Real-Time Trajectory Generation for Car-like Vehicles Navigating Dynamic Environments

Vivien Delsart, Thierry Fraichard, Luis Martinez-Gomez

Real-Time Trajectory Generation for Car-like Vehicles Navigating Dynamic Environments

Vivien Delsart, Thierry Fraichard and Luis Martinez

***Abstract*— This paper presents `Tiji`, a trajectory generation scheme, *ie* an algorithm that computes a feasible trajectory between a start and a goal state, for a given robotic system. `Tiji` is geared towards complex dynamic systems subject to differential constraints, such as wheeled vehicles, and its efficiency warrants it can be used in real-time. Above all, `Tiji` is able to compute a trajectory that reaches the goal state at a prescribed final time in order to avoid collision with the moving objects of the environment. The method proposed, which relies upon a parametric trajectory representation, is variational in nature. The trajectory parameters are incrementally updated in order to optimize of a cost function involving the distance between the end of the trajectory computed and the (goal state, final time) pair. Should the goal state be unreachable (if the final time is ill-chosen), the method returns a trajectory that ends as close as possible to the (goal state, final time) pair, which can be useful in certain applications.**

***Index Terms*— Trajectory generation; Differential constraints; Dynamic environments.**

I. Introduction

A. Background and Motivations

Trajectory generation for a given robotic system is the problem of determining a feasible trajectory (that respects the system's dynamics) between an initial and a final state. So far, all existing trajectory generation approaches have been concerned with optimizing a feasible trajectory to the goal state (see §II). In some cases, the path length would be optimized, or the energy, or the travel time. It is interesting to note that, in no circumstances, people have tried to compute a trajectory reaching the goal state at a specific *time instant*. Yet, when a robotic system is placed in a dynamic environment, *ie* featuring moving objects, it becomes important not only to compute the trajectory reaching a goal state but also to reach this goal state at a certain time in order to avoid collision with the moving objects of the environment. This constraint, henceforth called *final time constraint*, seems novel in the trajectory generation field. Although, it is intrinsic to all navigation problems in dynamic environments.

When dealing with dynamic environments, another constraint immediately arises: the *decision time constraint*: dynamic environments impose a strict upper bound on the time available to compute a trajectory (the robotic system can be in danger solely by remaining passive). The decision time constraint is determined by the nature of the moving objects, the faster they go, the harsher the decision time is with a shorter time to compute a trajectory.

INRIA, CNRS-LIG & Grenoble University, France.

While the trajectory generation problem has been well studied in the literature, it remains complex to solve especially for systems subject to differential constraints (such as nonholonomic wheeled vehicles). The final time constraint further increases the complexity of the trajectory generation problem since time now becomes an additional dimension to the problem. Local controllability properties might tell us whether a given robotic system can or cannot reach a given state. They do not tell us however whether it can do so at a specified time. To answer this question (decidability issue), one has to consider the set of reachable states embedded in the state×time space of the system. For realistic systems, such analysis is too complex for real-time applications (*cf* [1], [2]).

B. Contributions

This paper presents a trajectory generation scheme called `Tiji` which integrates the final time constraint. `Tiji` is geared towards complex dynamic systems subject to differential constraints, such as car-like vehicles, and its efficiency warrants it can be used in real-time (thus meeting the decision time constraint). The approach is similar in spirit to that of [3] or [4] (*cf* Section II): a parametric trajectory representation is assumed in order to reduce the search space. An initial set of parameters is selected yielding a trajectory that does not necessarily reach the goal state. The parameter space is then searched and efficient numerical optimization is used to optimize a cost function involving the distance between the end of the trajectory computed and the (goal state, final time) pair. Should the goal state be unreachable (if the final time is ill-chosen), the method returns a trajectory that ends as close as possible to the (goal state, final time) pair, which can be useful in certain applications. `Tiji` differs from previous works first because it takes into account the final time constraint and also because the control definition chosen is such that it ensures that all trajectory constraints are met.

C. Outline of the Paper

Works related to the problem at hand are reviewed in Section II. The approach proposed is outlined in Section III. The case of a car-like vehicle is addressed in Section IV and simulation results are given in Section V. Conclusions are finally presented in Section VI.

