Fast Smoothing of Motion Planning Trajectories using B-Splines

3 authors, including:

Jia Pan
University of North Carolina at Chapel Hill
47 PUBLICATIONS **752** CITATIONS

SEE PROFILE

Dinesh Manocha
University of Maryland, College Park
796 PUBLICATIONS **21,735** CITATIONS

SEE PROFILE

Fast Smoothing of Motion Planning Trajectories using B-Splines

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*
Videos available at http://gamma.cs.unc.edu/SPATH

***Abstract*— We present a novel trajectory smoothing algorithm to improve the collision-free paths computed by sample-based motion planners. Our approach uses cubic B-splines to represent the smooth, collision-free paths and can result in C^2 smoothness along most of the trajectory. Moreover, the algorithm performs local refinement on the control points in order to satisfy collision-free and other constraints. We also describe an efficient continuous collision detection algorithm that checks for collisions between the robot and the environment along the B-spline trajectories. We test our algorithm on different benchmarks, including rigid bodies and a high-DOF articulated robot. In practice, our method can generate high-quality trajectories efficiently, and also able to handle cluttered environments with narrow passages.**

I. Introduction

Sample-based planning algorithms such as probabilistic roadmaps (PRMs) [9] or rapidly-exploring random trees (RRTs) [12], [13] 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. 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 practice, most of these planners generate piecewise linear paths in the configuration space. The vertices of the collision-free path correspond to the samples and the edges are computed by linear or higher order interpolating motion between the samples.

In this paper, we address the problem of generating smooth collision-free paths for high DOF robots. The smoothness criteria is important for practical robots such as mobile service robots or unmanned aerial vehicles (UAV), as we need to take into account the physical limits of robot actuators and safety issues. Many other applications, including virtual prototyping and character animation, also require smooth trajectories. Due to randomization, the trajectories computed by sample-based planners may not be smooth. The resulting path may contain unnecessary turns or the velocities at the vertices may change arbitrarily.

Many techniques have been proposed in the literature to generate smooth paths. There is extensive literature on non-holonomic motion planning for car-like robots or kinodynamic planning of low-DOF robots. Other methods tend to smooth the piecewise linear paths computed using sample-based motion planners. These include shortcut algorithms [9], [7], [12], [30] that replace the linear path between two vertices by a higher order curve (e.g. parabolic segment) and check whether it is collision-free or satisfy other constraints. However, prior shortcut methods tend to ignore the obstacles and may not be able to compute collision-free paths in cluttered environments or narrow passages. Other smoothing methods perform trajectory optimization using numerical techniques [18], [23], [28]. However, it is hard to express some of the constraints, such as collision-free motion, in an analytic form for non-convex shapes. Therefore, these optimization algorithms are mostly limited to relatively simple environments. Moreover, their complexity increases with the dimension of the configuration space and the number of constraints.

Main Results: In this paper, we present a fast and simple algorithm to smooth the paths of sample-based planners by using spline interpolation. Our algorithm randomly selects a sequence of points along the original path and constructs a cubic B-spline that interpolates these points. We use the exponential map to construct smooth rotation motion in $\mathcal{SO}(3)$ and thereby handle translational as well as rotational motion. We also present a continuous collision detection algorithm based on conservative advancement to check the new paths for collisions with the environment.

As compared to prior path smoothing methods, our algorithm is relatively simple and effective. We use knowledge of piecewise linear paths to compute the shortcuts, and apply spline subdivision techniques to perform local refinement for generating collision-free trajectories.

We have implemented our spline-based path smoothing algorithm for 6-DOF rigid robots and also a 40-DOF articulated model. In practice, our approach works well and can also smooth the paths in cluttered environments and narrow passages. Furthermore, our method is generally as fast as prior shortcut algorithms.

The rest of paper is organized as follows. In Section II, we briefly survey related work. We introduce the notation and present our spline motion representation in Sec III. We present the basic spline-based smoothing algorithm in Section IV. We further improve the spline curve using local refinement in Sec V. In Section VI, we present an efficient collision checking algorithm for spline curves. We discuss the implementation and highlight the performance on different benchmarks in Section VII.

II. Related Work

Previous methods for improving the paths computed by motion planning algorithms can be classified into shortcut algorithms and optimization-based algorithms.

A. Shortcut Algorithms

Shortcut algorithms are widely used in robotics and computer animation. These algorithms iteratively replace jerky or unnatural portions of a path with shorter linear or parabolic segments [9], [7], [8], [30]. Simple velocity and acceleration constraints are further imposed over these parabolic segments [7]. Due to their simplicity, shortcut algorithms have been widely used to improve the quality of paths computed by randomized planners [6] or used as a preprocess for optimization methods.

Most of the prior shortcut algorithms tend to ignore the obstacles or the environment during the shortcut step. As a result, the proposed curves may collide with the obstacles. Moreover, prior algorithms usually only handle Euclidean configuration spaces [7].

B. Trajectory Optimization

Various optimization-based methods have been developed to improve the quality of paths or trajectories for robots or animated characters [23], [28]. Global numerical optimization methods such as gradient-based methods are employed to compute the optimal trajectory, where the optimality criteria may be defined based on execution time, total torque, energy consumption of the overall robot [2], [23], and are subject to the joint limits, velocity and acceleration constraints. However, numerical optimization methods require analytical formulation of constraints. Due to the difficulty of formulating the collision-free constraints between non-convex geometric models, other techniques such as potential fields, distance fields, or velocity dampers are used for collision avoidance [3], [5], [27], [18].

The trajectory optimization methods are often computationally intensive due to the high dimension of the optimization problem and the large number of constraints. Furthermore, they need a good initial guess and may suffer from local minima issues.

