ijrr

The International Journal of Robotics Research
31(10) 1155–1175

DOI: 10.1177/0278364912453186
ijr.sagepub.com
SAGE

# Collision-free and smooth trajectory computation in cluttered environments

**Jia Pan[1], Liangjun Zhang[2] and Dinesh Manocha[1]**

**Abstract**

*We present a novel trajectory computation algorithm to smooth piecewise linear collision-free trajectories computed by sample-based motion planners. Our approach uses cubic B-splines to generate trajectories that are $\mathcal{C}^2$ almost everywhere, except on a few isolated points. The algorithm performs local spline refinement to compute smooth, collision-free trajectories and it works well even in environments with narrow passages. We also present a fast and reliable algorithm for collision checking between a 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 cluttered environments.*

**Keywords**

Animation, simulation, virtual reality, path planning for manipulators

## 1. Introduction

Sample-based planning algorithms such as probabilistic roadmaps (PRMs) (Kavraki et al. 1996) or rapidly-exploring random trees (RRTs) (Kuffner and LaValle 2000; LaValle 2006) 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 degree-of-freedom (DOF) robots, and can also handle cluttered environments or narrow passages.

In many applications, including computer animation (Yamane et al. 2004) and physical robotic systems (Yang and Sukkarieh 2010), it is important that the paths are not only collision free, but also satisfy other constraints in terms of smoothness and solution quality. There has been considerable work on the computation of smooth paths in mobile robotics, because non-smooth motion can cause slippage and over-actuation, and are difficult for proportional–integral–derivative (PID) controllers to track, and thus difficult for the robot's actuator to execute. In computer-aided design (CAD) and virtual prototyping, path-smoothing algorithms are also considered important, as they are used to model accurate physical interactions (Chang 1995).

It is well known that sample-based planners can sometimes generate jerky, unnatural paths that may contain unnecessary turns, or that the velocities at the vertices may change arbitrarily (Hauser and Ng-Thow-Hing 2010; Kavraki et al. 1996; LaValle 2006). Furthermore, these issues become more significant when the free space of the robot has narrow passages (Pan et al. 2010; Zhang et al. 2008) as the search space for path computation becomes more constrained.

Many techniques have been proposed in the literature for generating smooth paths. At a broad level, they can be classified into shortcut methods (Hauser and Ng-Thow-Hing 2010; Kallmann et al. 2003; Kavraki et al. 1996; Kuffner and LaValle 2000) or optimization-based approaches (Ratliff et al. 2009; Lengagne et al. 2011). 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 linear shortcut methods tend to be fast and simple, and can produce high-quality paths in many cases (Geraerts and Overmars 2007). 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

[1]Department of Computer Science, University of North Carolina at Chapel Hill, US
[2]Department of Computer Science, Stanford University, US

**Corresponding author:**
Jia Pan, Campus Box 3175, Sitterson Hall, UNC Chapel Hill, Chapel Hill, North Carolina 27599-3175, US
Email: panj@cs.unc.edu

**Fig. 1.** Comparison between trajectories computed for the maintenance of a windscreen wiper. Left: Jerky, piecewise linear collision-free path for the wiper. Right: Smooth collision-free path computed by our spline-based shortcut algorithm.

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 an elastic problem from differential geometry. Some of the commonly used solutions are gradient-based methods, which tend to compute minimum-energy paths, or use elastic band or elastic strip planning, which models 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. Moreover, optimization-based methods are usually off-line methods because of their low speeds.

Ideally, we need a smoothing algorithm that can generate high-quality trajectories suitable for the actuator to execute and is also efficient enough so that the robot can generate a feasible trajectory as quickly as possible.

### 1.1. Main results

In this paper, we present a fast and simple algorithm to smooth the paths generated using sample-based planning by using spline interpolation (Figure 1). 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 or approximates these points. We use the exponential map to construct smooth rotational motion in $SO(3)$ and thereby handle translational and rotational motion in a uniform way. The cubic spline formulation provides sufficient flexibility in terms of providing higher-order smoothness, i.e. computing trajectories that are almost $\mathcal{C}^2$ except on a few isolated points. 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* techniques (Tang et al. 2011). The overall approach is reliable, guaranteed to not miss any collisions and is significantly faster than prior continuous collision detection algorithms based on checking discrete samples on the trajectory. The conservative advancement-based collision checking algorithm for spline trajectories has already been implemented in the FCL package (Pan et al. 2012), available at https://kforge.ros.org/projects/fcl/.

