Multi-Level Path Planning for Nonholonomic Robots using Semi-Holonomic Subsystems*

Sepanta Sekhavat[1], Petr Švestka[2], J.-P. Laumond[1], Mark H. Overmars[2]

(1) LAAS/CNRS, Toulouse, France
and
(2) Department of Computer Science, Utrecht University, the Netherlands
e-mail : petr@cs.ruu.nl, sepanta@laas.fr

Abstract

We present a new and complete multi-level approach for solving path planning problems for nonholonomic robots. At the first level a path is found that disrespects (some of) the nonholonomic constraints. At each next level a new path is generated, by transformation of the path generated at the previous level. The transformation is such that more nonholonomic constraints are respected than at the previous level. At the final level all nonholonomic constraints are respected.

We present two techniques for these transformations. The first, which we call the *Pick and Link* technique, repeatedly picks pieces from the given path, and tries to replace these by more feasible ones. The second technique restricts the free configuration space to a "tube" around the given path, and a roadmap, capturing the free space connectivity within this tube, is constructed by the *Probabilistic Path Planner*. From this roadmap we retrieve a new, more feasible, path.

In the intermediate levels we plan paths for what we refer to as *semi-holonomic* subsystems. Such a system is obtained by taking the real (physical) system, and removing some of its nonholonomic constraints.

In this paper, we apply the scheme to car-like robots pulling trailers, that is, *tractor-trailer robots*. In this case, the real system is the tractor-trailer robot, and the ignored constraints in the semi-holonomic subsystems are the kinematic ones on the trailers. These are the constraints of rolling without slipping, on the trailers wheels.

Experimental results are given that illustrate the time-efficiency of the resulting planner. In particular, we show that using the multi-level scheme leads to significantly better performance (in computation time and path shape) than direct transformations to feasible paths.

1 Introduction

Even in the absence of obstacles, planning motions for nonholonomic systems is not an easy task. So far there exists no general algorithm for planning the motions of any nonholonomic system, that guarantees to reach a given goal. The only existing results deal with approximation methods (i.e., methods that guarantee to reach a neighbourhood of the goal, e.g., [LS90, BLC93]) and exact methods for special classes of nonholonomic systems (e.g.,[LS90, MS90, RFLM93]); fortunately, these special classes contain several real robot models.

Obstacle avoidance adds a second level of difficulty : Not only does one have to take into account the constraints imposed by the kinematic nature of the system (i.e., linking the parameter

* This research has been partially supported by the ESPRIT III BRA Project 6546 (PROMotion) and by the Dutch Organisation for Scientific Research (N.W.O.)

derivatives), but also the constraints due to the obstacles (i.e., dealing with the configuration parameters of the system). It appears necessary to combine geometric techniques addressing the obstacle avoidance with control techniques addressing the nonholonomic motions. Such a combination is possible through subtle topological arguments ([Lau93]).

Treating the holonomic constraints separately from the nonholonomic ones is nowadays almost a "classical" approach. It has resulted in planners for various nonholonomic robots ([LJTM94], [SŠLO95], [SL91]). The idea is that the problem is solved in two separate steps. In the first, a collision-free path is computed without taking into account the nonholonomic constraints. Subsequently, in the second step, this geometric path is transformed into one that respects the nonholonomic constraints.

For relatively simple systems, that is, with few nonholonomic constraints, efficient path planners are obtained with the described scheme. This is for example the case for car-like robots ([LJTM94]). However, for systems with a higher degree of nonholonomy, it turns out that the second step is too time-consuming. This observation leads to the idea of further decomposition of the nonholonomic constraints, and introducing them separately.

Our paper, which develops this idea, is organised as follows : We first relate our planner to other works in nonholonomic motion planning (Section 2). Then, in order to introduce our *multi-level* planning scheme (Section 4), we discuss the concept of nonholonomy and we define what we refer to as *semi-holonomic subsystems* (Section 3). Our method consists of an initial search for a collision-free (but not necessarily feasible) path, and a number of subsequent transformation steps. Each such transformation step produces a path respecting more nonholonomic constraints than its input path. In Section 5 we present two general methods (the *PL* technique and *tube-PPP*) for the transformations. Section 6 is devoted to applying the multi-level planner to the particular example of a tractor towing a number of trailers. In order to use our planner for a given system, we need some specific local planners for the system and its semi-holonomic subsystems. The local planners that we use for the tractor-trailer problem (based on *RTR* and *sinusoidal inputs*) are presented in Section 6.2. A method for obtaining the first collision-free path (for a car towing fictive holonomic trailers) is given in Section 6.3. The paths directly generated by the different steps of the algorithm are typically very long and consist of many maneuvers. The final path, that is feasible for the tractor-trailer system is therefore hardly executable by a real (physical) robot. Hence effort must be done (and time must be spent) on smoothing the paths generated by our algorithm. Section 6.4 treats this problem in general (*probabilistic path shortening*) and for the tractor-trailer system in particular (*geometric NH-approximation*). Finally, the experimental results of Section 7 aim to illustrate that decomposing the nonholonomic constraints, and introducing separately, clearly reduces the computation time and the quality of the computed paths. Also, we show that application of the (tractor-trailer specific) *geometric NH-approximation* algorithm to the intermediate paths gives further improvement of the computation times in difficult cases, without severely penalising the search in easy ones.

2 Previous work

In the past few years, there has been a great deal of interest in motion planning algorithms that generate collision-free paths for nonholonomic systems. The tractor-trailer system is one of the examples frequently used to illustrate different algorithms. For a given system, the first question we have to answer is : can a robot reach a given goal, while avoiding collisions with the obstacles of its environment? This is the *decision* problem. For *locally controllable systems* [BL93], the existence of a feasible path between two configurations in the interior of the *free configuration space* CS_{free} is equivalent to the existence of any path between them in the interior of CS_{free}. This has led to a family of algorithms, decomposing the search in two phases. They first try to solve the geometric problem (i.e., the problem for the holonomic system that is geometrically equivalent to the nonholonomic one). Then they use the obtained path to build a feasible and collision-free one. So in the first phase the decision problem is solved, and only in the second phase the nonholonomic constraints are taken into account.

The first general result was presented by Sussmann and Liu [SL91], who proposed an algorithm constructing a sequence of feasible paths that *uniformly* converges to any given path. This guarantees that one can choose a feasible path *arbitrarily close* to a given collision-free path. The method uses high frequency sinusoidal inputs. Though this approach is general, it is quite hard to implement in practice. In [TLM+92], Tilbury *et al* exploit this idea for a mobile robot with two trailers. Experimental results however show that the approach cannot be applied in practice, mainly because the convergence is very slow. Therefore this method has never been connected to a geometric planner in order to obtain a global planner that would take into account both environmental and kinematic constraints.

Another approach was developed in LAAS for car-like robots ([LJTM94]), using Reeds and Shepp works on optimal control to approximate the geometric path. In [RS91] Reeds and Shepp presented a finite family of paths composed by of line segments and circle arcs containing a length-optimal path linking any two configurations (in absence of obstacles). The planner introduced in [LJTM94] replaces the collision-free geometric path by a sequence of Reeds and Shepp paths. This complete and fast planner was extended to the case of Hilare with one and two trailers, using near optimal paths numerically computed [LSV94, FGL93] (so far the exact optimal paths for tractor-trailer system in absence of obstacle are unknown). The resulting planners are however neither complete nor time-efficient.

The same scheme was used for systems that can be put into the *chained form*. For these systems, Tilbury et al. [TMS93] proposed different controls to steer the system from one configuration to another, in absence of obstacles. Sekhavat and Laumond prove in [SL96] that the *sinusoidal inputs* proposed by Tilbury et al. can be used in a complete algorithm transforming any collision-free path to a collision-free and feasible one. This algorithm was implemented for a car-like robot towing one or two trailers, which can be put into the chained form, and finds paths in reasonable times ([SL96]).

Another important class of motion planning algorithms consists of search based methods that build and explore a graph, with nodes being robot-configurations and edges corresponding to (simple) feasible paths. With increasing computation time, the graph tends to cover CS_{free}. Often heuristics are used to guide the search, in order to reduce the computation time required for obtaining a graph capturing sufficiently the connectivity of CS_{free}. These methods are of course penalised when the size of the search space grows.

The *Probabilistic Path Planner PPP* ([KŠLO95, ŠO95]) builds a graph of states in the free configuration space CS_{free}. The nodes are chosen randomly, and a local planner searches for paths linking pairs of admissible and close configurations. Whenever such a (local) connection succeeds, a corresponding edge is added to the graph. Given path planning problems can be then solved by performing searches in this graph (or roadmap). The critical point of *PPP* when applied to nonholonomic robots is the speed of the nonholonomic local planner.

For car-like robots very fast local planners have been developed. Thanks to this, *PPP* applied to the car-like robots resulted in fast and probabilistically complete planners for car-like robots that move both forwards and backwards, as well as for such that can only move forwards ([ŠO95]).

Local planners for tractor-trailer robots however tend to be much more time-consuming, which makes direct use of *PPP* less attractive. In [ŠV95] a local planner is presented and integrated into *PPP*, that uses exact closed form solutions for the kinematic parameters of a tractor-trailer robot. In [SŠLO95] the local planner using sinusoidal inputs for chained form systems is used. For practical use, both local planners appear to be too expensive for capturing the connectivity of CS_{free}. For this reason, in [SŠLO95] a two-level scheme is proposed, where at the first level the portion of CS_{free} is reduced to a neighbourhood of a geometric path, and at the second level a (real) solution is searched for within this neighbourhood. The multi-level algorithm proposed in this paper can in fact been seen as a generalisation of this two level scheme.

Barraquand and Latombe [BL93] propose a heuristic brute-force approach to motion planning for tractor-trailer robots. It consists of heuristically building and searching a graph whose nodes are small axis-parallel cells in configuration space. Two such cells are connected in the graph if there exists a basic path between two configurations in the respective cells. The completeness of this algorithm is guaranteed up to appropriate choice of certain parameters. The main drawback of this planner is that when the heuristics fail it requires an exhaustive search in the discretised configuration space. Furthermore, only the cell containing the goal configuration is reached, not the goal configuration itself. Hence the planner is inexact. Nevertheless, in many cases the method produces nice paths (with minimum number of reversals) for car-like robots and tractors pulling one trailer. For systems of higher dimension however it becomes too time consuming.

Ferbach [Fer] builds on this method in his *progressive constraints* algorithm in order to solve the problem in higher dimensions. First a geometric path is computed. Then the nonholonomic constraints are introduced progressively in an iterative algorithm. Each iteration consists of exploring a neighbourhood of the path computed in the previous iteration, searching for a path that satisfies more accurate constraints. In fact, for an intermediate path the tangent vector at each point is not constrained to lie in a given tangent space (imposed by the nonholonomic constraints), but only in a larger space containing this tangent space. This "larger space" tends to the exact tangent space during iterations. The underlying assumption of this approach is that the current path can actually converge towards a nonholonomic solution, when the forbidden areas of the phase space[1] expand. This assumption is however not true in general, for which reason the algorithm is not complete. Nevertheless, smooth collision-free paths in non-trivial environments were obtained with this method for car-like robots towing two and three trailers. The *multi-level* path planner introduced in this paper also decomposes the search into several phases of computation using intermediate paths. We however want to stress that the basic principle (leading to *complete* planners) is fundamentally different from the *progressive constraints* method. In the *progressive constraints* method all nonholonomic constraints are relaxed at the beginning and then, all together, taken into account more and more during the iterations, whereas in the *multi-level* scheme the nonholonomic constraints are added one by one. At each step a path is computed whose tangent vectors are chosen in a "subspace" containing the tangent subspace imposed by the nonholonomic constraints of the system. Whereas in the *progressive constraints* method only the *size* of this space is reduced in each iteration, our multi-level planner decreases the *dimension* of this space. Thanks to this, the number of intermediate paths is deterministic and known at the very beginning of the search. Besides, there is no discretization of the space and the elementary paths are exact (not computed by numerical integration).

