomy level of vehicles. One important solvable problem of autonomous navigation is planning a geometric path from an initial state (or pose) to a given goal state. From the perspective of path planning, the state of a robot can be described by its *configuration*. For a vehicle which moves in a planar workspace $\mathcal{W} \subset \mathbb{R}^2$, the configuration can be given by $q = (x, y, \theta)$, a vector of its position and orientation in the configuration space $\mathcal{C}$. The set of not allowed configurations (e.g. because of collision with obstacles) is called the configuration space obstacle $\mathcal{C}_{obs}$, its complement is the free space $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$.

The locomotion system of a car is based on rolling wheels. The rolling without slipping constraint of these wheels induce *nonholonomic constraints* which cause difficulties in the control of such robots, even in the absence of obstacles. These constraints cause that a car can only move in the direction of its heading (forwards or backwards) and that the turning radius is lower bounded, which should be taken into account in the planning algorithm.

The path planning problem for cars is solved and already available in commercial products for the case of simple parking situations or for large scale environments with broad free areas. However, one can find much less useful solutions for scenarios that require more complicated manoeuvrings in narrow environments. This paper describes a path planning algorithm that is invented especially for such environments. These include scenarios where narrow corridors and tight corners have to be passed or there are a lot of obstacles which cannot be avoided without a number of reversals. Real-life examples of such situations can be found in cluttered parking lots or historical quarters of small towns with narrow streets and tight parking places.

The structure of the paper is the following. Section II summarizes the available path planning approaches in the literature for solving nonholonomic planning problems. The proposed RTR+C*CS planning method is described in Section III, which is a summary of our previous papers [1], [2] and [3]. Section IV shows simulation results and a detailed comparison of our algorithm to other available methods in benchmark situations. Conclusions are drawn and directions of future work is marked out in Section V.

II. Related Work

Generating feasible paths for nonholonomic robots is not trivial even in the absence of obstacles. For car-like robots which can move only forward (Dubins-car [4]) or both forward and backward (Reeds–Shepp-car [5], [6]) exact methods exist for computing optimal (minimum length) paths between two configurations q_I and q_G. These are called local planners or steering methods. Reeds and Shepp showed in [5] that the shortest path between any two configurations of a car can be chosen from 48 possibilities. These consist of maximum five circular or straight segments and at most two cusps, where the circular arcs have minimal radius.

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. 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 involves sampling-based roadmap methods, which build one or more topological graphs in order to capture the topology of $\mathcal{C}_{free}$ and use local steering methods to connect the graph nodes. The second 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 [7], [8].

A. Sampling-Based Roadmap Methods

A good survey of sampling-based planning methods can be found in [9]. 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 $\mathcal{C}_{obs}$, only a black-box collision detector module which can

978-1-4799-7370-5/15/$31.00 ©2015 IEEE

Algorithm 1 The basic RRT construction algorithm [12]

```
1: T.init(q_init)
2: for all k = 1 to K do
3:     q_rand ← RANDOM_CONFIG()
4:     q_near ← NEAREST_NEIGHBOR(q_rand, T)
5:     if CONNECT(T, q_rand, q_near, q_new) then
6:         T.add_vertex(q_new)
7:         T.add_edge(q_near, q_new)
8:     end if
9: end for
10: return T
```

tell whether a given configuration is in $\mathcal{C}_{obs}$ or not. These methods proved to be successful in many planning problems, including high-dimensional configuration spaces. For example, the Probabilistic Roadmap Method (PRM) [10] 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 [11]. 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.

B. Rapidly Exploring Random Trees

The basic RRT construction process can be seen in Algorithm 1. The building of the tree $\mathcal{T}$ starts from the initial configuration q_{init}. RANDOM_CONFIG returns a random configuration q_{rand} from $\mathcal{C}$ (sampling step). 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 (this version is called RRT-Connect in the literature). 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 [13]. In the latter case the first tree is extended towards q_{rand} and the second tree is tried to connect to the first one. After every iteration, the two trees change their role.