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 modified algorithm is an extension of the approach described in Pan et al. (2011). The main improvements of this paper over Pan et al. (2011) include: (a) We have modified the algorithm so that it generates $\mathcal{C}^2$ B-splines almost everywhere, as the $\mathcal{C}^2$ continuity is desired by most physical robots. (b) We have improved the motion bound formulation, which can provide tighter bounds and thereby improve the overall performance of the smoothing algorithm. (c) We describe the detail for how to compute the motion bound for a rigid body and articulated bodies, for different motion trajectories. (d) We have included experiments on the PR2 robot to show our method's performance for a real robot.

The rest of the paper is organized as follows. We survey related work in Section 2. We introduce our notation and present our spline motion representation in Section 3. We present the basic spline-based smoothing and local refinement algorithm in Section 4. In Section 5, we describe an efficient collision checking algorithm for spline trajectories. We highlight the performance against different benchmarks for rigid and articulated bodies in Section 6. Further analysis of our algorithm is provided in Section 7.

## 2. Related work

In this section, we briefly review prior work on generating smooth motion and performing collision checking for trajectories.

### 2.1. Smooth motion generation

Smooth motion generation is a topic of interest in computer animation and robotics. For computer animation, the goal is to generate natural avatar motion, which should necessarily be smooth. In robotics, a robot cannot change velocity instantaneously because of its motor's limited output power, which requires its motion to have continuous velocity and/or acceleration. Besides the smoothness requirement, valid motion must satisfy additional constraints, including the collision-free constraint and the bounded velocity/acceleration constraint. The collision-free

constraint makes sure that the robot does not collide with obstacles; the bounded velocity/acceleration constraint is usually justified for reasons of safety and servo stability.

Optimization is one common technique to generate smooth motion and various methods have been developed. 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 and energy consumption of the overall robot (Lee et al. 2005; Lengagne et al. 2011), and are subject to the joint limits, and velocity and acceleration constraints. However, numerical optimization methods require the 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 (Brock and Khatib 2002; Ratliff et al. 2009; Toussaint et al. 2007). Trajectory optimization methods are often computationally intensive due to the high dimension of the optimization problem and the large number of constraints. Furthermore, they may suffer from local minima issues and are sensitive to the numerical precision of the gradient and Hessian matrices.

Shortcut techniques are widely used in robotics and computer animation as lightweight methods for generating smooth motion. They use heuristics to iteratively replace jerky or unnatural portions of a path with shorter linear or parabolic segments (Kavraki et al. 1996; Hauser and Ng-Thow-Hing 2010; Kallmann et al. 2003; Yang and Sukkarieh 2010). Simple velocity and acceleration constraints are further imposed over these parabolic segments (Hauser and Ng-Thow-Hing 2010). In practice, these methods can produce short paths quickly, though they usually cannot achieve optimality or even local optimality. Due to their simplicity and efficiency, shortcut algorithms have been widely used to improve the quality of paths computed by randomized planners (Geraerts and Overmars 2007) or used as a preprocess for optimization methods.

Another disadvantage shared by previous optimization-based methods and shortcut methods is that they formulate motion as a curve in Euclidean configuration spaces. In other words, they assume that the robot only consists of joints with a single degree of freedom, such as prismatic joints and revolute joints with joint limits. However, the Euclidean configuration space cannot adequately describe the movement of universal joints or rigid bodies in 3D space. For example, we can decompose the motion of a universal joint into rotations around three axes and then perform smoothing in the 3D configuration space. However, the smooth motion in the 3D configuration space may not map to the smooth motion in the workspace because the linear interpolation of the Euler angles does not provide a linear interpolation of the orientations (Fang et al. 1998; Shoemake 1985). Moreover, the Euler angle also suffers from the loss of one degree of freedom in some cases (the so-called gimbal lock effect). To handle such disadvantages, many other representations have been proposed to provide a more convenient representation of smooth motion, such as quaternion curves (Kim et al. 1995; Shoemake 1985) and screw motion (Nielson 2004; Powell and Rossignac 2008).

There are also other smoothing techniques besides the optimization and shortcut methods. Kanehara et al. (2007) modified the grid-based A* method and generated collision-free paths with clothoidal curvature. Some recent work has attempted to construct high-quality roadmaps that can generate smooth trajectories or near-optimal paths directly (Jaillet and Simeon 2008; Karaman and Frazzoli 2011; Marble and Bekris 2011; Nieuwenhuisen and Overmars 2004). The basic idea is adding edges or cycles that are critical for the quality of the resulting trajectory. These methods are usually less efficient than the shortcut methods.