Apart from the above general nonholonomic motion planners, some specific planners have been developed for special cases.

Dealing with tractor-trailer problem for example, we can cite [Luz94] in which a rule-based incremental control was applied to the special problem of parallel parking.

Another example is [SSB94] in which the goal is reached approximately (there is no control on the final position of trailers). This work concerns only cases in which a path without reversals between the extremal configurations exists. In these cases, the difference is computed between the area swept by the front car and that swept by a given trailer. Then, in many realistic cases, planning for the first car (going only forwards) can guarantee obstacle avoidance for the whole tractor-trailer system. This method is especially useful for industrial applications in which the exact position of trailers is not important, and the environment can be adapted to mobile robots that can perform only forward motions. At first glance, the method shows some similarities with the first planning step of our multi-level planner for tractor-trailer robots, as presented in this paper. However, conceptually there are again fundamental differences. In [SSB94], the planning is done only for the front car, and the trailers are ignored. The trailers just follow the car with feasible motions, induced by their nonholonomic constraints. In our case, the first path will be computed for a fictive system composed by a real car pulling "holonomic" trailers, and the planning is done for the entire system. That is, not the trailers are ignored, but merely their nonholonomic constraints. Through this, completeness of the multi-level planner is guaranteed. The concept of fictive, or *semi-holonomic subsystems*, is developed in the next section.

[1] The *phase space* is the set of (c,c') where c is a configuration and c' a velocity (not necessarily adapted to c for any particular nonholonomic system).

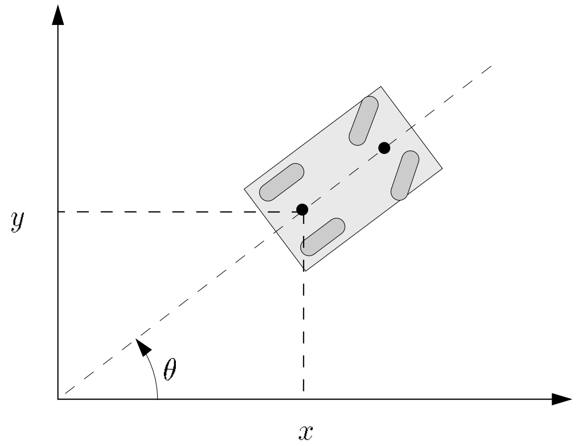


Figure 1: A car-like robot.

3 Nonholonomic systems and fictive simplifications

Holonomic constraints can be characterised by a set of equations involving only the system-state. This means that they only reduce the free configuration space CS_{free}, which makes it possible to apply any classical geometric approach (e.g., cell-decomposition, potential field) for tackling such problems. Nonholonomic constraints can only be characterised by a set of non-integrable equations involving, apart from the system-state, also the derivative (with respect to time) of this state. Because the equations cannot be integrated, the nonholonomic constraints do not reduce CS_{free}. However, they restrict the *direction* in which a CS motion is allowed. So, for a nonholonomic robot, not every collision-free path (that is, a path lying in the CS_{free}) is a feasible path.

For example, a car-like robot is nonholonomic. Physically, the nonholonomy is induced by the fact that the wheels cannot slide. This imposes a relation between the robot orientation and its admissible velocity. This relation can be expressed by the following non-integrable equation (see also Figure 1):

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0$$

Now let us consider a car-like tractor pulling n trailers. A configuration c of this system can be represented by $n+3$ parameters : $c = (x, y, \theta_0, \ldots, \theta_n)$. The position of the tractor is defined by x and y, the orientation of the tractor by θ_0, and the orientation of the k^{th} trailer by θ_k. See also Figure 2. The orientation of the tractors front wheels (Φ) is not taken into account here.

For each trailer we have an equation analog to the one above, linking the velocity of the (rear) axle midpoint and the orientation of the trailer. So the nonholonomic constraints of the tractor-trailer system are represented by $n+1$ equations of type $(0 \leq i \leq n)$:

$$E_i : \quad F_i(c, \dot{c}) = 0$$

These can be transformed to a non-integrable system of equations of the form :

$$\dot{c} = G(c, u)$$

where u is a control vector of dimension 2 (see [BL93]).

For such a system we can define $n+1$ fictive systems : $S_0, \ldots, S_n$. S_i is defined as the system respecting the nonholonomic constraints represented by the equations E_0 to E_i. We refer to S_i as the *semi-holonomic subsystem of degree i*.

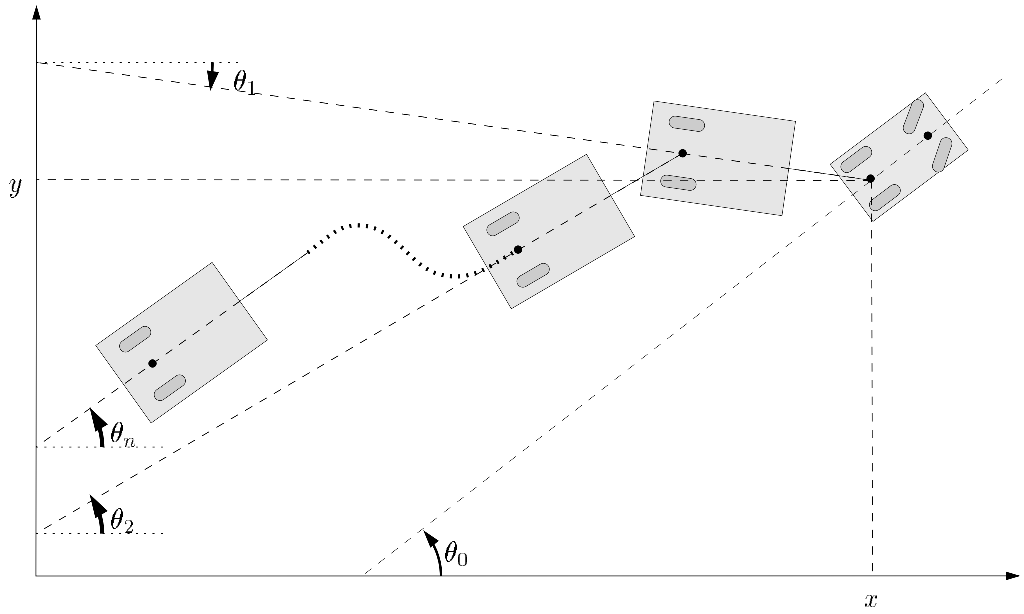


Figure 2: A tractor-trailer robot (with n trailers) placed at configuration $c = (x, y, \theta_0, \theta_1, \ldots, \theta_n)$.

Notice that the semi-holonomic subsystem, as explained above by the example of tractor-trailers, is a general concept. Indeed any nonholonomic system is defined by a set $\mathcal{E}$ of constraint equations of type E_i. So we can associate to it semi-holonomic subsystems by considering only subsets of $\mathcal{E}$.

4 The multi-level scheme using transformations between semi-holonomic subsystems

For simple systems, that is, with few nonholonomic constraints, efficient path planners are obtained by separating the holonomic constraints from the nonholonomic ones, and computing paths in two steps, according to the following (classical) scheme :

1. Compute a collision-free path without taking into account the nonholonomic constraints. If such a path does not exist, there is no solution to the problem.
2. Replace the (holonomic) path by a sequence of feasible ones.

The complexity of the transformation step (2) and the quality of the final path however depend on the initial path. There is a trade-off. The more the initial path respects the nonholonomic constraints, the faster will be the transformation, and the nicer will be the resulting path. However, more computation time is required to compute such initial paths. For robots with many nonholonomic constraints, it turns out that it pays off to spend more time on getting "good" initial paths.

The multi-level path planning scheme that we propose in this paper aims at this, by introducing a sequence of transformation steps, instead of just one. It uses the concept of semi-holonomic subsystems, as defined in the previous section. Given a real system S with $n+1$ semi-holonomic subsystems $S_0, \ldots, S_n$, we first find a path P_0 for system S_0. Then, in n subsequent steps, we transform it to a path feasible for the real system S. At step i $(0 \leq i < n)$, path P_i, feasible for S_i, is transformed to a path P_{i+1} feasible for S_{i+1}. Hence, the multi-level scheme that we propose is as follows :

1. Compute a collision-free path P_0, respecting the nonholonomic constraints of system S_0. If such a path does not exist, there is no solution to the problem.

2. **for** $i = 0$ **to** $n-1$:
   Transform path P_i to a collision free path P_{i+1},
   respecting the nonholonomic constraints of system S_{i+1}

Of course, the key ingredient is the transformation of the paths, in a way that they become feasible for the "next" system. In the following section we present two such techniques. Also, means for obtaining the initial path P_0 are required.

5 Obtaining initial paths and transformation techniques

The initial path P_0 is a path feasible for the (simple) system S_0, which has only one nonholonomic constraint (described by equation E_0). It is typically an easy task to compute such a path, with existing planners. For example, in the planner that we have implemented for tractor-trailer robots (as described in the Sections 6 and 7), an initial roadmap is computed with the *Probabilistic Path Planner* (See Section 6.3). This roadmap stores paths feasible for S_0, that is, paths respecting the tractors nonholonomic constraints, but ignoring those of the trailer(s).

In the rest of this section, we focus on ways of performing the transformation steps in the multi-level algorithm. We assume that S_i is the (sub)system for which we want to transform a path P_{i-1} to a path P_i. Furthermore, we assume that L_i is a *local planner* for system S_i. That is, L_i is a function that takes two robot configurations as arguments, and returns a path connecting its arguments, that (in absence of obstacles) is feasible for system S_i. A local planner must posses a *topological property*, in order to guarantee *completeness* of the transformation step. We give this property in Section 5.3.

5.1 PL method

One way of performing the transformation of a path P_{i-1} to a path P_i is the *Pick and Link (PL)* method. Let us assume that the collision-free path P_{i-1} is parametrised by $s \in [0..1]$. $P_{i-1}(0)$ and $P_{i-1}(1)$ are the extremal configurations. We first try to join these two configurations using the local planner L_i . If the obtained path is collision-free then we have a feasible path avoiding obstacles, and the problem is solved. If not, we take an intermediate configuration (let us say $P_{i-1}(\frac{1}{2})$) on the collision-free path P_{i-1} and we apply recursively the same treatment to the portion of the collision-free path between $P_{i-1}(0)$ and $P_{i-1}(\frac{1}{2})$ and to the portion of the path between $P_{i-1}(\frac{1}{2})$ and $P_{i-1}(1)$. As the algorithm proceeds, the considered extremal configurations will lie closer and closer to each other. Thanks to the *topological property* of the local planner, when the final configuration tends to the initial one, the length of the local path linking them tends to zero. This guarantees the convergence of the algorithm. For a serious demonstration see [SL96].

Strong points of this technique are its *completeness* and relative time-efficiency in cluttered regions of CS. The paths produced are however often very long and "ugly", and therefore require significant *smoothing* (See also Section 6.4). Furthermore, the completeness of the algorithm is only guaranteed if the input path P_{i-1} has a non-zero clearance from the obstacles.

5.2 Tube-PPP

The second transformation technique that we describe is based on the *Probabilistic Path Planner*, or *PPP* ([KŠLO95, ŠO95]).

PPP is conceptually quite simple. A roadmap R is constructed incrementally, by repeatedly generating a *random*[2] free configuration c, and trying to connect c to a number of previously generated configurations with the local planner. Whenever such a connection succeeds (that is,

[2]Heuristics have been developed for generating more nodes in certain interesting/difficult areas of CS_{free}.

