Proceedings of the 2007 IEEE/RSJ International
Conference on Intelligent Robots and Systems
San Diego, CA, USA, Oct 29 - Nov 2, 2007

From Path to Trajectory Deformation*

Hanna Kurniawati
National University of Singapore
http://www.comp.nus.edu.sg/~hannakur

Thierry Fraichard
Inria Rhône-Alpes & Grenoble University (FR)
http://emotion.inrialpes.fr/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. This paper introduces the first trajectory deformation scheme as an effort to improve path deformation. 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 deformation scheme presented operates in two steps, *ie*, a collision avoidance step and a connectivity maintenance step, hence its name 2-Step-Trajectory-Deformer (2-STD). In the collision avoidance step, repulsive forces generated by the obstacles deform the trajectory so that it remains collision-free. The purpose of the connectivity maintenance step is to ensure that the deformed trajectory remains feasible, *ie*, that it satisfies the robot's kinematic and/or dynamic constraints. Moreover, unlike path deformation wherein spatial deformation only takes place, 2-STD features both *spatial and temporal* deformation. It has been tested successfully on a planar robot with double integrator dynamics moving in dynamic environments.**

I. Introduction

A. Background and Motivation

Motion determination is a fundamental issue in designing autonomous robotic systems. To operate in the physical world, an autonomous robot needs to sense, reason, and act. Without a reliable motion determination strategy, the results of sensing and reasoning become void. Many motion determination strategies have been proposed (a summary can be seen in [2]). Most of them can be classified into two approaches, *ie*, *deliberative* and *reactive*. Deliberative approach computes a complete motion all the way to the goal using motion planning techniques. This approach requires a model of the environment as complete as possible and their intrinsic complexity is such that it may preclude their application in dynamic environments [3], [4]. On the other hand, reactive approach determines the motion to execute during the next few time-steps only. This approach can be used in any kind of environment including unknown and dynamic, but convergence towards the goal is not guaranteed.

To bridge the gap between deliberative and reactive approaches, recent works have proposed a complementary approach called *motion deformation*. Before execution, this approach computes a complete motion to the goal based on a priori environment information. Then, during 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. Provided that the motion connectivity can be maintained, convergence towards the goal is achieved (Fig. 1).

*) An extended version of this paper is in [1]

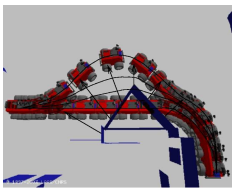


Fig. 1. Path deformation principle. The initial path is deformed so as to account for the triangular obstacle that was unknown at planning time [5].

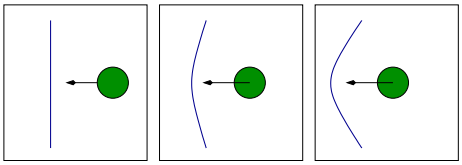


Fig. 2. Path deformation problem. As the moving disk approaches the path, the path is increasingly deformed until it snaps [6].

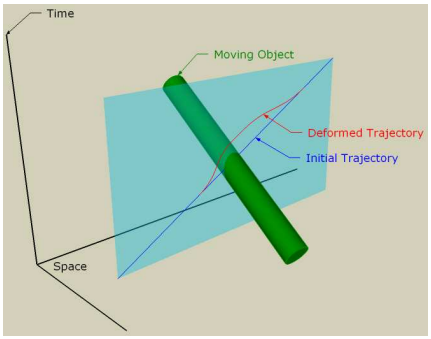


Fig. 3. Temporal deformation of a trajectory.

Although several motion deformation methods have been proposed [6], [7], [8], [9], [10], all deform only the geometric path and ignore the time dimension of a dynamic environment. This is not entirely satisfactory. For instance in Fig. 2, when a moving obstacle is to cross the path of the robot, path deformation takes place. As the moving obstacles approach the path, deformation increases. At some point, the path violates the robot's motion constraints and hence path connectivity can no longer be maintained. When this happens, it is necessary to resort to path planning in order to

1-4244-0912-8/07/$25.00 ©2007 IEEE.

determine a new path to the goal. However, if instead of keep deforming the geometric path, we also adjust the velocity, we can avoid the obstacles by slowing down to let the obstacles pass through first, and then continue following the geometric path at the next time step. Fig. 3 shows an illustration of the deformation in the state×time space. This indicates that the above problem can be alleviated if we deform in the state×time space.

