Navigating Dynamic Environments Using Trajectory Deformation

Vivien Delsart, Thierry Fraichard

Navigating Dynamic Environments Using Trajectory Deformation

Vivien Delsart[†] and Thierry Fraichard[†]

***Abstract*— Path deformation is a technique that was introduced to generate robot motion wherein a path, that has been computed beforehand, is continuously deformed on-line in response to unforeseen obstacles. In an effort to improve path deformation, this paper presents a trajectory deformation scheme. The main idea is that by incorporating the time dimension and hence information on the obstacles' future behaviour, quite a number of situations where path deformation would fail can be handled. The trajectory represented as a space-time curve is subject to deformation forces both external (to avoid collision with the obstacles) and internal (to maintain trajectory feasibility and connectivity). The trajectory deformation scheme has been tested successfully on a planar robot with double integrator dynamics and a car-like vehicle.**

***Index Terms*— Autonomous navigation; Motion deformation; Collision avoidance; Dynamic environments.**

I. Introduction

Where to move next? is a key question for an autonomous robotic system. This fundamental issue has been largely addressed in the past forty years. Many motion determination strategies have been proposed (see [1] for a review). They can broadly be classified into *deliberative* versus *reactive* strategies: deliberative strategies aim at computing a complete motion all the way to the goal, whereas reactive strategies determine the motion to execute during the next few time-steps only. Deliberative strategies have to solve a motion planning problem. They require a model of the environment as complete as possible and their intrinsic complexity is such that it may preclude their application in dynamic environments. Reactive strategies on the other hand can operate on-line using local sensor information: they can be used in any kind of environment whether unknown, changing or dynamic, but convergence towards the goal is difficult to guarantee.

To bridge the gap between deliberative and reactive approaches, a complementary approach has been proposed based upon *motion deformation*. The principle is simple: a complete motion to the goal is computed first using a priori information. It is then passed on to the robotic system for execution. During the course of the execution, the still-to-be-executed part of the motion is continuously deformed in response to sensor information acquired on-line, thus accounting for the incompleteness and inaccuracies of the a priori world model. Deformation usually results from the application of constraints both external (imposed by the obstacles) and internal (to maintain motion feasibility and connectivity). Provided that the motion connectivity can be maintained, convergence towards the goal is achieved.

[†]INRIA, CNRS-LIG & Grenoble University, France.

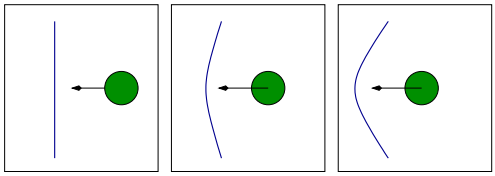


Fig. 1: Path deformation problem: in response to the approach of the moving disk, the path is increasingly deformed until it snaps (like an elastic band).

The different motion deformation techniques that have been proposed [2], [3], [4], [5], [6] all performs *path deformation*. In other words, what is deformed is a geometric curve, *ie* the sequence of positions that the robotic system is to take in order to reach its goal. The problem with path deformation techniques is that, by design, they cannot take into account the time dimension of a dynamic environment. For instance in a scenario such as the one depicted in Fig. 1, it would be more appropriate to leave the path as it is and adjust the velocity of the robotic system along the path so as to avoid collision with the moving obstacle (by slowing down or accelerating). To achieve this, it is necessary to depart from the path deformation paradigm and resort to *trajectory deformation* instead. A trajectory is essentially a geometric path parametrized by time. It tells us where the robotic system should be but also when and with what velocity. Unlike path deformation wherein spatial deformation only takes place, trajectory deformation features both *spatial and temporal* deformation meaning that the planned velocity of the robotic system can be altered thus permitting to handle gracefully situations such as the one depicted in Fig. 1.

The first trajectory deformation scheme has been proposed by one of the authors in [7]. It operates in two stages (collision avoidance and connectivity maintenance stages) and was geared towards manipulator arms. The contribution of this paper is a new trajectory deformation scheme, henceforth called `Teddy` (for Trajectory Deformer). It operates in one stage only and is designed to handle arbitrary robotic systems.

`Teddy` is designed to be one component of an otherwise complete autonomous navigation architecture. A motion planning module is required to provide `Teddy` with the nominal trajectory to be deformed. `Teddy` operates periodically with a given time period. At each cycle, `Teddy` outputs a deformed trajectory which is passed to a motion control module that determines the actual commands for the actuators of the robotic system. The paper focuses on `Teddy`

only. It is organised as follows: Teddy is overviewed in §II Its application to the case of a planar robot with double integrator dynamics is detailed in §III whereas §IV addresses the case of a car-like vehicle. Simulation results are then presented in §V

II. 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 $\mathbb{R}^3$). $q \in \mathbf{C}$ denote a configuration of $\mathcal{A}$. The dynamics of $\mathcal{A}$ is described by a differential equation of the form:

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

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

For the sake of trajectory deformation, a trajectory is discretized in a sequence of nodes. A node is a state-time, it is denoted by $n_i = (s_i, t_i)$. The discrete trajectory of $\mathcal{A}$ is $\Gamma_0 = \{n_0, n_1 \cdots n_N\}$ with n_0 (resp. n_N) the initial (resp. final) node of the trajectory.

B. Trajectory Deformation Principle

Algorithm 1: Teddy.