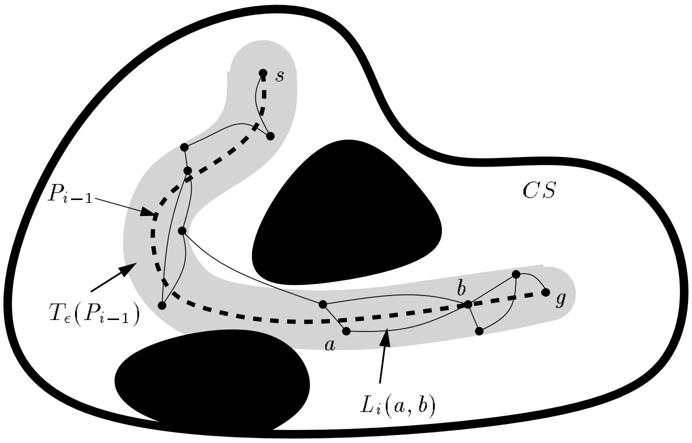


Figure 3: A path P_{i-1} defines a tube in CS, in which a roadmap, feasible for system S_i, is constructed. Note that, although all nodes lie in the tube, the local paths are allowed to exit it.

the computed local path is collision-free), the roadmap is extended with the corresponding local path. Once a roadmap has been constructed in the above manner, it can be used for retrieving feasible paths. We denote PPP with a specific local planner L by $PPP(L)$.

Given a path P we now define $T_\epsilon(P)$ to be the subset of CS that lies within distance ϵ of P, where ϵ is a (small) constant. We refer to $T_\epsilon(P)$ as the *CS-tube around P*. Formally :

$$T_\epsilon(P) = \{c \in CS | \exists \tilde{c} \in P : |\tilde{c} - c| \leq \epsilon\}$$

The transformation is now performed by executing $PPP(L_i)$ on $T_\epsilon(P_{i-1}) \cap CS_{free}$. That is, instead of picking the node configurations randomly from the whole CS_{free}, they are only picked from the free portion of $T_\epsilon(P_{i-1})$. The concept is illustrated in Figure 3. The start and goal configurations s and g are added as nodes at the very beginning, and the roadmap is extended in the standard way until s and g are graph-connected. Then, a graph search and concatenation of appropriate local paths gives a path P_i, feasible for the (sub)system S_i.

With respect to the PL transformation algorithm, *tube-PPP* has advantages as well as disadvantages. The paths produced are typically much shorter than those constructed by the PL technique. This means that less time has to be spent on smoothing them. Also, no minimal clearance between the (input) paths and the obstacles is required. However, the method is only *probabilistically* complete, and transformations of path segments where only concatenations of many short paths bring a solution tend to be more time-consuming than with the PL method.

Choosing the transformation techniques for the different transformation steps requires sensible choices to be made. In the implementation that we present in Section 7 we have based these choices on experiments.

5.3 Completeness of the transformation steps

Let us consider a configuration space CS equipped with a metric $d : CS \times CS \longrightarrow \Re+$. Let $B(c, r)$ be the ball of radius r around the configuration c. Let $\ell : CS \times CS \longrightarrow CS^{[0,1]}$ be a *local planner*; for two configurations $a, b \in CS$, $\ell(a, b)$ is a path $\ell_{a,b}(t), t \in [0, 1]$ such that $\ell_{a,b}(0) = a$ and $\ell_{a,b}(1) = b$.

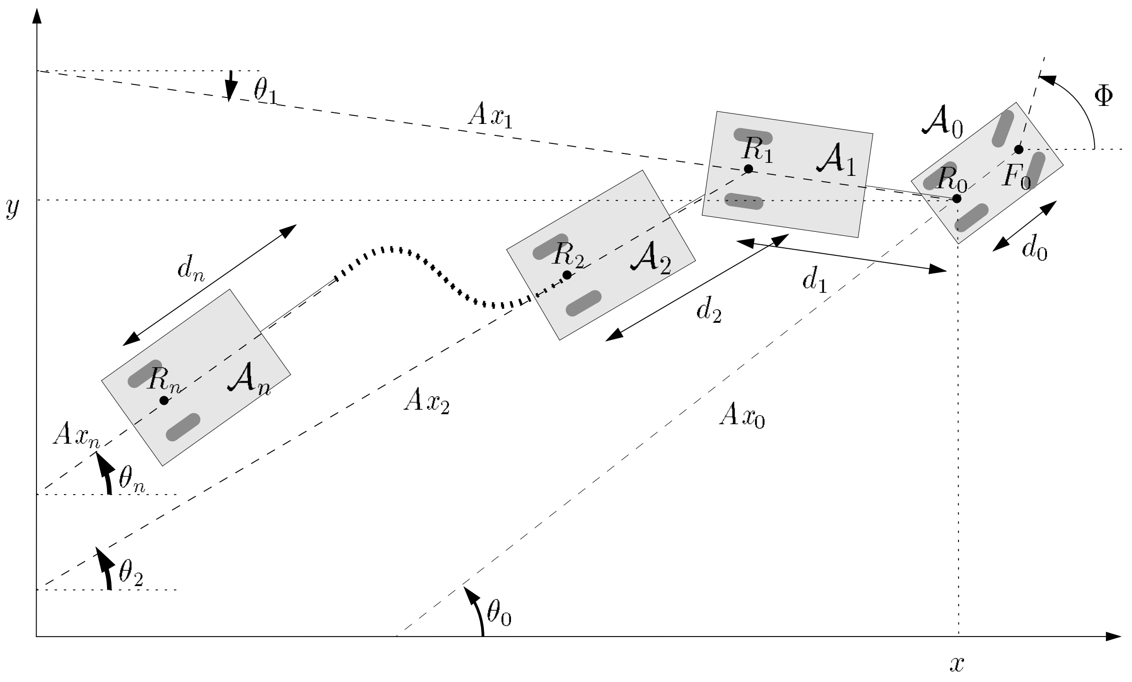


Figure 4: A tractor-trailer robot placed at configuration $(x, y, \Phi, \theta_0, \theta_1, \ldots, \theta_n)$, with also shown the tractors front point F_0, the vehicles rear points R_i, the main axes Ax_i and the rear point distances d_i.

Definition : A local planner ℓ verifies the *Topological Property (*TP*)* if:

$$\forall \epsilon > 0, \exists \eta > 0 \text{ such that } \forall c_0 \in CS, \forall c \in B(c_0, \eta) :$$
$$\forall t \in [0,1] : \ell_{c_0,c}(t) \in B(c_0, \epsilon)$$

Property : Let ℓ be a local planner that respects the topological property *TP*. The completeness of both the *PL* algorithm as *PPP* using ℓ is guaranteed. For proofs we refer to, respectively, [SL96] and [ŠO95].

6 Application to tractors with trailers

In this section we describe how the multi-level planning scheme can be applied to robots consisting of a car-like tractor, towing a number of trailers. First, in Section 6.1, we formally define tractor-trailer robots, and we give formulas that describe their nonholonomic constraints. We have seen, in Section 5, two transformation methods for the intermediate steps, requiring local planners respecting the topological property *TP*. In Section 6.2, we describe such local planners for tractor-trailer robots and their semi-holonomic subsystems. In addition to the transformation methods, we need an initial path. One way of obtaining such a path for S_0 is described in Section 6.3. Finally, in Section 6.4, we present methods for heuristically improving the quality of the paths produced by the transformation algorithms.

6.1 The tractor-trailer system

As described in Section 3, a configuration of a tractor with n trailers can be represented by $c = (x, y, \theta_0, \theta_1, \ldots, \theta_n)$. We have seen that the nonholonomic constraints of this system can be expressed by $n+1$ equations that lead to a control system with two inputs. For example, we can choose as inputs the tangential and the angular velocities of the tractor. If we consider a real tractor, the mechanical stops on the front wheels constrain the motion of the vehicle to have a bounded curvature. For the motion planning problem, this imposes a relation between the two inputs. Rather than treating the problem in this way, we consider the angle Φ of the car front wheels (with respect to the x-axis) as a coordinate of the configuration $c = (x, y, \Phi, \theta_0, \theta_1, \ldots, \theta_n)$ (See also Figure 4). Then the mechanical stops are simply geometric constraints on this coordinate. We use $\frac{1}{3}\pi$ as *maximal steering angle*. That is, when the front wheels are in a "straight" position, they cannot turn over an angle of more than $\frac{1}{3}\pi$ (clockwise or counterclockwise). This corresponds approximately to normal cars.

We now first introduce some notations and terminology, which we will use throughout this section (See also Figure 4). The tractor is denoted by $\mathcal{A}_0$, and the j-th trailer by $\mathcal{A}_j$. We refer to the midpoint between front wheels of $\mathcal{A}_0$ as *the tractors front point*, and the midpoint between its rear wheels as *the tractors rear point*. These points are denoted by, respectively, F_0 and R_0. The line going through these two points is referred to as *the tractors main axis*, denoted by Ax_0. Furthermore, for the trailers (that is, for $j > 0$), the midpoint between the wheels of $\mathcal{A}_j$ is *the rear point of trailer* j, denoted by R_j, and the line going through R_j and R_{j-1} is *the main axis of trailer* j, denoted by Ax_j. If we want to specify any of the above variables for a specific configuration c at which the robot is placed, we parameterise the variable by this configuration. Finally, we denote the (fixed) distance between the rear point R_{i-1} of a trailer and the rear point R_i of the vehicle in front by d_i.

For a tractor pulling two trailers, the kinematic model of the system can be expressed by the following equations :

$$\begin{cases} \dot{x} = cos\theta_0 v_0 \\ \dot{y} = sin\theta_0 v_0 \\ \dot{\Phi} = \omega \\ \dot{\theta}_0 = \frac{1}{d_0} tan(\Phi) v_0 \\ \dot{\theta}_1 = \frac{1}{d_1} sin(\theta_0 - \theta_1) v_0 \\ \dot{\theta}_2 = \frac{1}{d_2} sin(\theta_1 - \theta_2) cos(\theta_0 - \theta_1) v_0 \end{cases}$$

The inputs here are v_0, the velocity of the car, and ω, the front wheels angular velocity (that is, the derivative (in time) of the tractors steering angle). For a realistic system, not only do we have to take into account some bounds for the steering angle Φ, but also for the angles between consecutive bodies composing the robot (that is, between their main axes). We use $-\frac{1}{2}\pi$ and $\frac{1}{2}\pi$ as bounds.

The nonholonomic constraints of this system are imposed by the rolling without slipping of the wheels. So a natural way of building subsystems for a system composed by a tractor and n trailers is to ignore the existence of some of the wheels. We define the subsystem S_k $(0 \leq k \leq n)$ as in Section 3. That is, S_k is a car-like robot with the first k trailers nonholonomic (that is, with wheels), and the remaining $n-k$ trailers being holonomic. This means that the last $n-k$ trailers can rotate freely (within $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$) around their linking point with the vehicle in front.

6.2 Local planners for the subsystems

As we have seen above, for the both described transformation schemes we need a local planner respecting the topological property. Let us assume that we posses such a local planner ℓ for the system composed of the first $i+1$ bodies of our robot (that is, for the tractor and the first i trailers). We can then define a local planner ℓ_i for the system S_i as follows : we first plan a path only for the tractor and the first i trailers with the local planner ℓ. Then we simply interpolate the linking angles of the remaining trailers between the extremal configurations (see also Figure 5). It is easy to prove that such a local planner ℓ_i always respects the topological property for S_i.