Based on the above observation, we propose a *trajectory deformation* method. A trajectory is a continuous time-sequence of states and can be represented as a geometric curve in the state×time space of the robotic system [11]. Each obstacle yields a forbidden subset of the state×time space which is equivalent to the well-known configuration-obstacle. To avoid collision, a trajectory in the state×time space must avoid these "state×time-obstacles". As in path deformation, trajectory deformation starts with an initial trajectory to the goal which is computed off line based on a priori information. We assume that the robot starts execution with this initial trajectory and then updates its environment model periodically based on information from its sensors using techniques such as [12], [13]. At each update, the corresponding state×time-obstacle may change and the current trajectory may no longer be collision-free. Hence, the trajectory should be deformed accordingly. Intuitively, trajectory deformation features both *spatial and temporal* deformation, which means that the planned velocity of the robot can be altered and hence permitting a more satisfactory deformation in situations such as the one depicted in Fig. 2.

We call our trajectory deformation method *2-Step-Trajectory-Deformer* (2-STD). It consists of two steps, *ie*, a collision avoidance step and a connectivity maintenance step. In the collision avoidance step, repulsive forces generated by the obstacles deform the trajectory so that it remains collision-free. This step applies repulsive forces in the workspace×time space to several points on the robot, and then transform these forces to generate a repulsive force in the robot's state×time space. By doing so, this step can be applied to high dofs robot. The purpose of the connectivity maintenance step is to ensure that the deformed trajectory remains feasible, *ie*, that it satisfies the robotic system's kinematic and/or dynamic constraints. Depending on the dynamic equation of the robot, the connectivity maintenance step is performed either independently per dimension or using a method similar to the bang bang approach. Hence, this step can be applied to high dofs robot efficiently, too. Therefore, 2-STD is applicable to robots with any number of dofs.

II. Overview of the approach

Throughout the paper, we consider that the robot's motion is governed by an equation of the form:

$$\dot{s} = f(s,u) \quad ; \quad u_{\min} \leq u \leq u_{\max} \tag{1}$$

where $s \in \mathcal{S}$ is the robot's state. In a first order system, a state refers to the robot's configuration, while in a second order system, a state refers to a tuple of the robot's configuration and velocity. The variable $\dot{s} \in \mathcal{V}$ is the state's first derivative with respect to time or velocity for short, and $u \in \mathcal{U}$ is a control input bounded by minimum control u_{min} and maximum control u_{max}. The set $\mathcal{S}$, $\mathcal{V}$, and $\mathcal{U}$ are the state space, velocity space, and control space, respectively.

2-STD assumes that a collision-free trajectory γ taking the robot to its goal has been computed prior to execution. Trajectory γ is discretized into a sequence of nodes. Each node is denoted by (s,t), where $t \in \mathcal{T}$ is the time. We assume that the discretization is fine enough such that any two subsequent nodes (s_i,t_i) and (s_{i+1},t_{i+1}) are connected, *ie*, a *constant* control input can move the robot from s_i at t_i to s_{i+1} at t_{i+1}. A node (s_{i+1},t_{i+1}) is in the *reachable space* of a node (s_i,t_i) and (s_i,t_i) is in the *back-reachable space* of (s_{i+1},t_{i+1}) iff (s_i,t_i) and (s_{i+1},t_{i+1}) are connected.

Periodically, the robot receives an updated model of its workspace $\mathcal{W}$. This world model includes the prediction of the future motion of the moving obstacles. Upon receiving the new world model, 2-STD checks whether the still-to-be-executed nodes of γ are still collision-free. If one or more nodes are in-collision, 2-STD deforms γ.

2-STD deforms γ by moving the nodes in $\mathcal{S} \times \mathcal{T}$ in two steps, *ie*, obstacle avoidance step and connectivity maintenance step. In the obstacle avoidance step, 2-STD adopts the external force mechanism of the Elastic Strip approach [6]. A set of points (called control points) are defined over the robot body. Based on a particular distance function, repulsive forces generated by the different obstacles are defined for each control point. However, unlike elastic strip, the repulsive forces in 2-STD are defined in $\mathcal{W} \times \mathcal{T}$, instead of $\mathcal{W}$. Applying a repulsive force to a given control point pushes the control point in $\mathcal{W} \times \mathcal{T}$ away from the obstacles. Each of the repulsive forces are mapped into a displacement vector in $\mathcal{S} \times \mathcal{T}$ and the sum of these displacement vectors will be applied to the obstructed node.