Input: $\Gamma_k = \{n_k, n_{k+1} \cdots n_N\}$, *workspace model*
Output: $\Gamma_{k+1} = \{n_k{}', n_{k+1}{}' \cdots n_N{}'\}$

```
Γ_{k+1} = ∅;
// Apply forces to each node
foreach n_i ∈ Γ_k do
    n_i' = F_ext(n_i) + F_int(n_i);
    Γ_{k+1} = Γ_{k+1} ∪ n_i';
end
// Resample trajectory
Γ_{k+1} = Resample (Γ_{k+1});
// Check trajectory validity
if not Valid (Γ_{k+1}) then
    Invoke Motion Planner;
end
return Γ_{k+1};
```

The main steps of Teddy are outlined in Algorithm 1. Teddy operates periodically with a time period of duration T_c. At time t_k, it takes as input the still-to-be-executed part of the trajectory $\Gamma_k = \{n_k, n_{k+1} \cdots n_N\}$ and an updated model of the workspace. The workspace model includes the position of the obstacles of $\mathbf{W}$ at time t_k along with information about their future behaviour. Teddy then deforms Γ_k in response to the updated position and future behaviour of the obstacles. At time $t_{k+1} = t_k + T_c$, Teddy outputs a deformed trajectory $\Gamma_{k+1} = \{n_k{}', n_{k+1}{}' \cdots n_N{}'\}$ with $n_i{}'$ the updated node corresponding to n_i.

Like a particle placed in a force field, a node is displaced in response to the application of a force which is the combination of two kind of forces: external and internal. External forces (denoted $\mathbf{F}_{ext}$) are repulsive forces exerted by the obstacles of the environment, their purpose is to deform the trajectory in order to keep it collision-free. They are detailed in §II-C. Internal forces (denoted $\mathbf{F}_{int}$) on the other hand are aimed at maintaining the feasibility and the connectivity of the trajectory, *ie* to ensure that the deformed trajectory still satisfies the dynamics of $\mathcal{A}$. They are detailed in §II-D.

Now, for the sake of both collision-checking and connectivity evaluation, it is desirable to maintain a regular sampling level along the trajectory. Depending on the situation, nodes are removed or added accordingly. This point is detailed in §II-E.

Finally, it is important to note that, like the path deformation scheme, the trajectory deformation scheme suffers from the following limitation: there is no guarantee that it will produce a collision-free and connected trajectory at each time step; both schemes are heuristic by nature. Failure to produce a valid trajectory typically happens when the topology of $\mathbf{S} \times \mathbf{T}$ changes (when a passage is blocked for instance, like when a door is closed). At each time step, the deformed trajectory is therefore checked for collision and connectivity. Should it become invalid, a global motion planner must be invoked to compute a new nominal trajectory. Strictly speaking, the motion planner is not part of Teddy, it is not discussed here.

C. External Forces

External forces are repulsive forces exerted by the obstacles of the environment for collision avoidance purposes. They are derived from a potential function V_{ext}. To explicitly take into account the future behaviour of the moving obstacles, V_{ext} is defined in the space-time $\mathbf{W} \times \mathbf{T}$ (instead of $\mathbf{S} \times \mathbf{T}$ for efficiency reason). In a manner similar to [4], a set of points p_j are selected on the body of $\mathcal{A}$. Each node n_i of the trajectory Γ_k yield a set of control points $c_i^j = (p^j, t_i)$ in $\mathbf{W} \times \mathbf{T}$. For a control point c^j corresponding to the configuration q and the state s along the trajectory, V_{ext} is defined as:

$$V_{ext}(c^j) = \begin{cases} k_{ext}(d_0 - d_{wt}(c^j))^2 & \text{if } d_{wt}(c^j) < d_0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $d_{wt}(c^j)$ is the distance from c^j to the closest obstacle in $\mathbf{W} \times \mathbf{T}$. d_0 is the region of influence around the obstacles and k_{ext} is a repulsion gain. d_{wt} is a distance function in $\mathbf{W} \times \mathbf{T}$. It is derived from the Euclidean distance by scaling the space versus the time dimension. In $\mathbb{R}^2$ for instance, the distance d_{wt} between (x_0, y_0, t_0) and (x_1, y_1, t_1) is given by:

$$\begin{aligned} {d_{wt}}^2 &= {w_s}^2(x_1 - x_0)^2 + {w_s}^2(y_1 - y_0)^2 \\ &\quad + {w_t}^2(t_1 - t_0)^2 \end{aligned} \quad (2)$$

with w_s (resp. w_t) the spatial (resp. temporal) weight. The force resulting from this potential function acting on c^j is then defined as:

$$\mathbf{F}_{ext}^{wt}(c^j) = -\nabla V_{ext}(c^j) = k_{ext}(d_0 - d_{wt}(c^j))\frac{\mathbf{d}}{||\mathbf{d}||} \quad (3)$$

where $\mathbf{d}$ is the vector between c and the closest obstacle point. Now, $\mathbf{F}_{ext}^{wt}$ has to be mapped into $\mathbf{S} \times \mathbf{T}$. The forces defined in $\mathbf{W} \times \mathbf{T}$ by each control point c^j yield a force in $\mathbf{C} \times \mathbf{T}$ defined as follows:

$$\mathbf{F}_{ext}^{ct}(q,t) = \sum_{j=1}^{r} J_{c^j}^T(q,t)\mathbf{F}_{ext}^{wt}(c^j) \quad (4)$$

where $J_{c^j}^T(q,t)$ represents the Jacobian at point c^j:

$$J_{c^j}^T(q,t) = \begin{pmatrix} \frac{\partial q^1}{\partial p_1^j} & \cdots & \frac{\partial q^1}{\partial p_m^j} & 0 \\ \vdots & \ddots & \vdots & \vdots \\ \frac{\partial q^n}{\partial p_i^j} & \cdots & \frac{\partial q^n}{\partial p_m^j} & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \quad (5)$$

with m the dimension of $\mathbf{W}$, p_l^j the l^{th} coordinate of p^j, n the dimension of $\mathbf{C}$ and q^l the l^{th} coordinate of q. The final mapping into $\mathbf{S} \times \mathbf{T}$ that yields $\mathbf{F}_{ext}(n) = \mathbf{F}_{ext}(s,t)$ is carried out by leaving the remaining parameters of s unchanged.

