Collision-Free and Curvature-Continuous Path Smoothing in Cluttered Environments

Jia Pan [1] and Liangjun Zhang [2] and Dinesh Manocha [3]
[1]*panj@cs.unc.edu,* [3] *dm@cs.unc.edu, Dept. of Computer Science, University of North Carolina at Chapel Hill*
[2]*zhanglj@stanford.edu, Dept. of Computer Science, Stanford University*

***Abstract*—We present a novel trajectory computation algorithm to smooth jerky collision-free paths computed by sample-based motion planners. Our approach uses cubic B-splines to generate G^2 or curvature continuous trajectories. The algorithm performs local spline refinement to compute smooth, collision-free paths in narrow passages and satisfy velocity and acceleration constraints. We also present a fast and reliable algorithm for collision checking between robot and the environment along the B-spline trajectories. We highlight the performance of our algorithm on complex benchmarks, including path computation for rigid and articulated models in tight spaces and cluttered environments.**

I. Introduction

Sample-based planning algorithms such as probabilistic roadmaps (PRMs) [9] or rapidly-exploring random trees (RRTs) [11, 12] are frequently used to compute collision-free paths for physical robots and virtual agents. These algorithms generate samples using randomized techniques and attempt to connect nearby samples using local planning methods. The final paths are represented as piecewise linear paths in the configuration space, where the vertices correspond to the samples. Overall, sample-based planners are able to compute collision-free paths for high DOF robots, and can also handle cluttered environments or narrow passages.

In many applications, including virtual prototyping [5], protein folding [21], computer animation [24], and physical robotic systems [18, 25, 26], it is important that the paths are not only collision-free, but also satisfy other constraints in terms of smoothness and solution quality. There is considerable work on computation of smooth paths in mobile robotics, because nonsmooth motions can cause slippage and overactuation.

It is well known that sample-based planners can sometime generate jerky, unnatural paths that may contain unnecessary turns or the velocities at the vertices may change arbitrarily [7, 9, 12, 27]. Furthermore, these issues become more significant when the free space of the robot has narrow passages [15, 28] and the search space for path computation becomes more constrained.

Many techniques have been proposed in the literature to generate smooth paths. At a broad level they can be classified into shortcut methods [7, 8, 9, 11] or optimization-based approaches [1, 2, 18, 19, 27]. Current shortcut methods replace jerky or unnatural portions of a path with shorter linear or curved segments. The curve segments may correspond to parabolic arcs, Bézier curves or Dubins curves. These

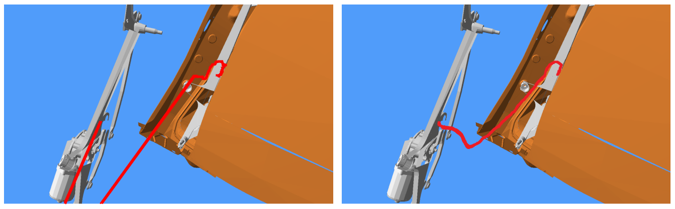
Fig. 1. Left: a jerky, piecewise linear collision-free path for a wiper for maintainability of the windscreen wiper motion. Right: a curvature-continuous collision-free path computed by our spline-based shortcut algorithm.

linear shortcut methods tend to be fast and simple, and can produce high quality paths in many cases [6]. However, current formulations may not provide enough flexibility in terms of generating higher order smoothness or handling narrow passages. Furthermore, exact checking for collisions along curved or higher-order trajectories can be relatively expensive. On the other hand, optimization-based methods tend to improve the quality of paths based on formulating the problem as an optimal control problem or as the elastic problem from Differential Geometry. Some of the commonly used solutions are based on gradient based methods, which tend to compute minimum-energy paths, or elastic bands or elastic strip planning that model the path using a mass-spring system. One of the challenges with optimization-based methods is to reliably compute collision-free paths, when there are a large number of obstacles or narrow passages.

Main Results: In this paper, we present a fast and simple algorithm to smooth the paths generated using sample-based planing by using spline interpolation. Our algorithm randomly selects a sequence of points along the original piecewise linear path and constructs a cubic B-spline in the configuration space that interpolates these points. We use the exponential map to construct smooth rotational motion in $SO(3)$ and thereby handle translational as rotational motion in a uniform way. The cubic spline formulation provides sufficient flexibility in terms of providing higher order smoothness, i.e. compute almost G^2 (second-order geometric or curvature-continuous) trajectories. Moreover, we use local spline refinement to satisfy velocity and acceleration constraints. We initially present the algorithm for rigid bodies and later extend it to high-DOF articulated models.

A key challenge in terms of using higher-order trajectories for the robot is performing fast and reliable collision detection with the obstacles. We present a novel collision-checking algorithm that uses tight motion bounds on the translational and rotational motions along the B-spline trajectories to perform fast collision checks using *conservative advancement* [22]. The overall approach is reliable, guaranteed to not miss any collisions and is significantly faster than prior exact algorithms based on continuous collision detection.

The overall smoothing algorithm is general and the cubic B-spline formulation offers sufficient flexibility and degrees-of-freedom to handle tight configuration spaces with narrow passages. We highlight its performance on many high-DOF complex CAD benchmarks used in virtual prototyping applications. The overall runtime performance is comparable to prior linear shortcut algorithms, though our formulation results in smoother trajectories and can also satisfy velocity and acceleration constraints.

The rest of paper is organized as follows. We introduce the notation and present our spline motion representation in Sec II. We present the basic spline-based smoothing and local refinement algorithm in Section III. In Section IV, we describe an efficient collision checking algorithm for spline trajectories. We highlight the performance on different benchmarks for rigid and articulated bodies in Section V.

II. Representation for Motions

In this section, we introduce our notation and present the underlying spline representation used to compute the trajectories.

A. Notation

Let $\mathcal{C}$ denote the d-dimensional configuration space and let $\mathcal{C}_{free}$ denote the subset of configurations that are collision-free. A configuration within $\mathcal{C}$ is denoted as $\mathbf{x}$. Its superscripts denote DOF or joint indexing (e.g. $\mathbf{x}^k$ is the value for k-th joint) and subscripts denote configuration indexing among multiple samples (e.g. $\mathbf{x}_k$ is the k-th sample in the configuration space).

A trajectory $\mathbf{u}(t), 0 \leq t \leq T$, represents a curve in the configuration space. $\mathbf{u}(t)$ is collision free if all the configurations on $\mathbf{u}(t)$ belong to $\mathcal{C}_{free}$. $\mathbf{u}(t)$ is considered to be (physically) feasible if it is collision free and satisfies other constraints. We use the well known definition of *geometric continuity* from approximation theory [4] to define smooth trajectories. Specifically, two curves meet at a common end point with $\mathcal{G}^n$ continuity, if there exists a reparameterization of the curves that meets at the same point with $\mathcal{C}^n$ continuity. In our case, we are interested in computing curvature-continuous trajectories, and ensure that our curve fitting and modification algorithms generate G^2 curves [4, 25]. In addition, we also impose constraints on the maximum velocity and acceleration along each curve, based on the following bounds:

1) Velocity $\mathbf{u}'(t)$ is bounded by a given limit $|\mathbf{u}'(t)| \leq \mathbf{v}_{max}$,
2) Acceleration $\mathbf{u}''(t)$ is bounded by a given limit $|\mathbf{u}''(t)| \leq \mathbf{a}_{max}$.

B. Spline Representation for Motion

Given an initial collision free piecewise linear trajectory $\mathbf{u(t)}$, parameterized over the interval $[0, T]$ with n vertices $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ computed by sample-based motion planning algorithms, our objective is to improve the smoothness of the input trajectory while satisfying various constraints. In the literature, many curved formulations including screw motion [14, 17], parabolic curves [7] and Bézier curves [16, 25] have been used to compute smooth trajectories. In our formulation, we use cubic B-splines to represent the trajectory of rigid or articulated robots. B-splines are well studied in approximation theory and correspond to a spline function that has minimal support with respect to a given degree, smoothness and domain partition. The B-splines are specified based on knot values, control points or de Boor points [4]. Moreover, they are evaluated in a recursive manner using the well-known *Cox-de Boor* recursion formula. In practice, B-splines provide sufficient flexibility to compute a curvature-continuous trajectory and to perform local refinement by adjusting the control points or knots. Moreover, the degree of B-spline curve is independent with the number of control points, which makes our method more flexible to control long trajectories, as compared to Bezier curve based methods [16, 25].