Once the obstacle avoidance step is completed, γ may no longer be connected. In the connectivity maintenance step, 2-STD tries to restore the connectivity. It operates locally on triples of subsequent nodes. Suppose $({s_j}', {t_j}')$ is the result of applying the obstacle avoidance step to (s_j,t_j) and $({s_j}'', {t_j}'')$ is the result of applying the connectivity maintenance step to $({s_j}', {t_j}')$. To ensure the connectivity of a trajectory, 2-STD tries to ensure the connectivity of each triplet $({s_{i-1}}', {t_{i-1}}')$, $({s_i}', {t_i}')$, $({s_{i+1}}', {t_{i+1}}')$ in the trajectory, sequentially. It tries to move these nodes such that $({s_i}'', {t_i}'')$ is in the reachability space of $({s_{i-1}}'', {t_{i-1}}'')$ and the back-reachability space of $({s_{i+1}}'', {t_{i+1}}'')$. For computational efficiency, the nodes are moved in $\mathcal{S}$ only. To respect the displacements generated by the obstacle avoidance step, 2-STD tries to minimize the total displacements,

$$d(s'_{i-1}, s'_i, s'_{i+1}) = \sum_{j=i-1}^{i+1} |s'_j - s''_j| \tag{2}$$

The overall strategy of 2-STD is shown in Algorithm 1. And the details are described in the next section.

Algorithm 1 2-Step-Trajectory-Deformer (2-STD)

Let γ be the current trajectory followed by the robot.
Receive new world model.
if the still-to-be-executed part of γ is obstructed **then**
  Let (s_j, t_j) be the first node of γ that becomes obstructed with respect to the new world model.
  i = j.
  repeat
    if node (s_i, t_i) is the last node **then**
      Add extra node (s_{i+1}, t_{i+1}).
    for k = i-1 to i+1 **do**
      if (s_k, t_k) has not been deformed by the obstacle avoidance step according to the new world model **then**
        Apply the obstacle avoidance step to (s_k, t_k) and keep the result in $({s_k}', {t_k}')$.
    Apply the connectivity maintenance step to triplet $({s_{i-1}}', {t_{i-1}}')$, $({s_i}', {t_i}')$, and $({s_{i+1}}', {t_{i+1}}')$ and keep the results in $({s_{i-1}}'', {t_{i-1}}'')$, $({s_i}'', {t_i}'')$, and $({s_{i+1}}'', {t_{i+1}}'')$, respectively.
    $i = i + 1$.
  until all nodes from (s_i, t_i) to the last node of γ are collision-free, (s_{i+1}, t_{i+1}) is reachable from $({s_i}'', {t_i}'')$, and the last node is at the goal state.
  if the deformed trajectory is collision free **then**
    Let the robot follow the deformed trajectory. So, γ is now : $<(s_1, t_1), \ldots, (s_{j-2}, t_{j-2}), ({s_{j-1}}'', {t_{j-1}}''), \ldots, ({s_i}'', {t_i}''), (s_{i+1}, t_{i+1}), \ldots >$.
  else
    Find a new trajectory starting from the current node.

III. 2-Step-Trajectory-Deformer

A. Obstacle Avoidance Step

2-STD avoids collisions by applying repulsive forces to the obstructed nodes. Suppose γ is the trajectory that is currently followed by the robot. Then, assuming the trajectory discretization is fine enough, when an obstacle obstruct an edge of γ, some of the nodes will also be obstructed. Now, suppose at time t, the robot receives a new world model of the future where node (s_i, t_i) of γ, $t_i > t$, becomes obstructed. To avoid collision, 2-STD generates repulsive forces in $\mathcal{W} \times \mathcal{T}$ to move the robot's control points away from obstacles. It then transforms each of these forces to a displacement vector in $\mathcal{S} \times \mathcal{T}$. The sum of these vectors is then applied to (s_i, t_i).

Since the repulsive force is defined based on distance function in $\mathcal{W} \times \mathcal{T}$, before describing the repulsive force, we need to first define a metric in $\mathcal{W} \times \mathcal{T}$. The main issue here is in deciding the scaling between a unit distance in the spatial dimension $\mathcal{W}$ and a unit distance in the temporal dimension $\mathcal{T}$. In this paper we simply assume that 1 m in $\mathcal{W}$ is the same as 1 sec in $\mathcal{T}$. Other scalings are of course possible and we discuss them in Section V. Once the scaling is resolved, we consider $\mathcal{W} \times \mathcal{T}$ as a Euclidean space and use Euclidean metric as the metric in $\mathcal{W} \times \mathcal{T}$.

