Path planning and control of differential and car-like robots in narrow environments

Conference Paper · March 2015

DOI: 10.1109/SAMI.2015.7061856

3 authors, including:

Akos Nagy
Budapest University of Technology and Economics
12 PUBLICATIONS **108** CITATIONS
SEE PROFILE

Domokos Kiss
Budapest University of Technology and Economics
13 PUBLICATIONS **115** CITATIONS
SEE PROFILE

Path Planning and Control of Differential and Car-like Robots in Narrow Environments

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

***Abstract*—This paper presents a comprehensive solution for path planning and control of two popular types of autonomous wheeled vehicles. Differentially driven and car-like motion systems are the most widespread structures among wheeled mobile robots. The planning algorithm employs a rapidly exploring random tree based global planner (RTR), which generates paths made of straight motion and in place turning primitives. Such paths can be directly followed by a differential drive robot. Car-like robots have a minimum turning radius constraint, hence we present a local steering method (C*CS) which obtains a path consisting circular and straight movements based on the primary RTR-path, without losing the existence of the solution. Additionally, a velocity profile generation algorithm is presented, which is responsible for the distribution of the time parameter along the geometric path, taking the physical limitations of the robot into account. Finally, control algorithms for path following are given for both robot types. Simulations and real experiments show the effectiveness of these methods, even is constrained environments containing narrow corridors and passages.**

I. Introduction

Looking at the trends of the automotive industry, the need for intelligent autonomous vehicles is increasing. To reach full autonomy of such vehicles, many subproblems have to be solved regarding environment sensing, self-localization, path planning and motion control. In this paper we turn our attention to path planning and control, and assume that the environment and the state of the vehicle (i.e. location and pose) is always known. 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 $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$.

The locomotion systems of most ground vehicles and mobile robots are based on rolling wheels. The rolling without slipping constraint of these wheels induce *nonholonomic constraints* which cause difficulties in the control of such robots. The two most popular wheel arrangements are the two-wheeled differential drive (usually having additional supporting wheels) and the four-wheeled, car-like structure (see Fig. 1). The kinematic motion equation of these models is given by

$$\dot{x} = v \cdot \cos(\theta), \quad \dot{y} = v \cdot \sin(\theta), \quad \dot{\theta} = \omega, \tag{1}$$

where v and ω stand for the translational and angular velocities of the robot. The reference point is the axle midpoint for the differential drive and the rear axle midpoint for the car. The

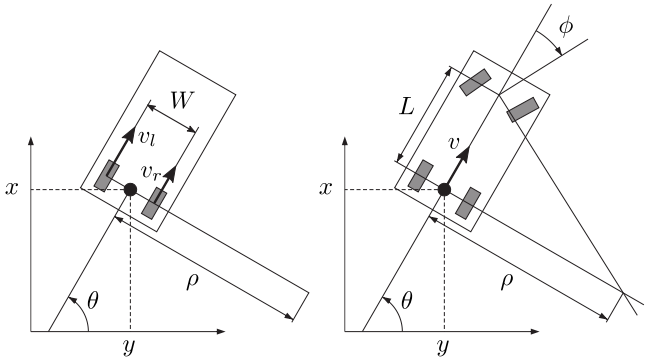


Fig. 1. Model of a differential drive (left) and a car-like robot (right)

nonholonomic constraint means practically that the robot can only move in its heading direction. The actuating variables are the independent left and right wheel velocities v_l and v_r for the differential drive, while in case of the car-like robot we treat the rear axle velocity v itself and the steering angle ϕ as control variables. It can be easily shown that $v = (v_r + v_l)/2$ and $\omega = (v_r - v_l)/W$ for the differential drive, and $\omega = (v \tan\phi)/L$ for the car. One can express the turning radii $\rho = v/\omega$ of these models as

$$\rho_{\text{diff}} = \frac{v_r + v_l}{v_r - v_l} \cdot \frac{W}{2} \quad \text{and} \quad \rho_{\text{car}} = \frac{L}{\tan\phi}. \tag{2}$$

It can be seen that ρ_{diff} can be zero with nonzero control inputs, which allows turning in place. At the other hand, ρ_{car} is lower bounded by the maximal steering angle.