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 [11]). On the other hand, the CONNECT version can be applied for systems where an explicit steering method is available to connect two configurations.

C. Nonholonomic planning by Approximation

Approximation methods proceed from a preliminary, not necessarily admissible collision-free path connecting the initial and final configurations, and perform a search for a sequence of feasible local paths connecting the same configurations. In the framework proposed in [7] the initial path is recursively subdivided until every part can be replaced by a collision-free admissible path obtained by an appropriate steering method. This work deals with car-like robots and applies the Reeds–Shepp optimal paths in the local planning phase. The quality of the resulting approximated feasible paths obtained by this algorithms is affected by the "goodness" of the initial geometric path. If the initial path leads mostly in directions which are non-admissible for the system, then the approximated path will be quite difficult (containing a great amount of reversals). An approach for increasing the "goodness" of the initial path can be found in [8]. It is shown in [14] that if the preliminary global geometric path has nonzero clearance and the applied steering method verifies the so called *topological property*, then the approximation method is complete.

III. The RTR+C*CS Path Planner

In our approach we adopt the ideas of both sampling-based geometric planning and approximation by a topological steering method. A primary global path is designed by the sampling-based RTR planner [2], which consists only of straight motion and turning in place primitives. The result is not feasible for the car, but is quite good for approximation, because every path segment which requires translation is feasible on its own. The approximation has the role to eliminate in-place turns by inserting circular path segments while avoiding collisions. The approximation is performed using the local C^*CS planner [1] which uses straight segments and circular arcs of lower bounded (but not necessarily minimal) radii.

A. Global Planning: RTR

The RTR (rotate–translate–rotate) 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 rotation (R) and translation (T) primitives for building the trees. The first difference to RRT can be found in the sampling step. It returns a guiding position p_G instead of a configuration, which can be treated as a one-dimensional continuous set of configurations, from which any element can serve as local goal in the tree extension step.

The vertex selection step 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.

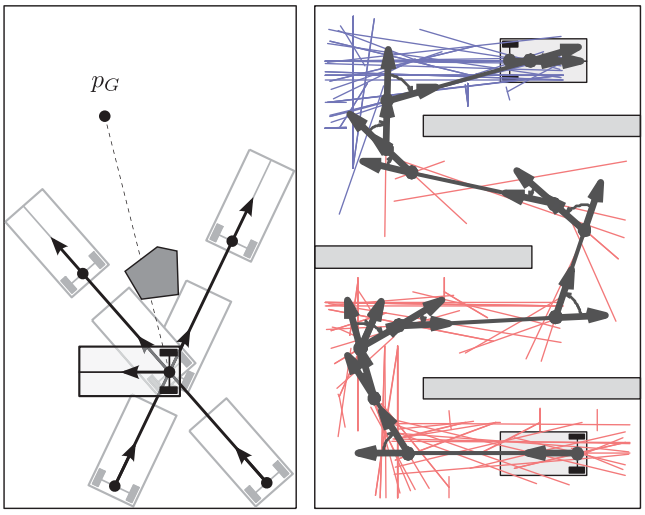


Fig. 1. (a) Illustration of the tree extension procedure if the direction to p_G is blocked, (b) A result of the RTR planner through a narrow corridor

The main difference to the RRT method can be found in the tree extension step. It performs some primitive rotate–translate motion pairs, guided by p_G as follows. A rotation is applied to reach the orientation pointing to p_G, and a consecutive translation is performed in both forward and backward directions. The translation is not stopped when p_G is reached, but continued until the first collision in both directions. On the other hand, if a collision occurs during the rotation, the two-directional translation is done at the colliding orientation, and the rotation is tried again in the other turning direction. This results an efficient tree extension, even if p_G is not reachable at the current step (see Fig. 1a). The operation of the RTR planner is described in more detail in our recent paper [2].