Now, we can define the repulsive force applied to a control point c. Let's denote the position of c when the robot is at (s_i, t_i) as c_i. The goal of the repulsive force is to move c_i in $\mathcal{W} \times \mathcal{T}$ away from obstacles. Since the robot's control points are supposed to represent the whole robot, 2-STD "enlarges" the obstacles at each time t_i of each node (s_i, t_i) in γ by a constant factor r_{inf}. The enlarged obstacles are then interpolated to generate the obstacles in the continuous $\mathcal{W} \times \mathcal{T}$ space. We call the enlarged obstacle as the obstacle's influence region and a point in $\mathcal{W} \times \mathcal{T}$ will be repulsed whenever it lies inside an obstacle's influence region. To compute the force, 2-STD will first find a point on the boundary of the obstacles' influence region that is nearest to c_i. For computational efficiency, 2-STD also tries to preserve the temporal ordering of the nodes in γ. For that, 2-STD sets the time deformation to be within $[t_{i-1,i}, t_{i,i+1}]$ where $t_{j,k} = (t_j + t_k)/2$. So, 2-STD finds a point on the boundary of the obstacles' influence region within $[t_{i-1,i}, t_{i,i+1}]$ that is nearest to c_i. Suppose this point is (w_n, t_n). The repulsive force (illustrated in Fig. 4) can then be defined as follows,

$$F_{\mathcal{W}}(s_i, c) = k_{ext}(w_n - c_i) \quad ; \quad F_{\mathcal{T}}(s_i, c) = t_n - t_i \quad (3)$$

where k_{ext} is the repulsion gain. The constant k_{ext} is not used in $F_{\mathcal{T}}$ to ensure that the deformation in $\mathcal{T}$ will not change the temporal ordering of the nodes in γ.

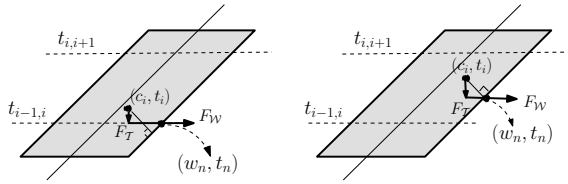


Fig. 4. The repulsive force in $\mathcal{W} \times \mathcal{T}$. The grey parallelogram is the influence region of an obstacle in $\mathcal{W} \times \mathcal{T}$. (a) When $t_n = t_{i-1,i}$. (b) When $t_{i-1,i} < t_n < t_{i,i+1}$.

The repulsive forces will then be transformed and summed to generate a displacement vector in $\mathcal{S} \times \mathcal{T}$ as follows,

$$D_{\mathcal{S}}(s_i) = \sum_{\forall c \in C} J_c^T(s_i) F_{\mathcal{W}}(s_i, c) \quad (4)$$
$$D_{\mathcal{T}}(s_i) = \frac{1}{|C|} \sum_{\forall c \in C} F_{\mathcal{T}}(s_i, c)$$

where C is the set of the control points of the robot, $|C|$ is the cardinality of C, and $J_C(s_i)$ is the Jacobian at point c of the robot while it is at s_i. The displacement vector will then be applied to (s_i, t_i) to move the robot away from obstacles.

B. Connectivity Maintenance Step

After the trajectory γ has been deformed to avoid obstacles, it may no longer be connected. The connectivity maintenance step tries to ensure the connectivity of the deformed trajectory while respecting the results of the obstacle avoidance step, as much as possible.

To ensure the connectivity of the trajectory, 2-STD ensures the connectivity of each triplet of nodes, sequentially. To keep the notation simple, we will use triplet (s_0, t_0), (s_1, t_1), (s_2, t_2) and assume that these nodes are the results of the obstacle avoidance step. They are connected whenever there is a pair of velocities $(\dot{s_0}, \dot{s_1})$ such that

$$s_1 = s_0 + \int_{t=t_0}^{t_1} \dot{s_0}\, dt \quad ; \quad s_2 = s_1 + \int_{t=t_1}^{t_2} \dot{s_1}\, dt \quad (5)$$

We denote the result of the connectivity maintenance step as $({s_0}', {t_0}')$, $({s_1}', {t_1}')$, $({s_2}', {t_2}')$. Maintaining connectivity of triplet means moving the nodes such that the resulting node $({s_1}', {t_1}')$ is in the reachability space of $({s_0}', {t_0}')$ and

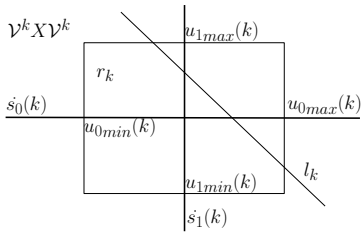


Fig. 5. All pairs of $\dot{s}(k)$ inside r_k satisfy the robot's motion equation for dimension-k. All pairs of $\dot{s}(k)$ on l_k satisfy (7).

in the back-reachability space of $(s_2{}', t_2{}')$. Thus, to ensure the connectivity of the triplet, we need to find a valid pair of velocities $\dot{s}_0$ and $\dot{s}_1$ that move the robot from $(s_0{}', t_0{}')$ to $(s_1{}', t_1{}')$ and from $(s_1{}', t_1{}')$ to $(s_2{}', t_2{}')$, respectively. Furthermore, in trying to respect the results of obstacle avoidance step, we try to minimize the displacement $d(s_0, s_1, s_2)$, as defined in (2). Hence, the main issue here is to find a desired valid pair of velocities $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$, *ie*, a pair of velocities that satisfies the motion equation of the robot and minimizes the displacement $d(s_0, s_1, s_2)$.

