A Planning Method to Obtain Good Quality Paths for Autonomous Cars

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

***Abstract*—Path planning among obstacles for nonholonomic systems is a widely researched area nowadays, but it is still one of the most challenging problems in autonomous navigation. We have recently presented a rapidly exploring random tree based global planner (RTR) and a steering method (C*CS) for car-like vehicles, which uses circular and straight movements. With the aid of these two methods it is possible to obtain a feasible path, even in narrow spaces and in situations requiring non-trivial maneuvers. This paper presents an improved solution for environments, where not all obstacles are known at the beginning and these are discovered during the motion of the robot. We also introduce an extension to the presented algorithm to achieve paths of better quality, i.e. more similar to a reasonable path generated by a human driver.**

***Keywords*—keyword**

I. Introduction

Driving support systems (e.g. parking assistant, collision avoidance) are increasingly widespread nowadays. In some application these intelligent systems plan the path for the vehicle from a start position to the desired goal position. Due to the presence of obstacles this is not an easy task at all. Kinematic constraints of the vehicle also make path planning difficult. Cars have 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. For this reason parallel parking with a car is relatively difficult.

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 $\mathcal{W} \subset \mathbb{R}^2$ 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 $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$.

In this paper we extend our previously create path planner algorithms (RTR and C*CS). In many cases the robot do not have complete knowledge of the environment, but a global path planner method requires global information. So we examine our planning methods and make some changes to get better results in dynamic environment. Another important aspect is the quality of the planned path. A good path is similar to a path generated by human. Our algorithms are extended to get more human-like paths.

The paper is organized as follows. A brief overview of path planning methods for autonomous robots are given in Section II. In Section III, we shortly discuss our previously developed RTR and C^*CS planning methods and show how these can be applied together to obtain feasible paths in narrow environments. Section IV shows our enhanced method for sampling the configuration space in the C^*CS algorithm. We show a planning strategy for environment, where farther obstacles are not known at the beginning in Section V. In Section VI, we present an extension to our path planning methods in order to generate human-like paths. Conclusions and directions of future work are summarized in Section VII.

II. Related Work

Path planning for nonholonomic robots is not trivial even in absence of obstacles in the environment [1]. Algorithms that can solve this task 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 our investigated car-like robot model [2], [3].

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 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 [4]. In this case firstly a primary geometric path is planned which takes the obstacles and the shape of the robot into account, but does not care about the kinematic constraints. In a second phase, this path is iteratively subdivided and the parts are tried to be connected by a steering method. If the geometric path has a nonzero clearance ε from the obstacles, and the steering method verifies the so called *topological property*, then the approximation succeeds in finite time [5].

The second category involves sampling-based roadmap methods, which build a graph in order to capture the topology of $\mathcal{C}_{free}$ and use local steering methods to connect the graph nodes. A good survey of sampling-based planning methods can be found in [6]. 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 $\mathcal{C}_{obs}$. These methods proved to be successful in many planning problems, including high-dimensional configuration spaces. For example, the Probabilistic Roadmap Method (PRM) [7] and the Rapidly exploring Random Trees (RRT) [8], [9] algorithms are common sampling based planners. The main idea of RRT method 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.

III. The RTR+C*CS Path Planner

Our global path planner (RTR) and steering method (C*CS) have been presented recently [10], [11]. The global path designed by RTR consists of translational motion (T) and turning in place (R). Therefore the resulted path is not feasible for car-like robots, but a good starting point for the steering method, because the path is collision free. Thus this path can be approximated by the C*CS local planner, which uses straight segments and circular arcs of lower bounded (but not necessary minimal) radii. The purpose of the approximation is to eliminate in-place turns while avoiding collisions, since these are not feasible for car-like robots.

A. RTR Global Planner

The RTR planner is based on the well-known Rapidly exploring Random Trees (RRT) algorithm [12]. RTR is similar to a bidirectional RRT method, because it also builds two trees from the start and goal configuration (*start tree*, *goal tree*). However, the phases (sampling, vertex selection and extension) of the RTR planner are different from RRT.