The paper is organized as follows. An overview of planning approaches for nonholonomic robots are given in Section II. After this, we recall our previously developed RTR and C^*CS planning methods in Section III and show how these can be applied together to obtain feasible paths in narrow environments. Section IV shows our algorithm for assigning the velocity reference signal to the geometric path, and Section V presents our path following control algorithms. We present our test results in Section VI. Conclusions and directions of future work are summarized in Section VII.

II. Related Work on Nonholonomic Planning

Generating feasible paths for nonholonomic robots is not trivial even in the absence of obstacles [1]. 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 our

978-1-4799-8221-9/15/$31.00 ©2015 IEEE

models under consideration, such the car-like and differential drive robots. The shortest length solutions for car-like robots are characterized by the Reeds–Shepp paths [2], [3], which consist of straight lines and minimum radius circular arcs. The optimal solution for differential drive robots minimizes the rotation of its wheels. The resulting paths consist of straight movements and in place turns, described by Chitsaz et al. [4].

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 [5], 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. This approach was proposed by Laumond et al. [6]. 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 (Reeds–Shepp paths in [6]). Some other approaches belonging to this category are [7] and [8]. 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 [1].

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 [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 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], [12]. 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.

Additionally, we mention two recently proposed approaches here, which were inspiring in the development of our algorithms. In [13] a practical method is presented for planning parking maneuvers of car-like robots in narrow environments, based on the idea of reducing the free space to regions that are reachable by a concatenation of simple motions. One motion primitive is a straight–circular–straight triplet, and the resulting solution is chosen from a set obtained by a breadth-first search of candidate sequences of these primitives. The choice is the result of a numerical optimization of an objective function consiting of more weighted terms. This method is practical in situations if two or three consecutive motion primitives suffice to reach the goal (e.g. parallel or perpendicular parking tasks) but the computational cost increases strongly with the number of required primitives. The other approach, presented in [14], works well in cluttered but not very narrow environments. It employs cell decomposition algorithms to abstract the environment to a topological roadmap which is used to obtain a preliminary path. This path consists of straight segments and is made feasible for car-like robots by simply smoothing the corners by circular arcs of minimum admissible turning radius. This implies that the resulting path contains no cusps, which is a strong limitation in a narrow environment. If a path cannot be smoothened this way, or the resulting path intersects with obstacles, the path is simply rejected and another one is searched in the roadmap. The algorithm proceeds until a feasible solution is found, or it reports failure if it runs out of candidate preliminary paths.

III. The *RTR+C*CS* Path Planner

In our approach we adopt the ideas of sampling-based geometric planning and approximation by a topological steering method. A primary global path is designed by the sampling-based RTR planner [15], which consists only of straight motion and turning in place primitives. The result is feasible for the differential drive, but not for the car. If a minimal turning radius is specified, then the local C^*CS planner [16] can be applied to obtain a secondary path containing straight segments and circular arcs of lower bounded (but not necessarily minimal) radii.

A. Global Planning: RTR

The RTR planner algorithm is based on the Rapidly exploring Random Tree (RRT) approach [11]. This suggests building a topological tree in $\mathcal{C}_{free}$ starting from the initial configuration. Three main steps are repeated: sampling, vertex selection and tree extension. The sampling step picks a (mostly random) configuration q_{rand} from $\mathcal{C}$. The vertex selection step determines the nearest configuration q_{near} in the tree, according to a metric defined on the configuration space. The tree extension step tries to extend q_{near} towards q_{rand} by a feasible local path until they are connected or a collision is detected. In order to reach the goal configuration, the random sampling can be biased to include it sometimes in the random sequence, or a bidirectional search can be performed by growing two trees from both the initial and goal configurations [12].

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. During random

sampling 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. 2a).

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 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. 2b). The operation of the RTR planner is described in more detail in our recent paper [15].