## *2.2. Collision checking for trajectories*

Before the shortcut algorithm replaces one portion of the trajectory with a shorter segment, it needs to check whether the new trajectory is collision free. The collision checking of a continuous trajectory is also useful in many applications of robotics, such as local planning in motion planning (LaValle 2006). A simple, inexact method for trajectory collision checking is to discretize the curve to a small resolution and test each sample for collisions. If the resolution is small, the checking process will be slow; but if the resolution is large, the checking method may miss some collisions. While adaptive sampling strategies and predictive methods can be used to alleviate this problem (LaValle 2006), they can be relatively slow. In order to provide a rigorous guarantee, continuous collision detection (CCD) techniques have been proposed (Redon et al. 2004), which compute the first time of contact between two moving objects along a continuous path. CCD is typically performed by using bounding volume hierarchies (BVH). The BVH used for CCD computations provides a conservative bound for the swept volume of an object generated during a given time interval. Many CCD algorithms have been proposed to handle various types of motion, including the deformation between two meshes (Govindaraju et al. 2005) and arbitrary in-between rigid motion (Redon et al. 2004). Another popular framework for CCD is conservative advancement (CA), which incrementally advances objects by a time step while avoiding collisions. In order to determine the conservative time step, it needs to compute the minimal separation distance between the objects and uses it to estimate conservative motion bounds. Tang et al. (2011) presented algorithms to compute the motion bound for screw motion. Conservative advancement algorithm can be applied to non-convex models based on BVHs (Tang et al. 2011) and can be further extended to articulated models (Zhang et al. 2007).

## 3. Motion representation

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

### 3.1. 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 $\mathbf{q}$. Its superscripts denote the DOFs or joint indexing (e.g. $\mathbf{q}^k$ is the value for the $k$-th joint) and subscripts denote configuration indexing among multiple samples (e.g. $\mathbf{q}_l$ is the $l$-th sample in the configuration space).

A trajectory $\mathbf{q}(t), 0 \leq t \leq T$, represents a curve in the configuration space. $\mathbf{q}(t)$ is collision free if all the configurations on $\mathbf{q}(t)$ belong to $\mathcal{C}_{free}$. $\mathbf{q}(t)$ is considered to be (physically) feasible if it is collision free and satisfies other problem-specific constraints.

We use the well-known definitions of $r$-th order *parametric continuity* (i.e. $\mathcal{C}^r$) and $r$-th order *geometric continuity* (i.e. $\mathcal{G}^r$) from approximation theory (Farin 1993) to define smooth trajectories. Specifically, two curves meet at a common end point with $\mathcal{C}^r$ continuity, if they have the same $r$-th order derivative at the common point. And two curves meet at a common end point with $\mathcal{G}^r$ continuity, if there exists a reparameterization of the curves that meet at the same point with $\mathcal{C}^r$ continuity. In our case, we are interested in computing $\mathcal{C}^2$ continuous trajectories, and ensuring that our curve fitting and modification algorithms generate almost $\mathcal{C}^2$ curves. Here the almost $\mathcal{C}^2$ nature of the curves means that the curves are $\mathcal{C}^2$ almost everywhere except on a few isolated points, which can be $\mathcal{C}^1$ or $\mathcal{G}^2$. We prefer such a robot trajectory because $\mathcal{C}^1$ continuity guarantees the feasibility of motion, as joint velocities cannot change instantaneously, while the $\mathcal{C}^2$ continuity guarantees the smoothness of a joint's acceleration or torque, which can reduce mechanical oscillations of the robot (Froissart and Mechler 1993). Finally, note that our smoothness criterion is more suitable for robot motion than the $\mathcal{G}^2$ criterion used in some previous work, such as Yang and Sukkarieh (2010), because these authors only required $\mathcal{G}^2$ on each point on the curve, which may even not be $\mathcal{C}^1$.

As Hauser and Ng-Thow-Hing (2010), we also impose additional constraints on the maximum velocity and acceleration along each curve, based on the following bounds:

(a) velocity $\mathbf{q}'(t)$ is bounded by a given limit $|\mathbf{q}'(t)| \leq \mathbf{v}_{max}$;
(b) acceleration $\mathbf{q}''(t)$ is bounded by a given limit $|\mathbf{q}''(t)| \leq \mathbf{a}_{max}$;

where the absolute value and inequality are taken element-wise.