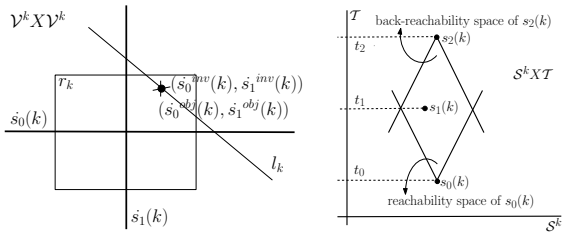
Let's now describe the overall method for finding $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$. First, 2-STD approximates the integration in (5) with summation to get

$$s_1 = s_0 + (t_1 - t_0)\dot{s}_0 \quad ; \quad s_2 = s_1 + (t_2 - t_1)\dot{s}_1 \qquad (6)$$

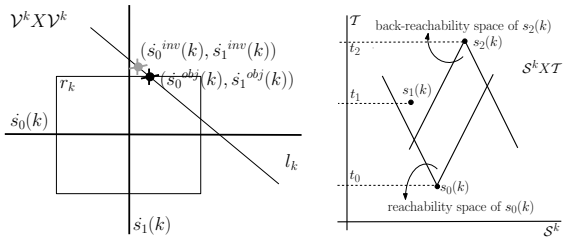
When $(t_1 - t_0)$ and $(t_2 - t_1)$ are small, any motion equation can be approximated quite accurately with the above summation. 2-STD will then find the pair of velocities $(\dot{s_0}^{inv}, \dot{s_1}^{inv})$ that satisfies (6). This pair of velocities minimizes the displacement since it will not move the states at all, but it may not satisfy the robot's motion equation. The idea is to use this pair as an "initial guess" to get $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$. Different motion equations may have different ways for finding this desired valid pair of velocities, efficiently. In the subsequent paragraphs, we give the details when the motion equation is governed by $f(s, u) = u$ and the details for arbitrary motion equation. Once the desired valid pair of velocities $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$ is found, 2-STD applies these velocities to (6) to get the new s_1' and s_2'. Notice that we do not change s_0. Although changing s_0 is possible, we prefer *not* to move s_0 to ensure that we do not need to go back and ensure the connectivity of the previous triplets again. This enables the robot to use a partially deformed trajectory, instead of having to wait until the whole deformation process is finished, which is useful when part of the trajectory that is obstructed by obstacles is quite long and the robot needs to move fast.

When the robot's motion is governed by $f(s, u) = u$, a valid pair of velocities can be found independently for each dimension. Let's now focus on dimension-k of $\mathcal{S}$ and of $\mathcal{V}$. When we consider the space of pair-of-velocities $\mathcal{V} \times \mathcal{V}$, the pair of velocities at dimension-k forms a 2-dimensional subspace $\mathcal{V}^k \times \mathcal{V}^k \subset \mathcal{V} \times \mathcal{V}$. In this subspace, the boundaries of the control input at dimension-k, *ie*, $u_{min}(k)$ and $u_{max}(k)$, form a rectangle r_k. Moreover, combining the two equations in (6) gives us the following line l_k equation in $\mathcal{V}^k \times \mathcal{V}^k$

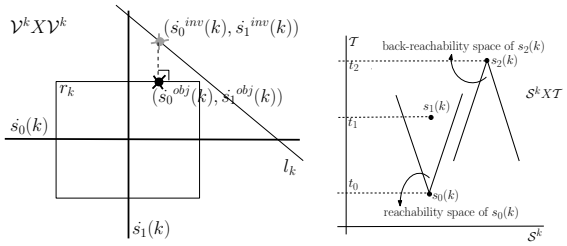
$$s_2(k) - s_0(k) = (t_1 - t_0)\dot{s}_0(k) + (t_2 - t_1)\dot{s}_1(k) \qquad (7)$$



(a) Case-1



(b) Case-2



(c) Case-3

Fig. 6. Three possible cases for $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$ when $f(s, u) = u$.