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.

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. A rotation is applied to reach the orientation pointing to p_G, and a consecutive translation is performed in both forward and backward directions until the first collision. If a collision occurs during the rotation, the rotation is tried again in the other turning direction. An example for the resulting RTR-path can be seen in Fig. 1, the two trees are depicted as well by different colors.

B. C*CS Steering Method

The main idea behind the $C{*}CS$ local planner is to use circular and straight movements as path primitives [13], [14]. With the help of these primitives we can arrive from a q_I initial configuration to any q_G goal. To simplify the notation we will use the letter C for the circular arcs and S for the straight segments in the sequel. It can be easily shown that without taking the curvature bound of a car-like robot and assuming that $\theta_I \neq 0$, the goal configuration can be reached with a CS pair as follows. With a tangential arc we turn to an intermediate

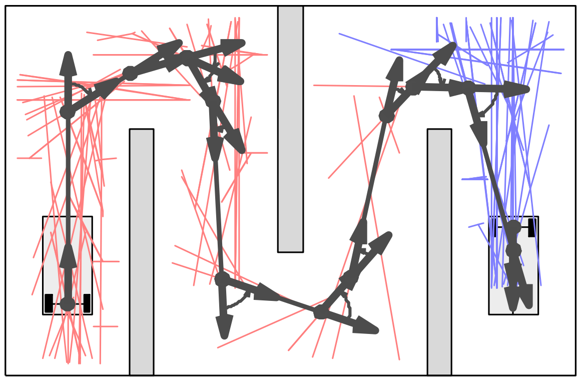
Fig. 1. A result of the RTR planner through a narrow corridor

goal $\tilde{q}_G$ which is in the line of the q_G (C). After that only a final straight segment is needed to arrive in the goal (S).

Unfortunately if the radius of the tangential arc is smaller than the minimal turning radius of the car or it is initially parallel to the line of the goal no feasible path can be found. This can be eliminated by reaching an intermediate $\tilde{q}_I$ configuration using a preceding S or C segment, from which the following tangential C segment does not violate the turning radius constraint. It is shown that we have infinitely many possibilities to choose such an intermediate configuration.

The set of configurations, which is reachable from the initial configuration, is called Arc Reachable Manifold (ARM [15]) Since this set contains both C and S segments we denote this with $C*$, where it stands for "a circular arc that can have zero curvature" thus $C{*}CS$ is a shorthand for both CCS and SCS paths.

In our previous work we examined this planning method with thorough tests even compared with other well known methods, and our results show that in most cases the resulting paths are more efficient and have better qualities than the others [11]. In the following we present other methods to reach better results.

IV. Improvements in Sampling Phase

Among obstacles the property of infinite solutions is useful, because there are more possibilities to find a feasible path, and there is a freedom to choose one of the "prettiest" solutions as well. Although in the implementation we have to use a form of sampling method. Naturally if the sample size is bigger, the calculation takes more time, but if only a few samples are taken the possible solution can be missed.

In our early implementation we used a Sukharev-grid [6] based sampling method, but this had more drawbacks. The necessary step size was depending on the actual environment, and a lot of calculation was unnecessary for example in far spaces or at the side of the car which is unreachable for a non-holonomic car-like robot.

Our new method is based on the behaviour of these robots and only the closed environment is sampled. To avoid the unnecessary calculation caused by the unreachable area at the two sides of the robot, we sampled the curvature of the possible circular arcs from 0 to κ_{max}, where $\kappa_{max} = \frac{1}{r_{min}}$. Also the

TABLE I. Run Times for Different Sampling Methods

Scene	Sukharev-grid		New method	
	Sample size	Run time	Sample size	Run time
Corridors	1225	7.067s	220	4.541s
Triangles	576	1.886s	40	0.102s
Parking lot	1600	4.089s	84	0.831s