### 3.2. Spline representation for trajectories

In this section, we give an overview of our representation of the trajectories using B-splines. We represent the piecewise linear collision-free trajectory computed by a sample-based planner as $\mathbf{q}_0(t)$. We assume that this trajectory is parameterized over the interval $[0, T]$ and has $n$ vertices $\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$. Our goal is to smooth out this trajectory and ensure that the resulting trajectory is collision free and satisfies various problem-specific constraints. In the literature, many curved formulations including screw motion (Nielson 2004; Powell and Rossignac 2008), parabolic curves (Hauser and Ng-Thow-Hing 2010), and Bézier curves (Pettre et al. 2002; Yang and Sukkarieh 2010), have been used to compute smooth trajectories. In our formulation, we use cubic B-splines to represent the trajectory of rigid or articulated robots. One advantage of using B-spline polynomials as the representation for motion is its consistency with results in neuroscience (Krebs et al. 1998), where human motion trajectory has been observed to be best described as a summation of bell-shaped basis functions.

B-splines have been comprehensively studied in approximation theory and correspond to a spline function that has minimal support with respect to a given degree, smoothness and domain partition. B-splines are specified based on knot values, control points or de Boor points (Farin 1993). 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 $\mathcal{C}^2$ trajectory and to perform local refinement by adjusting the control points or knots. Moreover, the degree of B-spline curve is independent of the number of control points, which makes our method more flexible for the control of long trajectories compared to Bézier curve-based methods (Pettre et al. 2002; Yang and Sukkarieh 2010).

*3.2.1. Spline motion representation for rigid bodies* We will first describe the motion and trajectory for a rigid body. The trajectory of a rigid body, $\mathbf{q}(t)$, consists of two parts: translation $\mathbf{T}(t)$ and rotation $\mathbf{R}(t)$. The 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 many issues when representing the motion of the rotational component $\mathbf{R}(t)$. The rotational motion is in fact a curve in $\mathcal{SO}(3)$, and we cannot represent $\mathbf{R}(t)$ using a B-spline curve in $\mathcal{R}^3$. One method is to represent the rotation by Euler angles $(\alpha, \beta, \gamma)$, which turns the rotational motion into a curve in $\mathcal{R}^3$ configuration space. However, Euler angles cannot formalize some rotational motion due to intrinsic singularities, and it can be difficult to generate motion with a smooth angular velocity.

In robotics and computer animation, many applications represent rotational motion as quaternions that are singularity free (Kim et al. 1995; Nielson 2004; Shoemake 1985). Other techniques tend to use the recursive formulation of

B-splines, i.e. applying the de Boor algorithm (Farin 1993) to generate quaternion curves (Shoemake 1985). However, the spline constructed by these approaches may not have a closed formulation and the collision checking along such trajectories can be rather expensive, as described in Section 5.

We represent rotational motion using an exponential map (Murray et al. 1994). This representation is relatively simple and can be also used to generate 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 unit vector $\mathbf{u}$ as the rotation axis and $\theta$ as the rotation angle. The inverse of the exponential map is called the logarithmic map $\log(\cdot)$. Given the underlying constraints, we first construct a cubic B-spline $\mathbf{w}(t)$ in $\mathcal{R}^3$, and then use the exponential map to map it back onto $\mathcal{SO}(3)$. We define the mapping result

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

as the cubic B-spline curve in $\mathcal{SO}(3)$.

It is easy to prove that the exponential map transforms a B-spline curve in $\mathcal{R}^3$ to a B-spline curve in $\mathcal{SO}(3)$. Let $\mathbf{w}_1(t)$ and $\mathbf{w}_2(s)$ be two joined curves with $\mathcal{C}^r$ continuity, i.e. $d^k\mathbf{w}_1(t)/dt^k = d^k\mathbf{w}_2(s)/ds^k$, for $k = 0, 1, \ldots, r$. Then it is easy to prove that $d^k e^{\mathbf{w}_1(t)}/dt^k = d^k e^{\mathbf{w}_2(s)}/ds^k$, for $k = 0, 1, \ldots, r$, according to the derivative property of a matrix exponential (Najfeld and Havel 1995)

$$\frac{d}{dt}e^{\mathbf{X}(t)} = \int_0^1 e^{\alpha\mathbf{X}(t)}\frac{d\mathbf{X}(t)}{dt}e^{(1-\alpha)\mathbf{X}(t)}d\alpha \tag{2}$$

where $\mathbf{X}(t)$ is a square matrix function. Therefore, the $\mathcal{C}^r$ curve $\mathbf{w}(t)$ is mapped to a $\mathcal{C}^r$ curve in $\mathcal{SO}(3)$.