II. Related Work

Many trajectory generation methods exist but they can roughly be classified into four categories. The first one is

the *primitive combination* category. The trajectories computed are geometric paths built with the concatenation of fixed geometric primitives. A popular one was proposed by Dubins [5]. It computes shortest paths for simplified car-like vehicles moving forward only, the paths comprise straight segments and circular arcs. Later Reeds and Shepp [6] addressed the case of a car moving forward and backward. More complex geometric primitives, like clothoids, have been also later considered [7], [8]. All these approaches compute geometric paths only. Path-velocity decomposition [9] could be used to equip the path with a velocity profile and thus attempting to solve the problem considered herein, *ie* meeting the final time constraint, but the nominal geometric path drastically reduces the solution space.

The second category of works attempts to solve a standard two-point boundary value problem. To simplify the problem at hand, the curve that will be used to connect the start and goal state is restricted to a specific type of curve, *eg* B-splines [10], quintic polynomials [11], or cubic spirals [12]. The main drawback with these approaches lies in the difficulty to obtain a curve satisfying internal constraints (such as an upper bound on the curvature).

The third category of works are variational approaches. An initial trajectory connecting the start and goal state (usually not feasible) is iteratively modified until it satisfies all the constraints of the robotic system considered. Such methods assume a parametric trajectory representation. The parameter space is searched and efficient numerical optimization is used to optimize a cost function defined over the trajectory, *eg* [13] where an initial linear spline is iteratively deformed.

The fourth and final category of works is similar to the previous one except for the fact that the initial trajectory is now feasible but does not connect the goal state. The initial trajectory is deformed through parameter optimization until the goal state is reached. For instance, Gallina [3] uses sums of harmonics to represent a trajectory and tries to reach the goal state by modifying the harmonics parameters, while Kelly et al. deals with simpler but faster polynomial spirals and triangular profiles [14].

Both the third and fourth categories achieve a high degree of generality and efficiency. However, sensitivity to the choice of the initial trajectory and convergence can be an issue in some cases.

III. Overview of the Approach

A. Notations and Definitions

Let $\mathcal{A}$ denote a robotic system operating in a workspace $\mathbf{W}$ ($\mathbb{R}^2$or $\mathbb{R}^3$). The dynamics of $\mathcal{A}$ is described by:

$$\dot{s} = f(s, \tilde{u}) \tag{1}$$

where $s \in \mathbf{S}$ is the state of $\mathcal{A}$, $\dot{s}$ its time derivative and $u \in \mathbf{U}$ a control. $\mathbf{S}$ and $\mathbf{U}$ respectively denote the the state space and the control space of $\mathcal{A}$. Let $\tilde{u} : [0, t_f[\longrightarrow \mathbf{U}$ denote a control trajectory, *ie* a time-sequence of controls. Starting from an initial state s_0 (at time 0) and under the action of a control trajectory $\tilde{u}$, the state of $\mathcal{A}$ at time t is denoted by $\tilde{u}(s_0, t)$. A couple $(s_0, \tilde{u})$ defines a state trajectory for $\mathcal{A}$, *ie* a curve in $\mathbf{S} \times \mathbf{T}$ where $\mathbf{T}$ denotes the time dimension. $\mathcal{A}$ is subject to a set of constraints over its control and state parameters:

$$\mathbf{h}(s, \tilde{u}) \leq 0 \tag{2}$$

B. Trajectory Generation by Constraints Optimization

Given two states s_0 and s_g, trajectory generation consists in finding the control trajectory $\tilde{u}$ to apply from state s_0 in order to reach the goal state s_g. A parametric representation of the control trajectory is assumed, in other words:

$$\tilde{u} = \tilde{u}(p) \tag{3}$$

where $p = (p_1, \ldots, p_k)$ is a vector of parameters. This parametrization reduces the search space to the set of trajectories defined by the parameter space considered. Although it is only a subset of all feasible trajectories, an appropriate parametrization usually suffices. Such a parametrization allows to express the dynamics as follows:

$$\dot{s} = f(s(p), \tilde{u}(p)) = f(p) \tag{4}$$

Trajectory generation by constraints optimization turns the initial problem, optimal control, to one of constraint optimization (also known as nonlinear programming): Let us consider a cost function $\mathbf{J}(s_0, \tilde{u})$ and the set of inequality constraints $\mathbf{h}$ of $\mathcal{A}$, constraint optimization is expressed as:

$$\begin{array}{ll} \text{minimize :} & \mathbf{J}(s_0, \tilde{u}) \\ \text{subject to :} & \mathbf{h}(s, \tilde{u}) \leq 0 \end{array} \tag{5}$$