The good extension properties and the effectiveness of the RTR algorithm even in narrow environments is illustrated in Fig. 1b. The two trees and the resulting path are drawn by different colors.

B. Local Planning: C*CS

The C^*CS planner obtains local paths for a car-like robot between two configurations. These consist of circular arcs (C) and straight segments (S). The C^* notation stands for "a circular arc that can have infinite radius", thus C^*CS is a shorthand for both CCS and SCS paths. The algorithm is detailed in our previous paper [1], we only summarize here its most important properties.

It is shown in [1] that for a given pair of initial and goal configurations (q_I, q_G) – denoted also as a query pair – there are infinitely many C^*CS solutions in the form

$$q_I \xrightarrow[\rho_{I,\tilde{I}}]{C} \tilde{q}_I \xrightarrow[\rho_{\tilde{I},\tilde{G}}]{C} \tilde{q}_G \xrightarrow[s_{\tilde{G},G}]{S} q_G, \qquad (1)$$

where $\tilde{q}_I$ and $\tilde{q}_G$ are intermediate configurations, $\rho_{I,\tilde{I}}$ and $\rho_{\tilde{I},\tilde{G}}$ are the radii of the first two circular segments and $s_{\tilde{G},G}$ is the length of the final straight segment. The position of the first intermediate configuration $\tilde{q}_I$ can be chosen arbitrarily, this is why we have an infinite number of solutions (see Fig. 2).

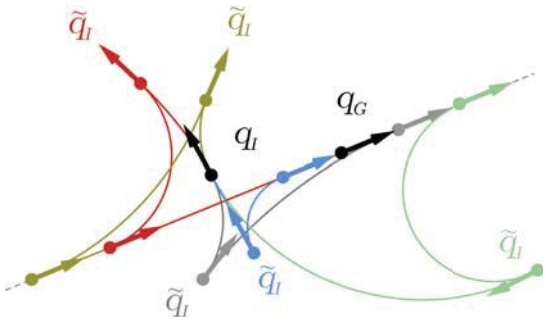


Fig. 2. There are many C^*CS paths between two configurations

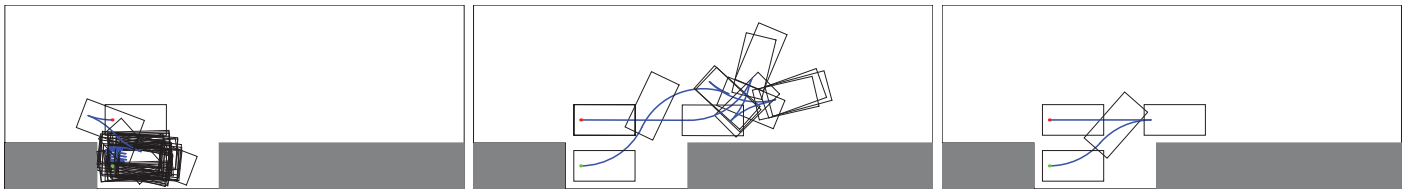
This property is useful in the presence of obstacles, because we can choose from a class of paths between q_I and q_G. In our implementation the algorithm takes a finite number of samples from the workspace at discrete distances from the car on circular paths of discrete curvatures, and computes a C^*CS path candidate for each. The colliding candidates are neglected, and the shortest one is chosen form the remaining set.

This local planning procedure is applied firstly to the starting and the goal configurations of the primary RTR-path. If no solution has been found, then the path is iteratively subdivided and further C^*CS paths are planned to replace the parts. However, because we have a finite number of C^*CS candidates, there is no guarantee that a solution will be found in this form. To overcome this fact, the $c\bar{c}S$ steering method is introduced in [1]. This steering method returns one exact solution from the class of C^*CS paths, where $\rho_{I,\tilde{I}}$ and $\rho_{\tilde{I},\tilde{G}}$ are minimal and have the same absolute value but opposite sign. It is proved in [1] that $c\bar{c}S$ verifies the topological property. This means that if q_I and q_G get closer to each other (by subdividing the primary path), then the C^*CS path between them gets closer to the collision-free geometric path. In other words, the C^*CS planner together with its $c\bar{c}S$ extension guarantees that a collision-free RTR-path can be approximated by a sequence of C^*CS paths, obeying a minimal turning radius constraint.