Overall, we formulate the motion of a rigid body by the translation spline $\mathbf{T}(t)$ and the rotation spline $\mathbf{R}(t) = \exp(\mathbf{w}(t))$, where $\mathbf{T}(t)$ and $\mathbf{w}(t)$ are both cubic B-splines in $\mathcal{R}^3$. We denote $\mathbf{f}^{t_0,\ldots,t_{m-2}}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}(t)$ as the cubic B-spline

$$\sum_{i=0}^{m}\mathbf{d}_i b_i(t) \tag{3}$$

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 $t_0 \le t \le t_{m-2}$. $\mathbf{f}$ is a uniform B-spline if $\{t_i\}_{i=0}^{m-2}$ correspond to uniform knot spacing. We also denote the spline function as $\mathbf{f}(t)$ for convenience.

*3.2.2. Spline motion representation for an articulated model* Given an articulated model $\mathcal{A}$, composed of $m$ rigid links $\mathcal{A}_1$, …, $\mathcal{A}_m$ with no closed loops, we use a directed acyclic graph to represent the link structure of $\mathcal{A}$. Each node in the graph denotes a link and each edge corresponds to the joint connecting two links. Many links may share the same parent link, but each individual link has only one parent. For the sake of simplicity, we assume that the index of the parent of link $i$ is $i-1$. Such notation can be easily modified when a parent has multiple child links.

For a given link $\mathcal{A}_i$, let $\{i\}$ denote its associated local frame, and let $\{0\}$ denote the global frame. We denote the orientation and translation of $\{i\}$ relative to $\{j\}$ as ${}^j\mathbf{R}_i(t)$ and ${}^j\mathbf{T}_i(t)$, respectively, where $j < i$. We define the articulated body $\mathcal{A}$ performing $\mathcal{C}^r$ or $\mathcal{G}^r$ spline motion when ${}^{j-1}\mathbf{R}_j(t)$ and ${}^{j-1}\mathbf{T}_j(t)$ are $\mathcal{C}^r$ or $\mathcal{G}^r$ splines in $\mathcal{SO}(3)$ and $\mathcal{R}^3$, respectively, for all $j = 1, 2, \ldots, m$. Here the definition of a spline for ${}^{j-1}\mathbf{R}_j(t)$ is the same as in rigid body motion. In other words, for articulated body spline motion, we require each link to perform rigid body spline motion in its parent link's local frame.

It is easy to see our definition of spline motion is consistent with the spline motion in the configuration space when each joint of the articulated model has a single degree of freedom, such as prismatic joints and revolute joints. However, when there are joints with more degrees of freedom, such as universal joints, our representation can provide more reasonable results.

## 4. 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. We use the cubic B-spline because it provides a good balance between smoothness constraints, spline refinement, cost of collision checking and oscillations 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, whether having a few or many obstacles. The output of the smoothing algorithm is a collision-free path, which is $\mathcal{C}^2$ almost everywhere except on some discrete points with only $\mathcal{C}^1$ or $\mathcal{G}^2$ continuity.

### *4.1. Algorithm overview*

The overall smoothing algorithm is shown in Algorithm 1. Given an input piecewise linear trajectory $\mathbf{q}_0(t), t \in [0, T]$, we first fit it with a smooth spline curve $\mathbf{q}(t)$ using spline interpolation, as shown in Figure 2(a). $\mathbf{q}(t)$ can approximate $\mathbf{q}_0(t)$ with arbitrary precision, if enough points are sampled along $\mathbf{q}_0(t)$. Therefore, we can always find a $\mathbf{q}(t)$ which is collision free and smooth. However, $\mathbf{q}(t)$ may still be jerky. Next, we shortcut it in an iterative manner. At each iteration, we randomly choose a portion $\mathbf{q}(t)|_{t_a \le t \le t_b}$ 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 4.2 and Figures 2(b) and 2(c)). We then use the spline collision detection (SCD) technique (refer to Section 5) to check whether every configuration along the trajectory $\mathbf{s}(t)$ is collision free. If there is a collision between the robot and an obstacle along that trajectory,