C. Smooth Curve in Non-Euclidean Space

The problem of interpolating a series of 'key-frame' or configurations with a smooth 3D trajectory is an important topic in both robotics and computer animation. Many previous algorithms use a spline-based representation of the 3D motion. Shoemake [24] uses quaternion curves to construct a spline in $\mathcal{SE}(3)$. Other algorithms [10] [11] use the explicit accumulative formulation for quaternion spline. Nonlinear optimization methods are used [1] [26] to obtain quaternion spline that minimizes the torque energy of the overall trajectory. Screw motion [15] [17] is also used as an alternative of spline motion to compute an interpolating curve. It is not clear whether these methods can be easily extended to high dimensional spaces.

III. Representation for Motions

In this section, we introduce our notation and present our spline representation for motions.

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. Specifically, we consider the constraints of velocity and acceleration limits:

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

B. Spline Representation for Motion

Given an initial collision free piecewise linear trajectory within n vertices $\mathbf{u}(t) = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ computed by sample-based motion planning algorithms, our objective is to improve the smoothness of the input trajectory while satisfying the collision-free, velocity and acceleration constraints. Our formulation choose B-spline to represent the motion of rigid or articulated robots, because it is smooth (e.g. a B-cubic spline is $\mathcal{C}^2$ smooth except on the boundary) and can be easily refined by adjusting the control points to satisfy various constraints.

The motion of a rigid body $\mathbf{u}(t)$ contains 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. The case for representing the motion of rotation component $\mathbf{R}(t)$ involves more issues: it is in fact a curve in $\mathcal{SO}(3)$ and a cubic B-spline in $\mathcal{R}^3$ can not formulate it well. Some previous methods represent the rotation as a 3×3 matrix $\mathbf{M}$ or the Euler angles (α, β, γ) and then use B-splines represent each curve $\mathbf{M}(i,j)(t)_{1\leq i,j\leq 3}$, or $\alpha(t)$, $\beta(t)$, $\gamma(t)$ respectively. However, $\mathbf{M}(i,j)(t)$ can not preserve the intrinsic property i.e. $\det(\mathbf{M}(t)) = 1$ easily. Moreover, the differential properties of $\mathbf{M}(t), \alpha(t), \beta(t), \gamma(t)$ may not be preserved in rotation $\mathbf{R}(t)$ due to singularities. In robotics and computer animation, most applications represent rotation as quaternions that are singularity-free [10] [11] [15][24][26]. Some of them directly change the recursive formulation of B-splines (i.e. the de Boor algorithm) for quaternions [24] [26]. However, the spline constructed by these approaches has no closed formulation and may not be suitable for our continuous collision detection method described in Section VI.

We represent the rotation motion based on exponential maps [14]. It is relatively simple, but can be also used to generate smooth splines. 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 necessary constraints, we can first construct cubic B-spline $\mathbf{w}(t)$ in $\mathcal{R}^3$, and then use the exponential map to map it back onto $\mathcal{SO}(3)$. The resulting map

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

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

As a result, 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 $\mathcal{R}^3$. For articulated bodies, we use the cubic B-spline to formulate the relative motion between two adjacent joints. Let $\mathbf{T}_i^{i-1}$ and $\mathbf{R}_i^{i-1}$ be the relative motion between joint $\mathcal{J}_i$ and its parent $\mathcal{J}_{i-1}$, they can be represented by two splines $\mathbf{T}_i^{i-1}(t)$ and $\exp(\mathbf{w}_i^{i-1}(t))$.

We denote $\mathbf{f}_{\mathbf{d}_0,...,\mathbf{d}_m}^{t_0,...,t_{m-2}}(t)$ 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 splines, and $t_0 \leq t \leq t_{m-2}$. $\mathbf{f}$ is uniform spline if $\{t_i\}_{i=0}^{m-2}$ are uniform knots. We also denote spline function as $\mathbf{f}(t)$ for convenience.

IV. Shortcut Smoothing based on Cubic B-Splines

In this section, we introduce our smoothing algorithm based on cubic B-splines. The basic framework is shown in Algorithm 1, which is quite similar to traditional shortcut algorithms such as [29] [7], though there are some differences. Prior methods tend to use the end-point information of the original path, i.e. $\mathbf{u}(t_a)$, $\mathbf{u}(t_b)$, $\mathbf{u}'(t_a)$, $\mathbf{u}'(t_b)$, etc. In other words, the original piecewise linear path provides the knowledge $\mathbf{u}(t) \in \mathcal{C}_{free}, \forall t \in [t_a, t_b]$ while the information utilized by the traditional approaches is $\exists \tilde{\mathbf{u}}(t) \in \mathcal{C}^0, \tilde{\mathbf{u}}(t) \in \mathcal{C}_{free}, \forall t \in [t_a, t_b]$. As a result, traditional shortcut methods usually only work when $|t_a - t_b|$ is small. Otherwise the shortcut attempts will fail frequently due to collisions, especially in environments with many obstacles or narrow passages.

Our new spline based algorithm [1] utilizes the complete information from the original piecewise linear path that needs to be smoothed. It can also adapt to different environments: when the environment is open (with a few obstacles), the new trajectory $\mathbf{s}(t)$ is smoother and can deviate a lot from the original path $\mathbf{u}(t)$; when the environment is constrained or has narrow passages, $\mathbf{s}(t)$ may not have high order continuity everywhere and maybe closer to $\mathbf{u}(t)$ more.

A. Spline based Shortcut Algorithm

Now we describe the SplineShortcut subroutine in Algorithm 1 which replaces trajectory $\mathbf{u}(t)|_{t_a \leq t \leq t_b}$ with a smoother and collision-free trajectory $\mathbf{s}(t)|_{t_a \leq t \leq t_b}$.