So in order to have a suitable local planner for the subsystem S_i, we need a local planner for a tractor with i nonholonomic trailers, respecting the topological property. We will now introduce

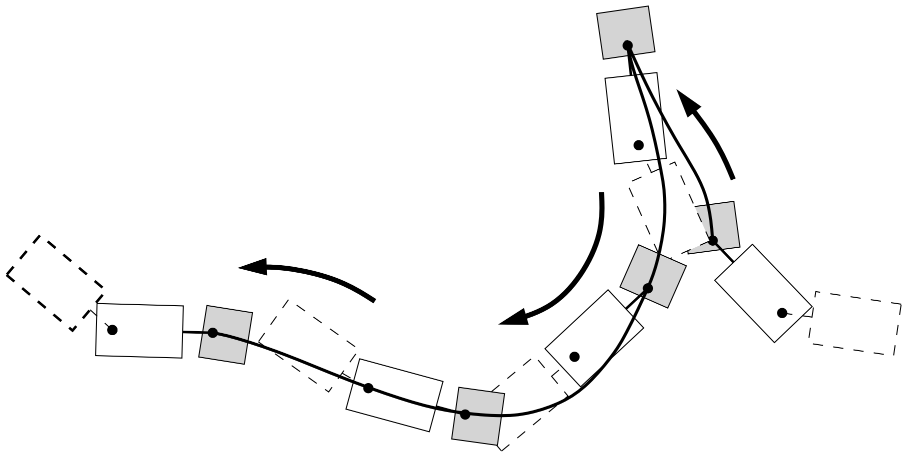


Figure 5: A path feasible for the tractor and the first trailer. The orientation of the second trailer, relative to the orientation of the first, is interpolated. In other words, we see a path feasible for system S_1.

such local planners.

6.2.1 RTR-planner for a car-like robot

For a car-like robot we use a quite simple local planner ℓ. Given two (car-like) configurations, it constructs the shortest path connecting them that consists of a (constant curvature) curve, a straight path segment, and another (constant curvature) curve. We refer to this local planner as the *RTR local planner*, and we denote it by L_{RTR}. This local planner is quite a cheap local planner, in the sense that construction and collision checking of the local paths can be done very efficiently. Furthermore, it guarantees probabilistic completeness of *PPP* (See also [ŠO95]).

6.2.2 Sinusoidal planner for tractor-trailers

Briefly, the principle of the *sinusoidal local planner* is to transform the state coordinates into the chained form and to apply sinusoidal inputs to the transformed system.

In what is considered as the *classical* tractor-trailer system in the literature (and that we consider in this paper), each vehicle is hitched to the centre of the rear axle of the front vehicle. Such system can be put in a form called the "chained form", locally around almost any point of the configuration space. The singularities are the points where one of the angles between two vehicles is $\frac{1}{2}\pi$. If we consider a physical system for which $\frac{1}{2}\pi$ is an upper bound for the admissible values of these angles (this is the assumption we have made for our tractor-trailer), then locally around any point of the configuration space the system can be put in the chained form :

$$\begin{cases} \dot{z_1} = u_1 \\ \dot{z_2} = u_2 \\ \dot{z_3} = z_2.u_1 \\ . \\ . \\ . \\ \dot{z_n} = z_{n-1}.u_1 \end{cases}$$

In the appendix the particular system of a tractor pulling two trailers is converted into chained form. For the classical system of tractor with n trailers, a general method of transformation into chained form is given by Sordalen [Sor93].

For chained form systems, there exists several types of control that bring them from one point to another : piecewise constant inputs, polynomial inputs or sinusoidal inputs [TMS93]. We need one of the corresponding local planners that respects the topological property.

The sinusoidal inputs for a chained form system of dimension n are defined as :

$$\begin{cases} u_1 = a_0 + a_1 sin(\omega t) \\ u_2 = b_0 + b_1 cos(\omega t) + b_2 cos(2\omega t) + ... + b_{n-2} cos((n-2)\omega t) \end{cases}$$

ω is fixed and $T = 2\frac{\pi}{\omega}$ is the integration time (the time required to steer the system from the start to the goal position). a_1 is chosen non-zero, and has influence on the shape of paths. The other parameters $(a_0, b_0, ..., b_{n-2})$ are functions of the extremal configurations.

We can prove that for a chained form system, a local planner using sinusoidal inputs respects the topological property TP ([SL96]).

6.3 Obtaining the initial paths for S_0

We have now all the ingredients required for transforming a S_i-path to a S_{i+1}-path, for any $i \geq 0$. This is however not enough. We must also have means for obtaining the initial S_0-paths, that is, paths for car-like robots towing a number of holonomic trailers. Path planning for such robots is however not a very challenging problem. One could first compute a fully holonomic path with, for example, a potential field method, and subsequently transform this path into a S_0-path with one of the transformation steps described in this paper.

In the planner that we implemented, and for which we will give experimental results in the next section, we however chose another solution. Given a particular scene, we construct a sufficient S_0-roadmap (that is, a roadmap storing paths feasible for system S_0) with *PPP*. As also described in Section 5.2, *PPP* builds a roadmap incrementally by probabilistically generating free configurations, and interconnecting these by a local planner, where possible. The local planner l_0 (based on *RTR*), as described in Section 6.2, is used. Given a constructed roadmap, solving a particular path planning problem (s, g) is done by connecting both the start configuration s and goal configuration g to the same connected component of the roadmap, with, for example, again the local planner. Of course, one may fail to compute such connections. In such a case, either there is no solution to the problem (that is, s and g lie in separate components of the free configuration space), or the roadmap is not yet sufficient, and has to be further extended.

An advantage of using *PPP* in the initial phase is that a roadmap is constructed only once, and paths can subsequently be retrieved very quickly, while for example a potential field method would require expensive computations to be performed each time a new (S_0-)problem is to be solved.

However, most time will typically be spent during the transformation phases, and hence we do not consider the use of *PPP* in the initial phase as a crucial ingredient of our nonholonomic planning scheme.

6.4 Smoothing the intermediate and final paths

We have not yet said anything about the *quality* of the paths generated by the different steps in our algorithm, and neither about the influence that the quality of a S_i-path has on its transformation to a S_{i+1}-path. For this cause, we should define what the quality of a path is. Unfortunately, it is not clear how to define such a measure formally.

One could say : the shorter a path, the better its quality. This is however problematic. Computing shortest paths for car-like robots (and cars towing trailers) amidst polygonal obstacles is still an open problem. Hence, one can compare two equivalent paths and determine which one is better, but since one does not know the length of the shortest path, it is impossible to determine the actual quality of a path. So if we apply some heuristic path shortening algorithm to paths that we find too long, we have no sensible stop criterion. Other intuitive quality measures are the number of reversals a path contains (the less, the better), and the minimal clearance of a path with the obstacles (the greater, the better). Again however we encounter the difficulties sketched for the length-measure, and, moreover, the measures contradict each other. For example, a shortest path typically touches obstacles, and a reversal-less path can be "unnecessarily" long.

For the initial and intermediate transformation steps however, we do not really have to worry about these considerations. We are merely interested in obtaining paths whose transformations are easy, that is, fast. Firstly, we recall that the *PL* method requires input-paths of a certain non-zero clearance for completeness. Furthermore, from experiments we noticed that the length of a path is of very large influence on the time-efficiency of its transformation. The shorter a path, the faster its transformation (on the average). However, there is another factor that plays an important role. Given a S_i-path P_i that is to be transformed to a S_{i+1}-path P_{i+1}, it appears that one can measure the extent to which P_i violates the $(i+1)$-*th* nonholonomic constraint. The more severe this violation is, the longer the transformation will take (again, on the average).

So, during each (initial and intermediate) step, we want generate paths that are short and do not violate severely the "next" nonholonomic constraint. Also, if the *PL* method is used, we must guarantee a non-zero clearance. The latter is quite easy (See also Section 7), so we now concentrate on the two first criterions.

Both presented transformation algorithms are actually very bad with respect to the two criterions. The generated paths are typically very long, up a hundred times longer then necessary, and they can hardly be used directly. Also, the transformation algorithms in no way aim at generating paths that somewhat respect the not yet introduced nonholonomic constraints. Fortunately, it appears to be quite easy to heuristically improve the paths sufficiently in order to make their transformations possible (and fast). First, in Section 6.4.1, we introduce a general heuristic *smoothing* technique for reducing path lengths. Then, in Section 6.4.2, we define the mentioned (tractor-trailer specific) "nonholonomy-violation" measure, and we propose an algorithm that improves paths with respect to this measure. This algorithm is again heuristic, and conceptually quite similar to the path-shortening one.

Regarding the final transformation, not much can be said in general about smoothing, for the reasons mentioned earlier. In the experiments that we present in the next section, we simply smoothed the paths (using path-shortening) until they intuitively looked nice (in our opinion), and their lengths seemed reasonable, with respect to the problems they solve.

6.4.1 Probabilistic path shortening

Let P_i be a S_i-path, and L_i be a local planner for the system S_i. Reducing the length of P_i can be done as follows :

> **loop** until ...
> Let Q be a *random* path segment of P_i, with start-configuration s and end-configuration e.
> Let Q_L be $L_i(s, e)$.
> **if** Q_L is *collision-free* and $length(Q_L) < length(Q)$
> **then** replace Q by Q_L in P_i.

This simple smoothing algorithm, that we refer to as *probabilistic path shortening*, works very well in practice. A problem however is the stop-criterion, as explained before. In our experiments, we simply run the algorithm until it stops making (significant) progress. See again Section 7.1 for experimental results.

6.4.2 Geometric approximation of nonholonomic constraints

Given an arbitrary (not necessarily feasible) path for a tractor with n trailers, one can measure in how far the i-th nonholonomic constraint of system S_n is violated. In Section 6.1 the nonholonomic constraints of a tractor-trailer robot have been described mathematically. However, using the terminology introduced in Section 6.1 (and in Figure 4), the nonholonomic constraints can also be characterised geometrically (See also Figure 6). During any motion of the robot :

1. the angle that F_0's velocity vector can make with Ax_0 is bounded, in absolute value, by a constant, i.e., the tractors *maximal steering angle* Φ_{max}, and

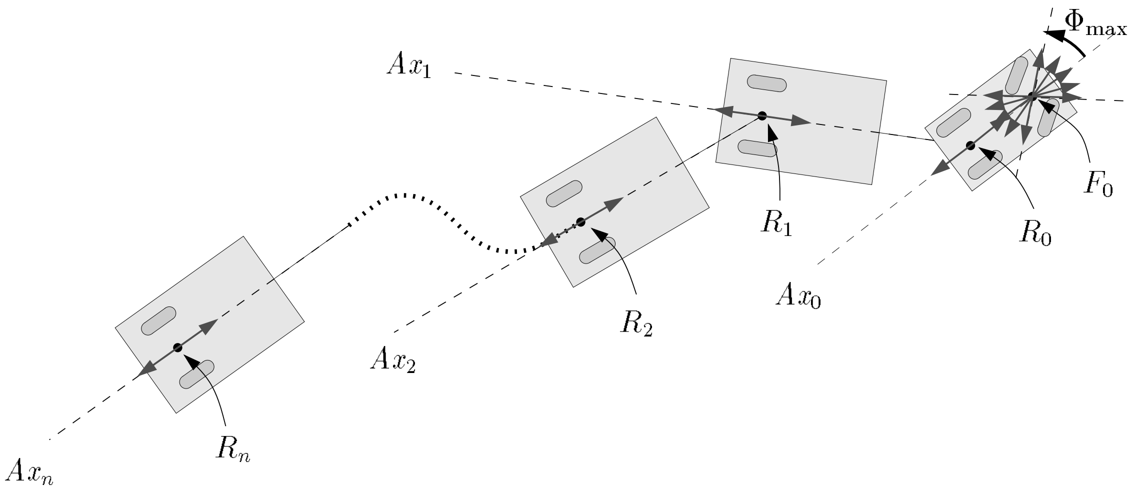


Figure 6: A tractor-trailer robot, with grey arrows indicating the directions of velocity that the front and rear points are allowed to have, in order for the robot not to violate any of its nonholonomic constraints.

2. for each $j \in \{0, \ldots, n\}$ R_j's velocity vector must lie on Ax_j.