Given an initial state s_0, constraint optimization for trajectory generation consists in updating the current set of parameters p in order to reduce the cost function $\mathbf{J}$. $\mathbf{J}$ is set as the distance from the goal state s_g (the state we want to reach) to the final state $s_f = \tilde{u}(s_0, t_f)$, *ie* the state reached by $\mathcal{A}$ from s_0 when applying the parametrized control trajectory $\tilde{u}$). The update of p must also guarantee that the resulting control trajectory remains feasible, *ie* satisfies (2).

C. Trajectory Generation Algorithm

Algorithm 1: `Tiji`

Input: s_0, s_g, t_g
Output: p, `success flag`

$i = 0$;
$p = InitialGuess(s_0, s_g, t_g)$;
repeat
  Compute $\tilde{u}_{bdd}$ and s_f;
  Compute $\mathbf{J}$;
  Compute cor_p and Update p;
  $i = i + 1$;
until $\mathbf{J} \leq \varepsilon$ `or` $i = i_{\max}$;
return $(p, s_f = s_g?)$;

Algorithm 1 describes our adaptation of a standard optimization algorithm to trajectory generation. This algorithm cycles until convergence or failure. A cycle has three main stages: (i) computation of an admissible control $\tilde{u}_{bdd}$ and

the resulting state s_f reached from the set of parameters p, (ii) computation of the cost function $\mathbf{J}$ (weighed distance between s_f and s_g), and finally (iii) computation of a correction cor_p to apply to the current set of parameters in order to reduce $\mathbf{J}$. Note that t_f, the duration of the control trajectory, is set to t_g in order to take into account the *final time constraint* described earlier. The algorithm ends when $\mathbf{J}$ is small enough or after a fixed number of iterations (in case of convergence towards a local minima).

In order to ensure that the trajectory computed is feasible, a novel method is used: it consists in truncating the parametric profiles of the control whenever necessary so that (2) is satisfied. The following sections describe the three main steps of Algorithm 1 and detail the truncating process.

D. Cost Function

The error between the reached state s_f and the goal state s_g is:

$$\Delta s = s_g - s_f \tag{6}$$

and the distance $\mathbf{J}$ is the following scalar function:

$$\mathbf{J}^2 = \sum_{j=1}^{p} \lambda_j (\Delta s_j)^2 \tag{7}$$

where Δs_j is the j-th feature of the error defined between the goal and final states, and λ_j are weighing coefficients. The resulting value $\mathbf{J}$ is used to determine whether the final state s_f is close enough to the goal state s_g to consider that convergence is achieved.

E. Computation of an admissible parametrization of the control

Given a parametric representation $\tilde{u}(p)$ of this control trajectory, a standard way to obtain the final state s_f is to integrate the equation (4):

$$s_f = s_0 + \int_{t_0}^{t_g} f(s(p), \tilde{u}(p))dt \tag{8}$$

However the input parametric control may violate the inequality constraints $\mathbf{h}(s, \tilde{u})$ (boundary constraints over the control and the state) of the system during the constraints optimization process, after application of one or several corrections on the set of parameters. Furthermore, given the *final time constraint* fixing the time $dt = t_g - t_0$ between both initial and final state, the goal state may be unreachable. Even if no solution exists in that case, our method is aimed to provide a feasible trajectory that ends as close as possible to the goal state. The main idea to ensure the feasibility of the trajectory is to truncate the parametric profiles of the control where the bounds defined over it are overreached. This process is then repeated at each step of the optimization process. We use so piecewise parametric profiles to represent the control and state profiles.