As shown in Algorithm 2, we first sample $m+1$ points $\mathbf{u}(t), t = t_0, ..., t_m$ on the original path to be replaced, where $t_0 = t_a, t_m = t_b$. The velocities at the two end-points are also given as $\epsilon\mathbf{u}(t_a)'$ and $\epsilon\mathbf{u}'(t_b)$, where $0 < \epsilon \leq 1$ is used to make sure $|\epsilon\mathbf{u}(t_a)'| \leq \mathbf{v}_{max}$ and $|\epsilon\mathbf{u}(t_b)'| \leq \mathbf{v}_{max}$. Next we use a cubic B-spline to approximate the original trajectory $\mathbf{u}(t)|_{t_a \leq t \leq t_b}$. We use an interpolation scheme, i.e. find a cubic B-spline $\mathbf{s}(t)|_{t_a \leq t \leq t_b}$ so that $\mathbf{s}(t_i) = \mathbf{u}(t_i), \forall i \in \{0, 1, ..., m\}$, $\mathbf{s}'(t_0) = \epsilon\mathbf{u}'(t_0)$ and $\mathbf{s}'(t_m) = \epsilon\mathbf{u}'(t_m)$.

According to [4], the $\mathcal{C}^2$-smooth interpolatory cubic spline with $\mathcal{G}^1$ smoothness on the boundary can be constructed by solving a linear system

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

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

Algorithm 1: Spline-based shortcut algorithm

Input : Piecewise linear path $\mathbf{u} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ parameterized on $[0, T]$, velocity bound $\mathbf{v}_{max}$, acceleration bound $\mathbf{a}_{max}$, iteration count N
Output: A smoothed, physically feasible trajectory $\mathbf{u}(t)$
begin
  for $i = 1$ **to** N **do**
    Sample t_a and t_b uniformly from $[0, T]$
    $\mathbf{s}(t) \leftarrow$ SplineShortcut($\mathbf{u}(t)|_{t_a \leq t \leq t_b}, \mathbf{u}'(t_a), \mathbf{u}'(t_b)$)
    $\mathbf{s}(t) \leftarrow$ SplineOptimize($\mathbf{s}(t)|_{t_a \leq t \leq t_b}, \mathbf{v}_{max}, \mathbf{a}_{max}$)
    if s **then**
      Replace $\mathbf{u}(t)|_{t_a \leq t \leq t_b}$ by $\mathbf{s}(t)$
end

Algorithm 2: SplineShortcut subroutine

Input : Piecewise linear sub-trajectory $\mathbf{u}(t)|_{t_a \leq t \leq t_b}$, end-point derivatives $\mathbf{v}_a, \mathbf{v}_b$
Output: A smoothed cubic B-spline trajectory $\mathbf{s}(t)$
begin
  Sample $m+1$ points $\mathbf{u}(t_i)$, with $t_a \leq t_i \leq t_b, i = 0, 1, ...m, t_0 = t_a, t_m = t_b$.
  Construct cubic B-spline $\mathbf{s}(t)|_{t_a \leq t \leq t_b}$ so that $\forall i \in \{0, ..., m\}, \mathbf{s}(t_i) = \mathbf{u}(t_i)$, $\mathbf{s}'(t_a) = \mathbf{v}_a$ and $\mathbf{s}'(t_b) = \mathbf{v}_b$.
  for $i = 0$ **to** $m-1$ **do**
    t_i^{cont} = CCDQuery($\mathbf{s}(t)|_{t_i \leq t \leq t_{i+1}}$).
  repeat
    Find in-colliding intervals $[t_i, t_j]$
    $\mathbf{s}(t)|_{t_i \leq t \leq t_j} \leftarrow$ SplineShortcut($\mathbf{u}(t)|_{t_i \leq t \leq t_j}$, $\lambda\mathbf{s}'(t_i) + (1-\lambda)\mathbf{u}'(t_i)$, $\lambda\mathbf{s}'(t_j) + (1-\lambda)\mathbf{u}'(t_j)$)
  until $\mathbf{s}(t)$ *is collision free within* $[t_0, t_m]$;
  return s
end

[1] Basically, our method can use B-splines of any degrees. However, the cubic B-splines provide an appropriate balance between smoothness constraints and high oscillations caused by high degree curves.

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

The parameters $\alpha_k, \beta_k, \gamma_k$ are completely decided by $\{t_i\}_{i=0}^m$. $\mathbf{r}_k$ are completely decided by $\{\mathbf{u}(t_i)\}_{i=0}^m$, $\mathbf{u}'(t_0)$ 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 $1 \leq k \leq m-2$; $\alpha_k = \frac{3}{2}, \beta_k = \frac{7}{2}, \gamma_k = \frac{3}{2}, \mathbf{r}_k = \epsilon\mathbf{u}'(t_k)$, for $k = 0$ or $m-1$. $\mathbf{r}_0$ and $\mathbf{r}_m$ are used to impose $\mathcal{G}^1$ smoothness on interval boundaries. Parameter setting for non-uniform spline is more complicated [4].

The solutions to the system are given by $\{\mathbf{d}_k\}_{k=0}^m$. Along with the additional two points $\mathbf{d}_{-1} = \mathbf{u}(t_0)$ and $\mathbf{d}_{m+1} = \mathbf{u}(t_m)$, they constitute the de Boor control point sets for the interpolatory spline $\mathbf{f}^{t_0,\ldots,t_m}_{\mathbf{d}_{-1},\ldots,\mathbf{d}_m,\mathbf{d}_{m+1}}(t)$ that we need to determine.