The good extension properties and the effectiveness of the RTR algorithm even in narrow environments is illustrated in Fig. 3a. The two trees and the resulting path are drawn by different colors. The depicted result was obtained after 51 iterations, and our tests showed that this situation is solvable in 20 to 200 iterations, with an average iteration count of 75 (based on 20 test runs). As a comparison, we implemented the original RRT algorithm as well with the same local planning method (rotate then translate towards q_{rand}), but with the original tree extension approach (stop at the first collision). We experienced the well known limitation of RRT, namely it has great difficulties in narrow environments. Fig. 3b shows the two trees grown by the RRT algorithm after 1000 iterations. During our tests we had no successful run of RRT in the illustrated situation (the number of iterations was maximized in 1000).

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)

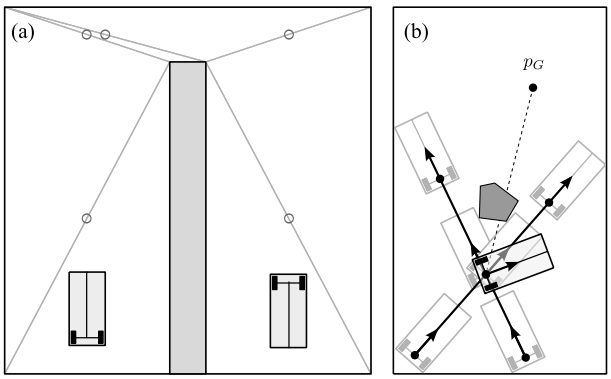


Fig. 2. (a) The random sampling is biased towards fixed guiding positions obtained by a triangular cell decomposition. (b) Illustration of the tree extension procedure if the direction to p_G is blocked

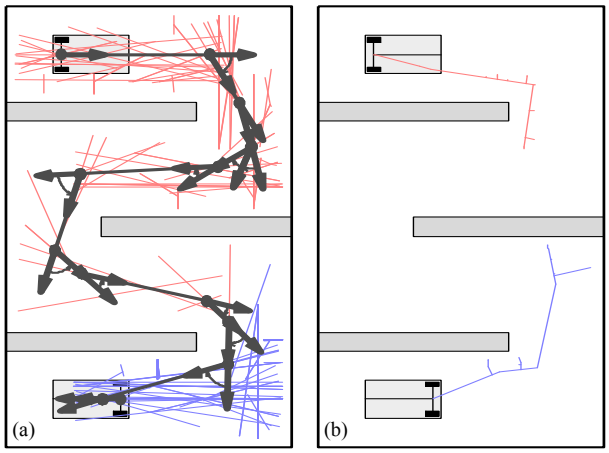


Fig. 3. (a) A result of the RTR planner through a narrow corridor, (b) The original RRT method fails

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 [16], we only summarize here its most important properties.

It is shown in [16] 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

$$q_I \xrightarrow[\rho_{I,\tilde{I}}]{C} \tilde{q}_I \xrightarrow[\rho_{\tilde{I},\tilde{G}}]{C} \tilde{q}_G \xrightarrow[s_{\tilde{G},G}]{S} q_G, \tag{3}$$

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. 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 uniformly from the workspace 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 [16]. 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 [16] 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. Velocity Profile Generation and Path Sampling

We present a time parameterization algorithm that generates a uniform time-sampled path based on a geometric path. The algorithm begins with a velocity profile generation, in which we assign a velocity value for every geometric path point. In the second step we resample the geometric path to obtain the same time interval between the samples.

A. Velocity Profile

We use the following constraints for velocity profile generation: on the one hand the maximal tangential velocity (v^{max}) and the maximal angular velocity (w^{max}) of the robot is given, on the other hand the maximum tangential acceleration (${a_{wt}}^{max}$) and the maximum resultant acceleration (${a_w}^{max}$) for the robot's wheels. The acceleration constraints are also used for deceleration. The wheel's resultant accelerations are defined by the coefficient of static friction ($\mu_{s_{max}}$), at which the wheels do not slip yet. We suppose that the static friction coefficient is constant between the ground and the robot wheels and does not depend on the direction of the force.