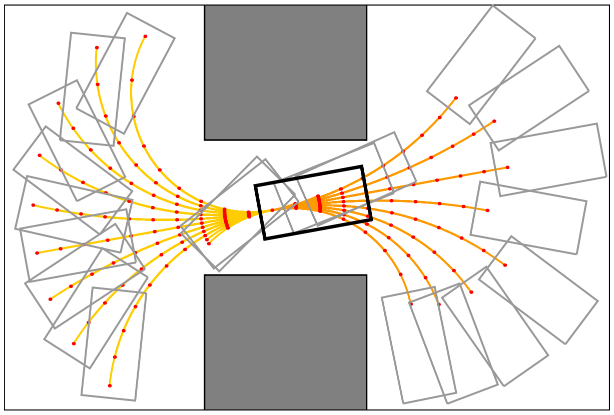
Fig. 2. Arc Reachable Manifold in an environment with two obstacles

distance from the robot is sampled from 0 to a chosen d_{max}. Our tests show that the $d_{max} = 2 \cdot r_{min}$ could be a suitable choice. The first step results tangential circles on both side of the robot and the second step results co-centric circles. The intersections of these circles will be the sample points. In this way the sample size can be reduced so that the possible solutions are still present.

In order to compare the two methods we ran the algorithm in different environments, and each time the sample size is set so that the algorithm barely finds a solution therefore the running times can be compared. Three environments are used, one with four narrow corridors where a lot of small segments are used, so the sampling phase called often (Fig. 8). The second scene is a small environment with few triangles as obstacles (Fig. 6), and the last one is similar to a parking lot with a lot of walls (Fig. 3).

In TABLE I we compare the running times of the two sampling method and as we expected the new algorithm is faster in every case, because it ignores many unreachable points. The algorithms were implemented in C++ and tested on a desktop computer with Intel E8400 processor running at $3.0GHz$ and having 4 GB RAM.

V. Planning in Dynamic Environment

Our presented path planning method requires global information about the environment. During collision detection in the RTR and the C*CS algorithm, all obstacles of the environment are checked. In most real situations we do not have global information about the environment or moving obstacles. In both cases the planner algorithm has to be run repeatedly if the environment is changed and the previously generated path would be collided.

Consider the following scenario. A 3D laser scanner is located on the top of a car-like robot. This sensor provides information about the obstacles near the car for the planning algorithm. In our simulation we model this situation the following way. The robot only detects the obstacles near the current configuration within a fixed radius. In every simulation iteration the program checks whether an obstacle within the sensing radius is discovered. If the new obstacle collide with the path of the robot, the path will be re-planned (see Fig. 3). After that, the robot steps forward to the next path configuration. It is important to note that for the sake of simplicity we assume that obstacles are discovered as whole geometric shapes. In contrast with that, a real sensor provides only partial information about an object at a time instant. In real robots, mapping algorithms have to be used to reconstruct the geometric shapes of obstacles, but these are out of scope of the current paper.