IV. Test Results and Comparison with Other Methods

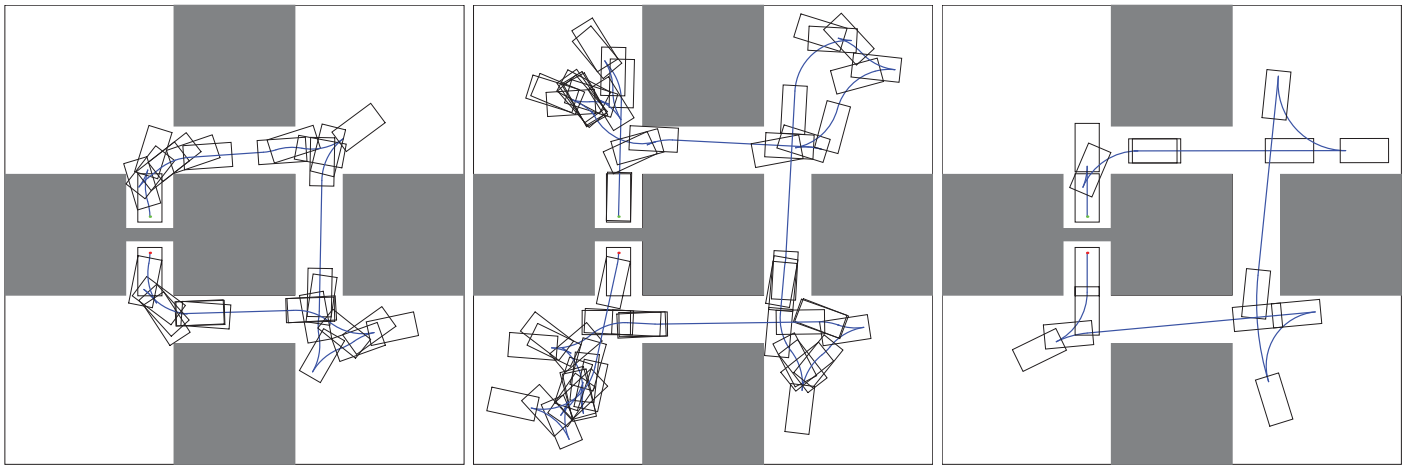
The RTR+C*CS planning algorithm was tested extensively in simulations. Together with it, we implemented some other variants of sampling based and approximation planning methods and made comparative measurements in different scenarios. The compared methods were the following:

- *RRT-Connect (RS)*: Bidirectional RRT using the Reeds–Shepp optimal steering method as local planner
- *RRT-Connect ($c\bar{c}S$)*: Bidirectional RRT using our $c\bar{c}S$ steering method as local planner
- *RRT-Connect (interp) + RS*: Bidirectional RRT with simple interpolation as local planner for generating a preliminary global path, which is approximated by Reeds–Shepp paths
- *RRT-Connect (interp) + C^*CS*: Bidirectional RRT with simple interpolation as local planner for generat-



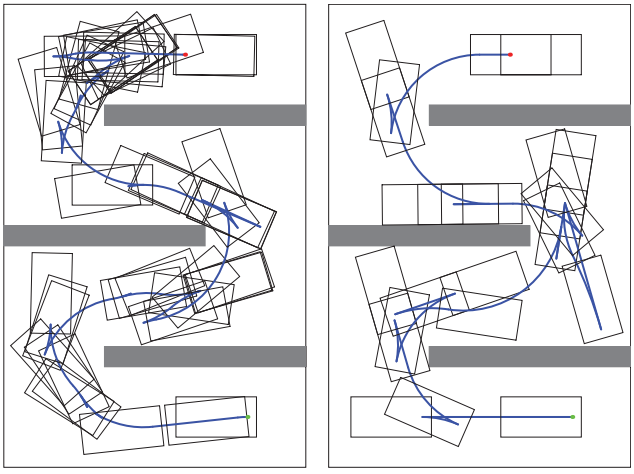
(a) (b) (c)