D. Internal Forces

The external forces defined above push each node of the trajectory away from the obstacles if they are inside their influence region. Internal forces are introduced to ensure that the trajectory remains connected, *ie* that there exists a trajectory verifying the dynamics of $\mathcal{A}$ between two consecutive nodes of the trajectory. Trajectory connectivity is related to the concepts of forward and backward reachability. The set of states that are reachable from a given state s_0 are defined as (forward-reachability):

$$\mathcal{R}(s_0) = \{s_f \in \mathbf{S} | \exists \xi, \exists t, s(s_0, \xi, t) = s_f\} \quad (6)$$

Likewise, the set of states from which it is possible to reach a given state s_0 are defined as (backward-reachability):

$$\mathcal{R}^{-1}(s_0) = \{s_b \in \mathbf{S} | \exists \xi, \exists t, s(s_b, \xi, t) = s_0\} \quad (7)$$

Let n_-, n and n_+ denote three consecutive nodes of the trajectory Γ_k. Γ_k is connected at n iff $n \in \mathcal{R}(n_-)$ and $n_+ \in \mathcal{R}(n)$. In other words, n must belong to $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$. Now, two cases arises depending on whether the intersection $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$ is empty or not (if this intersection is not empty, it means that n_- and n_+ are connected together also). The next two sections detail how the internal forces are defined in both cases.

1) Case 1: n_- and n_+ connected: In that case, the purpose of the internal force is to ensure that n remains within $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$. To that end, a virtual spring is defined between n and a selected point H belonging to $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$. It yields a potential function V_{int} defined in the space-time $\mathbf{S} \times \mathbf{T}$ as:

$$V_{int}(n) = k_{int} d_{st}(n)^2 \quad (8)$$

where $d_{st}(n)$ is the distance between n and H. It is defined in a manner similar to d_{wt}. k_{int} is an attraction gain.

$$\mathbf{F}_{int}(n) = -\nabla V_{int}(n) = k_{int} d_{st}(n) \frac{\mathbf{d}}{||\mathbf{d}||} \quad (9)$$

where $\mathbf{d}$ is the vector between n and H.

2) Case 2: n_- and n_+ disconnected: In that case, $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+) = \emptyset$ and it is not possible to find a point H belonging to $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$. The solution proposed then is aimed at restoring the connectivity with n_- only. To that end, H is simply selected within $\mathcal{R}(n_-)$ and $\mathbf{F}_{int}$ is defined as in §II-D.1 above.

3) Selecting H: Depending on whether n_- and n_+ are connected together (*ie* whether $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$ is empty or not), H should be selected within $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$ or $\mathcal{R}(n_-)$. In the former case, a natural choice for H would be the centroid of $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$. In the latter case, H could for instance be defined as the point of $\mathcal{R}(n_-)$ which is the closest to n.

Other choices are possible of course but the important thing to note is that, in theory, determining H requires, in the worst case, the characterization of the three sets $\mathcal{R}(n_-), \mathcal{R}^{-1}(n_+)$ and $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$. Computing reachable sets for arbitrary robotic systems is a process whose complexity is dependent upon the dimensionality of the system considered and whether its dynamics is linear or not (*cf* [8], [9]). Since `Teddy` has a limited time T_c only to deform the trajectory, it is therefore critical that `Teddy` be able to compute $\mathbf{F}_{int}(n)$ as efficiently as possible. To that end, it is important to exploit as much as possible the properties of the robotic system considered, and in some cases, to resort to various approximation or linearization schemes.

In the case where n_- and n_+ are connected, another possibility is to compute a feasible trajectory from n_- to n_+ and to select, say its intermediate state, to define H. Once again, it is the particulars of the robotic systems at hand that determines how the internal forces are actually computed.

E. Trajectory Resampling

In the course of the deformation process, the nodes of the trajectory may either move away from their neighbours or, on the contrary, move very close to them (whether it be in the spatial or the temporal dimensions). For the sake of both collision-checking and connectivity evaluation, it is desirable to maintain a regular sampling level of the trajectory Γ_k. Depending on the situation, nodes are removed or added accordingly.

Let n_-, n and n_+ denote three consecutive nodes of the trajectory Γ_k. A space-time distance similar to d_{wt} is used to compute the distance between two nodes (*cf* (2)). To begin with, if the distance between n_- and n_+ is less than a given threshold, n is removed from Γ_k. Then, the distance between n_- and n is computed. If is is greater than a given threshold then a new intermediate node n_i is added to Γ_k. n_i can be defined as the centroid of $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n)$. This node-adding procedure is repeated recursively for both pair of nodes (n_-, n_i) and $((n_i, n)$ (in case n_- and n are really far from one another). The same node-adding procedure is repeated for the nodes n and n_+.