where $s(k)$ and $\dot{s}(k)$ are dimension-k of s and $\dot{s}$, respectively. See Fig. 5 for an illustration. Each point that lies inside r_k and on l_k satisfies the robot's motion equation at dimension-k, and if $s_1'(k) = s_0(k) + (t_1 - t_0)\dot{s}_0(k)$ then $s_1'(k)$ is in the reachability space of $(s_0(k), t_0)$ and in the back-reachability space of $(s_2(k), t_2)$. The point $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$ always lie on l_k. The issue here is to bring $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$ to be inside r_k. Three cases are possible (Fig. 6), *ie*,

- Case-1 (Fig. 6*a*), $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$ is valid. In this case, $(\dot{s_0}^{obj}(k), \dot{s_1}^{obj}(k))$ = $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$. This automatically minimizes the displacement as none of the nodes are moved.
- Case-2 (Fig. 6*b*), $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$ is invalid but $r_k \cap l_k \neq \oslash$. In this case, $(\dot{s_0}^{obj}(k), \dot{s_1}^{obj}(k))$ is the point on $r_k \cap l_k$ that is closest to $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$. Simple algebraic manipulation is enough to prove that this pair of velocities minimizes the displacement, assuming that the approximation in (6) is accurate enough.
- Case-3 (Fig. 6*c*), $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$ is invalid and $r_k \cap l_k = \oslash$. In this case, 2-STD sets $(\dot{s_0}^{obj}(k), \dot{s_1}^{obj}(k))$ to be the point on r_k that is nearest to $(\dot{s_0}^{inv}(k), \dot{s_1}^{inv}(k))$. Note that in this case both $s_1(k)$ and $s_2(k)$, instead of just $s_1(k)$, are moved. Here, the pair of velocities is not guaranteed to minimize the displacement.

The above steps are performed to each dimension indepen-

dently to get the desired valid pair of velocities $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$.

Now, in general $f(s,u) \neq u$ and a valid pair of velocities can not be found independently for each dimension. Nevertheless, the pairs of valid velocities that satisfy $f(s,u)$ form a subspace in $\mathcal{V} \times \mathcal{V}$. When efficient methods are known for projecting a point in $\mathcal{V} \times \mathcal{V}$ to the subspace, 2-STD finds valid pair of velocities that minimizes the displacement, by first projecting $(\dot{s_0}^{inv}, \dot{s_1}^{inv})$ to the subspace. When the projected point satisfies the bounds on the control input, this projected point becomes $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$. When the projected point does not satisfy the bounds on the control input, 2-STD generates pairs of velocities using the combination of minimum and maximum control input and chooses the pair of velocities that minimizes the displacement $d(s_0, s_1, s_2)$ to be $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$. When this happens, the deformed nodes will be achieved by applying combinations of minimum and maximum control input, similar to the bang-bang approach [2]. In this case too we can not guarantee that $(\dot{s_0}^{obj}, \dot{s_1}^{obj})$ minimizes the displacement. When the motion equation is too complicated for projection to take place, 2-STD bypasses the projection step to directly use the combination of minimum and maximum control.

C. Adding nodes

When node (s_{goal}, t_{end}) at the end of the trajectory is deformed, the trajectory may not end at the goal state anymore. When the trajectory does not end at the goal state anymore, 2-STD adds an additional node (s_{add}, t_{add}), $s_{add} = s_{goal}$ and $t_{add} = t_{end} + t_d + t_\epsilon$ at the end of the deformed trajectory. Of course this additional node is not guaranteed to be collision-free nor reachable from its previous node. Therefore, 2-STD will again apply both obstacle avoidance and connectivity maintenance step. This process of adding node and applying the two steps of 2-STD will be repeated until the goal state is collision-free and reachable.

IV. Experimental Results

We implemented and tested 2-STD on a simulator written in C++ and ran on an Intel Pentium 4 3GHz 1GHz RAM with Linux operating system. In the test, we assume that a new world model is given every 20ms. Throughout the test, we use a planar square robot with double integrator subject to velocity and acceleration bounds. We ran this robot on two different dynamic environment scenarios.

The first scenario is used to assess the performance of 2-STD in handling scenarios as in Fig. 2. Initially, there is no obstacle and the original trajectory is a straight line. At t = 20ms, the robot receives a new world model (Fig. 7*b*) where there is a new obstacle that crosses the original trajectory. Upon receiving this world model, 2-STD deforms the trajectory to avoid obstacles (Fig. 7*c*). By incorporating the information on the future behaviour of the obstacle, 2-STD is able to deform the trajectory from behind the obstacle. This avoids 2-STD from deforming the trajectory too much such that it breaks the connectivity. Next, 20ms later, the robot receives a new world model where again there is another obstacle moving and crossing the original trajectory (Fig. 7*d*). As before, 2-STD deforms the trajectory to avoid obstacles (Fig. 7*e*). Each of the deformation process took less than 2ms. This scenario shows that 2-STD is able to avoid obstacles where path deformation methods tend to fail. This is expected because by taking the time dimension into account, 2-STD uses information on the future behaviour of the robot in order to deform the trajectory. This enables 2-STD to anticipate the obstacle's motion and deform the trajectory more appropriately.