For the translational motion, $\mathbf{f}(t)$ is exactly the spline curve that we need in 3D. For rotational motion, spline $\mathbf{f}(t)$ is the $\mathbf{w}(t)$ in Eq. 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 $\mathbf{s}(t)$. For articulated body, $\mathbf{s}(t)$ is computed in a similar manner, but splines are computed for the relative motion between joints. Moreover, notice that $\tilde{\mathbf{A}}$ is the same for different components of the same motion curve, we can pre-compute the LU-decomposition of $\tilde{\mathbf{A}}$ and then reuse it while computing translation and rotation curve.

Notice that we apply spline interpolation schemes for both translational motion in $\mathcal{R}^3$ and rotational motion in $\mathcal{SO}(3)$. In $\mathcal{R}^3$, the spline interpolation produces a path $\mathbf{p}$ which minimizes the L_2 energy: $\int \|\mathbf{p}''\|^2 dt$, i.e. the variational principle [4] [11]. However, the spline in $\mathcal{SO}(3)$ does not necessarily have such a property. In this sense, the exponential map does not preserve the optimality similar to splines in $\mathcal{R}^3$. In principle, it may be possible to compute a spline in $\mathcal{SO}(3)$ that satisfies variational principles (i.e. minimizes the torque energy). However, the computation may involve non-linear optimization, which can be expensive [1]. In practice, the above interpolation scheme works well for rotation.

After $\mathbf{s}(t)$ is computed, we need to check whether $\mathbf{s}(t) \in \mathcal{C}_{free}, \forall t \in [t_a, t_b]$ by using continuous collision detection technique (refer to Section VI). If it is collision free, we use local optimization method to satisfy the constraints corresponding to velocity and acceleration bounds, as discussed in Section V. Otherwise, we need to resolve the collisions by using a recursive method that modifies the curve.

B. Recursive Spline Modification

When a collision is found between $\mathbf{f}^{t_0,\ldots,t_m}_{\mathbf{d}_{-1},\ldots,\mathbf{d}_{m+1}}(t)$ and the environment, we need to resolve it by modifying the spline curve. The simplest solution is to sample more knots (e.g. we can change each spline segment into a m-knot spline, and the overall knot number is $m' = m^2$) within $[t_0, t_m]$ and construct a new spline $\mathbf{f}^{t_0,\ldots,t_{m'}}_{\mathbf{d}_{-1},\ldots,d_{m'+1}}(t)$ from scratch. However, this method requires solving linear systems with increasing sizes (as the new system matrix $\tilde{\mathbf{A}}$ will be of the size $(m'+1)^2$), which can be time consuming.

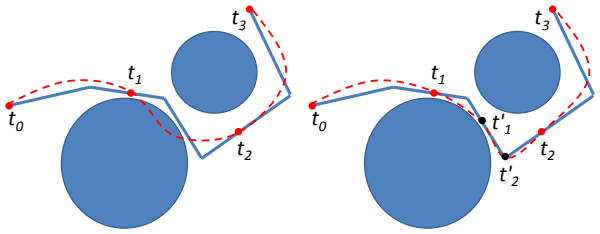

Fig. 1. For cubic B-spline $\mathbf{f}^{t_0,\ldots,t_3}(t), t \in [t_0, t_3]$, a collision happens during the interval $[t_1, t_2]$. We recursively add new knots within the interval and construct a spline $\mathbf{f}^{t_1,t'_1,t'_2,t_2}(t), t \in [t_1, t_2]$. The splines on $[t_0, t_1]$ and $[t_2, t_3]$ are kept unchanged by applying the subdivision for $\mathbf{f}^{t_0,\ldots,t_3}(t)$ on the two intervals.

Our solution is based on the locality of spline curves. Suppose that the collision between the B-spline curve and the obstacles is detected within the interval $[t_i, t_j]$, our goal is to keep the other portions of the spline unchanged during the intervals that correspond to collision free portions of the spline. Therefore, as shown in Fig 1, we only sample more points within $[t_i, t_j]$ which can pull the spline near the original piecewise linear, but a collision-free trajectory. We further need to compute the derivatives at the boundaries of $[t_i, t_j]$. We use the linear combination of derivatives on the original trajectory $\mathbf{u}(t)$ and the in-colliding spline $\mathbf{s}(t)$. That is $\mathbf{v}_a = \lambda\mathbf{s}'(t_i) + (1-\lambda)\mathbf{u}'(t_i)$ and $\mathbf{v}_b = \lambda\mathbf{s}'(t_j) + (1-\lambda)\mathbf{u}'(t_j)$. A larger value of λ makes the $\mathbf{s}(t)$ smoother after the adjustment, while a smaller value of λ makes $\mathbf{s}(t)$ fit $\mathbf{u}(t)$ better to avoid collision.

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. The algorithm returns failure for the current shortcut attempt if the collisions are not removed by local modifications. If $\lambda = 0$ in the boundary derivative setting, the $\mathcal{G}^1$ smoothness condition is not satisfied at t_i and t_j, but the shortcut attempt will succeed because the spline will converge to $\mathbf{u}(t)$ after several iterations.

For the collision-free spline intervals, we keep them unchanged. We still need the spline representation for these sub-splines for the local optimization step in Section V, in order to satisfy constraints on the velocity and acceleration. Therefore, we use the spline subdivision technique [4] to compute the $m+3$ new de Boor points $\tilde{\mathbf{d}}_{-1}, \ldots, \tilde{\mathbf{d}}_{m+1}$ for each sub-spline defined upon one collision-free interval.

The final output of SPLINESHORTCUT is a smooth collision-free trajectory.