Fig. 3. Typical resulting paths for parallel parking. (a) RRT-Connect (interp) + RS, (b) RRT-Connect (RS) and (c) RTR+C*CS.



(a) (b) (c)

Fig. 4. Typical resulting paths for square-shaped corridors. (a) RRT-Connect (interp) + RS, (b) RRT-Connect (RS) and (c) RTR+C*CS.



(a) (b)

Fig. 5. Typical resulting paths for the M-shaped corridors. (a) RRT-Connect (RS) and (b) RTR+C*CS.

ing a preliminary global path, which is approximated by C^*CS paths

- *RTR+RS*: RTR planner for generating a preliminary global path, which is approximated by Reeds–Shepp paths
- *RTR+C*CS*: RTR planner for generating a preliminary global path, which is approximated by C^*CS paths

We present results of solving planning queries in three different scenarios:

- A simple parallel parking problem (leaving a single parallel parking place), see Fig. 3
- Four perpendicular narrow corridors with bigger free areas at the corners (Fig. 4)
- An M-shaped narrow corridor with tight corners (Fig. 5)

The algorithms were implemented in C++ and tested on a notebook computer with Intel i7-3630QM processor running at $2.4GHz$ and having 8 GB RAM. The algorithms are compared by success rate, computational time and by the quality of resulting paths. To obtain more or less practical results, we constrained the maximum number of iterations of the tree building process to 4000 in every algorithm, and applied a lower bound of path subdivision at the approximation phase (if the size of the subdivided path piece goes below $0.005mm$, then the approximation is stopped). Every algorithm has been run 50 times in every scenario, and the average results are shown in Fig. 6. The performance is measured by the following metrics:

- *Success ratio*: Percentage of successful runs
- *Computation time*: The average running time of all (successful and unsuccessful) attempts
- *Effective computation time*: The computation time divided by the success ratio. This gives an estimate of the average time the algorithm needs to find a result.