We determine the robot's velocity by assuming maximal tangential acceleration along the path. For this purpose we maximize the wheel's tangential accelerations by taking the constraints defined above into account:

$$a_{wt}(k) = \min\left(\sqrt{({a_w}^{max})^2 - {a_{wc}(k)}^2},\ {a_w}_t^{max}\right) \quad (4)$$

The acceleration for all wheels can not be chosen freely, because the path curvature determines the ratio between them. The exact ratio depends on the robot type (differential or car-like). After we calculated the tangential acceleration of the wheels along the path, the robot's acceleration and velocity can be obtained easily.

$$v(k+1) = \min\left(v^{max}(k+1),\ \sqrt{v(k) + 2 \cdot a_t(k) \cdot \Delta s_c(k)}\right) \quad (5)$$

We assume that the robot moves on a circular path between two path points, these circles can be defined by the curvature of the path. $\Delta s_c(k)$ stands for the distance between points k and $k+1$ on the circular path.

From the robot's velocity and angular velocity constraints we determine a maximal velocity constraint for each point of the path regardless of the previous path point:

$$v^{max}(k) = \min\left(v^{max},\ \frac{\omega^{max}}{c(k)}\right) \quad (6)$$

where $c(k)$ is the path curvature at point k. In two cases, it is possible that the velocity of the previous path point should be modified. Firstly, if the centripetal acceleration alone exceeds the maximal resultant acceleration, then the velocity of the previous path point must be reduced. Secondly, in (5) the robot's acceleration constraint may be violated, so we have to decrease the velocity at the previous path point. This could happen at the end of the path, where we specify zero velocity for the robot. In this case we have to modify the velocity profile using a back propagation algorithm, in order to avoid violation of the acceleration constraint during braking at the last path point. In both cases we use the same method for backward propagation. This backward propagation means that we have to go back and modify the velocity values along the profile until all constraints are satisfied.

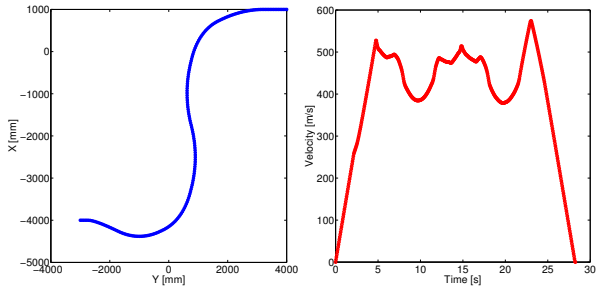


Fig. 4. The resampled path and the velocity profile for a car-like robot

B. Path Sampling

After we finished the velocity profile generation for the geometric path, we create the uniform time-sampled path for the path following algorithm. As a first step of the resampling, we determine the robot's velocity at each time sample, based on geometric velocity profile. We assume that between two path points the acceleration is constant. This was also true for the geometric velocity profile generation. A simple linear interpolation is used, which makes the above condition satisfied. From the generated velocity profile we can determine the distance between path points and due to uniform time-sampling the elapsed time between path points are already known.

Now we have to determine the time-sampled path coordinates. Since we know the distance between two path points, the resampled path can be generated by an iterative method. The first point is placed on the first point of the geometric path. After the first time-sampled point, each point is located on a circle defined by the distance from the previous point. An additional condition is that the points must be located on the geometric path and we can define an arc based on geometric path curvature for every path point. The time-sampled path points must intersect the circle and the arc. For all points there are two solutions, but we have to choose only one. The selection of the intersection points can be done by the position of the previous point.

The presented algorithm can be used for car-like robots and for differential robots as well.

V. Path Following Control

Although the path following algorithm for differential robots and for car-like robots has some common features, we discuss them separately.

A. Path Following Control of the Differential Robot

For differential robots we have two motion primitives: turning in place (e.g. at corner points of the RTR path) and path following (e.g. along straight segments or circular arcs of the C^*CS path). We use different controllers for the two primitives.

In case of turning in place, the control signal is the robot's angular velocity alone. There are two constraints that limit this:

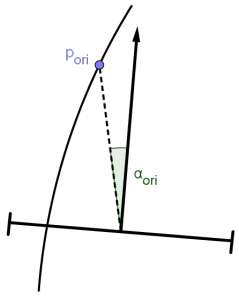


Fig. 5. The orientation controller setpoint signal

the maximal angular velocity (ω^{max}, same as in Section IV). and the maximal angular acceleration (β^{max}, this constraint can be derived by other robot constraints from Section IV).

