International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

# TP-Space RRT – Kinematic Path Planning of Non-Holonomic Any-Shape Vehicles

Invited Feature Article

Jose Luis Blanco[1*], Mauro Bellone[2] and Antonio Gimenez-Fernandez[1]

1 Department of Engineering, University of Almeria, La Cañada de San Urbano, Almería, Spain
2 CETMA Consortium, Brindisi, Italy
* Corresponding author(s) E-mail: jlblanco@ual.es

Received 20 August 2014; Accepted 20 February 2015

DOI: 10.5772/60463



## Abstract

The autonomous navigation of vehicles typically combines two kinds of methods: a path is first planned, and then the robot is driven by a local obstacle-avoidance controller. The present work, which focuses on path planning, proposes an extension to the well-known rapidly-exploring random tree (RRT) algorithm to allow its integration with a trajectory parameter-space (TP-space) as an efficient method to detect collision-free, kinematically-feasible paths for arbitrarily-shaped vehicles. In contrast to original RRT, this proposal generates navigation trees, with poses as nodes, whose edges are all kinematically-feasible paths, suitable to being accurately followed by vehicles driven by pure reactive algorithms. Initial experiments demonstrate the suitability of the method with an Ackermann-steering vehicle model whose severe kinematic constraints cannot be obviated. An important result that sets this work apart from previous research is the finding that employing several families of potential trajectories to expand the tree, which can be done efficiently under the TP-space formalism, improves the optimality of the planned trajectories. A reference C++ implementation has been released as open-source.

**Keywords** Path planning, Rapidly-exploring random tree (RRT), Obstacle avoidance

## 1. Introduction

Reliable autonomous vehicles and mobile robots represent a long sought goal for their potential impact in all sectors of the industry. From the list of complex problems that must be solved in order to achieve such an objective, one of paramount importance is autonomous *navigation*, that is, driving the vehicle toward a desired goal. Although commercial self-guided vehicles have been available for industrial environments for some years, e.g., Kiva warehouse robots [1], they are often restricted to operating in controlled areas, the vehicles' tend to be round to ease collision detection on rotating, and they use differential-drive kinematics for increased maneuverability. Avoiding these restrictions is highly desirable and it is thus the goal of intense recent research [2 - 4] since it would bring practical autonomous driving to industrial vehicles, which typically have non-round shapes and Ackermann steering kinematics that severely limit their maneuverability.

Techniques for autonomous navigation can be broadly split in two categories: (i) global *path planners* which take all available information about the workspace at once, and (ii) local obstacle avoidance, or the *reactive* approach, where only immediately sensed data determine how to control the vehicle. Pure reactive navigation methods have demonstrated their reliability and flexibility to cope with robot

navigation, especially for holonomic platforms [5, 6]. In spite of their practical utility, they cannot guarantee finding the optimal path towards the desired target; indeed, local minima prevent valid paths from always being found. In contrast, the efforts in *path planning* research are focused on the search for optimal trajectories, normally in the sense of shortest paths [7]. Such efficiency comes at a cost: planners may require a few seconds to generate a result for typical office-like navigation scenarios. This is in contrast to the ability of reactive methods to quickly respond to any dynamic change sensed in the environment, typically within one millisecond [8]. Current architectures for autonomous navigation tend to fuse global (planned) and local (reactive) methods, e.g., ROS `navigation` [9] and MoveIt! [7, 10]. In such a system, a robot first generates a plan, and then it starts moving using a reactive navigator. During the execution, global planning is typically run in the background to provide a better plan in a real-time fashion.

Path planning research during the 1990s discovered the potential of sampling-based algorithms such as the Rapidly-exploring random tree (RRT) algorithm [11], which will be described later in Section 2.3. RRT itself has been the starting point for other researchers to propose extensions, for example, Jacobs *et al.* in [12] propose parallelizing tree exploration for efficiency. Another improvement, called 'transition-based RRT' (T-RRT) was introduced in [13], where a global cost map is used to guide the random exploration phase. RRT methods were further enhanced leading to many variants, with two of the most widespread being RRT* [14] and RRT-connect [15]. However, none of the above methods intrinsically considers kinematic constraints, such as those of Ackermann-driven vehicles. Inspired by [16], kinodynamic RRT* [17] was proposed in 2013 as an interesting RRT* extension to include such constraints. Despite the large step forward represented by that work, it must be noted that nonlinear constraints (as those of a planar car) are only fulfilled approximately. Noting that a robot working in real-world conditions does not always need the exact optimal solution (since a suboptimal that could be found at a fraction of the computational cost may work well), recent works, like [18], present methods for asymptotically near-optimal motion planning as a trade-off between optimality and computational time.