III. Case Study 1: Double Integrator System

A. Model of the System

To begin with, `Teddy` has been applied to the case of a 2D planar robot $\mathcal{A}$ with double integrator dynamics (point mass model). A state of $\mathcal{A}$ is characterized by (p, v) that respectively denote the 2D position and velocity of $\mathcal{A}$: $p = (x, y)$ and $v = (v^x, v^x)$. The dynamics of $\mathcal{A}$ is given by:

$$\begin{pmatrix} \dot{p} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ a \end{pmatrix} \tag{10}$$

where a denotes the acceleration control applied to $\mathcal{A}$: $|a| \leq a_{\max}$ and $|v| \leq v_{\max}$.

B. Internal Forces Computation

Algorithm 2: H selection (double integrator system).

Input: $\{n_-, n, n_+\}$
Output: H

```
   // Check chronology
1  if t+ < t− then
   |  // Select H within R(n−)
2  |  H = ForwardSelect (n−, n);
3  else
   |  // Compute intermediate time ti
4  |  ti = (t+ − t−)/2;
   |  // Compute R(n−) ∩ R⁻¹(n+) at time ti
5  |  I = R(n−, ti) ∩ R⁻¹(n+, ti);
6  |  if I ≠ ∅ then
7  |  |  H = Centroid (I);
8  |  else
9  |  |  H = ForwardSelect (n−, n);
10 |  end
11 end
12 return H;
```

As mentioned earlier in §II-D.3, one key point in the adaptation of `Teddy` to a particular robotic systems lies in the determination of the point H that is used to compute the internal force $\mathbf{F}_{int}$. It is important that H can be computed efficiently. Algorithm 2 outlines the way H is computed in this case. The main idea is to compute both $\mathcal{R}(n_-)$ and $\mathcal{R}^{-1}(n_+)$ for a particular time slice only, namely the intermediate time slice $t = (t_+ - t_-)/2$. It is also taken advantage of the fact that, it is possible for the system (10) to compute the sets $\mathcal{R}(n_-), \mathcal{R}^{-1}(n_+)$ and $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$ for each spatial dimension independently.

Let us first consider how to compute $\mathcal{R}(n_-)$ for the time slice t (henceforth denoted $\mathcal{R}(n_-, t)$), and for the x-dimension only, *ie* in the case where $n_- = (x_-, v_-^x, t_-)$. The y-dimension is dealt with similarly and so is the computation of $\mathcal{R}^{-1}(n_+, t)$.

1) Computing $\mathcal{R}(n_-, t)$: First, the extremal positions reachable at time t from n_- are computed. It is easily achieved by integrating forward (10) while applying the extremal control $\pm a_{\max}$ (until $\pm v^x_{\max}$ is reached). Let $p_{\min}(t)$ and $p_{\max}(t)$ denote these extremal positions. Then, for a discrete set of positions $p_i \in [p_{\min}(t); p_{\max}(t)]$, the corresponding extremal velocities $v_{\min}(p_i, t)$ and $v_{\max}(p_i, t)$ are computed. Now, the convex hull of the corresponding set of position-velocity pairs yields a 2D polygonal approximation of $\mathcal{R}(n_-, t)$. $\mathcal{R}^{-1}(n_+, t)$ is computed in a similar manner.

2) Computing $\mathcal{R}(n_-, t) \cap \mathcal{R}^{-1}(n_+, t)$: Both $\mathcal{R}(n_-, t)$ and $\mathcal{R}^{-1}(n_+, t)$ are represented by 2D polygons of the position-velocity space. A straightforward polygon intersection yields $\mathcal{R}(n_-, t) \cap \mathcal{R}^{-1}(n_+, t)$.

3) Selecting H: If $\mathcal{R}(n_-, t) \cap \mathcal{R}^{-1}(n_+, t)$ is not empty then its centroid is computed, it becomes H (line 10 of Algorithm 2). Now, if $\mathcal{R}(n_-, t) \cap \mathcal{R}^{-1}(n_+, t)$ is empty, H must be selected within $\mathcal{R}(n_-)$ in order to try to maintain the connectivity between n_- and n. To that end, a discrete set of time instants $t_j > t_-$ is defined and the corresponding reachable sets $\mathcal{R}(n_-, t_j)$ are computed as above. Their centroids H_j are computed as well. Finally the point H_j whose distance to n is minimal becomes H (lines 2 and 12 of Algorithm 2).

IV. Case Study 2: Car-Like System

A. Model of the System

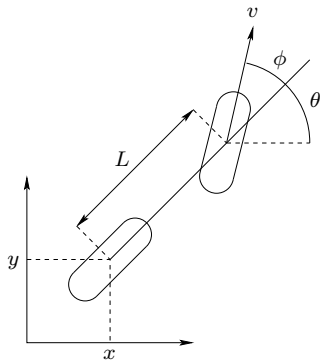


Fig. 2: The car-like vehicle $\mathcal{A}$ (bicycle model).

`Teddy` has then been applied to the case of a planar car-like vehicle $\mathcal{A}$. A state of $\mathcal{A}$ is characterized by (x, y, θ, ϕ, v) where (x, y) are the coordinates of the rear wheel, θ is the main orientation of $\mathcal{A}$, ϕ is the orientation of the front wheels (steering angle), and v is the linear velocity of the front wheel (Fig. 2). A control of $\mathcal{A}$ is defined by the couple

$u = (a, \zeta)$ where a is the front wheel linear acceleration and ζ the steering velocity. The dynamics of $\mathcal{A}$ is given by:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ v\tan(\phi)/L \\ \zeta \\ a \end{pmatrix} \quad (11)$$

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

$$v \in [0, v_{\max}], |\phi| \leq \phi_{\max}, |a| \leq a_{\max} \text{ and } |\zeta| \leq \zeta_{\max} \quad (12)$$

B. Internal Force Computation

Algorithm 3: H selection (car-like system).