Definition 1 *Let c_1 and c_2 be configurations of a tractor with n trailers, and let D denote the Euclidean distance in $\mathcal{R}^2$. For $i \in \{0, \ldots, n\}$, the* i-violation of (c_1, c_2), *denoted by $Vl_i(c_1, c_2)$, is defined as follows (See also Figure 7) :*

$$Vl_i(c_1, c_2) = D(\mathrm{Ax}_i(c_1), R_i(c_2))$$

If a S_n-robot performs a (very) short (and feasible) motion from a configuration c_1 and c_2 then, for each $j \in \{1, \ldots, n\}$, $\mathrm{Vl}_j(c_1, c_2)$ will be approximately 0. Now let P be a (not necessarily feasible) path for a tractor with trailers, parameterised by $t \in [0, 1]$, and let $\overline{P}_\epsilon$ be an ϵ-discretisation of P. That is, $\overline{P}_\epsilon$ is an ordered list of configurations $[c_1, \ldots, c_k]$ with $c_1 = P(0)$ and $c_k = P(1)$, such that no two consecutive configurations lie more that a distance ϵ apart (in configuration space).

If ϵ is chosen small and the i-th nonholonomic constraint is not violated in path P, then for any pair of consecutive configurations (c_j, c_{j+1}) in $\overline{P}_\epsilon$, $Vl_i(c_j, c_{j+1})$ will be approximately zero. If however P does violate the i-th nonholonomic constraint, then $\overline{P}_\epsilon$ will contain some pairs (c_j, c_{j+1}) with $Vl_i(c_j, c_{j+1})$ non-zero. Moreover, the more R_i's velocity vector deflects from Ax_i, the higher will be the value of Vl_i, for configuration pairs in the $\overline{P}_\epsilon$-segments where this violation occurs. This all follows from the geometric characterisation of the nonholonomic constraints given above.

Hence, it is sensible to define the extent to which the i-th nonholonomic constraint is violated by a path P in terms of the i-violation function Vl_i applied to consecutive pairs of configurations in $\overline{P}_\epsilon$. We chose for the following definition :

Definition 2 *Let P be a (not necessarily) feasible path for a tractor pulling n-trailers, and let $\overline{P}_\epsilon = [c_1, \ldots, c_k]$ be a ϵ-discretisation P. For $i \in \{0, \ldots, n\}$, the* i-violation of $\overline{P}_\epsilon$, *denoted by $\mathrm{Vl}_i(\overline{P}_\epsilon)$, is defined as follows :*

$$\mathrm{Vl}_i(\overline{P}_\epsilon) = \sum_{j \in \{1, \ldots, k-1\}} \mathrm{Vl}_i(c_j, c_{j+1})$$

So Definition 2 gives us means for measuring the violation of the i-th nonholonomic constraint in a given path, and this is exactly what we need. I.e., when a path P_{i-1} (for system S_{i-1}) is

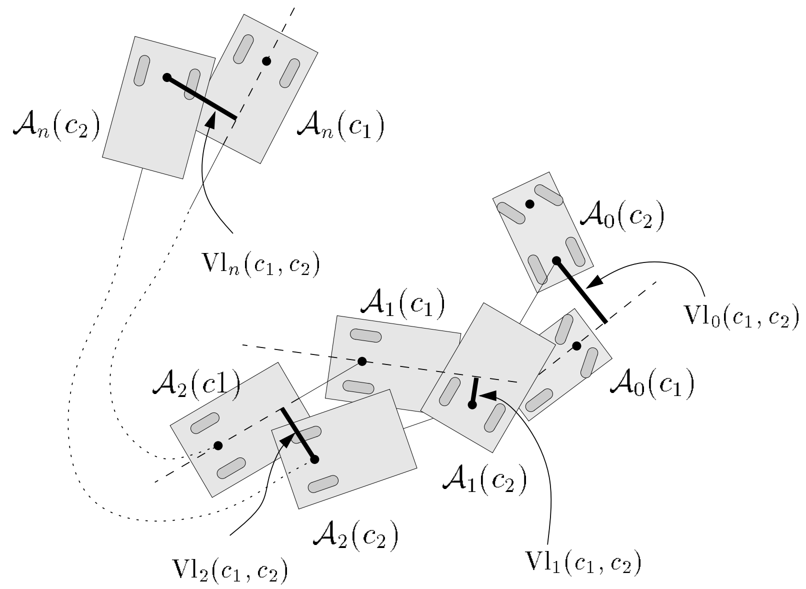


Figure 7: A tractor with n trailers is placed at two configurations c_1 and c_2. The i-violations of the configuration pair, defined as the distances between main axes $Ax_i(c_1)$ and rear points $R_i(c_2)$, are shown by thick line segments.

to be transformed to a path P_i (for system S_i), it is the i-th nonholonomic constraint that is to be introduced. The more this constraint is already respected in path P_{i-1}, the easier (and hence faster) will the transformation be. Hence, when a path P_{i-1} has been yielded by a transformation, one should try to reduce its i-violation, before passing it on to the next transformation step.

We again do this with a probabilistic algorithm, similar to *probabilistic path shortening*. However, instead of trying to replace random path segments by shorter ones computed by the local planner, we now try to replace the path segments by alterations of themselves that are of lower i-violation. Both the path segments as the alterations are generated probabilistically. We will refer to this algorithm, described bellow, as *geometric NH-approximation*.

Let P_{i-1} be the path for which the i-th nonholonomic constraint is to be approximated.
loop until ...
  Let Q be a *random* path segment of P_{i-1}.
  Let $\tilde{Q}$ be an *i-alteration* of Q.
  if $\tilde{Q}$ is *collision-free* and $Vl_i(\overline{\tilde{Q}_\epsilon}) < Vl_i(\overline{Q}_\epsilon)$ **then** replace Q by $\tilde{Q}$ in P_{i-1}.

The i-alterations, aimed at modifying the path (slightly) with respect to orientation of the i-th trailer, can, for example, be generated as follows :

Let $(x, y, \Phi, \theta_0, \ldots, \theta_n) \in \mathcal{C}^{[0\ldots1]}$ be the path (segment) to be "i-altered".
Let $\delta \in]0, \frac{1}{2}\pi[$ be a (experimentally) chosen constant.
Let $\tilde{\theta} = Random[-\delta, \delta]$.
Replace $\theta_i(t)$ by $\theta_i(t) + \sin(\pi t) \cdot \tilde{\theta}$ (for all $t \in [0, 1]$).

This simple method works fine, but other alterations are of course also possible, as long as the keep they path continuous.

In the next section we will present experimental results that show the positive influence of *geometric NH-approximation* on the transformation steps, and, through this, on the overall performance of our multi-level path planner.

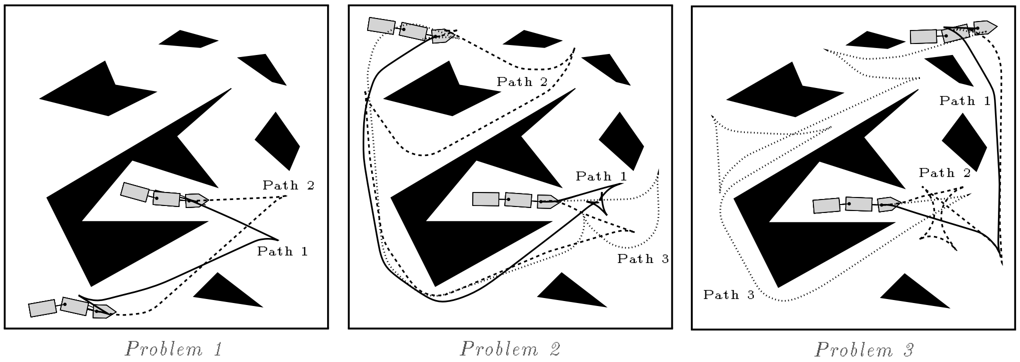


Figure 8: S_0-paths solving the problems 1, 2, and 3 in Scene 1.

7 An implementation of the algorithm and experimental results

We now describe an implementation of the multi-level algorithm for tractor-trailer robots with 2 trailers. The initial S_0-paths are obtained, as explained, by retrieval from a roadmap computed with *PPP*. For the $S_0 \to S_1$ transformation step we use *tube-PPP*, while for the $S_1 \to S_2$ transformation step we use the *PL* method. Because the *PL* method requires a non-zero clearance of its input-paths, we use a slightly grown robot (by a factor of 0.035) in the initial (*PPP*) step and the first (*tube-PPP*) transformation step. We based these choices on experiments. For the $S_0 \to S_1$ step both transformation algorithms seem to be of approximately the same speed, but the quality of the paths is better with *tube-PPP*. For the second step however, the *PL* method is much faster. We apply *probabilistic path shortening* to the intermediate paths. Furthermore, we have also implemented the *geometric NH-approximation* algorithm, and we will give experimental results for both cases where it is used, as for cases where it is not. Finally, we also give experimental results for the case where the $S_0 \to S_2$ transformation is performed in one single step (with the *PL* method). From these results it follows that a large gain is obtained by doing the transformations separately, according to the multi-level scheme that we propose in this paper.

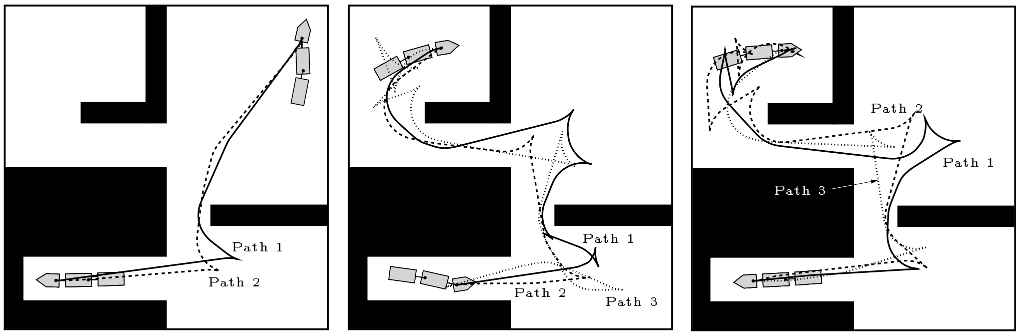
We have performed experiments with our planner in 2 different environments, which we refer to as *Scene 1* and *Scene 2*. For both scenes, we have defined 3 different path planning problems (that is, 3 pairs of start and goal configurations), and we have measured the performance of our planner for solving these problems. We are mainly interested in the performance of the transformation steps (this is the difficult part). For this reason, we take, for each problem, a number of different S_0-paths solving it, and we measure the performance of the transformation steps for each of the (equivalent) S_0-paths. We will refer to these S_0-paths, as the *initial test paths*. Scenes 1 and 2 are shown in Figures 8 and 9, together with the initial test paths that we use. In Figures 10 and 11 we see, for each problem, a S_0-path and a (smoothed) S_2-path to which it has been transformed by the multi-level planner. Figure 12 shows a 3D visualisation of a path, feasible for system S_2, computed by our multi-level planner.

The set of equivalent (but different) S_0-paths is obtained by retrieval from independently constructed roadmaps by *PPP*. We think that the set of S_0-paths that we present for each problem is fairly representative for paths that *PPP*, in general, generates for S_0-robots.

7.1 Experimental results

All experiments have been performed on a Silicon Graphics Indigo^2 workstation rated with 96.5 SPECfp92 and 90.4 SPECint92 (136 MIPS).

For both scenes we have first constructed a sufficient S_0-roadmap with *PPP*. For Scene 1 this



Problem 1 *Problem 2* *Problem 3*

Figure 9: S_0-paths solving the problems 1, 2, and 3 in Scene 2.

took about 40 seconds, and the resulting roadmap had 550 nodes. For Scene 2 about 30 seconds where required, resulting in a roadmap with 650 nodes.