In this work, we introduce a new RRT variant which is implicitly aware of the exact vehicle shape and its kinematic constraints, in contrast with original RRT where generated paths cannot be followed by real vehicles. Our algorithm exploits trajectory parameter-space (TP-space) [8] transformations, introduced elsewhere for accelerating the pure reactive navigation of arbitrarily-shaped robots [8, 19]. Thus, unlike some previous reactive navigators and planners which ignore or relax non-holonomic kinematic constraints and assume circular or point-like robots, the present proposal fully observes them. Furthermore, a fundamental property of TP-space transformations is their ability to be parametrized in a number of different ways depending upon the choice of a *trajectory family*, as explained in Section 2.2, a feature which was demonstrated to improve reactive navigation in the past [8] and which also improves the quality of paths found by RRT, as found experimentally in this paper.

The rest of this paper is organized as follows. Section 2 discusses the theory behind the TP-space and introduces classical RRT. The modified RRT algorithm to solve planning problems in a parameterized space is introduced in Section 3. Next, we discuss how to configure the proposed method for an Ackermann steering vehicle, and then Section 5 provides statistical experiments. Finally, we draw some final conclusions in Section 6.

## 2. Theoretical Background

Since this work builds upon previous works on (i) RRT [11] for fast path planning and (ii) TP-space transformations [19] for efficient collision-checking, it is convenient to introduce at this point the most relevant concepts from those works.

### *2.1 Preliminary definitions*

We start by briefly defining the elements involved in path planning —the reader is referred to [20] for an excellent related tutorial. Let us define the vehicle as a rigid body $B$ in a planar workspace $\mathbf{W}=\mathbb{R}^2$. Obstacles in this Euclidean space occupy an arbitrary region $\mathbf{O}_{WS} \subset \mathbf{W}$. The configuration space (C-space) for a planar vehicle has the topological structure $\mathbf{C}=\mathbb{R}^2 \times S^1$ [20], with configurations denoted as $\mathbf{q} \in \mathbf{C}$ and $B(\mathbf{q})$ denoting the robot shape translated as a rigid body in configuration $\mathbf{q}$. Obstacles in the workspace are mapped to C-obstacles in the C-space, $\mathbf{O}_C$, which can be defined as:

$$\mathbf{O}_C = \left\{ \mathbf{q} \in \mathbf{C} \mid B(\mathbf{q}) \cap \mathbf{O}_{WS} \neq \varnothing \right\}. \tag{1}$$

The free C-space is defined as the relative complement of those obstacles, that is, $\mathbf{C}_{free}=\mathbf{C} \setminus \mathbf{O}_C$. Motion planning can be stated as the problem of finding a continuous path $\pi:[0,1] \rightarrow \mathbf{C}_{free}$ such that $\pi(0)=\mathbf{q}_i$ and $\pi(1)=\mathbf{q}_g$ given an initial configuration $\mathbf{q}_i$ and a goal configuration $\mathbf{q}_g$ fulfilling $\mathbf{q}_i, \mathbf{q}_g \in \mathbf{C}_{free}$.

### *2.2 TP-space and trajectory generators*

The TP-space was introduced in [8] as a generalization of space transformations that abstract the kinematics and shape of non-holonomic vehicles [21]. The basic idea behind those works is that the robot C-space can be partly charted by means of 2D manifolds in the parameter space of a family of trajectories (TP-space) —refer to Fig. 1 (a). By

transforming C-obstacles into the trajectory space, a representation of the free space around the robot (*TP-obstacles*) is obtained. In such a space, the robot can be treated as a free-flying point. In other words, the space transformation embeds both the vehicle shape and its non-holonomic kinematic constraints, allowing motion planning methods to deal with it as if it were a holonomic platform that can move in any direction. An illustrative example of such a transformation is depicted in Fig. 1 (b)–(c), where obstacle points in the robot workspace are mapped into the TP-space using one particular 2D manifold corresponding to a family of trajectories. Notice how a free "gap" between obstacles becomes evident in Fig. 1 (c) when inspecting TP-obstacles.

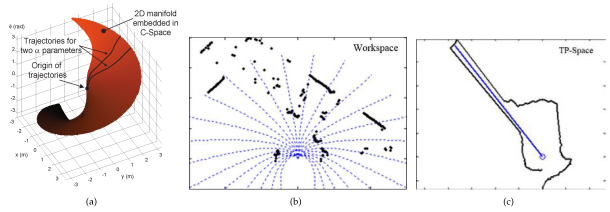


**Figure 1.** (a) C-space surface corresponding to a particular family of parameterized trajectories. (b) Obstacles in the robot workspace and a (decimated) view of a trajectory family. (c) The same obstacles, as seen in the trajectory family TP-space. In blue, the direction with the largest collision-free distance. Notice that straight lines from the origin in TP-space map to kinematically-correct trajectories in C-space.

Previous works on these ideas initially focused on their application to pure reactive navigation with the trajectory family of circular arcs [21]. It was later mathematically proven that a much wider family of paths were suitable for this end [19]. Each family is defined by means of one design function, leading to parameterized trajectory generators (PTGs). Thus, different PTGs define distinct 2D manifolds, all embedded in the vehicle C-space as illustrated in Fig. 1 (a). Since, in this paper, we address motion planning as opposed to reactive methods, the set of potential trajectories that can be used includes classical optimal path primitives [22], as explained in [8].