We first describe the motion representation for a rigid body. For articulated models, we represent them as composed of multiple rigid bodies. The motion of a rigid body $\mathbf{u}(t)$ consists of two parts: translation $\mathbf{T}(t)$ and rotation $\mathbf{R}(t)$. Translation motion $\mathbf{T}(t)$ is a curve in $\mathcal{R}^3$, so it can be naturally formulated as a 3D cubic B-spline. However, there are more issues in terms of representing the motion of the rotation component $\mathbf{R}(t)$. The rotational motion is in fact a curve in $\mathcal{SO}(3)$ and a cubic B-spline in $\mathcal{R}^3$ may not be able to represent it well. One method is to represent the rotation by Euler angles (α, β, γ). However, Euler angles can not formalize some rotational motions due to some intrinsic singularities and it can be difficult to generate motion with smooth angular velocity.

In robotics and computer animation, many applications represent rotational motion as quaternions that are singularity-free [10, 14, 20, 23]. Other techniques tend to use the recursive formulation of B-splines, i.e. based on de Boor algorithm [4], for quaternion curves [20, 23]. However, the spline constructed by these approaches may not have a closed formulation and collision-checking along such trajectories can be rather expensive, as described in Section IV.

We represent the rotation motion based on exponential maps [13]. This representation is relatively simple and can be also used to generate curvature-continuous trajectories. The exponential map $\exp(\cdot)$ is a continuous map between $\mathcal{R}^3$ and $\mathcal{SO}(3)$: $\exp(\mathbf{u}\theta) = (\cos\theta, \mathbf{u}\sin\theta)$, where $q = (\cos\theta, \mathbf{u}\sin\theta)$ is a quaternion with $\mathbf{u}$ as the rotation axis and θ as the rotation angle. The inverse of exponential map is called the logarithmic map $\log(\cdot)$. Given the underlying constraints, we first construct cubic B-spline $\mathbf{w}(t)$ in $\mathcal{R}^3$, and then use the exponential map

Algorithm 1: Spline-based Shortcut Algorithm

```
Input  : Trajectory $\mathbf{u}(t)|_{0\leq t\leq T}$, iteration count $N$
Output: A smooth and collision-free trajectory $\mathbf{s}(t)$
begin
    for $i = 1$ to $N$ do
        Randomly choose $\mathbf{u}(t)|_{t_a\leq t\leq t_b}$ from the input trajectory;
        Construct a cubic B-spline $\mathbf{s}(t)$ interpolating $m+1$ points on $\mathbf{u}(t)|_{t_a\leq t\leq t_b}$;
        Perform spline collision checking (SCD) for $\mathbf{s}(t)$;
        Resolve the colliding segments in $\mathbf{s}(t)$ based on spline modification;
        Refine $\mathbf{s}(t)$ locally to satisfy velocity and acceleration constraints;
end
```

to map it back onto $\mathcal{SO}(3)$. The resulting map

$$\exp(\mathbf{w}(t)) : \mathcal{R}^3 \to \mathcal{SO}(3) \tag{1}$$

is a cubic B-spline curve in $\mathcal{SO}(3)$ that is C^2 continuous.

Overall, we formulate the motion of rigid body by translation spline $\mathbf{T}(t)$ and rotation spline $\mathbf{R}(t) = \exp(\mathbf{w}(t))$, where $\mathbf{T}(t)$ and $\mathbf{w}(t)$ are both cubic B-spline in $\mathcal{R}^3$. We denote $\mathbf{f}^{t_0,\dots,t_{m-2}}_{\mathbf{d}_0,\dots,\mathbf{d}_m}(t)$ as the cubic B-spline $\sum_{i=0}^{m}\mathbf{d}_i b_i(t)$, where $\{\mathbf{d}_i\}_{i=0}^{m}$ are $m+1$ de Boor control points, $\{t_i\}_{i=0}^{m-2}$ are $m-1$ knots, $\{b_i(t)\}_{i=0}^{m}$ are $m+1$ basis functions, and $t_0 \leq t \leq t_{m-2}$. $\mathbf{f}$ is uniform B-spline if $\{t_i\}_{i=0}^{m-2}$ correspond to uniform knot spacing. We also denote spline function as $\mathbf{f}(t)$ for convenience.

III. Shortcut Smoothing based on Cubic B-Splines

In this section, we introduce our smoothing algorithm based on cubic B-splines. Our formulation can use B-splines of any degree. However, the cubic B-splines provide a good balance between smoothness constraints, spline refinement, cost of collision checking and oscillations that can be caused by high degree curves. Our algorithm utilizes the information from the original piecewise linear path that needs to be smoothed. Furthermore, our approach is general and applicable to all environments, composed of a few or a high number of obstacles.

A. Algorithm Overview

The overall smoothing algorithm is shown in Algorithm 1. Given an input trajectory $\mathbf{u}(t), t \in [0,T]$, we iteratively smooth it. At each iteration, we randomly choose a portion $\mathbf{u}(t)|_{t_a\leq t\leq t_b}$ from the input trajectory and construct a smoother B-spline trajectory $\mathbf{s}(t)$ that interpolates the end points and some intermediate points (refer to Section III-B). We then use spline collision detection (SCD) technique (refer to Section IV) to check whether every configuration along the trajectory $\mathbf{s}(t)$ is collision-free. If there is a collision between the robot and obstacles along that trajectory, we resolve the collisions by using a recursive method that modifies the spline curve as discussed in Section III-C. Once a collision-free trajectory is computed, we further perform local refinement (refer to Section III-D) to satisfy the constraints corresponding to velocity and acceleration bounds.

B. Spline Interpolation

We first sample $m+1$ points along $\mathbf{u}(t)$ at $t = t_0, ..., t_m$, where $t_0 = t_a, t_m = t_b$. The velocities at the two end-points are given as $\mathbf{u}(t_a^-)'$ and $\mathbf{u}'(t_b^+)$, which can be computed from the trajectory segments adjacent to $\mathbf{u}(t_a, t_b)$ and are used to guarantee C^1 continuity on the boundaries of resulting spline. We use an interpolation scheme, i.e. compute a cubic B-spline $\mathbf{s}(t)|_{t_a\leq t\leq t_b}$ that interpolates the sampled points, i.e. $\mathbf{s}(t_i) = \mathbf{u}(t_i), \forall i \in \{0,1,...,m\}$, $\mathbf{s}'(t_0) = \mathbf{u}'(t_0)$ and $\mathbf{s}'(t_m) = \mathbf{u}'(t_m)$.

According to [4], the C^2-smooth interpolatory cubic spline can be constructed by solving a linear system:

$$\tilde{\mathbf{A}}\tilde{\mathbf{d}} = \tilde{\mathbf{r}}$$

$$\tilde{\mathbf{A}} = \begin{pmatrix} 1 & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & \\ & & & \ddots & & \\ & & & \alpha_{m-1} & \beta_{m-1} & \gamma_{m-1} \\ & & & & & 1 \end{pmatrix} \tag{2}$$

$$\tilde{\mathbf{d}} = \begin{pmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{m-1} \\ \mathbf{d}_m \end{pmatrix} \text{ and } \tilde{\mathbf{r}} = \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{m-1} \\ \mathbf{r}_m \end{pmatrix},$$

where $\tilde{\mathbf{d}}$ is the vector of unknown variables for the de Boor control points; the parameters $\alpha_k, \beta_k, \gamma_k$ are function of the knot values $\{t_i\}_{i=0}^{m}$; $\mathbf{r}_k$'s are computed based on $\{\mathbf{u}(t_i)\}_{i=0}^{m}$, $\mathbf{u}'(t_0)$ and $\mathbf{u}'(t_1)$. For uniform cubic B-spline, $\alpha_k = 1, \beta_k = 4, \gamma_k = 1, \mathbf{r}_k = 6\mathbf{u}(t_k)$, for $1 \leq k \leq m-2$; $\alpha_k = \frac{3}{2}, \beta_k = \frac{7}{2}, \gamma_k = \frac{3}{2}, \mathbf{r}_k = \mathbf{u}'(t_k)$, for $k = 0$ or $m-1$. $\mathbf{r}_0$ and $\mathbf{r}_m$ are used to impose C^1 smoothness at interval boundaries. A similar linear system can also be computed for non-uniform B-splines [4].

The solution to the linear system is represented as $\{\mathbf{d}_k\}_{k=0}^{m}$. Along with two additional points $\mathbf{d}_{-1} = \mathbf{u}(t_0)$ and $\mathbf{d}_{m+1} = \mathbf{u}(t_m)$, these points constitute the de Boor control point sets for the interpolatory spline $\mathbf{f}^{t_0,\dots,t_m}_{\mathbf{d}_{-1},\dots,\mathbf{d}_m,\mathbf{d}_{m+1}}(t)$ that is used to represent the trajectories.