Input: $\{n_-, n, n_+\}$
Output: H

```
// Check chronology
if t_+ < t_- then
    // Select H within R(n_-)
    H = ForwardSelect (n_-, n);
else
    // Compute Trajectory from n_- to n_+
    π = TrajectoryGeneration (n_-, n_+);
    if Valid (π) then
        H = IntermediateState (π);
    else
        H = ForwardSelect (n_-, n);
    end
end
return H;
```

With its 5-dimensional state space and non-linear dynamics, the car-like system (11) has a complexity that prevents the efficient characterization of the reachable sets $\mathcal{R}(n_-), \mathcal{R}^{-1}(n_+)$ and $\mathcal{R}(n_-) \cap \mathcal{R}^{-1}(n_+)$ that are required in order to define H, the attraction state used in the definition of the internal force (9).

For efficiency reason, the following approach has been adopted instead in order to determine H (*cf* Algorithm 3): a *steering method* is used to compute a feasible trajectory between (n_-) and (n_+). If it succeeds, *ie* if n_- and n_+ are connected, the intermediate state of the trajectory computed becomes H and is used in the definition of $\mathbf{F}_{int}(n)$. Should the steering method fail to find a trajectory (when n_- and n_+ are not connected), an approximation of $\mathcal{R}(n_-)$ is computed: the control space of $\mathcal{A}$ is randomly sampled and the corresponding states are computed using classical Runge-Kutta method [10]. The sample closest to n becomes H.

The steering method used is derived from the trajectory generation method presented in [11]. It uses parametrized vehicle controls and nonlinear programming to search the control space for an optimum trajectory between two given states. It was modified in order to compute a trajectory to a goal state with a prescribed arrival time (*cf* [12]). It achieves efficiency by using fast numerical optimization techniques and effective initial guesses for the control parameters.

V. Simulation Results

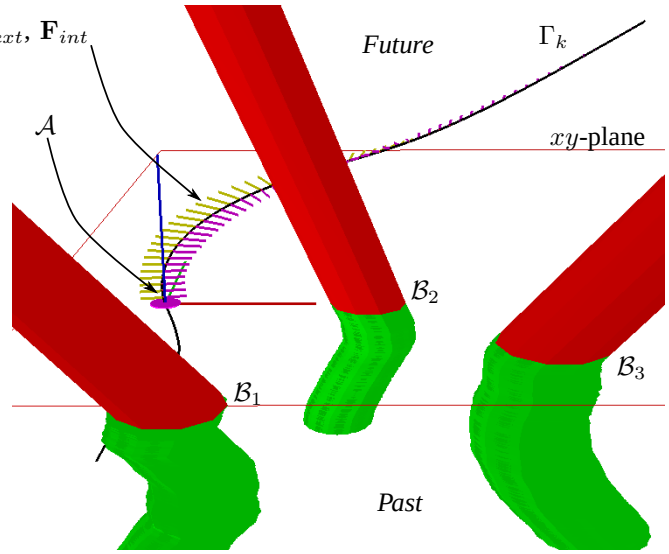
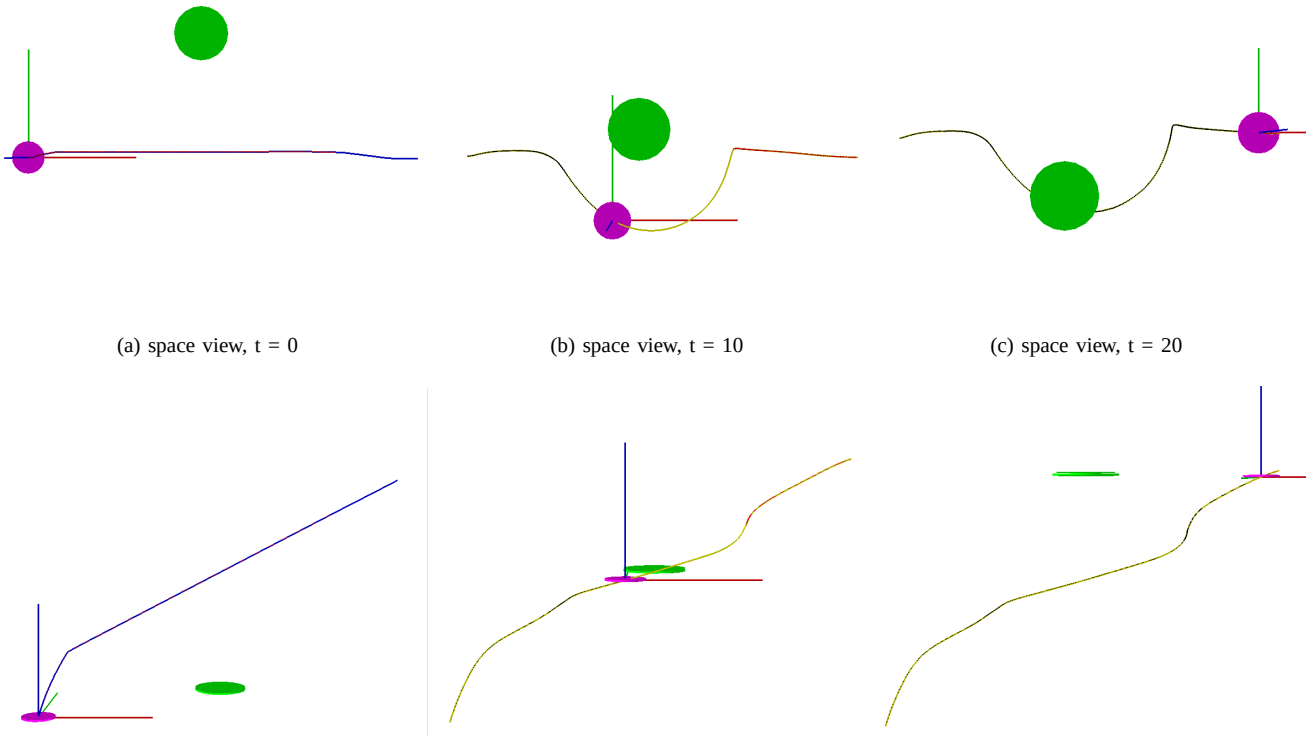