In the following, we introduce the basic concepts of TP-space that are required by our RRT-based method. PTG bi-dimensional manifolds are conveniently charted in polar coordinates $(\alpha, d)$ with the robot's current or initial pose being the origin, where $\alpha \in ]-\pi, \pi]$ is a parameter that modifies the shape of the trajectory and $d$ is the distance along the trajectory. Polar coordinates are preferred, since the simplest path for a holonomic robot is the straight line from its current pose, which corresponds to a path with a constant $\alpha$ value, i.e., a constant trajectory parameter in the TP-space formalism. In principle, there exists an infinite number of PTGs that map path families to the TP-space. Mathematically, the TP-space is the 2D Euclidean space of those $(\alpha, d)$ parameters, homeomorphic to the manifold in the C-space of vehicle poses $\mathbf{q}=(x,y,\phi)$ by means of the smooth transformation defined by a PTG function:

$$\begin{aligned} &PTG: \text{TP}-\text{Space} \rightarrow \text{C}-\text{Space} \\ &(\alpha, d) \rightarrow (x, y, \phi). \end{aligned} \tag{2}$$

For any fixed $\alpha$, the trajectory resulting from $PTG(\alpha,d)$ for $d \geq 0$ is kinematically feasible by definition. Let us insist again on the fact that, represented in polar coordinates centred at the current robot position, straight lines in the TP-space are mapped into valid trajectories in the real world. Workspace obstacles can be mapped into the TP-space, which allows us to quickly find the collision-free distance for any trajectory in the PTG. It was shown that building these TP-obstacles can be done efficiently for any PTG by means of pre-computed look-up tables [8]. Finally, it is important to define the inverse PTG transformation (homeomorphism requires the transformation to have a continuous inverse), which assigns TP-space points to each C-space configuration belonging to a specific trajectory family. For convenience, PTGs are defined in such a way that each $(x,y)$ position (relative to the robot pose at each instant of planning) is uniquely assigned one orientation $(\phi)$. Therefore, there is no ambiguity in defining the PTG inverse as a function of the target position only, thereby ignoring the orientation:

$$\begin{aligned} &PTG^{-1}: \text{Workspace} \rightarrow \text{TP}-\text{Space} \\ &(x, y) \rightarrow (\alpha, d) \end{aligned} \tag{3}$$

Notice that this does not imply that vehicle orientations are ignored or undefined; it only means that, given a starting pose and a particular trajectory family, the orientation becomes a function of the target Euclidean coordinates for each PTG.

### *2.3 The rapidly-exploring random tree algorithm*

Classical RRT [11] provides a reasonable trade-off between plan feasibility and computational time in solving the motion planning problem. Moreover, its probabilistic completeness and asymptotic stability guarantee convergence on a solution, if one exists [14].

The algorithm incrementally builds a tree of collision-free trajectories rooted at the initial condition. Hence, RRT is initialized as a tree, including the initial state as its unique vertex and no edges. At each iteration, a state $\mathbf{x}_{rand} \in X_{free}$ is randomly sampled from the free-object space $X_{free}$. An attempt is made to connect the nearest vertex $v \in \mathbf{V}$ to the tree through a new edge, with $\mathbf{V}$ being the set of currently existing vertexes. If such a connection is verified as collision-free, a new vertex $\mathbf{x}_{new}$ is added to the vertex set and a connection $(v;\mathbf{x}_{new})$ is added to the edge set. In the original version of RRT, iterations stop as soon as the tree contains a node in the goal region. In addition, connections between nodes in the tree have a prefixed-length and the edges are all straight lines.

**Algorithm 1** `tp_space_rrt`

**Input:** $\mathbf{x}_{init}$, $\mathbf{x}_{goal}$, $\mathbf{O}_{WS}$ ▷Starting and goal states and obstacles in WS

**Output:** $\mathbf{E}_{sol}$ ▷Sequence of path segments connecting $\mathbf{x}_{init}$ and $\mathbf{x}_{goal}$

1: $\mathbf{V} \leftarrow \{\mathbf{x}_{init}\}$, $\mathbf{E} \leftarrow \emptyset$ ▷Initialize empty tree

// Repeat until goal is added to the tree:

2: **while** $\mathbf{x}_{goal} \notin \mathbf{V}$ **do**

3: $\mathbf{x}_{rand} \leftarrow$ `SampleFree()` ▷Sample a random state in $C_{free}$ with goal bias

4: $\mathbf{S} \leftarrow \emptyset$ ▷Start with an empty set of candidate nodes

5: **for each** $i \in$ **PTGs do** ▷For each $\mathbf{x}_{random}$ n-PTG states are explored

6: $\mathbf{x}_{nearest} \leftarrow \mathbf{NN}(\mathbf{V}, \mathbf{x}_{rand}, i)$ ▷Search in the tree the nearest neighbor to $\mathbf{x}_{rand}$ using manifold $\mathcal{P}_i$