7.1.1 Results for the multi-level algorithm with *geometric NH-approximation*

First we give results for the *multi-level planner*, using *geometric NH-approximation*. For each initial test path $\mathcal{P}_0$, we measured the computation times required for :

1. Retrieving and smoothing $\mathcal{P}_0$ from the S_0-roadmap.
2. Transforming $\mathcal{P}_0$ to a S_1-path $\tilde{\mathcal{P}}_1$, with *tube-PPP*. Since *tube-PPP* is probabilistic, we give *averages* over 20 independent runs.
3. Transforming $\tilde{\mathcal{P}}_1$ to a "smooth" path $\overline{\mathcal{P}}_1$, with *probabilistic path shortening*. The algorithm was iterated until the point where no longer any significant length reduction was achieved.
4. Transforming $\overline{\mathcal{P}}_1$ to a path $\mathcal{P}_1$, respecting more the second trailers nonholonomic constraint, with *geometric NH-approximation*. We executed this algorithm for prespecified amounts of time.
5. Transforming $\mathcal{P}_1$ to a S_2-path $\tilde{\mathcal{P}}_2$, with the *PL* method.
6. Transforming $\tilde{\mathcal{P}}_2$ to a "smooth" path $\mathcal{P}_2$, with *probabilistic path shortening*. Again, the algorithm was iterated until the point where no longer any significant length reduction was achieved.

The results are presented in Tables 1 and 2. Each problem has its own subtable. Within each such subtable, a row gives the computation times for the corresponding initial test path, except for the last row, which gives average values over the different initial test paths. Furthermore, the most right column gives the total required computation times. The values between square brackets give the *lengths*[3] of the corresponding paths.

[3] We measure the distance that the tractors rear point travels, considering the scene to be (tightly) bounded by a unit square.

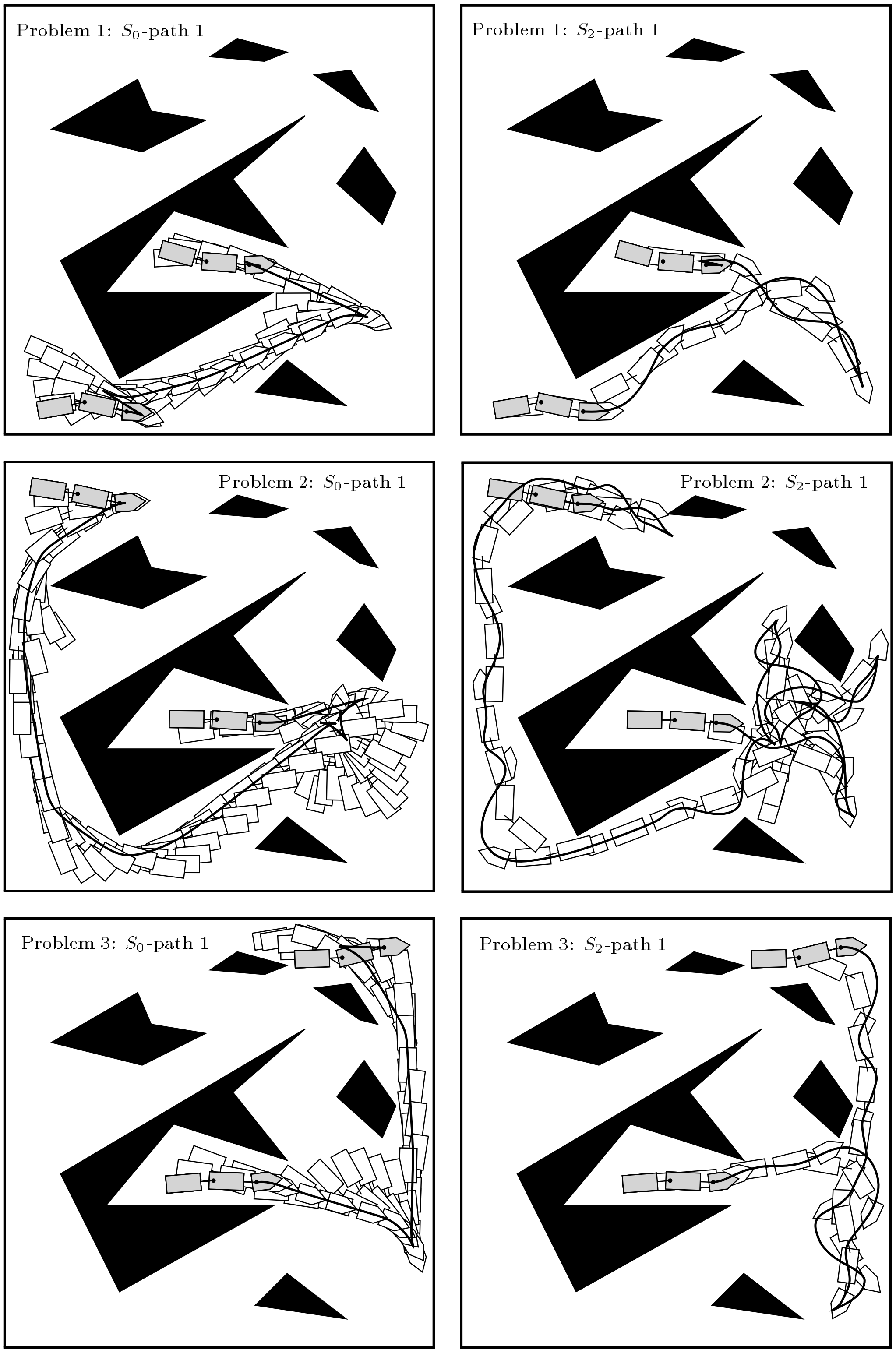


Figure 10: For each problem in Scene 1, a S_0-path and a (corresponding) S_2-path solving the problem is shown.

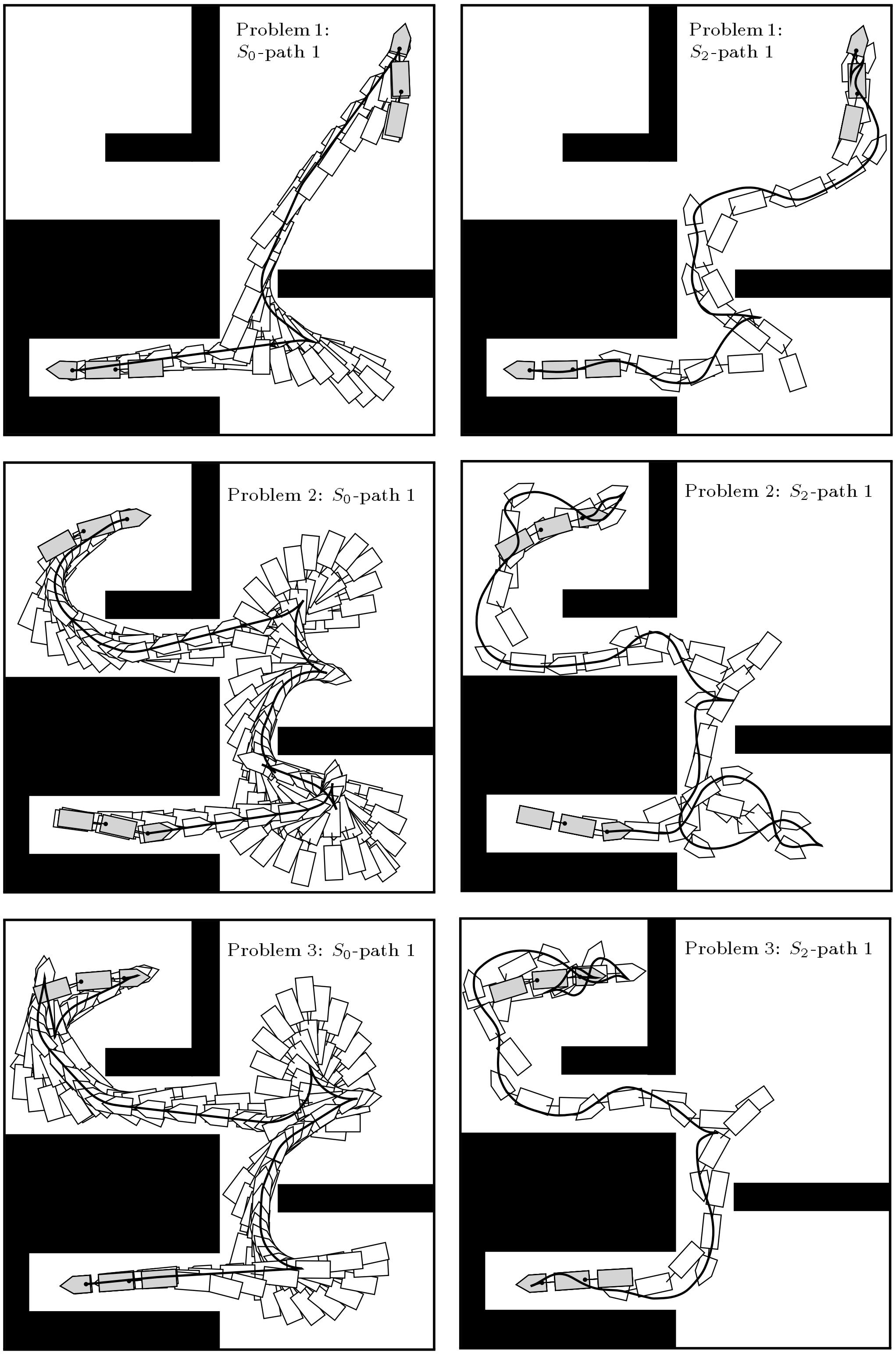


Figure 11: For each problem in Scene 2, a S_0-path and a (corresponding) S_2-path solving the problem is shown.

Figure 12: A 3D visualisation of a S_2-path, computed by our planner. It is a transformation of Path 2 solving Problem 2 in Scene 1 (See Figure 8).

Table 1 :

Scene 1	$\rightarrow \mathcal{P}_0$	$\xrightarrow{P} \tilde{\mathcal{P}}_1$	$\xrightarrow{S_P} \overline{\mathcal{P}}_1$	$\xrightarrow{S_G} \mathcal{P}_1$	$\xrightarrow{L} \tilde{\mathcal{P}}_2$	$\xrightarrow{S_P} \mathcal{P}_2$	Total
Problem 1							
Path 1	0.3	1.9	0.9	4.3	0.8 [8.6]	1.7 [1.7]	9.9
Path 2	0.3	0.6	0.2	4.3	3.9 [28.8]	19.4 [1.7]	28.7
Avg.	0.3	1.3	0.6	4.3	2.3 [18.7]	10.5 [1.7]	19.3
Problem 2							
Path 1	2.4	7.4	2.2	6.5	13.7 [88.3]	129.0 [7.4]	161.0
Path 2	1.8	6.7	4.3	6.5	14.1 [87.4]	64.5 [4.9]	97.8
Path 3	1.9	3.7	2.2	6.5	9.9 [63.2]	25.8 [3.8]	49.9
Avg.	2.0	5.9	2.9	6.5	12.6 [79.6]	73.1 [5.4]	103.0
Problem 3							
Path 1	1.5	2.6	2.2	6.5	7.1 [61.1]	51.6 [4.8]	71.4
Path 2	1.2	8.2	4.3	6.5	3.1 [15.4]	8.6 [3.3]	31.8
Path 3	1.5	34.1	4.3	6.5	18.1 [128.8]	322.5 [7.8]	386.9
Avg.	1.4	15.0	3.6	6.5	9.4 [68.4]	127.6 [5.3]	163.4

Table 2 :

Scene 2	$\rightarrow \mathcal{P}_0$	$\xrightarrow{P} \tilde{\mathcal{P}}_1$	$\xrightarrow{S_P} \overline{\mathcal{P}}_1$	$\xrightarrow{S_G} \mathcal{P}_1$	$\xrightarrow{L} \tilde{\mathcal{P}}_2$	$\xrightarrow{S_P} \mathcal{P}_2$	Total
Problem 1							
Path 1	0.3	0.6	0.4	4.3	0.9 [4.5]	2.2 [2.0]	8.6
Path 2	0.5	1.4	0.4	4.3	0.5 [3.0]	2.2 [2.0]	9.2
Avg.	0.4	1.0	0.4	4.3	0.7 [3.8]	2.2 [2.0]	9.0
Problem 2							
Path 1	1.4	34.4	8.6	6.5	7.3 [52.4]	38.7 [6.5]	96.8
Path 2	0.5	1.0	2.2	6.5	2.9 [17.5]	8.6 [6.3]	21.6
Path 3	1.3	9.7	4.3	6.5	13.2 [100.1]	387.0 [6.5]	422.0
Avg.	1.1	15.1	5.0	6.5	7.8 [56.6]	144.8 [6.4]	180.1
Problem 3							
Path 1	0.9	1.4	2.2	6.5	8.5 [69.6]	25.8 [3.7]	45.2
Path 2	0.9	38.6	4.3	6.5	25.9 [268.1]	645.0 [5.8]	721.2
Path 3	0.5	5.3	2.2	6.5	4.4 [34.9]	17.2 [4.0]	36.0
Avg.	0.8	15.1	2.9	6.5	12.9 [99.3]	229.3 [4.5]	267.5

We see that the total computation times are on the order of seconds for the two easy problems (Problem 1 in Scene 1 and Scene 2), and on the order of a few minutes for the other, more difficult, problems. When we look at the distribution of the computation times over the different steps of our algorithm, we see that most time is spent on the second transformation and, especially, the final smoothing step. Hence, it appears to be much easier to obtain an arbitrary S_2-path than one which is (intuitively) nice.