V. LOCAL REFINEMENT FOR CUBIC B-SPLINES

The trajectory $\mathbf{u}(t)$ also needs to satisfy other constraints beside a smooth path, e.g. velocity constraints and acceleration constraints, as defined in Section III-A. Unlike [7], we consider them as 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. [2]

We first discuss how to constrain the velocity bound. For a uniform cubic B-spline $\mathbf{f}^{t_0,\ldots,t_{m-2}}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}(t)$, the derivative at the i-th knot is $\mathbf{f}'(t_i) = \frac{\mathbf{d}_{i+2}-\mathbf{d}_i}{2T}$ and the derivative at the midpoint between i-th knot and $i+1$-th knot is $\mathbf{f}'(\frac{t_i+t_{i+1}}{2}) = \frac{\mathbf{d}_{i+3}+5\mathbf{d}_{i+2}-5\mathbf{d}_{i+1}-\mathbf{d}_i}{8T}$, where $T = t_{i+1} - t_i$.

We define the limit velocity as the velocity at time $\tilde{t}$ when $\mathbf{f}''(\tilde{t}) = 0$. Limit velocity will be the maximum/minimum velocity on one interval $[t_a, t_b]$ when $\tilde{t} \in [t_a, t_b]$. The limit velocity within the interval $[t_i, t_{i+1}]$ is $\mathbf{f}^e[t_i, t_{i+1}] = \frac{1}{2T}((\mathbf{d}_{i+2} - \mathbf{d}_i) + \frac{(\mathbf{d}_{i+2}-2\mathbf{d}_{i+1}+\mathbf{d}_i)^2}{-\mathbf{d}_{i+3}+3\mathbf{d}_{i+2}-3\mathbf{d}_{i+1}+\mathbf{d}_i})$. Based on some computation, we can find that $\mathbf{f}^e[t_i, t_{i+1}]$ is uniquely determined by the velocity at the begin, end, and middle knots of interval $[t_i, t_{i+1}]$: $\mathbf{f}^e[t_i, t_{i+1}] = \mathbf{f}'(t_i) + \frac{1}{8}\frac{(4\mathbf{f}'(\frac{t_i+t_{i+1}}{2})-\mathbf{f}'(t_{i+1})-3\mathbf{f}'(t_i))^2}{2\mathbf{f}'(\frac{t_i+t_{i+1}}{2})-2\mathbf{f}'(t_{i+1})-\mathbf{f}'(t_i)}$. 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, one approach is to check $\mathbf{f}'(t_i)$ and $\mathbf{f}'(\frac{t_i+t_{i+1}}{2})$ in order 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 corresponding $\mathbf{d}_{i+2}$ to $\lambda\mathbf{d}_{i+2}$ with $0 < \lambda < 1$. We repeat this procedure for several times and then most $t \in [t_0, t_{m-2}]$ would be constrained within velocity limit. Notice that the iteration will not influence $\mathbf{f}'(t_0)$ and $\mathbf{f}'(t_{m-2})$ on the boundaries of the spline.

Another better method with better convergence properties is as follows. We successively check the derivative at each knot, the derivative in the middle of previous interval and the limit velocity of previous interval, i.e. in the order of $\mathbf{f}'(t_i)$, $\mathbf{f}'(\frac{t_{i-1}+t_i}{2})$, $\mathbf{f}^e[t_{i-1}, t_i]$, $\mathbf{f}'(t_{i+1})$, ..., etc. When $\mathbf{f}'(t_i)$ does not satisfy the velocity limit, we reduce $\mathbf{d}_{i+2}$ to $\lambda\mathbf{d}_{i+2}$; when $\mathbf{f}'(\frac{t_{i-1}+t_i}{2})$ does not satisfy the velocity limit, we reduce $\mathbf{d}_{i+1}$ to $\lambda\mathbf{d}_{i+1}$; when $\tilde{t} \in [t_{i-1}, t_i]$ and $\mathbf{f}^e[t_{i-1}, t_i]$ violates the velocity limit, we use local search on $\mathbf{d}_{i+1}$ to find a value that can reduce $\mathbf{f}^e[t_{i-1}, t_i]$. Here $0 < \lambda < 1$. We choose $\mathbf{d}_{i+1}$ to reduce $\mathbf{f}^e[t_{i-1}, t_i]$ because it is the most sensitive parameter for $\mathbf{f}^e$. This adjustment approach is similar to [21] [22] for humanoid robotics.

The bound on the acceleration is relatively simple to satisfy, because the acceleration of the entire spline is a linear function. The acceleration at the i-th knot is $\mathbf{f}''(t_i) = \frac{\mathbf{d}_{i+2}-2\mathbf{d}_{i+1}+\mathbf{d}_i}{T^2}$. We only need to adjust $\{\mathbf{d}_i\}_{i=3}^{m-3}$ to make sure the acceleration is bounded by $\mathbf{a}_{max}$.

Whenever de Boor control points are adjusted, we need to perform continuous collision query to ensure that the modified spline is also collision-free.

[2]Similar optimization is also available for non-uniform spline, by using cubic B-spline's representation on interval $[t_i, t_{i+1}]$ in coefficient matrix form $\mathbf{f}_i(t) = (1, u, u^2, u^3)\mathbf{M}_i[\mathbf{d}_i^T, \mathbf{d}_{i+1}^T, \mathbf{d}_{i+2}^T, \mathbf{d}_{i+3}^T]^T$, where $u = \frac{t-t_i}{t_{i+1}-t_i}$, $\mathbf{M}_i$ is the coefficient matrix on $[t_i, t_{i+1}]$ and $\{\mathbf{d}_k\}_{k=i}^{i+3}$ are control points.

VI. Continuous Collision Detection for Cubic B-Spline Trajectories

Continuous collision detection (CCD) checks for collisions along a motion curve $\mathbf{f}(t)$. This involves collision checking along the entire path, as opposed to taking a few discrete samples along the path and checking for collisions at those specific configurations, which can avoid the tunneling problem of discrete collision detection. There are many efficient CCD algorithms available to handle some specific interpolating motions between two discrete configurations, e.g. linear motion or screw motion [19] [25]. They can also be applied to articulated models [32] [20].