7: **if** $\mathbf{x}_{nearest} \neq \emptyset$ **then**

8: $\hat{\mathbf{x}}_{rand} \leftarrow \mathbf{x}_{rand} \ominus \mathbf{x}_{nearest}$ ▷Relative target in Workspace

9: $(\alpha_{rand}, d_{new}) \leftarrow \mathtt{PTG}^{-1}(\hat{\mathbf{x}}_{rand}, i)$ ▷Relative target in TP-space

10: $\mathbf{O}_{TP} \leftarrow \mathtt{TPS\text{-}transform}(\mathbf{O}_{WS} \ominus \mathbf{x}_{nearest}, P_i)$ ▷Build TP-obstacles

11: $d_{free} \leftarrow \mathbf{O}_{TP}[\alpha_{rand}]$ ▷Collision-free distance in the trajectory heading to $\mathbf{x}_{rand}$

12: $d_{new} \leftarrow \min(d_{MAX}, d_{rand})$

13: **if** $d_{free} \geq d_{new}$ **then** ▷Do we have enough free space?

14: $\hat{\mathbf{x}}_{new} \leftarrow \mathtt{PTG}(\alpha_{rand}, d_{new})$

15: $\mathbf{x}_{new} \leftarrow \mathbf{x}_{nearest} \oplus \hat{\mathbf{x}}_{new}$ ▷New candidate state in global coordinates

16: $\mathbf{S} \leftarrow \mathbf{S} \cup \{(d_{new}, \mathbf{x}_{new})\}$ ▷Add to set of temptative new nodes

17: **end if**

18: **end if**

19: **if** $\mathbf{S} \neq \emptyset$ **then**

20: $\mathbf{x}^*_{new} \leftarrow \min_d(\mathbf{S})$ ▷Select candidate node with the shortest path

21: $\mathbf{V} \leftarrow \mathbf{V} \cup \mathbf{x}^*_{new}$, $\mathbf{E} \leftarrow \mathbf{E} \{(\mathbf{x}_{nearest}, \mathbf{x}^*_{new})\}$ ▷Add a new vertex and a new edge towards the relative target

22: **end if**

23: **end for**

// Recover the solution path:

24: $\mathbf{E}_{sol} \leftarrow$ `tree_backtrack`$(\mathbf{E}, \mathbf{x}_{goal}, \mathbf{x}_{init})$

25: **end while**

As will be seen below, the new strategy for edge creation presented in this paper employing a TP-space has the advantage of only proposing new edges that represent kinematics-compliant trajectories. In addition, collision checking for edges is done implicitly and efficiently by transforming obstacles into the TP-space.

## 3. Rapidly-exploring random trees in the TP-space

Next we present the modified RRT method, the main contribution of the present work. The algorithm, analyzed step by step in the following, is described in pseudo-code in Algorithm 1.

The algorithm works by incrementally constructing a tree ($\mathbf{V}$,$\mathbf{E}$) where the vertices $\mathbf{V}$ represent vehicle configurations (poses) and the edges $\mathbf{E}$ are the trajectories between them. Initially, the tree only contains the starting pose $\mathbf{x}_{init}$. At each iteration, a random "target" state ($\mathbf{x}_{rand}$) is drawn from the free space, with a statistical bias (typically a probability of $P = 0.05$) towards selecting the actual final goal $\mathbf{x}_{goal}$ (line 3). Next, several families of trajectories (PTGs) are employed while attempting to grow the tree (line 5). For each such PTG, the tree node that is closest to $\mathbf{x}_{rand}$ is sought ($\mathbf{x}_{nearest}$), which in our approach implies evaluating distances over each PTG manifold (line 6). As illustrated in Fig. 2, the edges between two nodes may be blocked for some PTGs and appear as collision-free for others, e.g., circular paths in Fig. 2 (a) may generate a collision, whereas the $\alpha$ -type generated trajectory in Fig. 2 (b) denotes a feasible path. Moreover, since distances are measured along the

trajectories manifold, the closest tree node to $\mathbf{x}_{rand}$ may differ among the PTGs. Note that not every target may be reachable for all PTGs (line 7) (i.e., the target may be outside of the manifold borders), but this will not become a limitation for the path searching algorithm so long as a diversity of the PTGs is used, each one having non-overlapping unreachable zones [19].