By default we control the robot with the maximum allowable angular velocity during turning motion. In every sampling period we have to check whether the robot can safely stop before the desired orientation, and if not, the angular velocity has to be decreased. The resulting angular velocity profile has a trapezoidal shape.

The path following process for differential robots is more complex. The robot's translational and angular velocities (v and ω) are controlled separately. The wheel velocities are controlled by low-level PI controllers, whose setpoints are based on the time-sampled path velocity profile. For this purpose, the translational and angular velocity values have to be converted to wheel velocities.

The translational velocity setpoint is obtained by a reference position (p_{vel}) along the path. We iterate through the path points in the vincinity of the robot's actual position and select the nearest path point as reference point. The translational velocity belonging to this point will be used as setpoint.

For the orientation controller an orientation reference point (p_{ori}) has to be defined. p_{ori} should be further away from the robot's current position along the path than the velocity reference point. The error signal for the orientation controller (α_{ori}) is the angle between the robot current orientation and the orientation of the line defined by p_{ori} and the current robot position. The orientation controller is a PD controller, its output is the desired angular velocity of the robot.

The reference point for the orientation controller can be determined by two methods. We can specify a constant distance or a constant time shift for p_{ori} related to p_{vel}. When constant time is used, then p_{ori} is defined by a given number of samples further than velocity reference point. We can use both methods at the same time by choosing the one which results in greater distance. In this case p_{ori} is always at least at the specified constant distance to the robot's current position, but in straight parts of the path a more distant point is used because of the constant time constraint.

B. Path Following Control of the Car-Like Robot

The method for car-like robots is very similar to the one for differential robots. It is not necessary to separate v to v_r and v_l, thus the velocity reference obtained from the profile can be passed directly to the low-level speed-controller of the car. Hence we do not need to modify the previously seen speed controller at all.

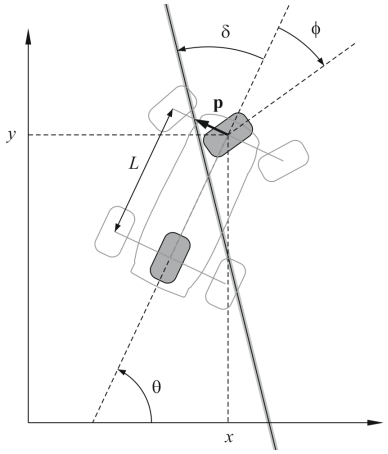


Fig. 6. Model of a line following car

The basic idea of controlling the orientation is that we have a "virtual" one-dimensional line sensor parallel to the front axle. We assume to have a system as illustrated in Fig. 6. First, we should modify the motion equation, because we treat the first axle midpoint as reference point in this model:

$$\dot{x} = v\frac{\cos(\theta+\phi)}{\cos\phi}, \quad \dot{y} = v\frac{\sin(\theta+\phi)}{\cos\phi}, \quad \dot{\theta} = v\frac{\tan\phi}{L}, \tag{7}$$

where v is the tangential speed of the center of the rear axle, θ is the orientation of the car and ϕ is the steering angle.

We first determine the tangent line of the path at the intersection point of the virtual line sensor and the path. Our goal is to control p, which is the signed lateral distance of the intersection point along the front axle, and δ, which is the angular distance between the orientation of the car and the tangent line. It can be shown that the motion equation of p and δ is

$$\begin{aligned} \dot{\delta} &= -\dot{\theta} = -v\frac{\tan\phi}{L} \\ \dot{p} &= v\cdot\tan\delta - v\cdot\tan\phi - v\cdot\frac{p}{L}\tan\delta\tan\phi \end{aligned} \tag{8}$$