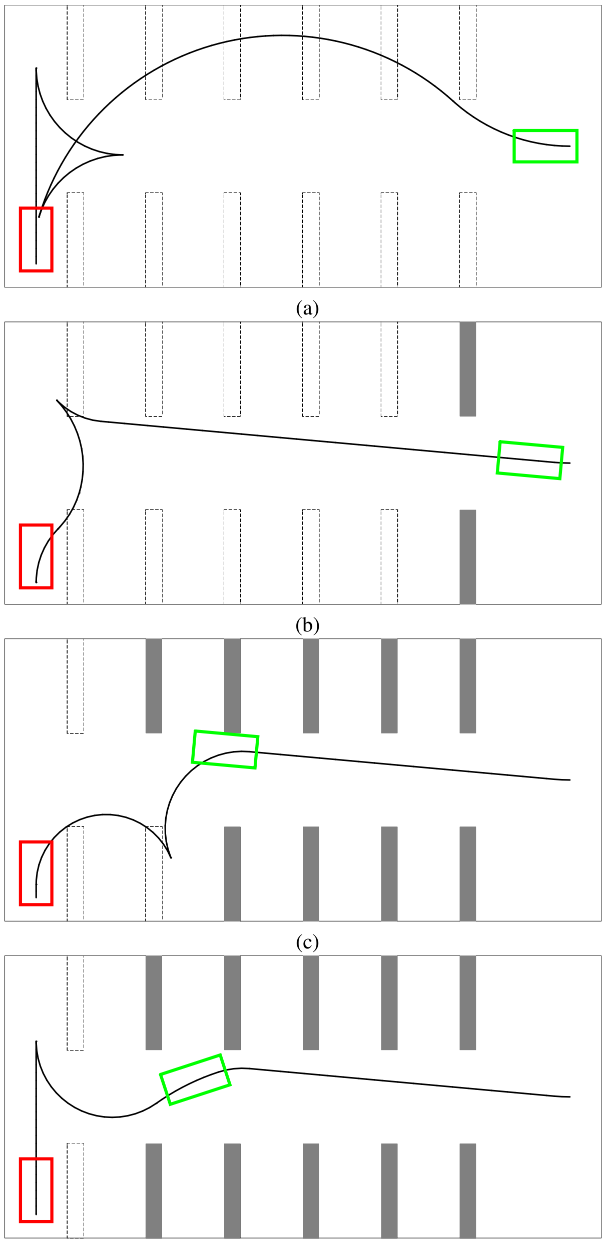

Fig. 3. Four iteration of the simulation process for dynamic environment. Dashed bordered obstacles are undiscovered.

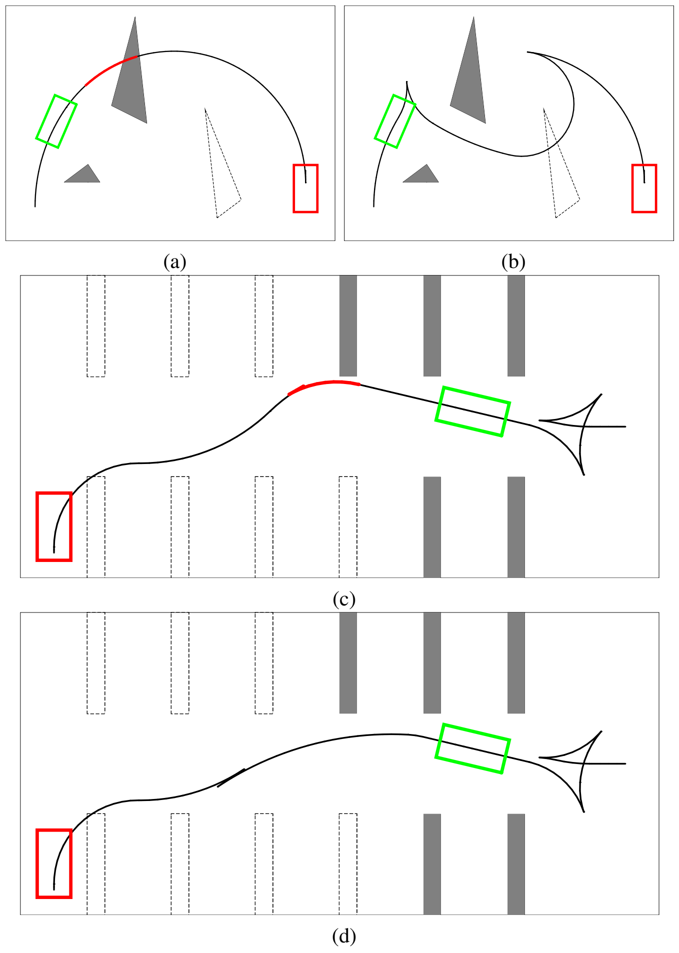
Fig. 4. Scenes for *Local Re-Planning* strategy. (a,c) The previous, collided path after discovering new obstacle, (b,d) The collision-free path after re-planning.