When we take into consideration the path lengths of the intermediate paths, we see very clearly that the length of the non-smoothed S_2-path $\tilde{\mathcal{P}}_2$ is of great influence on the final smoothing step. Also, not surprisingly, we see a strong correlation between the computation time of a (non-smoothed) S_2-path, and its length.

7.1.2 Other results

We have also done experiments with variations of the multi-level algorithm. Below, we give results obtained by (1) omitting *geometric NH-approximation*, and (2) skipping the S_1-level. We have not managed to smooth sufficiently all S_2-paths generated by the two-level planner (that is, the one without the S_1-level). We therefore do not give results for the final smoothing phase of this planner (the averages would be meaningless).

Without *geometric NH-approximation* **:**

Scene 1	$\to \mathcal{P}_0$	$\xrightarrow{P} \tilde{\mathcal{P}}_1$	$\xrightarrow{S_P} \mathcal{P}_1$	$\xrightarrow{L} \tilde{\mathcal{P}}_2$	$\xrightarrow{S_P} \mathcal{P}_2$	Total
Problem 1						
Path 1	0.3	1.9	0.9	1.9 [13.6]	2.2 [1.8]	7.2
Path 2	0.3	0.6	0.2	3.2 [24.4]	8.6 [1.9]	12.9
Avg.	0.3	1.3	0.6	2.6 [18.9]	5.4 [1.9]	10.1
Problem 2						
Path 1	2.4	7.4	2.2	29.2 [180.8]	150.5 [5.1]	191.6
Path 2	1.8	6.7	4.3	12.3 [87.5]	38.7 [4.3]	63.7
Path 3	1.9	3.7	2.2	20.3 [151.5]	38.7 [3.0]	66.7
Avg.	2.0	5.9	2.9	20.6 [139.9]	76.0 [4.0]	107.4
Problem 3						
Path 1	1.5	2.6	2.2	37.2 [331.4]	1032.0 [4.9]	1075.5
Path 2	1.2	8.2	4.3	11.7 [119.9]	51.6 [3.6]	77.0
Path 3	1.5	34.1	4.3	28.7 [210.1]	361.2 [8.1]	429.8
Avg.	1.4	14.9	3.6	25.9 [220.5]	481.6 [5.5]	527.4

Scene 2	$\to \mathcal{P}_0$	$\xrightarrow{P} \tilde{\mathcal{P}}_1$	$\xrightarrow{S_P} \mathcal{P}_1$	$\xrightarrow{L} \tilde{\mathcal{P}}_2$	$\xrightarrow{S_P} \mathcal{P}_2$	Total
Problem 1						
Path 1	0.3	0.6	0.4	2.2 [15.2]	6.5 [1.9]	10.0
Path 2	0.5	1.4	0.4	4.0 [34.0]	8.6 [1.8]	14.9
Avg.	0.4	1.0	0.4	3.1 [24.6]	7.5 [1.9]	12.5
Problem 2						
Path 1	1.4	34.4	8.6	51.7 [354.1]	141.9 [4.7]	238.0
Path 2	0.5	1.0	2.2	7.3 [55.9]	17.2 [6.0]	28.1
Path 3	1.3	9.7	4.3	27.0 [269.2]	774.0 [7.5]	816.4
Avg.	1.1	15.1	5.0	28.6 [226.4]	311.0 [6.1]	360.8
Problem 3						
Path 1	0.9	1.4	2.2	35.9 [311.0]	129.0 [3.3]	169.4
Path 2	0.9	38.6	4.3	32.3 [355.6]	1032.0 [7.1]	1108.1
Path 3	0.5	5.3	2.2	8.8 [86.0]	25.8 [3.6]	42.6
Avg.	0.8	15.1	2.9	25.7 [250.9]	395.6 [4.7]	440.0

Direct $S_0 \rightarrow S_2$ transformations :

Scene 1	$\mathcal{P}_0 \xrightarrow{L} \tilde{\mathcal{P}}_2$

Problem 1	
Path 1	21.6 [231.4]
Path 2	5.3 [40.5]
Avg.	13.5 [136.0]

Problem 2	
Path 1	77.8 [394.7]
Path 2	66.3 [408.3]
Path 3	72.0 [463.2]
Avg.	72.0 [422.1]

Problem 3	
Path 1	30.9 [208.7]
Path 2	53.8 [393.6]
Path 3	264.7 [875.1]
Avg.	92.6 [492.5]

Scene 2	$\mathcal{P}_0 \xrightarrow{L} \tilde{\mathcal{P}}_2$

Problem 1	
Path 1	2.1 [12.2]
Path 2	4.6 [39.0]
Avg.	3.4 [25.6]

Problem 2	
Path 1	335.5 [1266.7]
Path 2	5.2 [45.4]
Path 3	61.3 [577.3]
Avg.	134.0 [629.8]

Problem 3	
Path 1	11.1 [95.3]
Path 2	197.8 [1066.3]
Path 3	28.4 [197.8]
Avg.	79.1 [453.1]

7.1.3 Comparison of the results

In order to have a comprehensive overview of the experimental data, we plot the average computation times for each of the 6 presented problems in separate charts (Figure 13). The (average) *cumulative* computation time is set against the computation phase of the algorithm. Again because of the smoothing problems for some of the S_2-paths obtained by the two-level planner, the dotted plots stop at the $\tilde{\mathcal{P}}_2$ point.

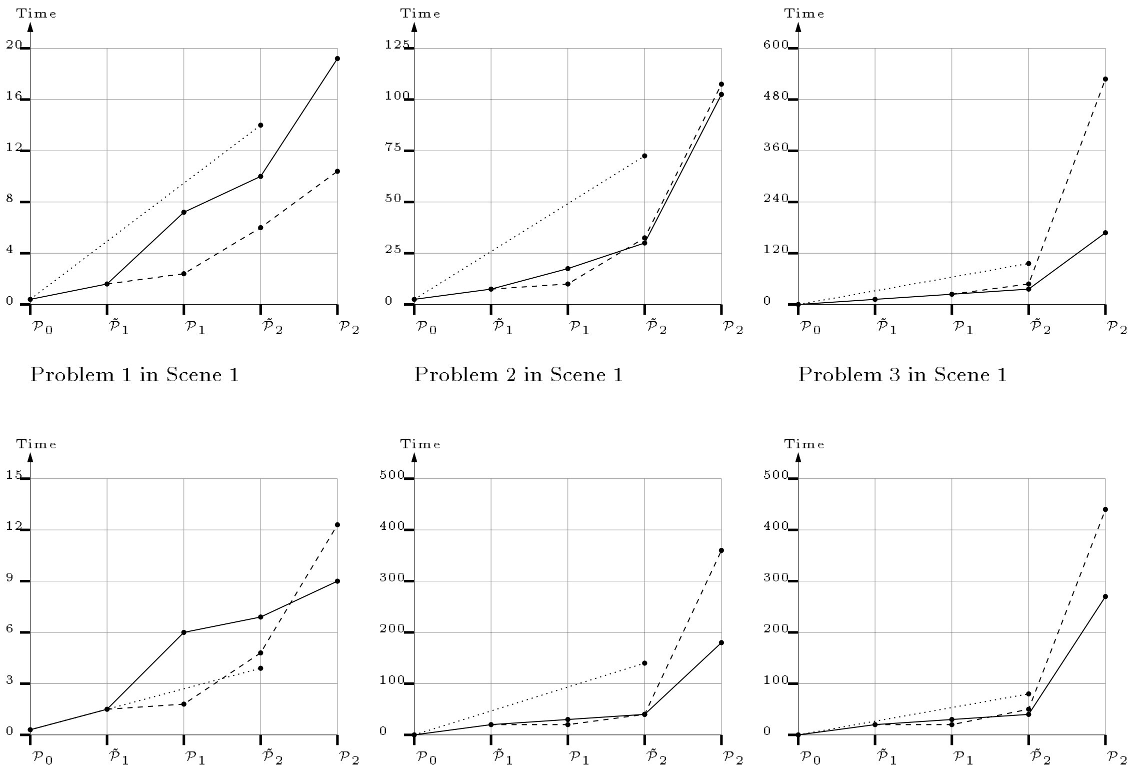
For the simple problems, that is, the problems 1, we do not see much structure. All three algorithms seem to perform well. For the other, more difficult, problems we do see significant differences. Over the whole, the *multi-level* algorithm with *geometric NH-approximation* is clearly the winner, while the *two-level* algorithm scores worst. Moreover, if we recall that most computation time is spent in the final smoothing phase and strongly depends on the length of the non-smoothed S_2-path $\tilde{\mathcal{P}}_2$, it is clear that the two-level algorithm performs very poorly indeed.

We have performed experiments with another two-level planner as well, namely one that transforms fully holonomic paths (that is, paths that also disrespect the tractors nonholonomic constraints) directly to S_2-paths, using the *PL* method. As could be expected, the results where considerably worse than even those for the $S_0 \rightarrow S_2$ planner. For sake of brevity, we do not include these results in our paper.

Summarising the experimental results, we come to the following conclusions. Firstly, over the whole the *multi-level* algorithm (with and without *geometric NH-approximation*) performs better, with respect to computation times and path qualities, than just *two-level* planning. Secondly, even though the *geometric NH-approximation* algorithm consumes some extra time, use of it does not severely penalise the overall performance in easy cases, while clearly improving the performance in more difficult ones.

8 Conclusions and future work

We have presented a new *multi-level* approach to path planning for nonholonomic robots amidst static obstacles. At the first level an initial path is generated that respects only some (easy) nonholonomic constraint(s). Then, at each subsequent level, this path is transformed to a more feasible path, that is, to a path that respects more nonholonomic constraints. The final path is fully feasible for the real robot. Two general transformation algorithms (*PL* and *tube-PPP*)



Problem 1 in Scene 1 Problem 2 in Scene 1 Problem 3 in Scene 1

Problem 1 in Scene 2 Problem 2 in Scene 2 Problem 3 in Scene 2

Figure 13: The average performances of the different algorithm settings for Scene 1 and Scene 2. The solid polylines correspond to the *multi-level* planner with *geometric NH-approximation*, the dashed polylines to the *multi-level* planner without *geometric NH-approximation*, and the dotted polylines to the *two-level* planner.

are described, as well means for obtaining initial paths (*PPP*). Also, techniques are presented for heuristically smoothing the intermediate and final paths (*probabilistic path shortening* and *geometric NH-approximation*).