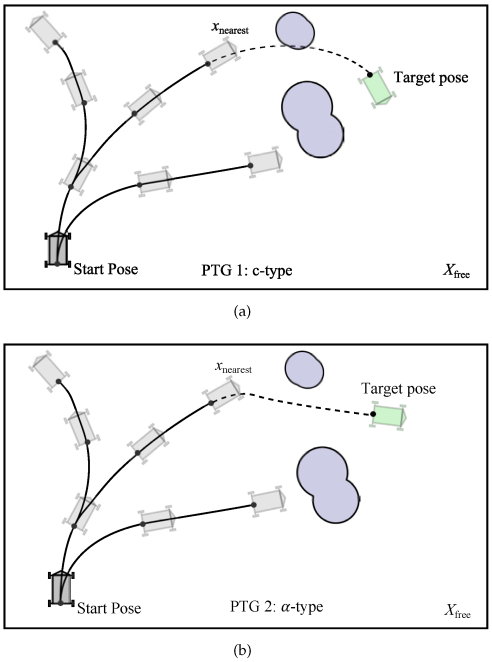


**Figure 2.** Unlike with classical RRT, the present approach only considers new tree edges that fulfill the vehicle kinematics. Moreover, several path families (PTGs) are evaluated simultaneously, a circular arc in (a) and an $\alpha$ -type path in (b), increasing the possibility of finding good collision-free trajectories.

In order to exploit the advantages of the TP-space, we must consider coordinates relative to the vehicle pose at the point where the tree is growing, i.e., at $\mathbf{x}_{nearest}$. Let $\hat{\mathbf{x}}_{rand} = \mathbf{x}_{rand} \ominus \mathbf{x}_{nearest}$ denote this relative pose of the randomly-picked state. Using the inverse PTG function, defined in Eq. (3), one can efficiently find out the trajectory parameter ($\alpha_{rand}$) and the distance over that path ($d_{rand}$) which takes it from $\mathbf{x}_{nearest}$ to that position (line 9). Once the obstacles have been mapped into the TP-space (line 10), checking for collisions throughout the trajectory of interest simply becomes a matter of testing for the minimum distance to the obstacles in the TP-space in the $\alpha_{rand}$ direction (line 11). It must be stressed that this process, which takes into account an arbitrary robot shape, takes a few microseconds in our present implementation based on look-up-tables. Since RRT defines a maximum length for new edges ($d_{MAX}$), those look-up-tables must be pre-computed only for obstacles up to the corresponding maximum distance.

Next, candidate tree nodes $\mathbf{x}_{new}$ are created for each PTG (lines 13–17), and finally only the shortest path is inserted into the tree (lines 20-21). One can find the possibilities shown in Fig. 3 while considering new tree nodes: (i) the random target is farther than $d_{MAX}$ and its path is collision-free, in which case the new node is placed along the PTG path at a distance of $d_{MAX}$ ; (ii) the path to the random target is collision-free and within range, in which case it is directly picked as the new node; and (iii) an obstacle blocks the way to the target, and hence it is discarded and the tree does not grow. Notice that these situations correspond to the same cases found in classic RRT for straight paths, with the advantage that truncating curved paths up to some arbitrary distance $d$ becomes a straightforward evaluation of the PTG function in Eq. (2) for the desired value of $d$. Eventually, the goal state will be reached and we can recover the collision-free and kinematically-valid path by backtracking the tree structure.

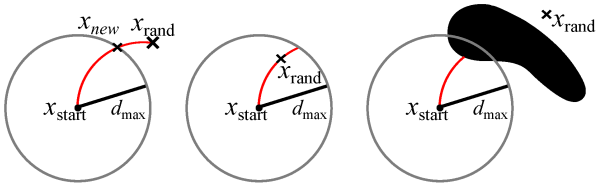


**Figure 3.** Possible situations found during edge generation. Refer to text for details.

## 4. PTG modeling for a real Ackermann-steering vehicle

Previous sections have discussed PTGs only generically regarding their role as smooth functions that chart 2D manifolds in C-space to TP-space coordinates. Next, we focus on two particular PTG models and discuss the practical problem of determining their parameters for a particular vehicle kinematics.

Adjusting the family of feasible trajectories to those attainable by a real vehicle is mandatory in order to achieve accurate and reliable path planning. In the case of Ackerman-steering vehicles, the critical parameter is the minimum turning radius. Previous works on TP-space and PTGs do not mention how to integrate this limitation, and hence we devote this section to this practical issue. In particular, we will analyze how this constraint affects two path families, circular arcs (C-PTG) and $\alpha$-asymptotic trajectories ($\alpha A$ PTG) [19], since they are the two PTGs employed in the experiments of Section 5.

### *4.1 C-PTG*

The generator functions for these paths, which define the linear $v(\alpha)$ and angular velocities $\omega(\alpha)$ for each trajectory parameter $\alpha$, are [19]:

$$v(\alpha) = K v_{max} \quad \text{(a)}$$
$$\omega(\alpha) = K \frac{\alpha}{\pi} \omega_{max} \quad \text{(b)} \qquad (4)$$

where $K$ is either 1 or -1 to select between forward or backward trajectories. It is obvious that the radius of curvature for this PTG is constant for each $\alpha$ value:

$$R(\alpha) = \frac{v(\alpha)}{\omega(\alpha)} = \frac{v_{max}}{\omega_{max}} \frac{\pi}{\alpha} \tag{5}$$

Since we are interested in setting the minimum turning radius, the worst case which must be observed is $\alpha = \pi$. Considering the geometry of the electric car model in Fig. 4, and given its wheelbase $L$ , track $W$ and maximum wheel angle $\alpha_{MAX}$ as defined in the figure, we have:

$$R_{min} = L \tan\left(\frac{\pi}{2} - \alpha_{MAX}\right) + \frac{W}{2} \tag{6}$$

which, once evaluated (for our vehicle we obtain $R_{min} = 2725$ mm), leaves one free parameter (e.g., the maximum linear velocity of the vehicle $v_{max}$) and imposes the condition $\omega_{max} = v_{max} / R_{min}$. As long as this condition holds, all the trajectories in this PTG manifold will be suitable for the real car.