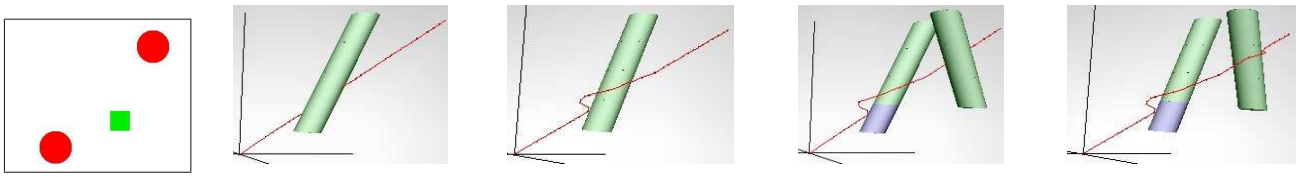
The second scenario is to assess the usefulness of 2-STD when the model of the future changes frequently. Initially, there is no obstacle and the original trajectory is shown in Fig. 8*a*. At t = 20ms, new obstacle A is detected to obstruct the trajectory. As a result, 2-STD deforms the initial trajectory (Fig. 8*b*). Then, 40ms later the robot detects new obstacle B that obstructs the trajectory. To avoid collision, 2-STD deforms the trajectory (Fig. 8*c*). Next, 20ms later at t = 80ms, the robot realizes that B has changed its motion and obstructs a different part of the trajectory and hence triggers 2-STD to deform the trajectory again (Fig. 8*d*). Then, 20ms later, the robot realizes that the motion of B, predicted at t = 80ms, is not entirely correct and the new predicted obstacle's motion obstructs again a different part of the trajectory. So 2-STD deforms the trajectory again (Fig. 8*e*). Note that the trajectory that has been deformed to avoid obstacles according to the world model received at t = 80ms will not be deformed back (to the original trajectory) unless it becomes obstructed. Last, at t = 160ms, new obstacle C is detected and this new obstacle passes through the goal position at the same time the trajectory reaches its goal position. 2-STD deforms the trajectory by adding nodes such that it will reach the goal position sometimes later (Fig. 8*f*). Each of these deformations took less than 8ms. This scenario shows that 2-STD is efficient enough to adapt to the frequent changes of the world model.

Although in our experiments, all the obstacles are dynamic, 2-STD respects static obstacles, too. Static obstacles are handled in the same way as dynamic obstacles are handled. And, although we only experimented with low dofs robot, 2-STD is applicable for high dofs robot, because the two steps of 2-STD, *ie*, obstacle avoidance and connectivity maintenance, are applicable to both low and high dofs robot.

V. Discussion

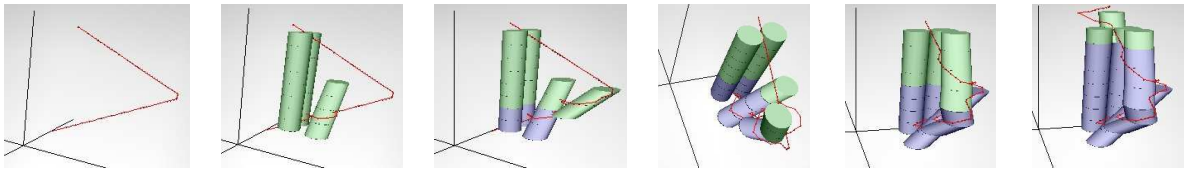
Several issues still need further investigation, *ie*,

- 2-STD uses distance function in $\mathcal{W} \times \mathcal{T}$ to avoid obstacles. This means, the distance metric is a combination of the distance in $\mathcal{W}$ and $\mathcal{T}$. Therefore, we need a good scale between a distance of one unit in $\mathcal{W}$ and a distance of one unit in $\mathcal{T}$. Too much weight on $\mathcal{W}$ causes the deformation to be dominated by spatial deformation, while too much weight on $\mathcal{T}$ causes the deformation to be dominated by temporal deformation. In the current implementation, we simply assume that 1m distance in $\mathcal{W}$ is the same as 1s distance in $\mathcal{T}$. A better scale may be to consider the average speed of the robot. Another



(a) The scenario. (b) t = 20ms, obstructed. (c) t = 20ms, deformed. (d) t = 40ms, obstructed. (e) t = 40ms, deformed.