Fig. 3: `Teddy`'s principle visualized in a scenario involving three moving disk obstacles $\mathcal{B}_i, i$=1-3. The time dimension is pointing upward. The past lies below the xy-plane (the present) and the future lies above. The obstacles are moving randomly but the model of the future assumes that they maintain a constant linear velocity. The internal and external forces acting upon the nodes of the trajectory Γ_k are represented by vectors.

`Teddy` has been implemented in C++ and tested on a desktop PC (Pentium 4@3GHz, 1GB RAM, Linux OS). `Teddy` has been evaluated in different scenarios featuring up to 40 circular obstacles moving randomly. At each time step, `Teddy` is provided with a new model of the environment that features the position and the geometry of the obstacles along with their current linear velocity (which is altered randomly at each time step). The model of the future is obtained by assuming that the obstacles maintain a constant linear velocity. This basic assumption is standard, it reflects the information that can be given by a tracking system able to determine the current position and speed of the moving obstacles. `Teddy` can of course handle more elaborate models of the future (using for instance long-term motion prediction models such as the ones developed by [13] among others). The important thing is that, at each time step, `Teddy` uses an updated workspace model and deforms the trajectory accordingly. Fig. 3 illustrates in a visual manner how `Teddy` operates.

The next two sections illustrates the workings of `Teddy` for the two robotics systems considered in different scenarios. The video which is attached to this paper presents similar results in a more lively fashion.



(a) space view, t = 0 (b) space view, t = 10 (c) space view, t = 20

(d) time view, t = 0 (e) time view, t = 10 (f) time view, t = 20

Fig. 4: Double integrator system, "cutting" scenario (spatial deformation): $\mathcal{A}$ is moving from the left to the right, the obstacle is moving downwards. The top snapshots depict the path at different time instant ($x \times y$ view). The bottom snapshots depict the velocity profile at the same instants ($x \times t$ view).

A. Double Integrator System

To emphasize the interest of trajectory deformation *vs* path deformation, a "cutting" scenario similar to the one depicted in Fig. 1 has been considered first. This scenario has been selected because it is problematic for classical path deformation schemes.

`Teddy` relies upon a number of parameters to operate properly: the repulsion gain k_{ext}, the attraction gain k_{int} and the distance functions d_{wt} and d_{st}. The two examples presented below have been selected to illustrate the importance of the distance function d_{wt} on the performance of `Teddy`. Recall that d_{wt} is used to determine the distance between a trajectory node and the closest obstacle in $\mathbf{W} \times \mathbf{T}$ (*cf* §II-C). In both examples, the initial trajectory had a duration of 20s and the discrete trajectory contained 320 nodes. `Teddy` would run at 28Hz.

For the same scenario, two very different deformation patterns can be obtained by properly selecting the weights w_s and w_t in (2). The first example is obtained by giving more weight to w_s thereby allowing more important spatial deformations to take place (Fig. 4). In this case, $\mathcal{A}$ has time to pass before the obstacle crosses its path. The path component of the trajectory is deformed downwards for safety reasons whereas the velocity component is only slightly modified. The second example on the other hand is obtained by giving more weight to w_t thereby allowing more important temporal deformations to take place (Fig. 5). In this case, $\mathcal{A}$ let the obstacle cross its path before proceeding. The path component of the trajectory is only slightly modified whereas the velocity component is largely deformed so as to allow $\mathcal{A}$ to slow down and stop in order to give way to the obstacle.