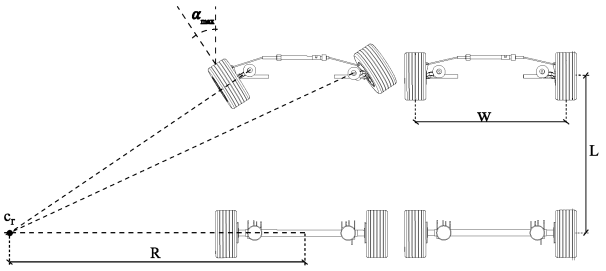


**Figure 4.** Ackerman-steering model of our electric car prototype

### *4.2 αA PTG*

In this case, the generator functions are [19]:

$$\begin{aligned} v(\alpha) &= v_{max}\, e^{-\left(\frac{\alpha-\phi(t)}{K_v}\right)^2} && \text{(a)} \\ \omega(\alpha) &= \omega_{max}\left(-\frac{1}{2} + \frac{1}{1+e^{-\frac{\alpha-\phi}{K_\omega}}}\right) && \text{(b)} \end{aligned} \tag{7}$$

with $K_v$ and $K_\omega$ as constants with angular dimensions that control how quickly the circular path tends to straighten out. From its dependency upon time, via $\phi(t)$, it can be seen that in this PTG the radius of the curvature of each trajectory is not constant for each $\alpha$ value. Trajectories generated from Eq. (7) always cause the vehicle to turn in the initial part of the trajectory and then smoothly switch towards a straight line with the orientation $\alpha$. In order to configure this PTG so as to satisfy the Ackerman kinematic constraints of the vehicle in all cases, we analyze the worst case, $|\alpha - \phi| = \pi$, where the maximum instantaneous curvature of the PTG occurs, and then equal it to the minimum turning radius of the real vehicle.

That is, $R_{min} = v(\alpha) / \omega(\alpha)$, which represents one equation with four parameters: the absolute maximum velocities ($v_{max}$, $\omega_{max}$) and the $\alpha A$ PTG shape parameters $K_v$ and $K_\omega$. By imposing the two maximum velocities according to safety considerations, we still have an additional degree of freedom in the PTG design. One arbitrary possibility is setting $K_v = K_\omega$, but any other ratio will lead to kinematically valid paths.

## 5. Experimental validation

The present section describes the results obtained with the novel RRT algorithm variant introduced in Section 3 in a benchmark of simulated environments. We should highlight that our C++ implementation has been released as an open-source library[1]. A comparison with classical RRT is omitted here due to the qualitatively-different nature of the obtained paths, whereas comparison with more optimal RRT* methods is left as future work to be done once the potential applicability of TP-space transformations to RRT* has been studied.

### *5.1 Experimental setup*

All the experiments were configured considering the future applicability of the proposed algorithm to a real electric car prototype developed at our laboratory. The simulator implements the Ackerman-steering model of such a real prototype, as shown in Fig. 4. In particular, $W = 1830$ mm is the track of the vehicle, $L = 1285$ mm mm is its wheelbase and $\alpha_{max} = 35.37$ is the maximum steering angle. With these data, it is possible to compute the minimum curvature radius $R$ with respect to the instantaneous center of rotation $c_r$.

Statistical results for the different path planning problems in the benchmark. The bold figures highlight the PTG combination that achieved the shortest average path, quickest average computation and highest success rate, respectively.

We have investigated the performance of the proposal in a benchmark comprising four path planning problems in synthetic environments, each exhibiting a different degree of difficulty. The benchmark scenarios, which can be seen in Figs. 5 (a)–(d), comprise:

1 http://www.mrpt.org/tp-rrt

| Scenario: | | 1 | | | 2 | | | 3 | | | 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PTG count: | | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
| Path length [m] | min | 9.63 | 9.56 | 9.63 | 17.08 | 16.76 | 18.76 | 34.54 | 34.09 | 36.32 | 62.93 | 62.36 | 70.54 |
| | max | 12.00 | 11.84 | 33.94 | 31.66 | 31.01 | 49.46 | 41.14 | 41.32 | 89.42 | 74.36 | 74.26 | 123.81 |
| | mean | 10.22 | **10.12** | 16.08 | **22.58** | 22.92 | 27.22 | 37.46 | **37.44** | 54.38 | 68.31 | **67.85** | 89.45 |
| | std | 0.45 | 0.41 | 5.12 | 4.74 | 4.71 | 5.87 | 1.20 | 1.22 | 8.72 | 2.38 | 2.37 | 9.70 |
| Solve time [ms] | min | 0.7 | 1.5 | 2.1 | 28.1 | 79.9 | 13.1 | 51.0 | 130.2 | 173.1 | 104.8 | 236.4 | 218.9 |
| | max | 76.1 | 91.7 | 60.8 | 4852.7 | 4462.2 | 1021.3 | 1783.9 | 4629.4 | 4550.1 | 1392.9 | 4647.2 | 1976.4 |
| | mean | **10.7** | 13.9 | 15.2 | 1433.3 | 1486.7 | **108.8** | **387.9** | 654.5 | 958.4 | **391.1** | 750.0 | 773.9 |
| | std | 11.2 | 12.7 | 9.0 | 1393.3 | 1303.8 | 104.6 | 277.9 | 525.0 | 605.3 | 186.2 | 445.2 | 308.2 |
| Success rate: | % | **100** | **100** | **100** | 22.4 | 11.2 | **100** | **100** | **100** | **100** | **100** | **100** | **100** |

**Table 1.** Statistical results for the different path planning problems in the benchmark. The bold figures highlight the PTG combination that achieved the shortest average path, quickest average computation and highest success rate, respectively.

- Scenario 1: A naive planning problem with only one obstacle in a large, empty workspace.
- Scenario 2: A "parking-like" problem that requires driving the vehicle into a relatively small space.
- Scenario 3: An easy maze-like scenario.
- Scenario 4: Another maze problem, with slightly narrower free spaces.

Additionally, the experiments analyzed the impact of choosing different sets of PTGs, a fundamental decision since they determine the range of available trajectories for the vehicle at each tree growth during RRT exploration. We define three possibilities:

1. **PTGs**=$\{C^+\}$. Only forward circular arcs.
2. **PTGs**=$\{C^+,\alpha A\}$. Forward circular arcs and $\alpha$ -asymptotic trajectories.
3. **PTGs**=$\{C^+,C^-,\alpha A\}$. Forward and backward circular arcs and $\alpha$ -asymptotic trajectories.

Due to the probabilistic nature of RRT planners, statistical experiments become mandatory. Therefore, each path planning problem has been repeated 250 times for each scenario and for each PTG set. We benchmarked the performance of the proposed method in all the scenarios by measuring: (i) the success rate, i.e., how often a valid solution is found within some predefined maximum computation time (in this experiment the limit is five seconds); (ii) the total length of the obtained paths, and (iii) the computational cost, measured as the time required to obtain a valid planned path solution. All the simulations were run on a desktop computer featuring a 3.2 GHz Intel Core i5 processor. Parallelization was not exploited during the present benchmark.

### *5.2 Results discussion*

Table 1 shows statistical results obtained from the 250 repetitions of each path planning problem for each PTG combination. A few remarkable conclusions can be established from this benchmark outcome, as discussed next.

First of all, by only considering whether a valid path is found or not found, it is clear from the success rates (bottom row in Table 1) that all the PTG combinations led to excellent results in Scenarios 1, 3 and 4 (valid paths are found in 100% of runs), while using backward circular arcs (only included in the case of three PTGs) becomes decisive in Scenario 2, where more complex maneuverability is required. Therefore, including several PTGs clearly increases the likelihood of finding a valid path.

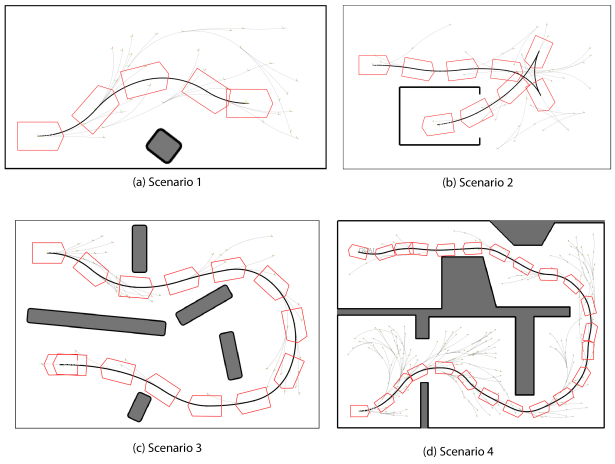


**Figure 5.** Representative results obtained by the proposed path planning method in four synthetic environments. The approximate vehicle shape (red polygon) is represented throughout the optimal path (thick black curve). Thin curves stand for other tree branches that do not belong to the optimal one. Small reference frames represent nodes in the motion tree.