For the translational motion, $\mathbf{f}(t)$ is exactly the spline curve that we need in 3D. For rotational motion, spline $\mathbf{f}(t)$ corresponds to $\mathbf{w}(t)$ in Equation 1, and we need to transform it into a spline in $\mathcal{SO}(3)$ via exponential mapping. The combination of translation and rotation curves constitutes the motion spline curve $\mathbf{s}(t)$.

We can also change the boundary conditions of Equation 2 to achieve G^2 continuity instead. According to [3], two curves $\mathbf{q}(s), s \in [0,1]$ and $\mathbf{r}(t), t \in [0,1]$ meet with G^2 continuity at $\mathbf{q}(1) = \mathbf{r}(0)$ if and only if there exist real numbers η_1 and η_2

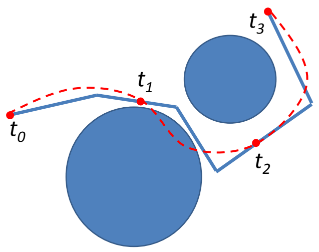

Fig. 2. For cubic B-spline $\mathbf{f}^{t_0,\dots,t_3}(t), t \in [t_0, t_3]$, a collision happens during the interval $[t_1, t_2]$. We recursively add new knots within the interval and refine the spline $\mathbf{f}^{t_1,t'_1,t'_2,t_2}(t), t \in [t_1, t_2]$ (shown on the right). The splines defined in the intervals $[t_0, t_1]$ and $[t_2, t_3]$ are unchanged based on this local refinement scheme. The modified spline is collision-free in $[t_0, t_3]$.

so that

$$\mathbf{r}'(0) = \eta_1 \mathbf{q}'(1) \quad (3)$$
$$\mathbf{r}''(0) = \eta_1^2 \mathbf{q}''(1) + \eta_2 \mathbf{q}'(1). \quad (4)$$

In other words, the two curves should have common unit tangent and curvature vectors at the shared boundary point. Notice that when $\eta_1 = 1$ and $\eta_2 = 0$, we have the condition for C^2 continuity.

For cubic B-spline in Equation 2, the G^2 boundary condition can be formalized as

$$\begin{aligned} \frac{1}{2}(\mathbf{d}_1 - \mathbf{d}_{-1}) &= \eta_1 \mathbf{u}'(t_a^-) \\ \mathbf{d}_{-1} - 2\mathbf{d}_0 + \mathbf{d}_1 &= \eta_1^2 \mathbf{u}''(t_a^-) + \eta_2 \mathbf{u}'(t_a^-) \\ \frac{1}{2}(\mathbf{d}_{m-2} - \mathbf{d}_m) &= \eta_3 \mathbf{u}'(t_b^+) \\ \mathbf{d}_m - 2\mathbf{d}_{m-1} + \mathbf{d}_{m-2} &= \eta_3^2 \mathbf{u}''(t_b^+) + \eta_4 \mathbf{u}'(t_b^+). \end{aligned} \quad (5)$$

As a result, to achieve an interpolated spline with G^2 continuity, we only need to construct a new algebraic system by replacing the C^1 boundary condition in Equation 2 with the new G^2 boundary condition. Unlike the original linear system, the new system is (1) nonlinear due to η_1^2 and η_3^2 terms; (2) under-determined with two additional degrees of freedom. To solve the system, we use additional constraints, such as maximizing η_1 and η_3 (i.e. try to be a C^2 spline if possible) or use velocity/acceleration constraints. Finally we solve a quadratic optimization problem to obtain all the control points.

C. Recursive Spline Modification to Resolve Collision

After $\mathbf{s}(t)$ is computed, we check whether all the configurations belonging to $\mathbf{s}(t)$ are collision free. The collision checking algorithm is presented in Section IV. If a collision is found, we resolve it by modifying the spline curve. As shown in Fig 2, our solution is based on the locality of spline curves. Suppose that the collision between the B-spline curve and the obstacles corresponds to a parameter that lies within the interval $[t_i, t_j]$, our goal is to keep the other portions of the B-spline unchanged, as they represent collision-free portions of the spline. Therefore, we only sample more parameter values within $[t_i, t_j]$ and use them to pull the spline near the original piecewise linear curve $\mathbf{u}(t)$, which is collision-free. The modified spline segment may no longer be C^2, but we refine it using the boundary condition in Equation 5 so that the new spline is guaranteed to be G^2.

The above procedure can be applied recursively whenever a collision is detected within any sub-interval. However, we only use this recursive formulation two or three times. If the collision is not resolved after a few recursive steps, the algorithm returns failure for the current shortcut attempt. If the algorithm is unable to compute a collision-free path, it changes the number of sample points along $\mathbf{u}(t)$ and repeats the process. This is similar to generating new samples for sample-based planning and use them to compute a collision-free piecewise trajectory.

For the other collision-free spline intervals, we keep them unchanged. However, the original spline has been divided into several parts. We still need the spline representation for each part in the local optimization step in Section III-D. Therefore, we use the spline subdivision technique [4] to compute the $m+3$ new de Boor points $\tilde{\mathbf{d}}_{-1}, ..., \tilde{\mathbf{d}}_{m+1}$ for each spline that is defined on the collision-free intervals. The final output is a G^2 collision-free trajectory.

D. Satisfying Velocity and Acceleration Constraints

The modified trajectory $\mathbf{s}(t)$ also needs to satisfy other constraints, e.g. velocity constraints and acceleration constraints, as defined in Section II-A. In our approach, we consider them as soft constraints.

Our solution depends on the properties of B-spline which iteratively adjusts the de Boor control points successively to refine the set of control points so that the resulting spline curve can satisfy these constraints. For convenience, from now on we assume the spline is a uniform B-spline and the similar formulation can be derived for non-uniform splines.

We first discuss how to constrain the velocity bound. For a uniform cubic B-spline $\mathbf{f}_{\mathbf{d}_0,\dots,\mathbf{d}_m}^{t_0,\dots,t_{m-2}}(t)$, the derivatives at the i-th knot, and the midpoint between i-th knot and $i+1$-th knot are:

$$\begin{aligned} \mathbf{f}'(t_i) &= \frac{\mathbf{d}_{i+2} - \mathbf{d}_i}{2A} \\ \mathbf{f}'(\frac{t_i + t_{i+1}}{2}) &= \frac{\mathbf{d}_{i+3} + 5\mathbf{d}_{i+2} - 5\mathbf{d}_{i+1} - \mathbf{d}_i}{8A}, \end{aligned}$$

where $A = t_{i+1} - t_i$.

We define the limit velocity as the velocity at time $\tilde{t}$ when $\mathbf{f}''(\tilde{t}) = 0$. The limit velocity $\tilde{\mathbf{V}}$ will be the maximum/minimum velocity on one interval $[t_a, t_b]$. The limit velocity within the interval $[t_i, t_{i+1}]$ is:

$$\begin{aligned} \tilde{\mathbf{V}}[t_i, t_{i+1}] &= \frac{1}{2A}(\mathbf{d}_{i+2} - \mathbf{d}_i + \frac{(\mathbf{d}_{i+2} - 2\mathbf{d}_{i+1} + \mathbf{d}_i)^2}{-\mathbf{d}_{i+3} + 3\mathbf{d}_{i+2} - 3\mathbf{d}_{i+1} + \mathbf{d}_i}) \\ &= \mathbf{f}'(t_i) + \frac{1}{8}\frac{(4\mathbf{f}'(\frac{t_i+t_{i+1}}{2}) - \mathbf{f}'(t_{i+1}) - 3\mathbf{f}'(t_i))^2}{2\mathbf{f}'(\frac{t_i+t_{i+1}}{2}) - 2\mathbf{f}'(t_{i+1}) - \mathbf{f}'(t_i)}. \end{aligned}$$

From this equation, we can find that $\tilde{\mathbf{V}}[\mathbf{t_i}, \mathbf{t_{i+1}}]$ is uniquely determined by the velocity at the beginning, ending, and middle knots of the interval $[t_i, t_{i+1}]$. As a result, if we can bound the magnitudes of $\{\mathbf{f}'(t_i)\}_{i=0}^{m-2}$ and $\{\mathbf{f}'(\frac{t_i+t_{i+1}}{2})\}_{i=0}^{m-3}$, we can control the magnitude of velocity of spline $\mathbf{f}'(t)$, for all $t \in [t_0, t_{m-2}]$.

Therefore, we check the bounds on $\mathbf{f}'(t_i)$ and $\mathbf{f}'(\frac{t_i+t_{i+1}}{2})$ for i from 1 to $m-3$. If $|\mathbf{f}'(t_i)|$ or $|\mathbf{f}'(\frac{t_i+t_{i+1}}{2})|$ is larger than $\mathbf{v}_{max}$, we reduce the corresponding $\mathbf{d}_{i+2}$ to $\lambda \mathbf{d}_{i+2}$ with $0 < \lambda < 1$. We repeat this procedure several times until the limit velocity $\tilde{\mathbf{V}}$ is constrained within the given velocity bound. Notice that the iteration will not influence $\mathbf{f}'(t_0)$ and $\mathbf{f}'(t_{m-2})$ at the boundaries of the spline.