The multi-level scheme is very general. It is applicable to any nonholonomic robot, provided that the robot is locally controllable, and that one has local planners for the corresponding subsystems. In order to obtain completeness, these local planners must respect a basic topological property.

We have applied the multi-level scheme to *tractor-trailer* robots (with two trailers). The resulting planner is *complete*, and, as experimental results show, quite *time-efficient*. Problems involving a tractor with two trailers in realistic environments are solved on the order of, at most, a few minutes. The experimental results also show that the multi-level algorithm performs significantly better than just two-level planning (that is, direct transformation of an initial path to a feasible one). Furthermore, use of the tractor-trailer specific *geometric NH-approximation* algorithm, that heuristically reduces violations of the nonholonomic constraints, gives further improvement.

There remain various open questions and topics of future research. For example, it is not clear in general what should be the order in which the nonholonomic constraints are to be introduced. For tractor-trailer robots, we made this choice just intuitively. Also, from the experimental results we can observe that the running times of our algorithm vary considerably over different initial paths solving the same problem. Therefore, in our opinion, it is an important topic of future research to formulate criteria for initial paths, that indicate the difficulty of transforming these initial paths to feasible ones. Such criteria could be used for guiding the generation of the initial paths. Also, better stop-criteria for the smoothing algorithms would be desirable.

Finally, we want to mention the possibility of *multi-level roadmap generation*. Instead of taking an initial *path*, and, in a number of steps, transforming it to a feasible path, one could also take an initial *roadmap*, and perform the transformations on this whole roadmap. One then ends up with a fully feasible roadmap, that is, a roadmap from which paths, respecting all nonholonomic constraints of the robot, can directly be retrieved. Of course, the total computation times will, in non-trivial scenes, be very high (on the order of hours or days). However, if one is dealing with a truly static environment, an algorithm that within a few hours or days computes a roadmap from which fully feasible (and smoothed) paths are directly retrievable can be favourable to one that requires no preprocessing, but takes a few minutes to compute each single path.

Acknowledgements

We are very grateful to Geert-Jan Giezeman, for helping us with the software. All basic geometric operations have been performed by routines contained in the Plageo library ([Gie93]), and also the software for our 3D path visualisation is due to Geert-Jan.

References

[BL93] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[BLC93] A. Bellaïche, J.P. Laumond, and M. Chyba. In *Proc. 32nd IEEE Conf. on Decision and Control*, December 1993.

[Fer] P. Ferbach. Technical report, Electricité de France. SDM Dept., Chatou, France.

[FGL93] C. Fernandes, L. Gurvits, and Z.X. Li. Optimal nonholonomic motion planning for a falling cat. In Z. Li and J.F. Canny, editors, *Nonholonomic Motion Planning*, Boston, USA, 1993. Kluwer Academic Publishers.

[Gie93] G.-J. Giezeman. *PlaGeo—A Library for Planar Geometry.* Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, August 1993.

[KŠLO95] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *To appear in IEEE Trans. Robot. Autom.*, 1995.

[Lau93] J.-P. Laumond. Singularities and topological aspects in nonholonomic motion planning. In Zexiang Li and J.F. Canny, editors, *Nonholonomic Motion Planning*, pages 149–199. Kluwer Academic Publishers, 1993.

[LJTM94] Jean-Paul Laumond, Paul E. Jacobs, Michel Taïx, and Richard M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. Robot. Autom.*, 10(5), October 1994.

[LS90] G. Lafferriere and H. J. Sussmann. Motion planning for controllable systems without drift: A preliminary report. Technical Report SYSCON-90-04, Rutgers Center for Systems and Control, 1990.

[LSV94] J.P. Laumond, S. Sekhavat, and M. Vaisset. Collision-free motion planning for a nonholonomic mobile robot with trailers. In *4th IFAC Symp. on Robot Control*, pages 171–177, Capri, Italy, September 1994.

[Luz94] D. Luzeaux. Parking maneuvers and trajectory tracking. In *Proc. Int. Workshop on Advanced Motion Control*, Berkeley, CA, USA, 1994.

[MS90] R. Murray and S. Sastry. Steering nonholonomic systems using sinusoids. In *Proc. IEEE Conf. on Decision and Control*, pages 2097–2101, 1990.

[RFLM93] P. Rouchon, M. Fliess, J. Lévine, and P. Martin. Flatness and motion planning. In *Proc. European Control Conference*, pages 1518–1522, 1993.

[RS91] J.A. Reeds and R.A. Shepp. Optimal paths for a car that goes both forward and backward. *Pacific Journal of Mathematics*, 145(2):367–393, 1991.

[SL91] H.J. Sussmann and W. Liu. Limits of highly oscillatory controls and the approximation of general paths by admissible trajectories. Technical Report SYSCON-91-02, Rutgers Center for Systems and Control, February 1991.

[SL96] S. Sekhavat and J.P. Laumond. Topological property of trajectories computed from sinusoidal inputs for nonholonomic chained form systems. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, April 1996.

[ŠO95] P. Švestka and M. Overmars. Motion planning for car-like robots using a probabilistic learning approach. *To appear in Intern. Journal of Rob. Research*, 1995.

[Sor93] O.J. Sordalen. Conversion of a car with n trailers into a chained form. In *Proc. IEEE Journal of Robotics and Automation*, pages 1382–13874, 1993.

[SSB94] A. Sahai, M. Secor, and L. Bushnell. An obstacle avoidance algorithm for a car pulling many trailers with kingpin hitching. Technical Report UCB/ERL M94/10, University of California, Berkeley, CA, USA, March 1994.

[SŠLO95] S. Sekhavat, P. Švestka, J.P. Laumond, and M. H. Overmars. Probabilistic path planning for tractor-trailer robots. Technical Report 96007, LAAS-CNRS, Toulouse, France, 1995.

[ŠV95] P. Švestka and J. Vleugels. Exact motion planning for tractor-trailer robots. In *Proc. IEEE Internat. Conf. on Robotics and Automation*, pages 2445–2450, Nagoya, Japan, 1995.

[TLM+92] D. Tilbury, J.-P. Laumond, R. Murray, S. Sastry, and G. Walsh. Steering car-like systems with trailers using sinusoids. In *Proc. 1992 IEEE Int. Conf. Robotics and Automation*, pages 1993–1998, Nice, France, May 1992.

[TMS93] D. Tilbury, R. Murray, and S. Sastry. Trajectory generation for the n-trailer problem using goursat normal form. Technical Report UCB/ERL M93/12, Univ. California, Berkeley, CA, USA, February 1993.

Appendix

If we consider the particular case of the car-like robot pulling 2 trailers, we have already seen a kinematic model of the system in section 6.1. Another one can be :

$$\left\{ \begin{array}{l} \dot{x_2} = cos\theta_2 cos(\theta_1 - \theta_2) cos(\theta_0 - \theta_1) cos(\theta - \theta_0) v_0 \\ \dot{y_2} = sin\theta_2 cos(\theta_1 - \theta_2) cos(\theta_0 - \theta_1) cos(\theta - \theta_0) v_0 \\ \dot{\Phi} = \omega \\ \dot{\theta}_0 = \frac{1}{d_0} sin(\theta - \theta_0) v_0 \\ \dot{\theta}_1 = \frac{1}{d_1} sin(\theta_0 - \theta_1) cos(\theta - \theta_0) v_0 \\ \dot{\theta}_2 = \frac{1}{d_2} sin(\theta_1 - \theta_2) cos(\theta_0 - \theta_1) cos(\theta - \theta_0) v_0 \end{array} \right.$$

where (x_2, y_2) is the rear point R_2 of the second trailer, and Φ is the angle of the tractors front wheels, with respect to the x-axis (See also Figure 4). The inputs are v_0, the tangential velocity of the tractors front point, and ω, the angular velocity of the front wheels.

The following change of coordinates convert the control system above into chained form :

$$\left\{ \begin{array}{l} z_1 = x_2 \\ z_2 = \Bigg(\frac{(1+\tan^2(\theta_0-\theta_1))(1+\tan^2(-\theta_1+\theta_2))}{\cos(-\theta_1+\theta_2)} \\ \times \left(-\frac{\tan(-\Phi+\theta_0)}{d_0 \cos(-\theta_1+\theta_2)\cos(\theta_0-\theta_1)} - \frac{\tan(\theta_0-\theta_1)\, v3}{d_1 \cos(-\theta_1+\theta_2)} \right) \\ + 2\, \frac{\tan(\theta_0-\theta_1)\tan(-\theta_1+\theta_2)(1+\tan^2(-\theta_1+\theta_2))}{\cos(-\theta_1+\theta_2)} \left(-\frac{\tan(\theta_0-\theta_1)}{d_1 \cos(-\theta_1+\theta_2)} - \frac{\tan(-\theta_1+\theta_2)}{d_2} \right) \\ + \frac{\tan(\theta_0-\theta_1)(1+\tan^2(-\theta_1+\theta_2))\sin(-\theta_1+\theta_2)}{\cos^2(-\theta_1+\theta_2)} \left(-\frac{\tan(\theta_0-\theta_1)}{d_1 \cos(-\theta_1+\theta_2)} - \frac{\tan(-\theta_1+\theta_2)}{d_2} \right) \\ - 4\, \frac{\tan(\theta_0-\theta_1)(1+\tan^2(-\theta_1+\theta_2))\sin(\theta_2)\tan(-\theta_1+\theta_2)}{d_2 \cos(-\theta_1+\theta_2)\cos(\theta_2)} \\ + 6\, d_2 \tan(-\theta_1 + \theta_2)\tan(\theta_2)(1 + \tan^2(-\theta_1 + \theta_2)) \\ \times \left(-\frac{\tan(\theta_0-\theta_1)}{d_1 \cos(-\theta_1+\theta_2)} - \frac{\tan(-\theta_1+\theta_2)\, v3}{d_2} \right) \\ - 3\tan^3(-\theta_1 + \theta_2)\, (1 + \tan^2\theta_2) - 12\, \frac{\tan^3(-\theta_1+\theta_2)\tan(\theta_2)\sin(\theta_2)}{\cos\theta_2} \\ + d_2\, (1 + \tan^2(-\theta_1 + \theta_2))^2 \left(-\frac{\tan(\theta_0-\theta_1)}{d_1 \cos(-\theta_1+\theta_2)} - \frac{\tan(-\theta_1+\theta_2)}{d_2} \right) \\ + 2\, d_2 \tan^2(-\theta_1 + \theta_2)(1 + \tan^2(-\theta_1 + \theta_2)) \\ \times \left(-\frac{\tan(\theta_0-\theta_1)}{d_1 \cos(-\theta_1+\theta_2)} - \frac{\tan(-\theta_1+\theta_2)}{d_2} \right) \\ - 4\, \frac{\tan^2(-\theta_1+\theta_2)(1+\tan^2(-\theta_1+\theta_2))\sin(\theta_2)}{\cos\theta_2} \Bigg) /(d_1\, d_2 \cos^5\theta_2) \\ z_3 = \frac{1}{d_1\, d_2\, cos^4\theta_2} . \frac{tan(\theta_0-\theta_1)}{cos(\theta_1-\theta_2)} \times (1 + tan^2(\theta_1 - \theta_2)) \\ \quad + \frac{1}{d_2\, cos^4\theta_2} \times tan(\theta_1 - \theta_2)(3tan(\theta_1 - \theta_2)tan\theta_2 - (1 - tan^2(\theta_1 - \theta_2))) \\ z_4 = \frac{1}{d_2} . \frac{tan(\theta_1-\theta_2)}{cos^3\theta_2} \\ z_5 = tan\theta_2 \\ z_6 = y_2 \end{array} \right.$$