**Algorithm 1:** Spline-based shortcut algorithm.

**Input** : Trajectory $\mathbf{q}_0(t)|_{0\le t\le T}$, iteration count $N$
**Output**: A smooth and collision-free trajectory $\mathbf{s}(t)$
**begin**
  Fit $\mathbf{q}_0(t)$ with a smooth but collision-free trajectory $\mathbf{q}(t)$;
  **for** $i = 1$ **to** $N$ **do**
    Randomly choose $\mathbf{q}(t)|_{t_a\le t\le t_b}$ from the input trajectory;
    Construct a cubic B-spline $\mathbf{s}(t)$ interpolating $m+1$ points on $\mathbf{q}(t)|_{t_a\le t\le t_b}$;
    Perform spline collision checking (SCD) for $\mathbf{s}(t)$;
    `success` = Resolve the colliding segments in $\mathbf{s}(t)$ based on spline modification;
    $T_{\mathbf{s}} \leftarrow$ runtime of $\mathbf{s}(t)$;
    **if** `success` and $T_{\mathbf{s}} < t_b - t_a$ **then**
      Refine $\mathbf{s}(t)$ locally to satisfy velocity and acceleration constraints;
      $\Delta \leftarrow t_b - t_a - T_{\mathbf{s}}$;
      $\mathbf{q}(t) \leftarrow$
      $\mathbf{q}(t)|_{0\le t\le t_a} + \mathbf{s}(t) + \mathbf{q}(t)|_{t_b-\Delta\le t\le T-\Delta}$;

we resolve the collision using a recursive method that modifies the spline curve as discussed in Section 4.3. Once a collision-free trajectory is computed, we perform further local refinement (refer to Section 4.4) to satisfy the constraints corresponding to velocity and acceleration bounds.

## 4.2. Spline interpolation

Spline interpolation is used to transform a piecewise linear curve $\mathbf{q}_0(t)$ to a smooth spline $\mathbf{q}(t)$ and to perform spline shortcut operations. The spline interpolation algorithm is similar in both cases; the only difference is the boundary conditions.

*4.2.1. Transform $\mathbf{q}_0(t)$ to $\mathbf{q}(t)$* When transforming from $\mathbf{q}_0(t)$ to $\mathbf{q}(t)$, we first sample $m+1$ points along the piecewise linear trajectory $\mathbf{q}_0(t)$ at $t = t_0, \ldots, t_m$, and make sure that the two end points are included in the samples. To make sure that $\mathbf{q}(t)$ is collision free, $\mathbf{q}(t)$ must not deviate from $\mathbf{q}_0(t)$ too much. The parameter $m$ is chosen automatically in an iterative manner. It is initialized as the number of vertices in $\mathbf{q}_0(t)$. If the resulting $\mathbf{q}(t)$ is in-collision, we repeatedly double $m$ until a collision free $\mathbf{q}(t)$ is obtained. The resulting $m$ can be very large in some cases, e.g. when the original jerk trajectory is very close to an obstacle. However, in practice, we find that it is usually 3 to 5 times the number of vertices in $\mathbf{q}_0(t)$. $\mathbf{q}_0(t)$ may be derived by the motion planner without timing information. In this case, the initial parameterization is according to the chord length so that the samples are distributed uniformly along the entire path. Next, we reparameterize the curve according to the chord length and the joints' velocity limit

$$u_i - u_{i-1} = \max_k \frac{\int_{t_{i-1}}^{t_i} \mathbf{q}_0^k(t)\,dt}{\mathbf{v}_{max}^k} \tag{4}$$

where $u_0 = 0$. That is, we decide the timing information $u_i$ to make sure that the motor can finish the motion with the maximum speed. Finally, the interpolation scheme computes a cubic B-spline $\mathbf{q}(u)|_{u_0\le u\le u_m}$ that interpolates the sampled points, i.e. $\mathbf{q}(u_i) = \mathbf{q}_0(t_i), \forall i \in \{0, 1, \ldots, m\}$.