The bound on the acceleration is relatively simple to satisfy, because the acceleration of the entire cubic spline is a linear function. The acceleration at the i-th knot is $\mathbf{f}''(t_i) = \frac{\mathbf{d}_{i+2}-2\mathbf{d}_{i+1}+\mathbf{d}_i}{A^2}$. We only need to adjust $\{\mathbf{d}_i\}_{i=3}^{m-3}$ to make sure the acceleration is bounded by $\mathbf{a}_{max}$. Whenever the de Boor control points are adjusted, we need to perform continuous collision query to ensure that the modified spline is also collision-free.

Overall, our spline fitting and refinement schemes tend to maintain second order geometric continuity and ensure that the computed trajectories are curvature-continuous. As a result, if our algorithm is able to compute a collision-free path, it satisfies the following properties:

Theorem 1: (Trajectory Smoothness Theorem) The trajectory in $\mathcal{C}$, composed by multiple cubic B-splines, is G^2 continuous.

IV. Exact Collision Detection

Collision checking is an integral component of any smoothing algorithm. Many optimization-based smoothing algorithms can't guarantee computation of collision-free paths, especially when the environment has a high number of obstacles or tight narrow passages. In case of shortcut algorithms, the choice of smoothing function is also restricted by the complexity of performing reliable collision checking along the trajectory.

The simplest algorithms for collision checking compute discrete samples along a path or trajectory and check them for collisions. However, the resulting techniques can miss collisions due to poor sampling, In contrast, continuous collision detection (CCD) methods check for collisions between a robot and obstacles when the robot moves along a given motion curve $\mathbf{f}(t)$. This can overcome the inaccuracy or the tunneling problem of discrete collision detection. However, prior CCD algorithms are mainly limited to linear trajectories in configuration space, or screw motion or parabolic arcs [22, 29] and no fast methods are known for CCD along arbitrary spline trajectories.

A. Efficient Spline Collision Detection Algorithm

Our SCD (spline collision detection) algorithm is based on *conservative advancement* (CA) [22]. Fig. 3 shows an overview of CA method. Suppose we are given two convex objects $\mathcal{A}$ and $\mathcal{B}$,where $\mathcal{A}$ is moving and $\mathcal{B}$ is fixed. Denote $\mathcal{A}(t)$ as $\mathcal{A}$ at time $t \in [0,1]$. The basic idea of CA is to incrementally advance $\mathcal{A}$ by a small time step Δt_k toward $\mathcal{B}$ while avoiding collision. In order to perform this step, we need to compute the minimal separation distance between $\mathcal{A}$ and $\mathcal{B}$. Let $d(\mathcal{A}(t), \mathcal{B})$ represent the distance and $\mathbf{n}$ be the direction of the closest vector. Then an upper bound μ on the motion of $\mathcal{A}(t)$ projected onto the direction $\mathbf{n}$ is estimated. More specifically, a *motion bound* μ for the moving object $\mathcal{A}$ within $[t, t+\Delta t]$ with respect to the given direction $\mathbf{n}$ is defined as an upper bound to $\frac{1}{\Delta t}\max_{\mathbf{p}\in\mathcal{A}}\int_t^{t+\Delta t}|\mathbf{p}'(s)\cdot\mathbf{n}|ds$. Finally, the advancing length at the step k is calculated by:

$$\Delta t_k = \frac{d(\mathcal{A}(t), \mathcal{B})}{\mu}.$$

$\mathcal{A}$ advances by Δt_k each step until $d(\mathcal{A}(t), \mathcal{B})$ is small enough. If $\tau = \sum_k \Delta t_k$ is equal to 1, the given path is collision-free. Otherwise, τ is the first time of contact.

Algorithm 2: Spline Continuous Collision Detection

```
Input  : Two objects $\mathcal{A}$ and $\mathcal{B}$; $\mathcal{A}$'s spline motion
         function $\mathbf{f}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}^{t_0,\ldots,t_{m-2}}(t)$
Output: collision-free or the first collision time $\tau$
begin
    $t \leftarrow 0$
    // Repeat CCD for each spline segment
    for $i = 1$ to $m-2$ do
        while $t \le t_i$ do
            Compute current distance $d(\mathcal{A}(t), \mathcal{B})$
            Estimate the motion bound $\mu$
            Compute conservative advancement
            $\Delta t = \frac{d(\mathcal{A}(t),\mathcal{B})}{\mu}$
            $t \leftarrow t + \Delta t$
            Check collision between $\mathcal{A}(t)$ and $\mathcal{B}$
            if collision then
                return $\tau = t$
    return collision-free
end
```

We apply the CA method to B-spline trajectories. Algorithm 2 shows the overall algorithm SCD. Lets assume $\mathcal{A}$'s trajectory is a B-spline with $m-2$ segments $\mathbf{f}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}^{t_0,\ldots,t_{m-2}}(t)$. Our algorithm performs collision checking for each segment iteratively.

The main challenge in terms of applying CA continuous collision detection is to find a tight motion bound μ for the given motion trajectory. Tang et al. [22] present the motion bounds for linear and screw motion in C-space. In our case, we need to find an appropriate motion bound for the spline motion. Its translation part $\mathbf{T}(t)$ is a B-spline in $\mathcal{R}^3$; its rotation part $\mathbf{R}(t)$ is represented as $\mathbf{w}(t)$ using exponential map, which is also a B-spline in $\mathcal{R}^3$.

B. Motion Bound for Cubic B-Splines

Our goal is to compute a tight motion bound for the moving object $\mathcal{A}$. We first present a theorem of computing the motion bound for a point $\mathbf{p}$ on the moving object $\mathcal{A}$.

Theorem 2: Suppose $\mathbf{p}$ *is a point on the rigid body with local coordinate* $\mathbf{r}$ *with respect to the origin.* $\mathbf{p}$ *moves according to spline motion function* $\mathbf{p}(t) = \mathbf{T}(t) + \mathbf{R}(t)\mathbf{r}$. *Within*

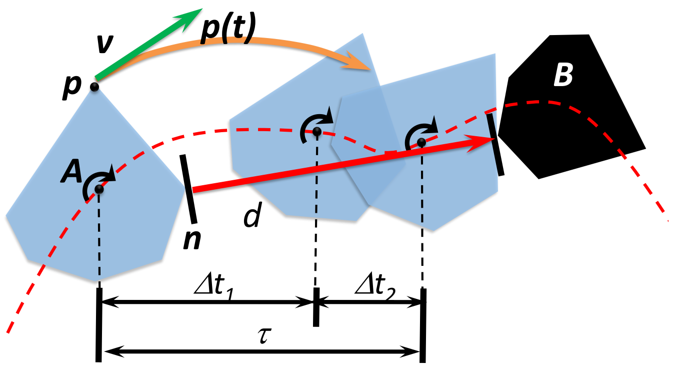

Fig. 3. Continuous collision detection between $\mathcal{A}(t)$ and $\mathcal{B}$. $\mathcal{A}(t)$ is a spline motion. $\mathbf{p}$ is a point on $\mathcal{A}$. $\mathbf{p}$'s motion function is $\mathbf{p}(t)$ and $\mathbf{v}(t) = \mathbf{p}'(t)$ is the velocity. d is the closest distance between two objects and $\mathbf{n}$ is the direction of closest distance. $\tau = \Delta t_1 + \Delta t_2$ is the first time of contact. Δt_1 and Δt_2 correspond to the advancements during two successive steps.

$[t, t + \Delta t]$, a motion bound $\mu_\mathbf{p}$ for $\mathbf{p}$ with respect to the direction $\mathbf{n}$ is given as:

$$\mu_\mathbf{p} = A \cdot W_1 + B \frac{W_1}{W_2} + W_3,$$

where $A = |\mathbf{r} \cdot \mathbf{n}| + 2|\mathbf{r} \times \mathbf{n}| + 2|\mathbf{r}|$, $B = |\mathbf{r} \times \mathbf{n}| + 8|\mathbf{r}|$, $W_1 = \max_{s\in[t,1]} |\mathbf{w}'(s)|$, $W_2 = \max_{s\in[t,1]} |\mathbf{w}(s)|$ and $W_3 = \max_{s\in[t,1]} |\mathbf{T}'(s)|$.