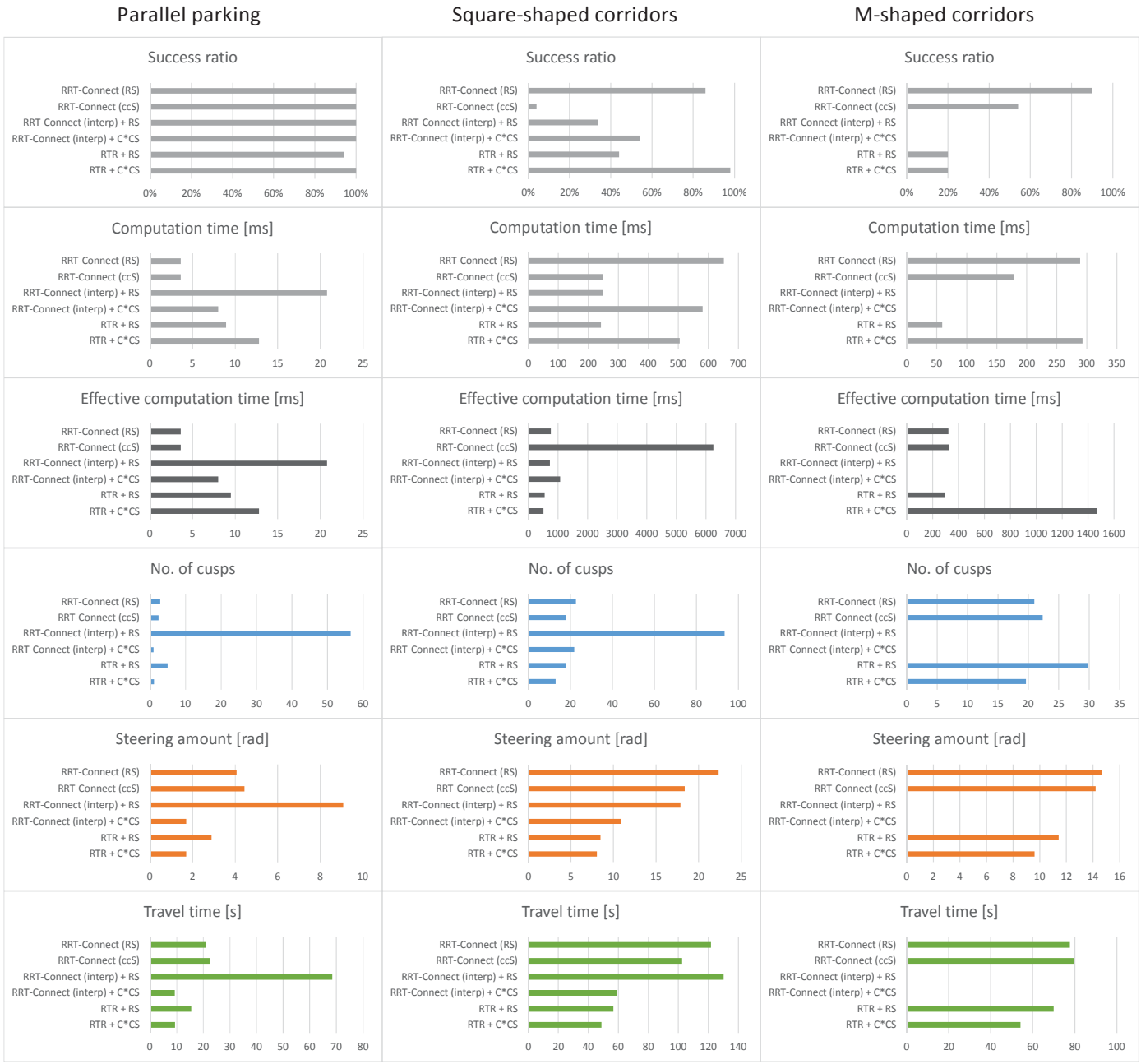


Fig. 6. Summary of the simulation results

We tried to characterize what a "good" path means. The following metrics give an estimate of how "comfortable" a passenger would feel while sitting in a car which travels along the path. Thus, the path quality is measured by:

- *No. of cusps*: The number of reversals along the path. A good path contains as few reversals as possible.
- *Steering amount*: The absolute integral of the path curvature. It gives the amount of turning along the path. A path with less turns and more straight segments is more comfortable than a path which requires turning from the beginning to the end
- *Travel time*: The time of driving along the resulting path is estimated as follows. A traveling speed function with $v_{max} = 5m/s$ and $v_{min} = 1m/s$ is assigned to the path, which is inversely proportional to the actual path curvature. The cusps are penalized with $0.5s$ "reversing time". Those paths are better, which require less time to travel along. Note that the car in the examples is $4m$ long and $2m$ wide.

The example scenarios represent increasing difficulty for the planning algorithms. The relative size of bigger free areas is decreasing from Fig. 3 to 5 while the amount of narrow corridors is increasing. A good solution has to avoid unnecessary maneuvering by using the available wider areas to obtain a "natural" path. However, unneeded detours towards

the free areas have to be avoided as well.

The first test case is exiting a parallel parking place. This is a relatively easy task even for the C^*CS local planner, because it can be solved by C–C–S segments in a single approximation step. Therefore the two C^*CS based methods design the same path for almost every attempt.