According to Farin (1993), the $\mathcal{C}^2$-smooth interpolating cubic spline can be constructed by solving a linear system

$$\bar{\mathbf{A}}\bar{\mathbf{d}} = \bar{\mathbf{r}} \tag{5}$$

where

$$\bar{\mathbf{A}} = \begin{pmatrix} \beta_0 & \gamma_0 & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & \\ & & \ddots & & & \\ & & & \alpha_{m-1} & \beta_{m-1} & \gamma_{m-1} \\ & & & & \alpha_m & \beta_m \end{pmatrix}$$

is an $(m+1) \times (m+1)$ tridiagonal coefficient matrix, $\bar{\mathbf{r}} = [\mathbf{r}_0\ \mathbf{r}_1 \cdots \mathbf{r}_{m-1}\ \mathbf{r}_m]^{\mathrm{T}}$ is an $(m+1) \times d$ matrix for the sampled points and $\bar{\mathbf{d}} = [\mathbf{d}_0\ \mathbf{d}_1 \cdots \mathbf{d}_{m-1}\ \mathbf{d}_m]^{\mathrm{T}}$ is an $(m+1) \times d$ matrix for unknown de Boor control points.

The matrices $\bar{\mathbf{A}}$ and $\bar{\mathbf{b}}$ are determined by the parameterization of the curve and its boundary condition. Here we use the natural boundary condition, i.e.

$$\frac{d^2}{du^2}\mathbf{q}\Big|_{u_0} = \frac{d^2}{du^2}\mathbf{q}\Big|_{u_m} = \mathbf{0}$$

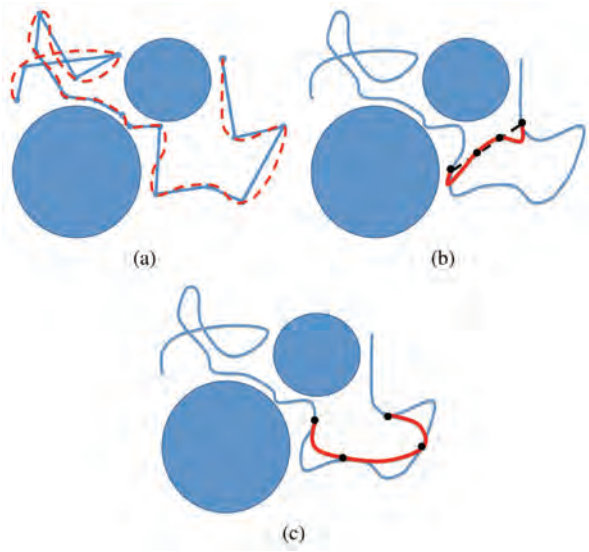


**Fig. 2.** Overview of the spline shortcut scheme: (a) Transform $\mathbf{q}_0(t)$ to $\mathbf{q}(t)$ as described in Section 4.2.1. (b), (c) Spline shortcut described in Section 4.2.2. (b) uses the linear trajectory $\overrightarrow{\mathbf{q}(t_a)\,\mathbf{q}(t_b)}$ as the guidance trajectory while (c) uses part of the spline trajectory $\mathbf{q}(t)|_{t_a\le t\le t_b}$ as the guidance trajectory.

As a result, we have the following setting for the linear system (Farin 1993)

$$\begin{aligned}
&\alpha_k = \frac{\Delta_k^2}{\Delta_{k-2}+\Delta_{k-1}+\Delta_k}, \quad 1 \le k \le m-1\\
&\beta_k = \frac{\Delta_k(\Delta_{k-2}+\Delta_{k-1})}{\Delta_{k-2}+\Delta_{k-1}+\Delta_k} + \frac{\Delta_{k-1}(\Delta_k+\Delta_{k+1})}{\Delta_{k-1}+\Delta_k+\Delta_{k+1}},\\
&\quad 1 \le k \le m-1\\
&\gamma_k = \frac{\Delta_{k-1}^2}{\Delta_{k-1}+\Delta_k+\Delta_{k+1}}, \quad 1 \le k \le m-1\\
&\alpha_m = -\Delta_{m-1}\\
&\beta_0 = \Delta_0 + 2\Delta_1\\
&\beta_m = \Delta_{m-2} + 2\Delta_{m-1}\\
&\gamma_0 = -\Delta_0\\
&\mathbf{r}_k = (\Delta_{k-1}+\Delta_k)\,\mathbf{q}(t_k), \quad 1 \le k \le m-2\\
&\mathbf{r}_0 = (\Delta_0+\Delta_1)\,\mathbf{q}(t_0)\\
&\mathbf{r}_m = (\Delta_{m-2}+\Delta_{m-1})\,\mathbf{q}(t_m)
\end{aligned} \tag{6}$$

where $\Delta_i = u_{i+1} - u_i$, $0 \le i \le m-1$, is the time step between two sample points and $\Delta_{-1} = \Delta_m = 0$.