Let $\mathbf{h}(s, \tilde{u}) \leq 0$ represents the set of constraints defined over the control and state profiles. An admissible input control trajectory $\tilde{u}_{bdd}$ is defined thus to take into account the constraints defined over the system:

$$\tilde{u}_{bdd}(t) = \begin{cases} \tilde{u}(p,t) & \text{if } \mathbf{h}(s,\tilde{u}) \leq 0 \\ \mathbf{U}_{extl}(t) & \text{otherwise} \end{cases} \tag{9}$$

where $\mathbf{U}_{extl}(t)$ is the extremal applicable control at time t such that $\mathbf{h}(s, \tilde{u}) \leq 0$, *ie.* the maximal or minimal control (depending of the case), that fulfill all the constraints defined over the system. We have then to consider two different time interval types: time intervals $\mathbf{I}^{fea} = \left\{\mathbf{I}_1^{fea}, \ldots, \mathbf{I}_n^{fea}\right\} \subseteq [t_0; t_g]$ on which the parametric representation of the control trajectory fulfill the constraints defined over the system *ie* st. $\mathbf{h}_i(s, \tilde{u}) \leq 0, \forall i \in [1; n]$, and time intervals $\mathbf{I}^{ovr} = \{\mathbf{I}_1^{ovr}, \ldots, \mathbf{I}_m^{ovr}\} = [t_0; t_g] \setminus \{\mathbf{I}^{fea}\}$ on which the parametric representation of the control violate the constraints.
Considering the new expression of the control trajectory and knowing the sets of intervals $\mathbf{I}^{fea}$ (resp. $\mathbf{I}^{ovr}$) over which the parametric representation fulfill (resp. violate) the constraints of the system, we can determine the final reached state as follows:

$$s_f = s_0 + \sum_{i=0}^{n} \int_{\mathbf{I}_i^{fea}} f(s(p), \tilde{u}(p))dt + \sum_{j=0}^{m} \int_{\mathbf{I}_j^{ovr}} \mathbf{U}_{extl}(t)dt \tag{10}$$

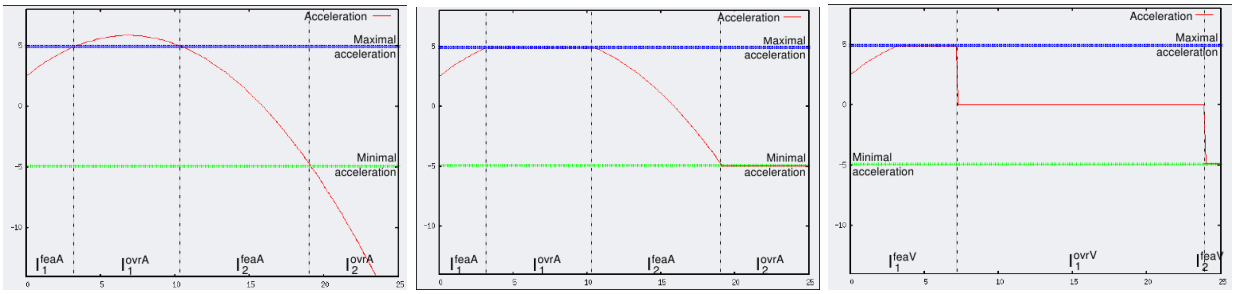
The final state reached is then obtained from the current set of parameters by using the resulting control if its parametric representation is included in the bounds defined over it, and by using the maximal control applicable where its parametric representation is out of the bounds.
Fig. 1 presents an example of the bounding process of the forward method. Let suppose a 1D double integrator system with velocity and acceleration bounds. Given an initial parametric acceleration profile (Fig. 1a), we truncate it on intervals I_1^{ovrA} and I_2^{ovrA} to ensure it fulfills the acceleration constraint of the system (Fig. 1b). After integrating it to obtain the velocity profile (Fig. 1e), we bound this new velocity profile (Fig. 1f) on intervals I_1^{ovrV} to fulfill the velocity constraints afterwards. It implies a second change over the acceleration profile: saturation of the velocity profile nullify consequently the acceleration profile over intervals I_1^{ovrV} (Fig. 1c). The final state reached is then accordingly modified, nevertheless it guarantees that the final acceleration (Fig. 1c), which will define the input control, is feasible.