Fig. 7. Scenario-1. (a) The robot is colored green, the obstacles are colored red. The obstacle at the left of the robot is heading up while the other obstacle is heading down. The robot has just avoided collision with the obstacle in the left. (b)-(e) The environment and trajectory in configuration×time space. The axis are the black lines, the time axis is the vertical line. The skewed tubes are the obstacles in the configuration×time space, the blue part is the past while the green part is the future. The trajectory is the red line.



(a) t = 0ms. (b) t = 20ms. (c) t = 60ms. (d) t = 80ms. (e) t = 100ms. (f) t = 160ms.

Fig. 8. Scenario-2. The environment and trajectory in configuration×time space. The axis are the black lines, the time axis is the vertical line. The skewed tubes are the obstacles in the configuration×time space, the blue part is the past while the green part is the future. The trajectory is the red line.

possibility is to relate it to our objective, *eg*, to reach the goal quickly, to minimize energy usage, *etc*.

- Currently, to keep the temporal ordering of the trajectory unchanged, the deformation of each node in $\mathcal{T}$ is limited. It would be interesting to see the result when the deformation in $\mathcal{T}$ is unlimited just as the deformation in $\mathcal{S}$. The key issue is of course how to efficiently reorder the trajectory such that the robot can still use the partially deformed trajectory.
- In the connectivity maintenance step, 2-STD fixes the time and assumes that only one control input can be applied for moving the robot from a node to its subsequent node. This is often too restricted, especially when the motion equation is complex. One improvement would be to allow changes in time and allow several changes of control between nodes.
- Although in the connectivity maintenance step, 2-STD tries to respect the deformation generated by the obstacle avoidance step, in general there is no guarantee that the connectivity maintenance step will not nullify the results of obstacle avoidance step. One way to improve on this maybe to "blend" the obstacle avoidance and connectivity maintenance step more smoothly.

VI. Conclusion

This paper introduces the first trajectory deformation strategy as an effort to improve path deformation technique for motion planning in dynamic environment. The main idea is that by incorporating the time dimension and hence information on the obstacles' future behaviour, motion deformation approach can be used for robotic system moving in complicated dynamic environment. Based on this idea, we propose a trajectory deformation method, 2-STD, that consists of two steps. First, the obstacle avoidance step to deform the trajectory to avoid obstacles. Second, the connectivity maintenance step to ensure that the deformed trajectory respects the robot's kinematic and/or dynamic constraints.

Our preliminary results on a planar robot with double integrator dynamics show that 2-STD is able to efficiently deform the robot's motion in cases where path deformation methods fail, *ie*, cases such as Fig. 2. Moreover, 2-STD is efficient enough to deform the robot's motion, avoiding collision with dynamic obstacles, even when the world model changes frequently.

References

[1] H. Kurniawati and T. Fraichard, "From path to trajectory deformation," *INRIA Research Report*, 2007.

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

[3] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance, Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[4] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *IROS*, Edmonton, AB (CA), Aug. 2005.

[5] O. Lefebvre, F. Lamiraux, C. Pradalier, and T. Fraichard, "Obstacles avoidance for car-like robots. integration and experimentation on two robots," in *ICRA*, New Orleans, LA (US), Apr. 2004, pp. 4277–4282.

[6] O. Brock and O. Khatib, "Elastic strips: a framework for motion generation in human environments," *IJRR*, vol. 21, no. 12, pp. 1031–1–52, Dec. 2002.

[7] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *ICRA*, Atlanta, GA (US), May 1993, pp. 802–807.

[8] M. Khatib, H. Jaouni, R. Chatila, and J.-P. Laumond, "Dynamic path modification for car-like nonholonomic mobile robots," in *ICRA*, Albuquerque, NM (US), Apr. 1997, pp. 2920–2925.

[9] F. Lamiraux, D. Bonnafous, and O. Lefebvre, "Reactive path deformation for nonholonomic mobile robots," *IEEE Trans. on Robotics and Automation*, vol. 20, no. 6, pp. 967–977, Dec. 2004.

[10] Y. Yang and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation," in *RSS*, Philadelphia, USA, August 2006.

[11] T. Fraichard, "Trajectory planning in a dynamic workspace: a 'state-time space' approach," *Adv. Robotics*, vol. 13, no. 1, pp. 75–94, 1999.

[12] A. Bruce and G. Gordon, "Better motion prediction for people-tracking," in *ICRA*, New Orleans, LA (US), Apr. 2004.

[13] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *IJRR*, vol. 24, no. 1, pp. 31–48, 2005.