Proof: See Appendix A. ■

We extend the spline motion bound to non-convex objects by using the swept sphere volume (SSV) hierarchy technique in [22]. SSV is one type of bounding volume for collision acceleration, which has a sphere radius parameter r and several medial axis parameters $\{\mathbf{c}_i\}_{i=1}^n$, where n can be 1, 2 or 3. Given a SSV, any point $\mathbf{p}$ in that SSV can be represented as $\mathbf{p} = r\mathbf{k} + g(\{\mathbf{c}_i\})$, where k is a unit vector and $g(\cdot)$ is a linear function of $\{\mathbf{c}_i\}$. Ultimately, we have following result about motion bound for the entire SSV:

Theorem 3: The motion bound for one SSV along the B-spline trajectory is given as

$$\mu_\alpha = (5r + \max_k |\mathbf{c}_k \cdot \mathbf{n}| + 2\max_k |\mathbf{c}_k \times \mathbf{n}| + 2\max_k |\mathbf{c}_k|)W_1 + (9r + \max_k |\mathbf{c}_k \times \mathbf{n}| + 8\max_k |\mathbf{c}_k|)\frac{W_1}{W_2} + W_3,$$

where the symbols are of the same meaning as in Theorem 2.

Proof: See Appendix A. ■

Given a complex non-convex object, we represent it using a hierarchy of SSVs and compute the motion bound for the non-convex object along that trajectory. We then use the hierarchy to compute the conservative time step. Eventually, we can test whether the robot traversing along a B-spline trajectory collides with any obstacle.

V. Results

In this section, we highlight the performance of our spline-based smoothing algorithm on different benchmarks with cluttered environments. We show both results on rigid robots and a 40-DOF articulated robot.

	Piano	Gear	Wiper	CarSeat	AlphaPuzzle	Bridge
#DOF	6	3	6	6	6	40
#face	952	7,188	26,766	245,127	2,088	31,718

TABLE I
Geometric complexity of our benchmarks.

A. Rigid Robots

The results of five different benchmarks on rigid robots are shown in Figs. 4, 5, and 6. Table I summarizes the geometric complexity and DOFs for each benchmark. Among the five benchmarks, only the piano benchmark doesn't have narrow passages. The rest of the benchmarks are challenging for sample-based motion planners and smoothing algorithms. We use a variant of sample-based motion planner [15] to compute an initial piecewise linear trajectory and apply our B-spline smoothing algorithms to these trajectories.

For each benchmark, we compare the initial piecewise-linear trajectory, new trajectory computed using linear shortcut smoothing algorithm [24] and the new trajectory using our spline-based algorithm. In particular, we visualize these trajectories for the local coordinate origin of each rigid robot (Figs. 4, 5, and 6). As compared to the linear shortcut algorithm, our spline-based smoothing algorithm computes curvature-continuous trajectories. Moreover, for the gear benchmark (Fig 4), we show the swept volume of the robot generated using the trajectories computed by different algorithms.

For the piano benchmark (Fig 5), we also visualize the rotation curves using quaternions. In this case, the distance to the sphere is the angle and the direction of the sphere center is the axis. The curve corresponding to the rotational motion, generated by the original sample-based planner, is not smooth. The linear shortcut algorithm results in a lot of corners and our spline-based smoothing algorithm results in a smooth trajectory.

We show the timing results in Table II. All methods perform 500 shortcut operations on different benchmarks. For each shortcut operation in linear shortcut and spline shortcut, we check collisions by using $30\frac{N|t_b - t_a|}{T}$ discrete samples, where N, t_b, t_a, T are introduced in Algorithm 1. We notice that the computational overhead of our spline-based shortcut algorithm over the linear shortcut method is rather small. Since the discrete collision checking algorithm is not reliable, we use our SCD algorithm and integrate it with the spline shortcut framework. The conservative advancement based SCD algorithm is significantly faster than prior methods used. This follows from the inherent efficiency of conservative advancement method over other CCD algorithms. The overall smoothing algorithm with SCD is comparable (or faster) than prior methods.

B. Extension to Articulated Robots

We extend our spline-based smoothing method to articulated robots. Consider a robot with n serial links. We perform the spline interpolation for each link by treating it as a rigid robot.

	Piano	Gear	Wiper	CarSeat	AlphaPuzzle	Bridge
Linear shortcut	59.92	8.533	24.06	19.98	10.533	163
Spline shortcut + Discrete collision detection	43.53	9.781	33.09	51.39	15.781	185
Spline shortcut + Spline collision detection	24.51	5.453	4.69	28.31	17.16	not implemented

TABLE II
TIMING OF SHORTCUT ALGORITHMS IN SECONDS. THE FIRST TWO ROWS PERFORM DISCRETE COLLISION CHECKING AND NEED TO CONSIDER A VERY HIGH NUMBER OF SAMPLES. THE LAST ROW USES OUR NOVEL SPLINE COLLISION DETECTION (SCD) ALGORITHM AND IT WORKS WELL IN CLUTTERED ENVIRONMENTS.

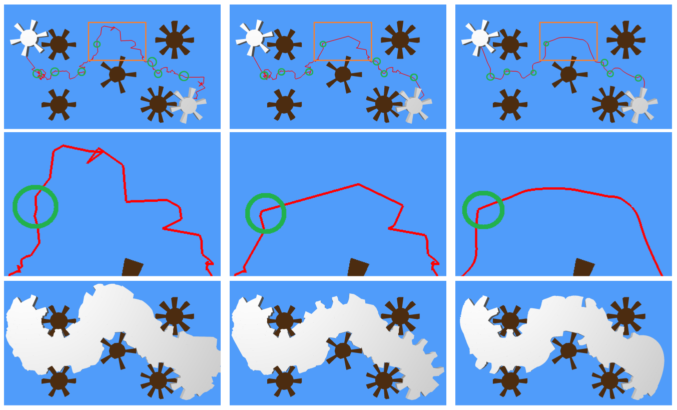
Fig. 4. Results for the gear benchmark. From left to right are results computed by planner, linear shortcut and our spline smoothing algorithm. The first row shows the trajectories traversed by the origin of the robot; the second row shows a zoomed view of the trajectories within the orange box in the first row; the third row shows the swept volumes for the trajectories computed by different methods. We use green circles to show the narrow passages in the configuration space. Our smoothing algorithm computes a $\mathcal{F}^2$ trajectory for most of the path, though it can be $\mathcal{C}^0$ in very cluttered areas.

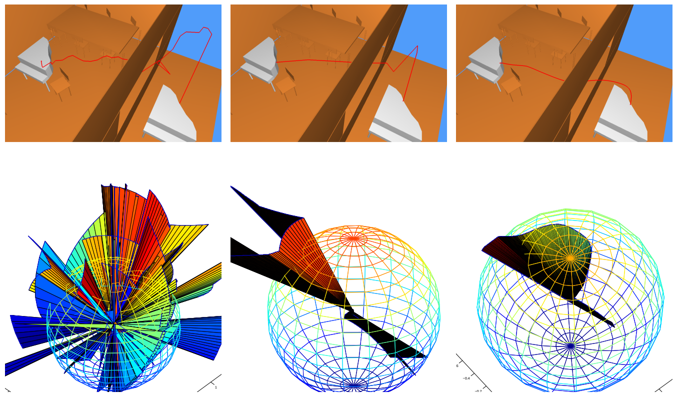
Fig. 5. Results for piano benchmark. From left to right, we show the paths computed by the sample-based planner, linear shortcut algorithm and our method, respectively. The first row shows the trajectory traversed by the origin of the robot. The second row visualizes the rotation motion of the robot obtained by different methods. Each vector originating from the sphere center represents a rotation with its magnitude for angular velocity and its direction for rotation axis. Our method can compute smooth rotational motions.

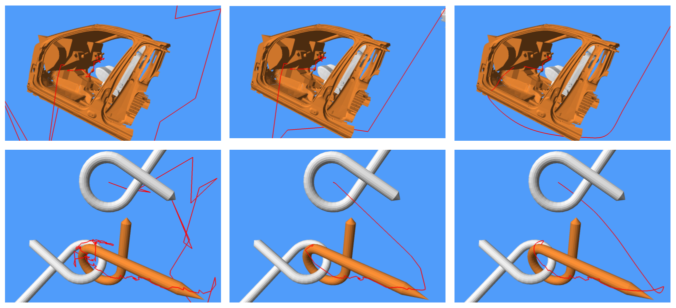
Fig. 6. Rigid robot benchmarks: car seat and alpha puzzle. From left to right, we show the results computed by a sample-based planner, the smooth result computed using linear shortcut and the smooth trajectory generated by our algorithm. We show the path in the workspace traversed by the origin of the rigid robot.