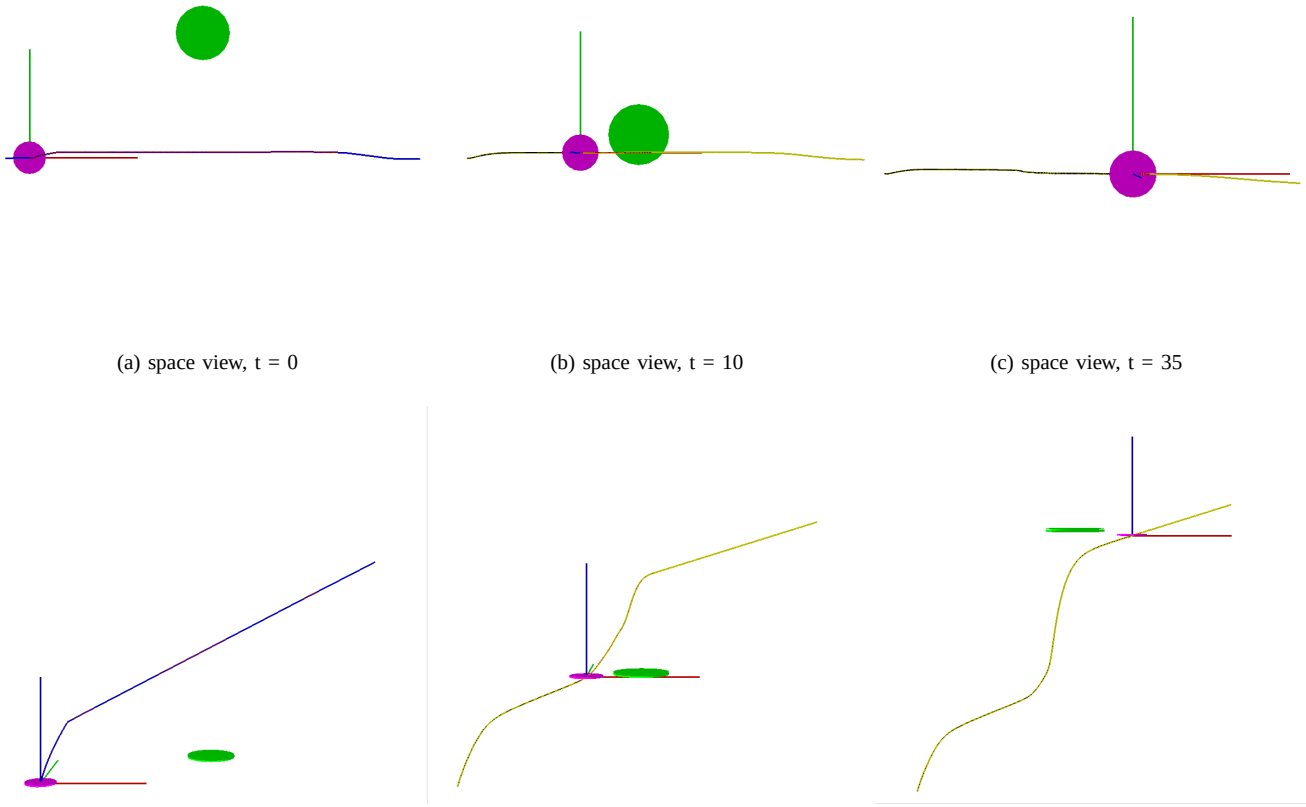
These two examples have shown the influence of the choice of the parameters in the final performance of `Teddy`. They have also illustrated the advantage of trajectory deformation versus path deformation. Afterwards, `Teddy` has been tested on different scenarios featuring both fixed and moving obstacles. The velocity of the moving obstacles change randomly at each time cycle. Such a scenario is depicted in Fig. 6. It features ten fixed and forty randomly moving obstacles.

B. Car-Like System

In the car-like system case, `Teddy` has also been tested on different scenarios featuring both fixed and moving obstacles. Fig. 7 depicts such a scenario wherein $\mathcal{A}$ is placed in an environment featuring ten fixed and twenty randomly moving obstacles. Fig. 7 illustrates the ability of `Teddy` to deform a trajectory while keeping its curvature and curvature derivative compatible with the dynamics of $\mathcal{A}$.

C. Performances

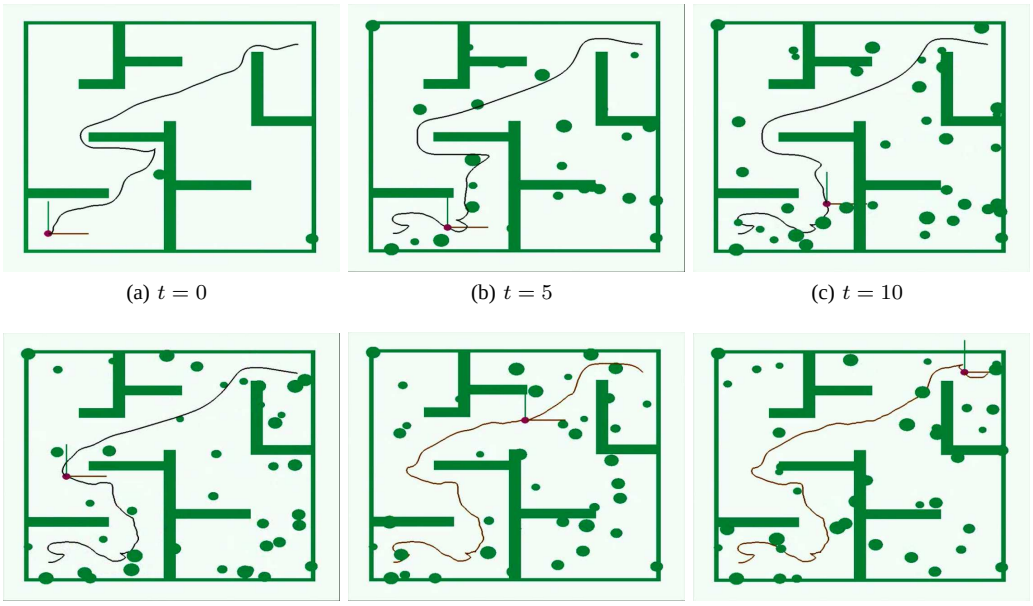
From a complexity point of view, the overall complexity of `Teddy` grows linearly with the number of nodes and