As can be seen in the graphs, the methods which use simple interpolation in the global planning phase obtain the worst end results. This is because the global preliminary path assumes free movement in every direction. Thus, the approximated path tends to contain a number of unnatural reversals even is simple situations (see Fig. 3a). Furthermore, according to the leftmost column of Fig. 6, these methods failed to solve the most difficult planning query.

The following observation is that the RTR global planner contributes greatly to the quality of paths. Since it uses straight movements for translation, the preliminary path is easy to approximate by a car-based local planner. The C^*CS method is very well suited for such preliminary paths, this is why the RTR+C*CS planner obtains the best quality paths in all situations. Looking at the measured values of the results in the most difficult M-shaped environment, it can be seen that the most obvious choice from the literature for solving such problems, the direct application of the RRT method with the Reeds–Shepp local planner is almost competing with the RTR+C*CS method regarding path quality. Furthermore, its effective computation time is much better than that of RTR+C*CS. The reason is that in such tight situations the result will most likely contain minimum radius turns, which can be obtained by the Reeds–Shepp paths directly. At the same time, the C^*CS local planner makes unnecessary iterations while trying to find less sharp turns. This drawback turns to a benefit in other situations, as can be seen in Fig. 4b and Fig. 4c, where the direct RRT method builds trees and finds solutions which make unnecessary roundabouts in the wider areas.

V. Conclusions and Future Work

We have presented a global geometric path planning method for car-like robots and a comparative analysis with possible alternatives. The (preliminary) RTR path planner is capable of designing paths consisting of straight movements and turning in place. When applying the C^*CS local planner to the RTR path in a second approximation phase, a path obeying the minimal turning radius constraint of cars is obtained. The simulation results showed that our planning algorithm can be applied universally in less and more difficult situations, and the obtained paths are quite "natural". Furthermore, its running time ranges from milliseconds to a few seconds on an everyday computer, which makes it feasible for application in real-life situations.

Our future work includes a comprehensive testing on real robot platforms which should verify that these algorithms are well-suited for problems requiring autonomous maneuvering. A further improvement of the path planner algorithm is in progress, which has the goal of generating paths with continuous curvature.

Acknowledgement

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] D. Kiss and G. Tevesz, "A steering method for the kinematic car using C*CS paths," in *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, Velké Karlovice, Czech Republic, May 2014, pp. 227–232.

[2] ——, "The RTR path planner for differential drive robots," in *Proceedings of the 16th International Workshop on Computer Science and Information Technologies CSIT2014*, Sheffield, England, September 2014.

[3] A. Nagy, G. Csorvási, and D. Kiss, "Path planning and control of differential and car-like robots in narrow environments," in *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Herlany, Slovakia, January 2015, pp. 103–108.

[4] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[5] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, pp. 367–393, 1990.

[6] P. R. Giordano, M. Vendittelli, J.-P. Laumond, and P. Souères, "Nonholonomic distance to polygonal obstacles for a car-like robot of polygonal shape," *IEEE Trans. Robot.*, vol. 22, pp. 1040–1047, 2006.

[7] J.-P. Laumond, P. E. Jacobs, M. Taïx, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. Robot. Autom.*, vol. 10, pp. 577–593, 1994.

[8] S. Sekhavat and M. Chyba, "Nonholonomic deformation of a potential field for motion planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, MI, 1999, pp. 817–822.

[9] S. M. LaValle, "Sampling-based motion planning," in *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available online at http://planning.cs.uiuc.edu/.

[10] L. E. Kavraki, P. Svetska, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, pp. 566–580, 1996.

[11] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep., 1998.

[12] A. Yershowa, L. Jaillet, T. Simeon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005, pp. 3856–3861.

[13] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, pp. 378–400, 2001.

[14] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, "Guidelines in nonholonomic motion planning for mobile robots," in *Robot Motion Planning and Control*, ser. Lecture Notes in Control and Information Sciences, J.-P. Laumond, Ed. Springer, 1998, vol. 229.