Let $\mathbf{T}_i^{i-1}$ and $\mathbf{R}_i^{i-1}$ be the translation and rotation for the link $\mathcal{L}_i$ relative to its parent $\mathcal{L}_{i-1}$. Then the relative motion for the link $\mathcal{L}_i$ to its parent in $\mathcal{S}(3)$ can be represented by two splines $\mathbf{T}_i^{i-1}(t)$ and $\exp(\mathbf{w}_i^{i-1}(t))$. Given the initial piecewise path for translation or rotation motion of a link, we construct cubic B-splines by using the spline interpolation scheme described in SectionIII-B. Note that in Section III-C, the matrix $\tilde{\mathbf{A}}$ in the linear system is same for any link of the articulated robot, because it solely depends on the chosen knots t_i of the cubic B-spline. So we can pre-compute the LU-decomposition of $\tilde{\mathbf{A}}$ and then reuse it to compute the B-spline trajectory for each link.

The exact spline collision checking (SCD) algorithm (Section IV) can be also extended to articulated robots. The underlying conservative advancement technique works well as long as the motion bounds can be estimated for articulated robots. We compute the motion bound for each rigid component using Theorem 3 and apply the technique used in [29] to estimate a tight motion bound for the articulated robot.

Fig. 7 shows a 40-DOF hyper-redundant (HRR) articulated robot with free rotation joints. The goal is to compute a path such that the robot goes through a hole and brackets of a bridge to perform the inspection. We use a variant of sample-based planner to compute an initial path and apply our spline-based algorithm to smooth the trajectory. Fig. 7 shows the intermediate configurations on the trajectory of the robot. As compared to the linear shortcut method, our method can compute a shorter and smoother trajectory.

VI. Analysis and Comparisons

There is extensive work on path smoothing in robotics, control and related areas, as highlighted in Section I. We use cubic B-splines for trajectory computation as it provides sufficient degrees of freedom to compute curvature-continuous trajectories and handle cluttered environments. Furthermore,

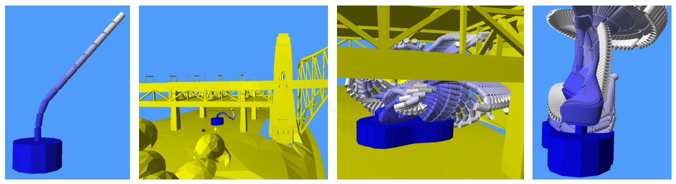

Fig. 7. Results for the 40-DOF hyper-redundant articulated model. The left two subfigures show the robot and the bridge inspection scenario. In the right three subfigures, we show the results computed by a sample-based planner and the smooth result computed by our algorithm.

we are able to perform fast and reliable continuous collision checking. None of the prior methods provide all these features or capabilities. Most of the prior shortcut algorithms use linear shortcuts or parabolic curves [7, 24], but can't provide high order smoothness or curvature continuity. Furthermore, it is not clear whether these methods would work in narrow passages or cluttered environments. Recently, a curvature-continuous trajectory computation algorithm has been proposed [25]. This approach uses Bézier curves to construct G^2 smooth, but the formulation appears to be limited to a point robot navigating in a 2D or 3D workspace.

There are many optimization-based methods, such as CHOMP [18], which have many useful properties and work well on robotic systems. Some of these methods, don't even need collision-free piecewise linear trajectories and can also model dynamics constraints. However, one challenge is to compute collision-free trajectories when there are a high number of obstacles or narrow passages.

Our spline continuous collision detection algorithm is the first reliable algorithm that can check for collisions along spline trajectories. The resulting formulation based on conservative advancement can also be applied to other trajectory formulations. Furthermore, our continuous collision detection algorithm is much faster than prior exact collision checking methods.

VII. Conclusions and Future Work

We present a trajectory smoothing algorithm using cubic B-splines. Our formulation can compute curvature-continuous trajectories and also works well in cluttered environments. We also describe a fast and reliable continuous collision detection algorithm along spline trajectories.

There are many avenues for future work. We would like to combine our approach with gradient optimization techniques, such as CHOMP [18], to perform path refinement on our trajectories. It may be useful to extend our approach to take into account kinematic and dynamics constraints and integrate the algorithm with robotic systems. Finally, we can also design fast and reliable collision detection algorithms, similar to SCD, for other trajectory formulations or optimization-based smoothing algorithms.

Acknowledgements This work was supported in part by ARO Contract W911NF-10-1-0506, NSF awards 0917040, 0904990 and 1000579, and RDECOM Contract WR91CRB-08-C-0137. L. Zhang was supported in part by NSF Computing Innovation Fellowship under Grant #0937060 to the Computing Research Association.

Appendix A
Motion Bound for cubic B-spline Motion

The motion of a rigid body contains two parts: the translation or the rotation. Both are modeled as spline. The translation is $\mathbf{T}(t)$ and the rotation is $\mathbf{R}(t)$. Then for one point $\mathbf{p}_i$ on the geometry, its position at time t is $\mathbf{p}_i(t) = \mathbf{T}(t) + \mathbf{R}(t)\mathbf{r}_i = \mathbf{T}(t) + q(t)\mathbf{r}_i q(t)^{-1}$. Here $\mathbf{r}_i$ is point $\mathbf{p}_i$'s coordinate in local frame, $q(t) = \exp(\mathbf{w}(t)) = \cos\theta(t) + \hat{\mathbf{u}}\sin\theta(t)$ with $\mathbf{w}(t) = \mathbf{u}(t)\theta(t)$ while $\mathbf{u}(t)$ is the rotation axis and $\theta(t)$ is the rotation angles. $\mathbf{w}(t)$ and $\mathbf{T}(t)$ are both spline curves in $\mathcal{R}^3$. The explicit form for $\mathbf{w}(t)$ in one segment is $w(t) = \mathbf{d}_0\frac{1-3t+3t^2-t^3}{6} + \mathbf{d}_1\frac{4-6t^2+3t^3}{6} + \mathbf{d}_2\frac{1+3t+3t^2-3t^3}{6} + \mathbf{d}_3\frac{t^3}{6} = \mathbf{d}_0\lambda_0 + \mathbf{d}_1\lambda_1 + \mathbf{d}_2\lambda_2 + \mathbf{d}_3\lambda_3$, where $\lambda_k(t) \geq 0, \forall k, \forall t \in [0,1]$ and $\sum_k \lambda_k = 1$. Explicit form for $\mathbf{T}(t)$ is similar.

To use the CCD algorithm in [22], we need to estimate an upper bound for $\max_i \int_t^{t+\Delta t} |\dot{\mathbf{p}}_i(\tau)\cdot\mathbf{n}|d\tau$ in the form of $\Delta t \cdot \mu$, where t is the current progress of conservative advancement procedure, $\mathbf{n}$ is the closest direction between rigid body and the obstacles, $\mu = \mu(t)$ is the motion bound function that does not depend on Δt.

We first estimate the bound for one point $\mathbf{p}_i$:

$$\begin{aligned} & \int_t^{t+\Delta t} |\dot{\mathbf{p}}_i(\tau)\cdot\mathbf{n}|d\tau \\ \leq\; & \int_t^{t+\Delta t} |\dot{\mathbf{T}}(\tau)\cdot\mathbf{n}|d\tau + \int_t^{t+\Delta t} |(q(\tau)\mathbf{r}_i q(\tau)^{-1})'\cdot\mathbf{n}|d\tau \end{aligned}$$

We first compute the second item. After some computation, we have

$$\begin{aligned} & (q(\tau)\mathbf{r}_i q(\tau)^{-1})' \\ =\; & -\sin 2\theta\ \theta'\mathbf{r}_i - 2\cos 2\theta\ \theta'\mathbf{r}_i\times\mathbf{u} + 2\sin 2\theta\ \theta'(\mathbf{u}\cdot\mathbf{r}_i)\mathbf{u} \\ & + \sin 2\theta\mathbf{u}'\times\mathbf{r}_i + 2\sin^2\theta((\mathbf{u}'\cdot\mathbf{r}_i)\mathbf{u} + (\mathbf{u}\cdot\mathbf{r}_i)\mathbf{u}') \end{aligned}$$

Then the bound for rotation part is

$$\begin{aligned} & \int_t^{t+\Delta t} |(q(\tau)\mathbf{r}_i q(\tau)^{-1})'\cdot\mathbf{n}|d\tau \\ \leq\; & \int_t^{t+\Delta t} |\theta'||\mathbf{r}_i\cdot\mathbf{n}| + 2|\theta'||(\mathbf{r}_i\times\mathbf{n})\cdot\mathbf{u}| + 2|\theta'||(\mathbf{u}\cdot\mathbf{r}_i)(\mathbf{u}\cdot\mathbf{n})| \\ & + |(\mathbf{r}_i\times\mathbf{n})\cdot\mathbf{u}'| + 2|(\mathbf{u}'\cdot\mathbf{r}_i)(\mathbf{u}\cdot\mathbf{n}) + (\mathbf{u}\cdot\mathbf{r}_i)(\mathbf{u}'\cdot\mathbf{n})|d\tau \end{aligned}$$