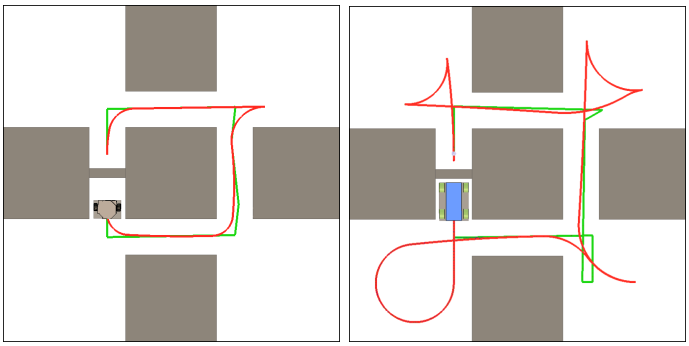
If we assume that the system will have small tracking errors, we can use a simpler model obtained by linearization around $(p,\delta)=0$:

$$\dot{\delta} = -\frac{v}{L}\phi, \quad \dot{p} = v(\delta-\phi) \tag{9}$$

From this, we can express the sate-space representation of the system:

$$\begin{aligned} x &= [\delta \quad p]^T \\ \dot{x} &= \begin{bmatrix} 0 & 0 \\ v & 0 \end{bmatrix} x + \begin{bmatrix} -v/L \\ -v \end{bmatrix}\phi \\ p &= [0 \quad 1]x + 0\cdot\phi \end{aligned} \tag{10}$$

Using a state feedback controller, we can define the poles of the closed-loop system. Based on the required poles, the feedback gains can easily be calculated by Ackermann's formula.



(a) (b)

Fig. 7. The RTR path (green) and its C*CS approximation (red) for (a) differential and (b) car-like robots

VI. Test Results

The previously presented algorithms are tested fully in simulation and partially on real robots as well. We used V-REP (Virtual Robot Experimentation Platform) as simulation environment [17]. To achieve full capabilities of the simulator program a framework is created using C++ language, together with the Fade2D library, which is a Delaunay Triangulation Library written in C++ [18]. The main reason of choosing this language is the implementation on embedded microcontrollers of the real robots.

Simulation results in an environment with more narrow corridors are shown in Fig. 7. It is not surprising that the resulting path for the car-like robot is more complicated than the path of the differential robot.

The algorithms have been tested with a differential robot in real world environment, and the results meet the simulation. The real measurements for a car-like robot (using a 1:10 scale RC-model) are under development.

VII. Conclusions and Future Work

We have presented a full solution set for path planning and path following control of differential drive and car-like robots. The RTR path planner is capable of designing paths consisting of straight movements and turning in place for differential robots. If we apply the C^*CS planner to the RTR path, then a path obeying the minimal turning radius constraint of cars is obtained. Of course, the result of the C^*CS planner is feasible for differential robots as well, and has the advantage that it can be followed by a higher speed than the primary RTR path because the robot has not to be halted at the corners. Our tests have shown a great efficiency of these algorithms in cluttered environments containing narrow corridors.

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] 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.

[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] H. Chitsaz, J. M. O'Kane, D. J. Balkcom, and M. T. Mason, "Minimum wheel-rotation paths for differential-drive mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2006, pp. 1616–1623.

[5] H. Chitsaz and S. M. LaValle, "Minimum wheel-rotation paths for differential drive mobile robots among piecewise smooth obstacles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007, pp. 2718–2723.

[6] 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.

[7] B. Mirtich and J. Canny, "Using skeletons for nonholonomic path planning among obstacles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1992, pp. 2533–2540.

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

[13] J. Kim, W. Chung, and S. Park, "Practical motion planning for car-parking control in narrow environment," *IET Control Theory and Applications*, vol. 4, pp. 129–139, 2010.

[14] N. Ghita and M. Kloetzer, "Trajectory planning for a car-like robot by environment abstraction," *Robotics and Autonomous Systems*, vol. 60, pp. 609–619, 2012.

[15] 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.

[16] ——, "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.

[17] C. Robotics, "V-REP – Virtual Robot Experimentation Platform," 2014, http://www.coppeliarobotics.com/.

[18] B. Kornberger, "Fade2D - an easy to use delaunay triangulation library for C++," 2014, http://www.geom.at/fade2d/html/.