Most of the prior algorithms for CCD assume linear or screw interpolating motion between the discrete instances. One direct way to handle CCD check for spline motion is to use the previous CCD algorithms in a piecewise manner. That is, we sample a series of configurations $\{\mathbf{x}_i\}$ on the curve (the number of configurations will be much fewer than discrete collision detection) and apply the CCD algorithm between adjacent configurations $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ by assuming that the motion between them is some specific trajectory that the CCD algorithm can handle easily. For example, we can use the C^2A algorithm [25], which can handle linear motion and screw motion. Such piecewise CCD is feasible for spline motion and usually can provide rather good results due to two reasons. First, given two configurations $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, there exists a unique linear motion that connects them. The claim is also true for screw motion due to the Chasles theorem in Screw Theory [14]. Secondly, spline motion can be approximated well by linear or screw motion. [19] mentioned that if there are enough discrete samples, any motion curve can be well approximated by linear or screw motion. For spline motion, it is possible to achieve good approximation without too many samples on the curve. The reason is that spline tries to minimize $f''(x)$ along the curve, which is an approximation of curvature $f''(x)(1 + f'(x))^{-3/2}$.

However, the piecewise CCD algorithm still suffers from the tunneling problem [25]. Moreover, when the sampling interval is large, the assumed linear/screw motion may deviate considerably from the underlying spline motion, which makes CCD result inaccurate. As a result, we present a CCD algorithm called Spline-CCD for spline motion based on special properties of spline motion. It uses the notion of conservative advancement (CA) [19] [25] [32] [20].

We first introduce the algorithm for a rigid body. Suppose there are two rigid bodies $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{B}$ is still and $\mathcal{A}$'s trajectory is the spline motion $\mathbf{f}^{t_0,\ldots,t_{m-2}}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}(t)$, which is composed of $m-2$ spline segments. Each segment is a degree-3 polynomial controlled by 4 de Boor points and is parameterized over $[0, 1]$. We use C^2A technique to handle these segments one by one. For one segment $\mathbf{p}(t)$, we need to find the first time of contact t_c between $\mathcal{A}$ and $\mathcal{B}$, i.e. $\min\{t \in [0, 1] | \mathcal{A}(t) \cap \mathcal{B}\}$. CA calculates a lower bound of t_c by repeatedly advancing $\mathcal{A}$ by Δt. Δt is computed based on a lower bound on the closest distance $d(\mathcal{A}(t), \mathcal{B})$ between $\mathcal{A}(t)$ and $\mathcal{B}$, and the upper bound $\mu(t)$ on the motion of

$\mathcal{A}(t)$ projected onto $d(\mathcal{A}(t), \mathcal{B})$. In other words, if we can have a motion bound $\mu(t)$ so that $\int_t^{t+\Delta t} |\dot{\mathbf{p}}(\tau) \cdot \mathbf{n}| d\tau \leq \int_t^{t+\Delta t} \mu(t) d\tau = \mu(t)\Delta t$, then $\Delta t \leq \frac{d(\mathcal{A}(t),\mathcal{B})}{\mu}$ is the safe advancement at time t. t_c can be obtained by repeatedly calculating the time step Δt_i and summing them up (i.e. $t_c = \sum_i \Delta t_i$) until $d(\mathcal{A}(t), \mathcal{B})$ becomes less than user-specified threshold. The overall contact time for spline motion $\mathbf{f}(t)$ is $T_c + t_c$ if the first contact is detected in T_c-th segment and the local contact time on that segment is t_c.

In Appendix I, we describe a solution to the main computation in terms of applying C^2A to spline trajectory, i.e. calculating the motion bound μ for spline motion. Our method is based on the subdivision and convex hull properties of B-splines. Similar to [25], we further use swept sphere volume (SSV) hierarchies to extend the CA formulation for arbitrary polygon-soup models.

The Spline-CCD algorithm can be extended to articulated bodies. The CA procedure works as long as the motion bounds can be estimated for articulated bodies. This turns out to be straightforward, e.g. we can use the technique used in [32] to estimate motion bounds for articulated body based on bounds of all rigid components.

VII. Results

In this section, we highlight the performance of our spline-based shortcut algorithm on different benchmarks shown in Table I. Among the six benchmarks used for rigid body motion planning, only the piano benchmark doesn't have narrow passages. The original collision-free paths are computed using a sample-based motion planner [31]. Piecewise linear collision-free path for the 40-DOF articulated benchmark is computed using a retraction-based planner [16].

The results for different benchmarks are shown in Fig 2, Fig 3, Fig 4 and Fig 5. For rigid bodies, we show the path traversed by the origin of the local frame associated with the rigid body. We compare the results of our method with the results computed by the sample-based motion planner and linear shortcut algorithm [29]. In all these benchmarks, our spline-based smoothing algorithm gives the best trajectories.

Moreover, for the gear benchmark (Fig 2), we show the swept volume of the robot generated using the paths computed by different algorithms. A smoother trajectory results in a smaller volume with smoother boundaries.

For the piano benchmark (Fig 3), we also visualize the rotation curves using quaternions, where the distance to the sphere is the angle and the direction of the sphere center is the axis. Obviously, the rotation curve generated by the original sample-based planner is rather uneven, the linear shortcut algorithm results in a lot of corners, and our spline-based smoothing algorithm results in smooth curves.

The narrow passages in the free configuration space add to the complexity of trajectory smoothing algorithms. Our spline-based shortcut operation may fail in extremely constrained narrow passages and the curve will degrade to $\mathcal{C}^0$ locally. However, as shown in Fig 2, where we use green circles to mark narrow passages, our method can still handle such cases effectively, as compared to prior methods.