To estimate an upper bound for $|(\mathbf{u}'\cdot\mathbf{r}_i)(\mathbf{u}\cdot\mathbf{n}) + (\mathbf{u}\cdot\mathbf{r}_i)(\mathbf{u}'\cdot\mathbf{n})|$, we first give the formulation for $\mathbf{u}'$ based on $\mathbf{u} = \frac{\mathbf{w}}{|\mathbf{w}|}$:

$$\mathbf{u}' = \frac{|\mathbf{w}|\mathbf{w}' - |\mathbf{w}|'\mathbf{w}}{|\mathbf{w}|^2} = \frac{|\mathbf{w}|\mathbf{w}' - \frac{\mathbf{w}\cdot\mathbf{w}'}{|\mathbf{w}|}\mathbf{w}}{|\mathbf{w}|^2}.$$

Moreover, the upper bound for $|\mathbf{u}'|$ can be obtained via

$$|\mathbf{u}'|^2 = \frac{|\mathbf{w}|^2|\mathbf{w}'|^2 - |\mathbf{w}\cdot\mathbf{w}'|^2}{|\mathbf{w}|^4} \leq \frac{|\mathbf{w}'|^2}{|\mathbf{w}|^2}.$$

As a result, we have the estimation $|(\mathbf{u}'\cdot\mathbf{r}_i)(\mathbf{u}\cdot\mathbf{n}) + (\mathbf{u}\cdot\mathbf{r}_i)(\mathbf{u}'\cdot\mathbf{n})| \leq 4|\mathbf{r}_i|\frac{|\mathbf{w}'|}{|\mathbf{w}|}$. Moreover, as $\mathbf{w} = \mathbf{u}\theta$, we have $\mathbf{w}' =$

$\mathbf{u}\theta' + \mathbf{u}'\theta$. We also have $\mathbf{u}\cdot\mathbf{u} = 1$, thus $\mathbf{u}\cdot\mathbf{u}' = 0$. As a result, $\mathbf{w}'\cdot\mathbf{u} = \theta'$ and $|\theta'| \leq |\mathbf{w}'|$.

Using the upper bounds estimated above and $|\mathbf{u}(t)| = 1$, we can give the estimation for rotation part as

$$\begin{aligned}
&\int_t^{t+\Delta t} |(q(\tau)\mathbf{r}_i q(\tau)^{-1})'\cdot\mathbf{n}|d\tau \\
\leq\ & (|\mathbf{r}_i\cdot\mathbf{n}| + 2|\mathbf{r}_i\times\mathbf{n}| + 2|\mathbf{r}_i|)\int_t^{t+\Delta t}|\mathbf{w}'|d\tau \\
& +(|\mathbf{r}_i\times\mathbf{n}| + 8|\mathbf{r}_i|)\int_t^{t+\Delta t}\frac{|\mathbf{w}'|}{|\mathbf{w}|}d\tau \\
=\ & A_i\int_t^{t+\Delta t}|\mathbf{w}'|d\tau + B_i\int_t^{t+\Delta t}\frac{|\mathbf{w}'|}{|\mathbf{w}|}d\tau \\
\leq\ & (A_i\max_{\tau\in[t,t+\Delta t]}|\mathbf{w}'| + B_i\frac{\max_{\tau\in[t,t+\Delta t]}|\mathbf{w}'|}{\min_{\tau\in[t,t+\Delta t]}|\mathbf{w}|})\Delta t \\
\leq\ & (A_i\max_{\tau\in[t,1]}|\mathbf{w}'| + B_i\frac{\max_{\tau\in[t,1]}|\mathbf{w}'|}{\min_{\tau\in[t,1]}|\mathbf{w}|})\Delta t \qquad (6)
\end{aligned}$$

Now we need to calculate $\max_{\tau\in[t,1]}|\mathbf{w}'(\tau)|$ and $\min_{\tau\in[t,1]}|\mathbf{w}(\tau)|$ separately.

$\max_{\tau\in[t,1]}|\mathbf{w}'|$ is simple. We have $\max_{\tau\in[t,1]}|\mathbf{w}'| = (\max_{\tau\in[t,1]}|\mathbf{w}'|^2)^{1/2}$. $m(\tau) = |\mathbf{w}'(\tau)|^2$ is a degree-4 polynomial whose maximum value can only be arrived at $\tilde{\tau}$ where $m'(\tilde{\tau}) = 0$. $m'(\tau)$ is a degree-3 polynomial, and its (at most 3) real roots can be computed analytically. These roots along with t and 1 are the 5 candidates for $\tau\in[t,1]$ that reaches the maximum value of $m(\tau)$ and thus $|\mathbf{w}'(\tau)|$. We only need to check the values at these 5 points and can obtain $W_1(t) = \max_{\tau\in[t,1]}|\mathbf{w}'|$.

Now we need to compute $\min_{\tau\in[t,1]}|\mathbf{w}|$. Of course we can follow the same way as above by computing the minimum value of $l(\tau) = |\mathbf{w}(\tau)|^2$. However, $l'(\tau)$ is a degree-5 polynomial and therefore there is no analytic solution for $l'(\tau) = 0$. We must calculate it via numerical methods. To avoid numerical root-finding, here we estimate a lower bound for $\min_{\tau\in[t,1]}|\mathbf{w}|$ based on the convex hull and subdivision properties of spline.

First, $\mathbf{d}_0,\mathbf{d}_1,\mathbf{d}_2,\mathbf{d}_3$ are the de Boor points of spline $\mathbf{w}(\tau),\tau\in[0,1]$. We can use subdivision technique to compute the de Boor points $\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t$ for sub-spline $\mathbf{w}(\tau),\tau\in[t,1]$, which is within the convex hull made by $\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t$ and thus within the tetrahedron $\triangle_{\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t}$. As a result, $\min_{\tau\in[t,1]}|\mathbf{w}|^2$ is bounded from below by the distance of $\mathbf{0}$ to points in $\triangle_{\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t}$: $\min_{\tau\in[t,1]}|\mathbf{w}|^2 \geq [d(\mathbf{0},\triangle_{\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t})]^2$. Here $d(\mathbf{0},\triangle_{\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t})$ can be simply computed according to geometry operators: if $\mathbf{0}$ is in the interior of tetrahedron, $d = 0$; otherwise it is the minimum of distances to the visible faces of tetrahedron. However, if $\mathbf{0}\in\triangle_{\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t}$, then the estimation for the rotation (Equation 6) is too loose: $\frac{\max_{\tau\in[t,1]}|\mathbf{w}'|}{\min_{\tau\in[t,1]}|\mathbf{w}|} = +\infty$. The problem is that though $\mathbf{w}(\tau)\neq\mathbf{0}$ (as $\mathbf{0}$ means no rotation), it is possible that $\mathbf{0}$ locates in the tetrahedron of four control points: the convex hull made by $\triangle_{\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t}$ is not compact enough. We can use spline subdivision to provide more compact convex hull and better bounds.

We use spline subdivision to split the spline into two parts, corresponding to the $t\in[t,\frac{t+1}{2}]$ and $t\in[\frac{t+1}{2},1]$ on original spline. Suppose the de Boor control points for the two splines are $\mathbf{a}_0,\mathbf{a}_1,\mathbf{a}_2,\mathbf{a}_3$ and $\mathbf{b}_0,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3$. $\mathbf{w}(\tau),\tau\in[t,\frac{t+1}{2}]$ is within tetrahedron $\triangle_{\mathbf{a}_0,\mathbf{a}_1,\mathbf{a}_2,\mathbf{a}_3}$ and $\mathbf{w}(\tau),\tau\in[\frac{t+1}{2},1]$ is within tetrahedron $\triangle_{\mathbf{b}_0,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3}$. We can estimate the bounds $\min_{\tau\in[t,\frac{t+1}{2}]}|\mathbf{w}|^2 \geq [d(\mathbf{0},\triangle_{\mathbf{a}_0,\mathbf{a}_1,\mathbf{a}_2,\mathbf{a}_3})]^2$ and $\min_{t\in[\frac{t+1}{2},1]}|\mathbf{w}|^2 \geq [d(\mathbf{0},\triangle_{\mathbf{b}_0,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3})]^2$. Therefore the more compact estimation is $\min_{\tau\in[t,1]}|\mathbf{w}(\tau)| \geq \min[d(\mathbf{0},\triangle_{\mathbf{a}_0,\mathbf{a}_1,\mathbf{a}_2,\mathbf{a}_3}),d(\mathbf{0},\triangle_{\mathbf{b}_0,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3})]$. If $\mathbf{0}$ is now not within $\triangle_{\mathbf{a}^0,\mathbf{a}^1,\mathbf{a}^2,\mathbf{a}^3}\cup\triangle_{\mathbf{b}^0,\mathbf{b}^1,\mathbf{b}^2,\mathbf{b}^3}$, then the estimation is acceptable. Otherwise, we can further subdivide the segments that contain $\mathbf{0}$. We repeat the procedure until a finite bound is obtained which is denoted as $W_2(t)$. The iteration will stop in finite steps because $\mathbf{0}$ is not on the curve.