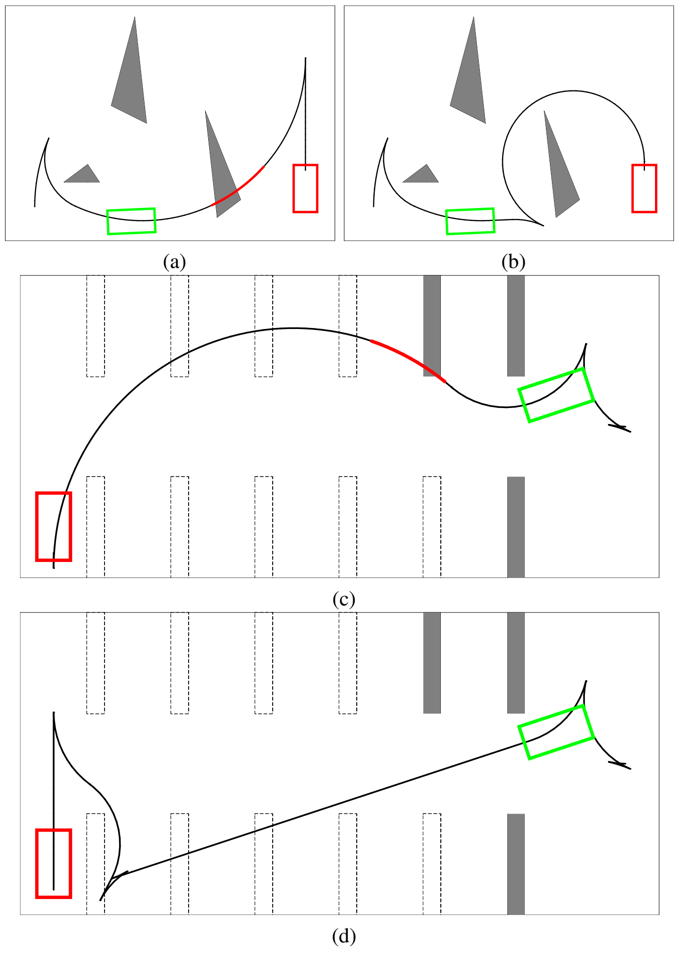
Fig. 5. Scenes for *Global Re-Planning* strategy. (a,c) The previous, collided path after discovering new obstacle, (b,d) The collision-free path after re-planning.

The most important factors for this simulation scenario are planning time and the quality of the path. The running time of the RTR and C*CS algorithms definitely determine the planning time. In this Section we will focus on planning time and path quality.

We compare three different planning strategies for dynamic environments. The strategies differ in the re-planning method when the path is collided:

- *Global Re-Planning*: The path is re-planned from the robot current configuration to the goal configuration.
- *Local Re-Planning*: The path is re-planned from the robot current configuration to the end of the collided path segment.
- *Global Re-Planning with Tree Reservation*: Same as the first strategy, but the goal tree in the RTR algorithm remains.

Fig. 5 shows *Global Re-Planning* strategy for two situations. In these figures, we marked the configuration of the robot when it detects a new obstacle, the collided path segment and the goal configuration as well. The undiscovered obstacles are drawn with dashed border. In the first scenario (see Fig. 5a, 5b), the path takes from the lower, left corner to the lower, right corner. We can see that planning to the goal configuration makes a natural path for car-like robots in this situation. However, in the next scenario (Fig. 5c, 5d) this planning method generates an unnecessarily difficult path after detecting the collision. One can guess that local re-planning, which replaces only the colliding path segment would result a more simple path in this situation.

The *Local Re-Planning* strategy is shown in Fig. 4. In the first situation (Fig. 4a, 4b), this method generates a quite complicated path, because the planner tries to connect the new path segment to existing one. In this strategy, there is an extra parameter, which determines the the length of the regenerated path segment. If this parameter set to infinity, the strategy will be *Global Re-Planning* method. In the second scene (Fig. 4c, 4d), the local planning makes a simple, natural solution and does not re-plan unnecessary.