F. Parametrization Update

The parametrization update is the last but not least part of the trajectory generation. We choose a steepest descent method with variable step length to ensure it because of its low memory and computation cost. So we linearize the equation (4), as follows:

$$\Delta s \simeq \left[\frac{\partial f}{\partial p}\right] \Delta p \tag{11}$$

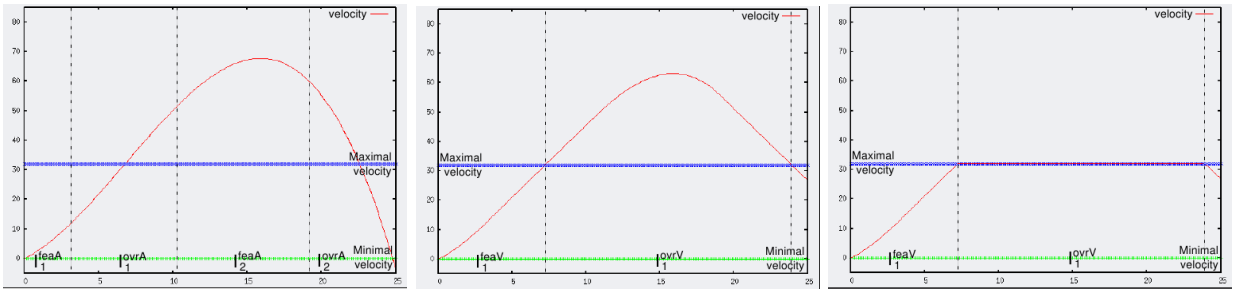
where Δs is the state error given in (Eq. 6), Δp is the supposed error made over the set of parameters and $\frac{\partial f}{\partial p}$ are the partial derivatives of the state variation wrt. parameters.



(a) Step 1: Linear acceleration profile without any bounding process.

(b) Step 2: Linear acceleration profile with acceleration bounding process.

(c) Step 3: Linear acceleration profile with acceleration and velocity bounding processes.



(d) Step 1: Linear velocity profile without any bounding process.

(e) Step 2: Linear velocity profile with acceleration bounding process.

(f) Step 3: Linear velocity profile with acceleration and velocity bounding processes.

Fig. 1: Different steps of the computation of the linear acceleration and velocity profile. The crosses represent the minimal and maximal bounds over these profiles.

To reduce the distance $\mathbf{J}$, a correction over the set of parameters can be computed:

$$cor_p = -\tau \left[\frac{\partial f}{\partial p}\right]^{-1} \Delta s \tag{12}$$

where τ is the application coefficient of the correction. The inverted matrix of the partial derivatives represents then the direction of the correction applied, and $\tau \Delta s$ represents the step length of the steepest descent method.

Now to illustrate the method, we present in the next section its application to a car-like system.

IV. Case Study: Car-like Vehicle

The trajectory generation method defined in the previous part was aimed to handle car-like vehicles. We present here the system considered and the resulting variational trajectory generation.

A. Model of the System

Our trajectory generation process has then been applied to the case of a planar car-like vehicle $\mathcal{A}$. The state of $\mathcal{A}$ is characterized by (x, y, θ, ϕ, v) where (x, y) are the coordinates of the rear wheels midpoint, θ is the heading of $\mathcal{A}$, ϕ is the orientation of the front wheels (steering angle), and v is the linear velocity of the rear wheels. A control of $\mathcal{A}$ is defined by the couple $u = (a, \zeta)$ where a is the rear wheel linear acceleration and ζ the steering velocity. The dynamics of $\mathcal{A}$ is given by:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ v\tan(\phi)/L \\ \zeta \\ a \end{pmatrix} \tag{13}$$

where L is the wheelbase of $\mathcal{A}$. It is assumed that $\mathcal{A}$ is moving forward only:

$$v \in [0, v_{\max}], |\phi| \leq \phi_{\max}, |a| \leq a_{\max} \text{ and } |\zeta| \leq \zeta_{\max} \tag{14}$$

B. Computation of the admissible parametrization

We choose a 2nd-order polynomial parametric representation of the control (a, ζ) as follows :