	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.

	Piano	Gear	Wiper	CarSeat	AlphaPuzzle	Bridge
linear shortcut	59.92	8.533	24.06	19.98	10.533	163
spline shortcut	43.53	9.781	33.09	51.39	15.781	185
spline shortcut + CCD	24.51	5.453	4.69	28.31	17.16	-

TABLE II

TIMING OF SHORTCUT ALGORITHMS IN SECONDS.

We also show the timing results in Table II where all methods perform 500 shortcut operations on different benchmarks. For each shortcut operation in linear shortcut and spline shortcut, we check collisions for $30\frac{n|t_b - t_a|}{T}$ discrete samples, where n, t_b, t_a, T are introduced in Algorithm 1. We can see that the overhead of our spline-based algorithm over linear shortcut method is rather small. Moreover, our spline-CCD algorithm may significantly reduce the overhead of collision checking, and results in overall speedup. (Notice that exact CCD is generally slower than discrete CCD because each step of exact CCD is more expensive. However, in shortcut algorithms, we may check collisions for long paths (i.e. when $|t_b - t_a|$ is large). In these cases, discrete CCD needs a lot of computation while exact CCD may complete quickly due to the smoothness of the splines).

VIII. Conclusions and Future Work

We present an efficient and simple trajectory smoothing algorithm using cubic B-spline interpolation. Our algorithm can transform uneven or jerky trajectories generated by sample-based motion planners into smooth trajectories. We also use local optimization to refine cubic B-spline control points in order to satisfy additional constraints. Our new CCD algorithm for B-spline trajectories is used for exact collision checking with the environment and results in overall speedup as compared to prior methods.

There are many avenues for future work. We are interested in extending our spline-based framework to handle more complex motions with other constraints, such as the non-holonomic motions and humanoid motions. Furthermore, since both spline shortcut and spline-CCD algorithms are parallelizable, we would like to design the parallel extensions of our algorithm and implement them on multi-core CPUs and many-core GPUs. Finally, we would like to apply these algorithms to physical robots.

Appendix I
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 $\mathbf{u}(t)$

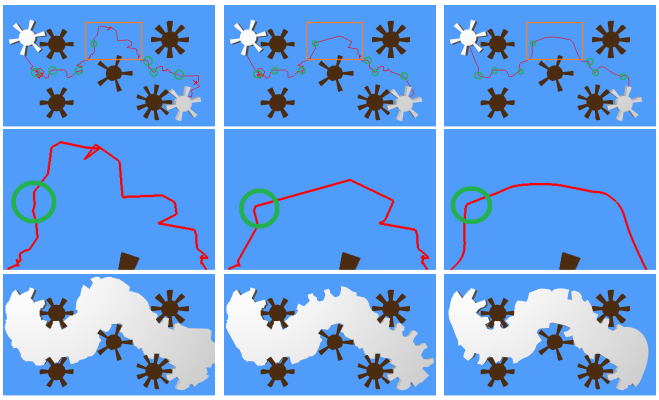

Fig. 2. Results for gear benchmark. From left to right are results computed by planner, linear shortcut in [29] and our method. The first row shows the translational curves; the second row shows a zoomed view of the trajectories within the orange box in the first row; the third row shows the swept volumes 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 $\mathcal{C}^2$ trajectory for most of the path, though it can be $\mathcal{C}^0$ in very cluttered areas.

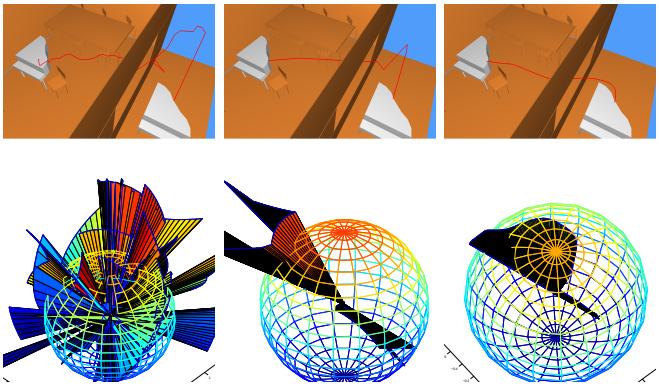

Fig. 3. Results for piano benchmark. From left to right, we show the paths computed by the sample-based planner, linear shortcut algorithm [29] and our method, respectively. The first row shows the translational curves while the second row visualizes the rotation curves obtained by different methods.

is the rotation axis and $\theta(t)$ is the rotation angles. $\mathbf{w}(t)$ and $\mathbf{T}(t)$ are both spline curves in $\mathcal{R}^3$. To use the CCD algorithm in [25], 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 $\mathbf{p}_i$:

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

Based on some computation, we can give the estimation

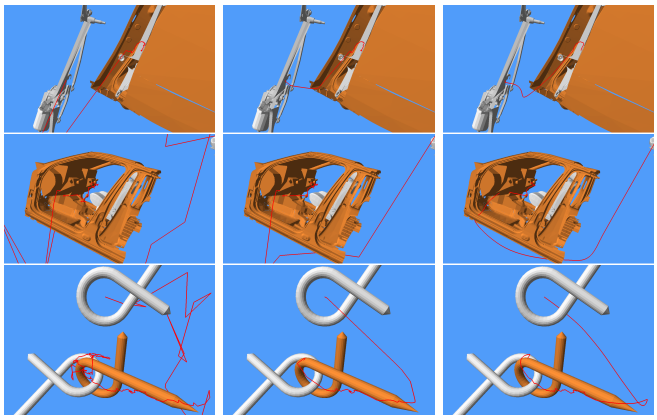

Fig. 4. Results for the benchmarks wiper, car seat, alpha puzzle. From left to right, we show the results computed by a sample-based planner, the smooth result computed computed using linear shortcut [29] and the smooth trajectory generated by our algorithm. We show the path in the workspace traversed by the origin of the rigid robot.