The *Global Re-Planning with Tree Reservation* method generates path to the goal configuration like the *Global Re-Planning* algorithm. The tree reservation is an improvement in the RTR algorithm. After the first run of the global planner, the RTR does not rebuild the entire *goal tree*, instead it uses the tree built in the first iteration. As a disadvantage of this

TABLE II. Run Times for Different Planning Strategies

Scene	Global Re-Pl.	Local Re-Pl.	Global Re-Pl. with T. R.
Corridors	64.9375ms	56.1083ms	44.375ms
Triangles	7.2250ms	7.625ms	6.425ms
Parking Lot	8.40875ms	8.27795ms	12.3391ms

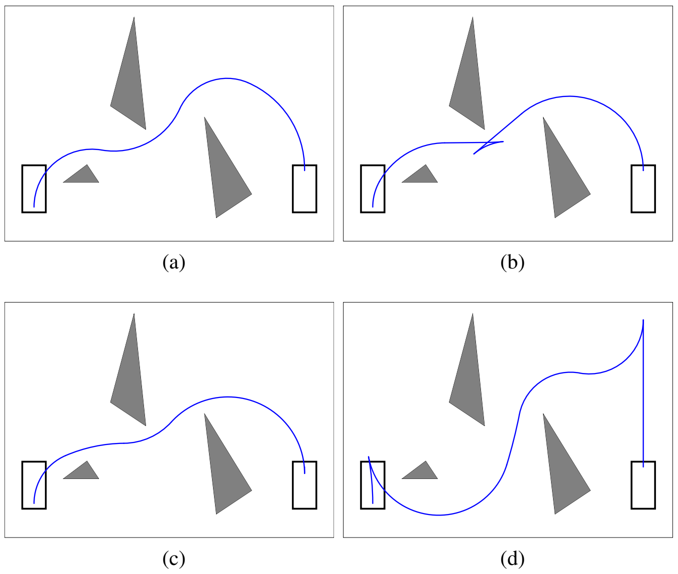
(a) (b) (c) (d)

Fig. 6. Different paths according to different parameters.

improvement, the planner has to remove the collided nodes and edges from the *goal tree* and it takes time to check the whole tree.

In TABLE II we compare the running times of the three different re-planning strategies. We have run twenty measurements for every scene, as can be seen in Section IV, with every strategy, and averaged the results. 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.

In the first scene the *Local Re-Planning* strategy gives better run time, but in the second scene *Global Re-Planning* has lower run time. We see the same result with path quality, so the performance of local or global re-planning strongly depends on the environment. The global strategy with tree reservation has lower run time for the first two scene. Thus in most cases the extra collision check in this method cause small run time growth.

VI. Improvements of the Final Path

In some cases the final paths are feasible but overly complicated to perform, because the random nature of the previous algorithms makes them sometimes contain too many short manoeuvres or drives too close to the edge of the obstacles [11]. Our goal was to create paths which are closer to the one which a human driver would plan.

In our previous work we compared different planning methods, and for this we introduced a way to measure the "quality" of the resulted path. In the sequel we use the same quality measures to improve the resulting paths. This is done

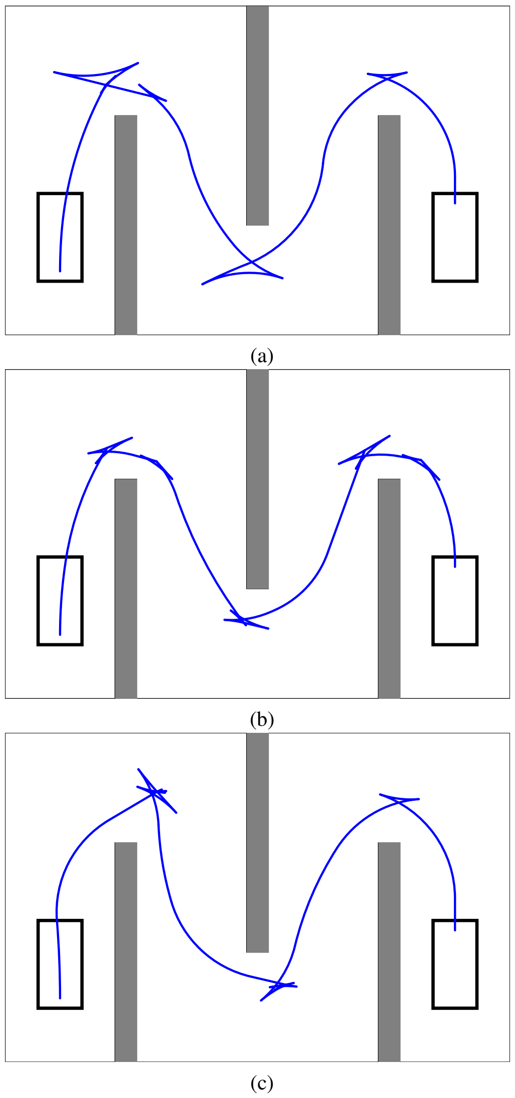
(a) (b) (c)