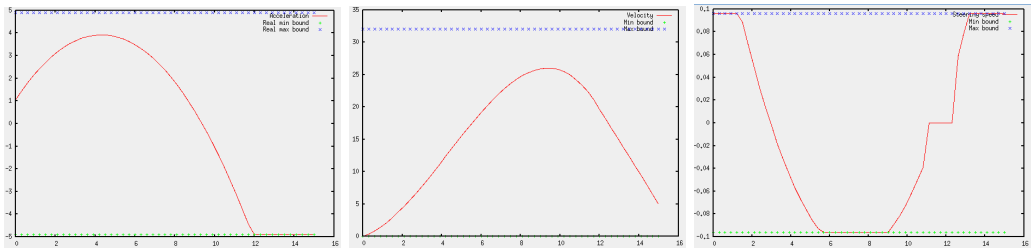
$$\zeta(t) = \alpha_1 + 2\beta_1 t + 3\gamma_1 t^2 \tag{15}$$

$$a(t) = \alpha_2 + 2\beta_2 t + 3\gamma_2 t^2 \tag{16}$$

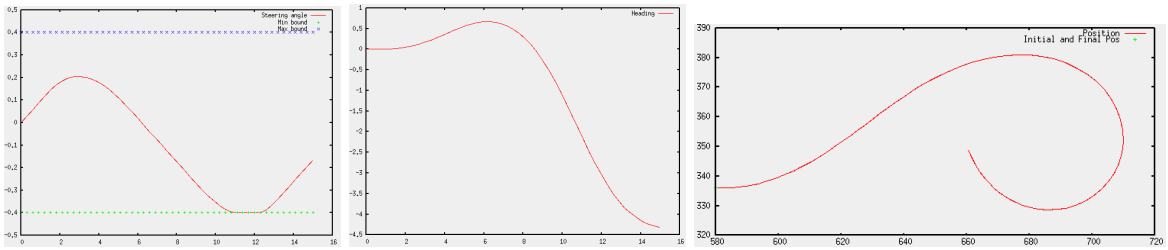
where $p = (\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2)$ is a selected set of parameters.

The computation of an admissible parameterization of the control of $\mathcal{A}$ from the initial set of parameters p consists then in the following steps:

1) Computation of the linear acceleration profile: First the linear acceleration profile is computed thanks to its parametric expression given in Eq. (16). Its initial polynomial profile could nevertheless overreach the acceleration bounds defined over the system. Let us note $I_{a_{min}}$ (resp. $I_{a_{max}}$) the set of time intervals where the minimal (resp. maximal) acceleration bound is overreached, and $I_a =$



(a) Linear acceleration profile (b) Linear velocity profile (c) Steering speed profile



(d) Steering angle profile (e) Heading profile (f) Path profile

Fig. 2: Different state profiles computed by the forward method to obtain a feasible trajectory from a given set of parameters. Green and blue crosses represent the minimal and maximal bounds over each profile.

$[t_0; t_g] \setminus \{I_{a_{min}}, I_{a_{max}}\}$ that respect them. Determination of these intervals allows to truncate the acceleration profile as follows:

$$a(t) = \begin{cases} \alpha_2 + 2\beta_2 t + 3\gamma_2 t^2 & \text{over } I_a \\ -a_{max} & \text{over } I_{a_{min}} \\ a_{max} & \text{over } I_{a_{max}} \end{cases} \quad (17)$$

Piecewise polynomial representation is needed to express this profile. However, it respects then the linear acceleration constraints.

2) Computation of the linear velocity profile: This second profile is computed by integration of the linear acceleration one. So it can be first expressed as follows:

$$v(t) = v_0 + \sum_{i \in I_a} (\alpha_2 dt_i + \beta_2 {dt_i}^2 + \gamma_2 {dt_i}^3) + \sum_{i \in I_{a_{min}}} (-a_{max}) dt_i + \sum_{i \in I_{a_{max}}} a_{max} dt_i \quad (18)$$