(a) space view, t = 0 (b) space view, t = 10 (c) space view, t = 35

(d) time view, t = 0 (e) time view, t = 10 (f) time view, t = 35

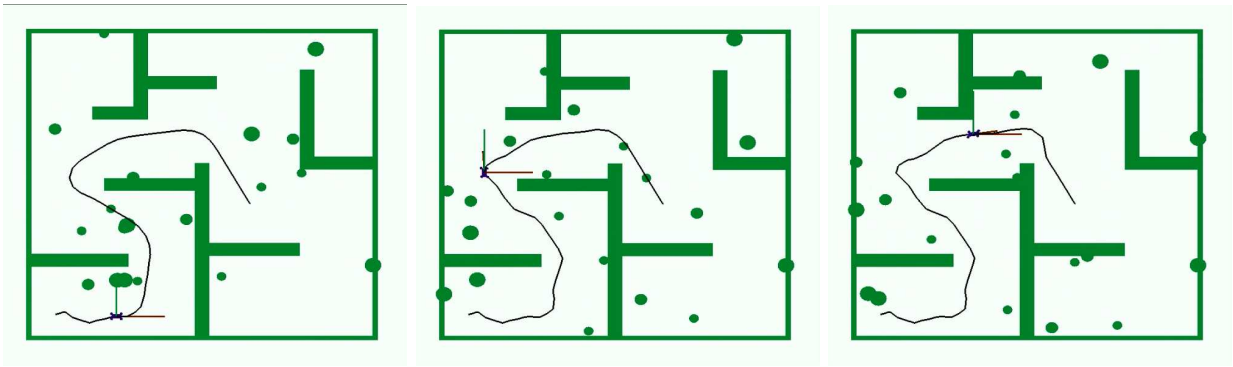
Fig. 5: Double integrator system, "cutting" scenario (temporal deformation): $\mathcal{A}$ is moving from the left to the right, the obstacle is moving downwards. The top snapshots depict the path at different time instant ($x \times y$ view). The bottom snapshots depict the velocity profile at the same instants ($x \times t$ view).



(a) $t = 0$ (b) $t = 5$ (c) $t = 10$

(d) $t = 20$ (e) $t = 25$ (f) $t = 30$

Fig. 6: Double integrator system: the snapshots depict the path at different time instant ($x \times y$ view).



(a) $t = 3$ (b) $t = 13$ (c) $t = 20$

Fig. 7: Car-like system: the snapshots depict the path at different time instant ($x \times y$ view).

number of obstacles	number of nodes				
	50	100	180	250	320
1	6	11	20	27	35
3	44	48	68	70	73
10	49	88	135	199	229

TABLE I: Running time (in ms) of one deformation cycle as a function of the number of nodes and obstacles.

the number of obstacles. Table I gives the running time for the double integrator system of one deformation cycle for different numbers of nodes and obstacles.

VI. Conclusion and Future Works

The paper has presented `Teddy`, a trajectory deformation scheme. Given a nominal trajectory reaching a given goal, `Teddy` deforms it reactively in response to updated information about the environment's obstacles. `Teddy` can handle robotic systems with arbitrary dynamics. It has been applied to the case of a 2D double integrator system and a car-like system. Because, `Teddy` explicitly takes into account information on the future behaviour of the obstacles, it is able to handle situations that are problematic for classical path deformation schemes. In the future, it is planned to further optimize `Teddy`. Considering for instance that the knowledge about the future behaviour is less reliable in the distant future, it could be interesting to monotonically decrease the influence of the obstacles with respect to time. Last but not least, `Teddy` remains to be integrated within a global navigation architecture and tested on an actual robotic system. It is planned to do so on the architecture and the vehicle presented in [14]

Acknowledgements

This work has been supported by the French Ministry of Defence (DGA Doctoral Grant) and by the European Commission contracts "Cybercars-2 FP6-IST-2004-028062" and "Have-It FP7-IST-2007-212154".

References

[1] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[2] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA (US), May 1993.

[3] M. Khatib, H. Jaouni, R. Chatila, and J.-P. Laumond, "Dynamic path modification for car-like nonholonomic mobile robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM (US), Apr. 1997.

[4] O. Brock and O. Khatib, "Elastic strips: a framework for motion generation in human environments," *Int. Journal of Robotics Research*, vol. 21, no. 12, Dec. 2002.

[5] F. Lamiraux, D. Bonnafous, and O. Lefebvre, "Reactive path deformation for nonholonomic mobile robots," *IEEE Trans. on Robotics and Automation*, vol. 20, no. 6, Dec. 2004.

[6] Y. Yang and B. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation," in *Proc. of the Int. Conf. Robotics: Science and Systems*, Philadelphia PA (US), Aug. 2006.

[7] H. Kurniawati and T. Fraichard, "From path to trajectory deformation," in *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, San Diego, CA (US), Oct. 2007.

[8] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler, "Recent progress in continuous and hybrid reachability analysis," in *Proc. of the IEEE Int. Conf. on Computer Aided Control Systems Design*, Munich (DE), Oct. 2006.

[9] 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.

[10] C. Lubich, "On projected runge-kutta methods for differential-algebraic equations," in *BIT Numerical Mathematics*, vol. 31, 1991.

[11] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int. Journal of Robotics Research*, vol. 26, no. 2, Feb. 2007.

[12] V. Delsart and T. Fraichard, "Reactive trajectory deformation," INRIA," Research report, In Press.

[13] D. Vasquez, T. Fraichard, O. Aycard, and C. Laugier, "Intentional motion on-line learning and prediction," *Machine Vision and Applications*, Jan. 2008.

[14] G. Chen, T. Fraichard, and L. Martinez-Gomez, "A real-time autonomous navigation architecture," in *Proc. of the IFAC Symp. on Intelligent Autonomous Vehicles*, Toulouse (FR), Sept. 2007.