Fig. 7. Resulted paths in an M-shape environment.

by incorporating these already in the planning process. The measures which characterize a "fine" path are:

- *No. of cusps*: The number of reversals along the path.
- *Steering amount*: The absolute integral of the path curvature.
- *Travel time*: The estimated time of driving along the resulting path.
- *Average clearance*: The average clearance from the obstacles along the path.

For a passenger, too many reversals could be uncomfortable, as well as a high amount of turning. The travel time is estimated as follows. A travelling speed function with $v_{max} = 5\frac{m}{s}$ and $v_{min} = 1\frac{m}{s}$ 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 requires less time to travel along.

In addition to the formerly used quality measures the average clearance is added as well. To specify the clearance,

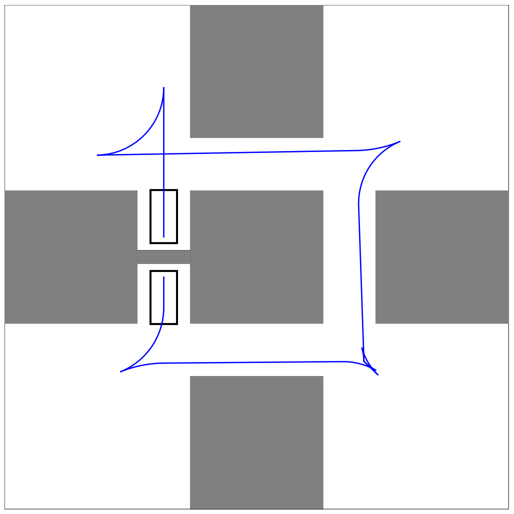

Fig. 8. Example of the four corridor environment

the path is sampled. For each sample point the minimal distance from the obstacles is calculated and summarized. Finally, the sum is normalized with the length of the path. The purpose of this parameter is to keep the robot away from the obstacles as a real driver would do to protect the car.

Examples can be seen in Fig. 6. In this scenario the robot starts from the left position facing upwards and the goal position is on the right facing down. In Fig. 6a the number of cusps are minimal. In Fig. 6b the steering amount is minimized, the straighter paths are preferred. In Fig. 6c the resulted path is longer, but it takes less time to travel along. At last the path in Fig. 6d is more complicated than the others, but the distance to obstacles is higher.

In order to tune the final path, the planning process is invoked several times for the same query and the quality measures are determined for the resulting candidate paths. The best solution is chosen by minimizing a cost function, which is a weighted sum of the different quality measures. The measures and the weighing factors are normalized to the $[0, 1]$ interval.

The weight factors can depend on different scenarios and on individual choice as well. In a narrow M-shaped environment, which is shown in Fig. 7, can be difficult to manoeuvre therefore the resulted paths have similarities. But even in this scenario the differences can be observed. In Fig. 7a, 7b the steering amount is lowered thus the longer path segments have a low curvature. In Fig. 7a the weight of cusps is higher (lower cusps number) and in Fig. 7b the travel time is shorter. In Fig. 7c the weight factor of cusps, travel time and clearance is increased so the steering amount can be higher but the other measures are better.

In a one corridor scenario the differences can be observed easier. In Fig. 9a the steering amount is low because of the straight middle segment, in Fig. 9b the number of cusps is lowered in order to achieve a simpler path. This path is the best in travel time as well. In Fig. 9c the clearance factor is increased. In this situation the car rather uses the larger open spaces to manoeuvre and drives in the middle of the corridor.