As for acceleration, this profile of the linear velocity can overreach the linear velocity bounds defined over the system. As in the previous case, we compute the set of intervals $I_{v_{min}}$ (resp. $I_{v_{max}}$) where the minimal (resp. maximal) linear velocity bound is overreach, and $I_v = [t_0; t_g] \setminus \{I_{v_{min}}, I_{v_{max}}\}$ that respect them. The final linear velocity profile v_{bd} (cf fig. 2b) is then given by:

$$v_{bd}(t) = \begin{cases} (18) & \text{over } I_v \\ -v_{max} & \text{over } I_{v_{min}} \\ v_{max} & \text{over } I_{v_{max}} \end{cases} \quad (19)$$

If the parametric velocity has been modified, the input control acceleration profile must be modified consequently once again. Its final profile a_{bd} is then computed by derivation of v_{bd} wrt. the time:

$$a_{bd}(t) = \begin{cases} 0 & \text{over } I_{v_{min}} \cup I_{v_{max}} \\ -a_{max} & \text{over } I_{a_{min}} \setminus \{I_{v_{min}} \cup I_{v_{max}}\} \\ a_{max} & \text{over } I_{a_{max}} \setminus \{I_{v_{min}} \cup I_{v_{max}}\} \\ \alpha_2 + 2\beta_2 t + 3\gamma_2 t^2 & \text{over } I_{av} \end{cases} \quad (20)$$

where I_{av} is the set of intervals where neither linear acceleration nor linear velocity bounds are overreached. Eq. (20) give then the final profile of the acceleration control of the current forward method (cf fig. (2a)).

3) Computation of the steering angle and steering speed profiles: Steering angle and steering speed profiles are computed in a same way than the linear acceleration and velocity, with the successive bounding processes of their profiles wrt. the steering angle and steering speed constraints (14). Examples of their final profiles ζ_{bd} and ϕ_{bd} are depicted in figures Fig.2c and Fig.2d.

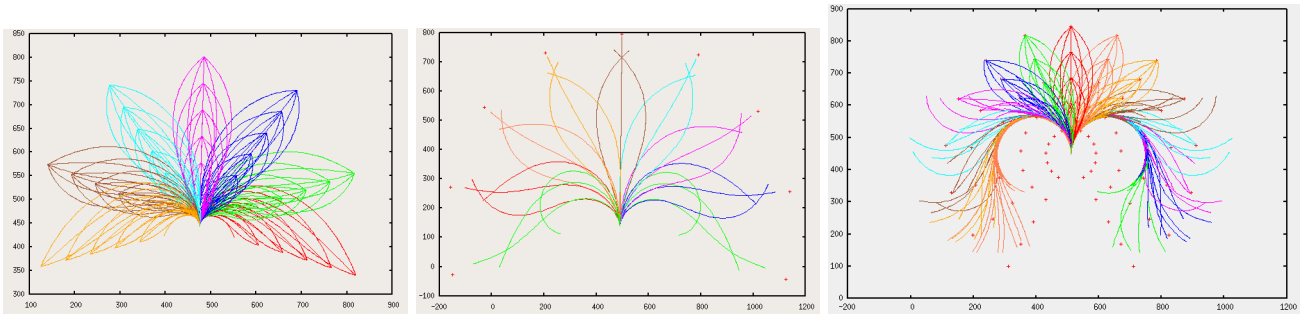
From the new control defined, the remaining states parameters profiles are obtained by integration of the equation (13). The new heading and path profiles are then computed thanks to the bounded velocity and steering angle profiles as follows (cf Fig. 2e,2f):

$$\theta_{bd}(t) = \theta_0 + \int_0^t v_{bd}(t) \frac{\tan(\phi_{bd}(t))}{L} dt \quad (21)$$

$$x_{bd}(t) = x_0 + \int_0^t v_{bd}(t) \cos(\theta_{bd}(t)) dt \quad (22)$$

$$y_{bd}(t) = y_0 + \int_0^t v_{bd}(t) \sin(\theta_{bd}(t)) dt \quad (23)$$

Figures 2 illustrate an example of the resulting control and state profiles of the car-like system $\mathcal{A}$ from a given set of parameters.



(a) Reachable goal states.

(b) Unreachable goal states due to the linear acceleration and velocity constraints.

(c) Unreachable goal states due to the steering angle and steering speed constraints.

Fig. 3: $x \times y$ views of the trajectories computed to try to reach different positions (crosses) around the different system considered. Should the final states cannot be reached, alternative trajectories are obtained nonetheless. All trajectories computed are however guaranteed to be feasible by the system considered.