Till now, we have finished the estimation for the rotation item in Equation 6. The translation item is much simpler: $\int_t^{t+\Delta t}|\dot{\mathbf{T}}(\tau)\cdot\mathbf{n}|d\tau \leq \Delta t\max_{\tau\in[t,1]}|\dot{\mathbf{T}}(\tau)|$, while $W_3(t) = \max_{\tau\in[t,1]}|\dot{\mathbf{T}}(\tau)|$ can be obtained by checking $|\dot{\mathbf{T}}(\tau)|$'s value at t, 1 and the real roots of $(|\dot{\mathbf{T}}(\tau)|^2)'$. Then we have the motion upper bound $\mu_i \triangleq A_iW_1 + B_i\frac{W_1}{W_2} + W_3$ so that $\mu_i\Delta t \geq \int_t^{t+\Delta t}|\dot{\mathbf{p}}_i(\tau)\cdot\mathbf{n}|d\tau$.

Finally we calculate the motion bound μ. Of course we can compute μ_i for every point $\mathbf{p}_i$ and then get μ directly by $\mu = \max_i\mu_i$. However, this may be time consuming for complicated geometry. Similar to [22], we build the swept sphere volume (SSV) hierarchy for the geometry and estimate the motion bounds for SSVs. Our estimation works for different types of SSV, such as PSS, LSS and RSS. We assume the sphere radius used to construct SSV is r.

For each point within one SSV α, it can be represented as $\mathbf{r}_i = \mathbf{r}^i + \mathbf{k}$, where $\mathbf{r}^i$ is a unit vector with radius r and $\mathbf{k}$ is a point in the inner medial structure of SSV. For PSS $\mathbf{k} = \mathbf{c}_1$; for LSS, $\mathbf{k} = s\mathbf{c}_1 + (1-s)\mathbf{c}_2$; for RSS $\mathbf{k} = s\mathbf{c}_1 + (1-s)\mathbf{c}_2 + t(\mathbf{c}_3 - \mathbf{c}_1)$, where $s,t\in[0,1]$ and $\mathbf{c}_k, k = 1,2,3$ are end points of α's medial structure. Then for points within α, we have $|\mathbf{r}_i\cdot\mathbf{n}| = |(\mathbf{r}^i+\mathbf{k})\cdot\mathbf{n}| \leq r + \max_k|\mathbf{c}_k\cdot\mathbf{n}|$, $|\mathbf{r}_i\times\mathbf{n}| = |(\mathbf{r}^i+\mathbf{k})\times\mathbf{n}| \leq r + \max_k|\mathbf{c}_k\times\mathbf{n}|$ and $|\mathbf{r}_i| = |\mathbf{r}^i+\mathbf{k}| \leq r + \max_k|\mathbf{c}_k|$. As a result, the motion bound μ^α for α that satisfies $\max_i\int_t^{t+\Delta t}|\dot{\mathbf{p}}_i(\tau)\cdot\mathbf{n}|d\tau \leq \mu^\alpha\Delta t$ is

$$\begin{aligned}
\mu^\alpha =\ & (5r + \max_k|\mathbf{c}_k\cdot\mathbf{n}| + 2\max_k|\mathbf{c}_k\times\mathbf{n}| + 2\max_k|\mathbf{c}_k|)W_1 \\
& +(9r + \max_k|\mathbf{c}_k\times\mathbf{n}| + 8\max_k|\mathbf{c}_k|)\frac{W_1}{W_2} + W_3.
\end{aligned}$$

Then we can use μ^α in the SSV hierarchy traverse procedure in order to calculate the μ for the whole rigid body efficiently.

References

[1] J. Biggs and W. Holderbaum, "Planning rigid body motions using elastic curves," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 20, pp. 351–367, 2008.

[2] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. Journal*

of Robotics Research, vol. 18, no. 6, pp. 1031–1052, 2002.

[3] T. D. DeRose and B. A. Barsky, "Geometric continuity, shape parameters, and geometric constructions for catmull-rom splines," *ACM Transactions on Graphics*, vol. 7, pp. 1–41, 1988.

[4] G. Farin, *Curves and surfaces for computer aided geometric design (3rd ed.): a practical guide*. San Diego, CA, USA: Academic Press Professional, Inc., 1993.

[5] E. Ferre and J.-P. Laumond, "A path planner for plm applications," in *isiCAD*, 2004.

[6] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *International Journal of Robotics Reseach*, vol. 26, no. 8, pp. 845–863, 2007.

[7] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *Proceedings of International Conference on Robotics and Automation*, 2010.

[8] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning collision-free reaching motions for interactive object manipulation and grasping," *Proceedings of Eurographics*, vol. 22, pp. 313–322, 2003.

[9] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, pp. 12(4):566–580, 1996.

[10] M.-J. Kim, M.-S. Kim, and S. Y. Shin, "A general construction scheme for unit quaternion curves with simple high order derivatives," in *SIGGRAPH*, 1995, pp. 369–376.

[11] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE International Conference on Robotics and Automation*, 2000.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006.

[13] R. Murray, Z. Li, and S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[14] G. M. Nielson, ""ν-quaternion splines for the smooth interpolation of orientations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 2, pp. 224–229, 2004.

[15] J. Pan, L. Zhang, and D. Manocha, "Retraction-based RRT planner for articulated models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[16] J. Pettre, T. Simeon, and J. Laumond, "Planning human walk in virtual environments," in *IROS*, vol. 3, 2002, pp. 3048–3053.

[17] A. Powell and J. Rossignac, "Screwbender: Smoothing piecewise helical motions," *IEEE Computer Graphics and Applications*, vol. 28, no. 1, pp. 56–63, 2008.

[18] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Proceedings of International Conference on Robotics and Automation*, May 2009.

[19] Z. Shiller and S. Dubowsky, "Global time optimal motions of robotic manipulators in the presence of obstacles," in *Proceedings of International Conference on Robotics and Automation*, vol. 1, apr. 1988, pp. 370–375.

[20] K. Shoemake, "Animating rotation with quaternion curves," *SIGGRAPH*, vol. 19, no. 3, pp. 245–254, 1985.

[21] G. Song and N. Amato, "Using motion planning to study protein folding pathways," *Journal of Computational Biology*, vol. 9, no. 2, pp. 149–168, 2002.

[22] M. Tang, Y. J. Kim, and D. Manocha, "C2a: controlled conservative advancement for continuous collision detection of polygonal models," in *Proceedings of International Conference on Robotics and Automation*, 2009, pp. 356–361.

[23] C. Tomlin, "Splining on lie groups," 1995.

[24] K. Yamane, J. J. Kuffner, and J. K. Hodgins, "Synthesizing animations of human manipulation tasks," *ACM Transactions on Graphics (SIGGRAPH 2004)*, vol. 23, no. 3, pp. 532–539, Aug. 2004.

[25] K. Yang and S. Sukkarieh, "An analytical continuous-curvature path-smoothing algorithm," *Robotics, IEEE Transactions on*, vol. 26, no. 3, pp. 561 –568, june 2010.

[26] E. Yoshida, C. Esteves, T. Sakaguchi, J.-P. Laumond, and K. Yokoi;, "Smooth collision avoidance: Practical issues in dynamic humanoid motion," in *IEEE/RSJ International Conference On Intelligent Robots and Systems (IROS)*, Beijing, China, Oct 2006, pp. 827–832.

[27] M. Zefran and V. Kumar, "Interpolation schemes for rigid body motions," *Computer-aided Design*, vol. 30, no. 3, pp. 179–189, 1998.

[28] L. Zhang, X. Huang, Y. Kim, and D. Manocha, "D-plan: Efficient collision-free path computation for part removal and disassembly," *Journal of Computer-Aided Design and Applications*, vol. 5, no. 6, pp. 774–786, 2008.

[29] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, "Continuous collision detection for articulated models using taylor models and temporal culling," *ACM Transactions on Graphics*, vol. 26, no. 3, p. 15, 2007.