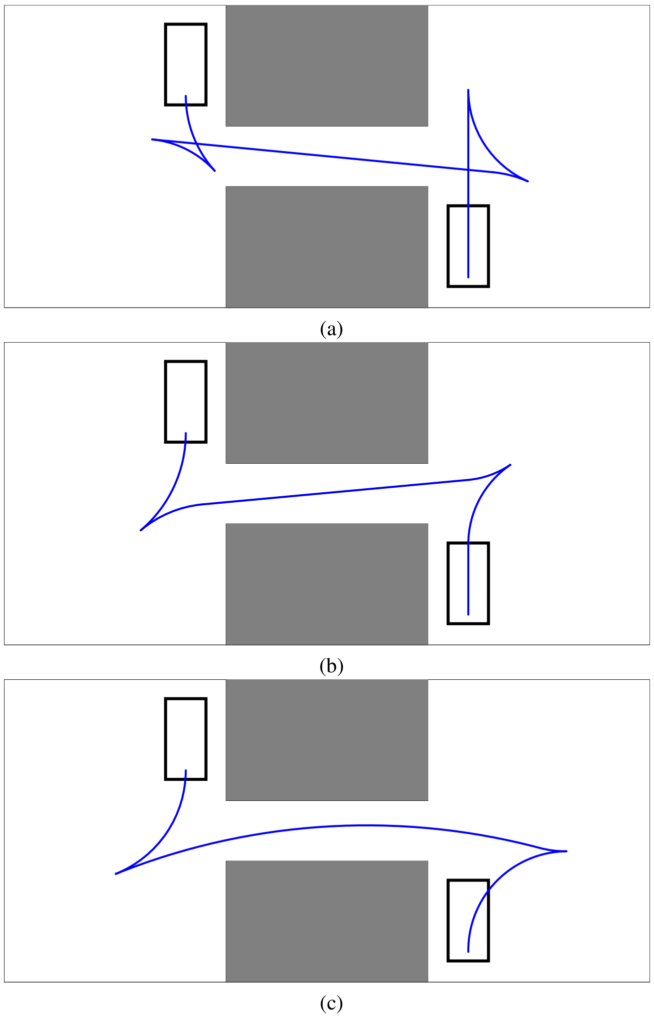

Fig. 9. Four different path between a corridor.

In a situation where the planning time is not a serious concern e.g. where the resulting path can be reused or the "quality" of the resulting path is important the above detailed method can be a suitable choice.

VII. Conclusions and Future Work

We have presented a global geometric path planning method for car-like robots and an analysis about the possible improvements. The RTR path planner with the $C^{*}CS$ steering-method is capable of designing paths even among obstacles and narrow spaces. The simulation results showed that our planning algorithm can be applied in partially unknown or changing environments and the obtained paths are quite "natural". Furthermore, its running time ranges at most a couple of seconds, 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 manoeuvring. 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] J.-P. Laumond, *Robot Motion Planning and Control*, ser. Lecture Notes in Control and Information Sciences. Springer, 1998, vol. 229.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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/.

[7] 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.

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

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

[10] 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.

[11] G. Csorvási, A. Nagy, and D. Kiss, "RTR+C*CS: An effective geometric planner for car-like robots (accepted, publishing in progress)," in *Proceedings of the 16th International Carpathian Control Conference (ICCC'2015)*, Szilvasvarad, Hungary, May 2015.

[12] D. Kiss and G. Tevesz, "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.

[13] D. Kiss, "A path planner for car-like robot maneuvering in narrow environments," in *Proceedings of the Automation and Applied Computer Science Workshop, AACS'2013*, Budapest, Hungary, June 2013, pp. 82–93.

[14] 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.

[15] J. Minguez and L. Montano, "Extending collision avoidance methods to consider the vehicle shape, kinematics, and dynamics of a mobile robot," *IEEE Trans. Robot.*, vol. 25, pp. 367–381, 2009.