Example	Standard case	Steering overreached	Velocity overreached
average required steps	12.29	20.00	20.0
average time by trajectory (ms)	3.9	7.0	7.2
average number of traj. by sec.	255.08	146.76	139.73

TABLE I: Performances of the trajectory generation for each case study.

V. Simulation Results

`Tiji` has been implemented in C++ and tested on a desktop computer (Core2Duo@1.8GHz, 2GB RAM, Linux OS). It has been evaluated in different scenarios featuring reachable and unreachable goal states. The maximum number of iterations was heuristically set to 20.

Three examples of trajectories are presented here. Fig. 3a depicts a case where the goal states are reachable from s_0. In that case, `Tiji` is able to find a trajectory connecting both initial and goal states. Figs. 3b and 3c present cases where the goal states are unreachable. In both cases, `Tiji` returns feasible trajectories ending as close as possible to the goal states.

From a complexity point of view, the trajectory generation algorithm depends on the number of required steps to converge to the goal state. Table I sum up the average number of required steps and the resulting computation time in the three different cases study.

VI. Conclusion

The paper has presented `Tiji`, a new trajectory generation scheme that can be used to *efficiently* compute *feasible* trajectories for system with complex dynamics. Besides, it can handle a *final time constraint*, *ie* reaching a goal at a prescribed time instant. Finally, when the goal is unreachable, it returns a trajectory ending as close as possible to the goal. `Tiji` is particularly suited for autonomous navigation situations featuring moving objects. As an example, `Tiji` has been integrated and has proven its efficiency in a trajectory deformation scheme called `Teddy` [15].

Acknowledgments

This work has been supported by the French Ministry of Defence (DGA Doctoral Scholarship) and by the European Commission contract “Have-It FP7-IST-2007-212154”.

References

[1] I. Mitchell, “Comparing forward and backward reachability as tools for safety analysis,” in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science. Springer, 2007, no. 4416.

[2] A. Marigo, B. Piccoli, and A. Bicchi, “Reachability analysis for a class of quantized control systems,” in *Proc. of the 39-th IEEE Conference on Decision and Control*, 2000.

[3] P. Gallina and A. Gasparetto, “A technique to analytically formulate and to solve the 2-dimensional constrained trajectory planning problem for a mobile robot,” *Journal of Intelligent and Robotic Systems*, 2000.

[4] A. Kelly and B. Nagy, “Reactive nonholonomic trajectory generation via parametric optimal control,” in *The International Journal of Robotic Research*, vol. 22, Jul.-Aug. 2003.

[5] L. Dubins, “On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents,” *American Journal of Mathematics*, vol. 79, 1957.

[6] J. Reeds and L. Shepp, “Optimal paths for a car that goes forward and backward,” *Pacific J. Math.*, vol. 145, 1990.

[7] Y. Kanayama and N. Miyake, “Trajectory generation for mobile robots,” in *In Proc. of the Int. Symp. on Robotics Research*, 1985.

[8] T. Fraichard and A. Scheuer, “From reeds and shepp to continuous curvature paths,” *Transactions on Robotics*, vol. 20, 2004.

[9] K. Kant and S. W. Zucker, “Toward efficient trajectory planning: The path-velocity decomposition,” *The International Journal of Robotics Research*, vol. 5, no. 3, 1986.

[10] K. Komoriya and K. Tanie, “Trajectory design and control of a wheel-type mobile robot using B-spline curve,” in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1989.

[11] A. Takahashi, T. Hongo, and Y. Ninomiya, “Local path planning and control for AGV in positioning,” in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1989.

[12] Y. Kanayama and B. I. Hartman, “Smooth local path planning for autonomous vehicles,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1989.

[13] H. Delingette, M. Hébert, and K. Ikeuchi, “Trajectory generation with curvature constraint based on energy minimization,” in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1991.

[14] T. Howard and A. Kelly, “Optimal rough terrain trajectory generation for wheeled mobile robots,” *Int. Journal of Robotics Research*, 2007.

[15] V. Delsart and T. Fraichard, “Navigating dynamic environments using trajectory deformation,” in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2008.