With respect to computational time, we observe the expected result that, in general, employing more PTGs leads to greater execution costs. An important exception can be seen in Scenario 2, where the average computational cost dramatically drops (~92%) with the introduction of the backward moving ($C^-$) PTG. The reason is that we only count the execution cost of successful path planning runs,

and Scenario 2 is particularly challenging to RRT when using forward-only trajectories, as illustrated with an unsuccessful RRT random tree in Fig. 6. This means that it requires the construction of a much larger tree until one random node falls in the narrow window that takes the vehicle towards the goal pose.

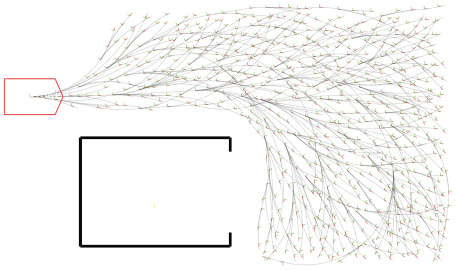

**Figure 6.** Example of an unsuccessful run of the proposed algorithm for Scenario 2 when using only two PTGS (forward circular arcs and $\alpha$-A). Not considering backward movements makes it highly unlikely that the RRT will find a path to the goal due to the minimum turning radius limitation. Compare with Fig. 5 (b).

Finally, regarding the length of the obtained paths, it should be recalled that RRT exhibits probabilistic completeness [14] but does not guarantee the global optimality of the trajectories that are found. This explains why introducing both forward and backward PTGs seems to increase the average path cost. Employing different forward-facing trajectories (in the case of using two PTGs) is only better than employing a single PTG in some scenarios (refer to row for the mean path length in Table 1), which makes sense since the shortest paths depend upon the particular geometry of the environment, and RRT itself is unable to achieve global optimality.

## 6. Conclusion

We have introduced an extension to classical RRT that enforces the creation of navigation trees whose edges are exactly collision-free and kinematically-compatible with nonholonomic vehicles. Through a statistical analysis, it has been shown that the use of different sets of parameterized trajectories can improve the obtained paths and dramatically increase the success ratio. Although for the purpose of this paper only two sets of PTGs have been used, the generality of this method allows the usage of arbitrary PTGs and even the development of new ones according to the specific constraints of a vehicle. Further research is required in order to explore the extension of the proposed ideas to optimal search methods, such as RRT* instead of RRT.

## 7. Acknowledgements

This work was partially funded by the Spanish "Ministerio de Ciencia e Innovación" under the contract DAVARBOT (DPI 2011-22513) and the grant program JDC-MICINN 2011.

## 8. References

[1] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9, 2008.

[2] Yoshiaki Kuwata, Gaston A Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P How. Motion planning for urban driving using rrt. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1681–1686. IEEE, 2008.

[3] John Markoff. Google cars drive themselves, in traffic. *The New York Times*, 10:A1, 2010.

[4] Thorsten Luettel, Michael Himmelsbach, and H-J Wuensche. Autonomous ground vehiclesâ€"concepts and a path to the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1831–1839, 2012.

[5] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1):23–33, 1997.

[6] Javier Minguez and Luis Montano. Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, 2004.

[7] Ioan Alexandru Sucan, Mark Moll, and EE Kavraki. The open motion planning library. *Robotics & Automation Magazine, IEEE*, 19(4):72–82, 2012.

[8] Jose-Luis Blanco, Javier González, and Juan-Antonio Fernández-Madrigal. Extending obstacle avoidance methods through multiple parameter-space transformations. *Autonomous Robots*, 24(1): 29–48, 2008.

[9] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, page 5, 2009.

[10] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit! *IEEE Robotics Automation Magazine*, 19(1): 18–19, 2012.

[11] Steven Michael LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept, Iowa State University, 1998.

[12] Sam Ade Jacobs, Nicholas Stradford, Cesar Rodriguez, Shawna Thomas, and Nancy M Amato. A scalable distributed RRT for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5088–5095. IEEE, 2013.

[13] Léonard Jaillet, Juan Cortés, and Thierry Siméon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4): 635–646, 2010.

[14] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[15] James J Kuffner Jr and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 995–1001. IEEE, 2000.

[16] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[17] D.J. Webb and J. van den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5054–5061, 2013.

[18] Andrew Dobson and Kostas E Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research*, 33(1):18–47, 2014.

[19] José-Luis Blanco, Javier González-Jiménez, and Juan-Antonio Fernández-Madrigal. *Foundations of Parameterized Trajectories-based Space Transformations for Obstacle Avoidance*, chapter 2. Mobile Robots Motion Planning: New Challenges. I-Tech Education and Publishing, 2008.

[20] S.M. LaValle. Motion planning. *Robotics Automation Magazine, IEEE*, 18(1):79–89, March 2011.

[21] J. Minguez, L. Montano, and J. Santos-Victor. Reactive navigation for non-holonomic robots using the ego-kinematic space. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3074–3080, 2002.

[22] Marilena Vendittelli, J-P Laumond, and Carole Nissoux. Obstacle distance for car-like robots. *IEEE Transactions on Roboticsand Automation*, 15(4):678–691, 1999.