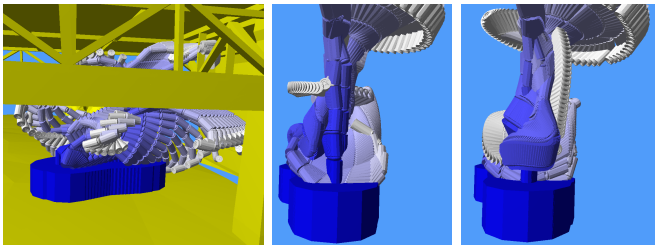

Fig. 5. Results for the 40-DOF articulated model. From left to right, we show the results computed by a sample-based planner, the smooth result of linear shortcut method in [29] and the smooth result computed by our algorithm.

for rotation part as

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

where $A_i = |\mathbf{r}_i\cdot\mathbf{n}| + 2|\mathbf{r}_i\times\mathbf{n}| + 2|\mathbf{r}_i|$, $B_i = |\mathbf{r}_i\times\mathbf{n}| + 8|\mathbf{r}_i|$. 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 $W_1(t)=\max_{\tau\in[t,1]}|\mathbf{w}'|$.

For $\min_{\tau\in[t,1]}|\mathbf{w}|$, 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 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, suppose $\mathbf{d}_0^t,\mathbf{d}_1^t,\mathbf{d}_2^t,\mathbf{d}_3^t$ are control points for sub-spline $\mathbf{w}(\tau),\tau\in[t,1]$. $\mathbf{w}(\tau)$ locates within the con-

vex hull made by the four control points, i.e. 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. 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 2) is too loose: $\frac{\max_{\tau\in[t,1]}|\mathbf{w}'|}{\min_{\tau\in[t,1]}|\mathbf{w}|} = +\infty$. We use spline subdivision to provide more compact convex hull and better bounds. Spline subdivision splits 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. A more compact estimation is then $\min_{\tau\in[t,1]}|\mathbf{w}(\tau)| \geq \min[d(\mathbf{0},\triangle_{\mathbf{a}_0,\mathbf{a}_1,\mathbf{a}_2,\mathbf{a}_3}), d(\mathbf{0},\triangle_{\mathbf{b}_0,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3})]$, where $\mathbf{a}_0,\mathbf{a}_1,\mathbf{a}_2,\mathbf{a}_3$ and $\mathbf{b}_0,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3$ are de Boor points for two sub-splines. We repeat the procedure until a finite bound is obtained which is denoted as $W_2(t)$.

The bound for translation item is similar to W_1 and we denote it as W_3. 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. Similar to [25], 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. We can find 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 $\mu^\alpha = (5r + \max_k|\mathbf{c}_k\cdot\mathbf{n}| + 2\max_k|\mathbf{c}_k\times\mathbf{n}| + 2\max_k|\mathbf{c}_k|)W_1 + (9r + \max_k|\mathbf{c}_k\times\mathbf{n}| + 8\max_k|\mathbf{c}_k|)\frac{W_1}{W_2} + W_3$. Then we can use μ^α in the SSV hierarchy traverse procedure in order to calculate the μ efficiently.

REFERENCES

[1] A. H. Barr, B. Currin, S. Gabriel, and J. F. Hughes, "Smooth interpolation of orientations with angular velocity constraints using quaternions," *SIGGRAPH*, vol. 26, no. 2, pp. 313–320, 1992.

[2] J. Bobrow, "Optimal robot plant planning using the minimum-time criterion," *Journal of Robotics and Automation*, vol. 4, no. 4, pp. 443 –450, aug. 1988.

[3] 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.

[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] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," *Proceedings of International Conference on Robotics and Automation*, pp. 1152–1159, 1987.

[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 c^2-continuous b-spline quaternion curve interpolating a given sequence of solid orientations," in *Computer Animation*, apr. 1995, pp. 72–81.

[11] ——, "A general construction scheme for unit quaternion curves with simple high order derivatives," in *SIGGRAPH*, 1995, pp. 369–376.

[12] 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.

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

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

[15] 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.

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

[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] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," *Proc. of Eurographics (Computer Graphics Forum)*, 2002.

[20] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *Proceedings of ACM symposium on Solid modeling and applications*, 2004, pp. 145–156.

[21] M. Ruchanurucks, S. Nakaoka, S. Kudoh, , and K. Ikeuchi, "Generation of humanoid robot motions with physical constraints using hierarchical b-spline," in *Proceedings of International Conference on Intelligent Robots and Systems*, aug. 2005, pp. 1869–1874.

[22] ——, "Humanoid robot motion generation with sequential physical constraints," in *Proceedings of International Conference on Robotics and Automation*, May 2006, pp. 2649–2654.

[23] 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.

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

[25] 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.

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

[27] M. Toussaint, M. Gienger, and C. Goerick, "Optimization of sequential attractor-based movement for compact behaviour generation," in *Proceedings of International Conference on Humanoid Robots*, nov. 2007, pp. 122 –129.

[28] A. Witkin and M. Kass, "Spacetime constraints," *SIGGRAPH*, vol. 22, no. 4, pp. 159–168, 1988.

[29] 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.

[30] K. Yang and S. Sukkarieh, "3d smooth path planning for a uav in cluttered natural environments," in *Proceedings of International Conference on Intelligent Robots and Systems*, sep. 2008, pp. 794 –800.

[31] 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.

[32] 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.