The solution to the linear system is represented by $\{\mathbf{d}_k\}_{k=0}^m$. Along with two additional points $\mathbf{d}_{-1} = \mathbf{q}_0(t_0)$ and $\mathbf{d}_{m+1} = \mathbf{q}_0(t_m)$, these points constitute the de Boor control point set for the interpolating spline $\mathbf{f}^{t_0,\ldots,t_m}_{\mathbf{d}_{-1},\ldots,\mathbf{d}_m,\mathbf{d}_{m+1}}(t)$, which is used to represent the trajectory.

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 $\mathbf{q}(t)$.

*4.2.2. Spline shortcut* When performing the spline shortcut, we first randomly sample two points on $\mathbf{q}(t)$ at $t_a$ and $t_b$. Then we sample $m+1$ points along a guidance trajectory $\mathbf{g}(t)$ at $t = t_0, \ldots, t_m$, where $t_0 = t_a$ and $t_m = t_b$. The guidance trajectory $\mathbf{g}(t)$ can be a linear trajectory $\overrightarrow{\mathbf{q}(t_a)\,\mathbf{q}(t_b)}$ (Figure 2(b)) or the spline portion $\mathbf{q}(t)\,|_{t_a \le t \le t_b}$ (Figure 2(c)). We randomly choose one type of guidance trajectory for the shortcut operation. When shortcut iteration starts, we choose the linear trajectory with a higher probability because it can provide a more efficient shortcut; we gradually increase the probability for applying the spline shortcut because it has a higher potential for giving a successful and smooth shortcut. The velocities at the two end points are $\mathbf{q}'(t_a^-)$ and $\mathbf{q}'(t_b^+)$, which can be computed from the trajectory segments adjacent to $\mathbf{q}(t)\,|_{t_a \le t \le t_b}$ and are used to guarantee $\mathcal{C}^1$ continuity on the boundaries of the resulting spline. Then the interpolation scheme computes a cubic B-spline $\mathbf{s}(u)\,|_{u_0 \le u \le u_0}$, which interpolates the sampled points, i.e. $\mathbf{s}(u_i) = \mathbf{g}(t_i)$, $\forall i \in \{0, 1, \ldots, m\}$, $\mathbf{s}'(u_0) = \mathbf{q}'(t_0)$ and $\mathbf{s}'(u_m) = \mathbf{q}'(t_m)$. The parameterization of the shortcut portion $\mathbf{s}(u)$ is determined according to the velocity limit and the Euclidean distance between adjacent samples, which is an estimate of the chord length after the shortcut

$$u_i - u_{i-1} = \max_k \frac{|\mathbf{g}^k(t_i) - \mathbf{g}^k(t_{i-1})|}{\mathbf{v}^k_{max}} \le t_i - t_{i-1} \tag{7}$$

The matrices $\bar{\mathbf{A}}$ and $\bar{\mathbf{b}}$ for the resulting linear system are almost the same as the ones in Section 4.2.1 except for a few terms

$$\begin{aligned}
&\alpha_m = 0; \quad \beta_0 = \beta_m = 1; \quad \gamma_0 = 0\\
&\mathbf{r}_k = (\Delta_{k-1}+\Delta_k)\,\mathbf{g}(t_k), \quad 1 \le k \le m-2\\
&\mathbf{r}_0 = \mathbf{q}(t_0) + \frac{\Delta_0 \mathbf{q}'(t_0)}{3}\\
&\mathbf{r}_m = \mathbf{q}(t_m) - \frac{\Delta_{m-1}\mathbf{q}'(t_m)}{3}
\end{aligned} \tag{8}$$

where $\Delta_i = u_{i+1} - u_i$, $0 \le i \le m-1$, is the time step between two sample points and $\Delta_{-1} = \Delta_m = 0$. The difference is due to the $\mathcal{C}^1$ boundary condition. From the solutions of the linear system and two additional points $\mathbf{d}_{-1} = \mathbf{q}(t_0)$ and $\mathbf{d}_{m+1} = \mathbf{q}(t_m)$, we can compute a shortcut motion portion $\mathbf{s}(u)$, which is then validated using the spline collision checking routine (Section 5). If it is collision free, then it is merged with $\mathbf{q}(t)\,|_{0 \le t \le t_a}$ and $\mathbf{q}(t)\,|_{t_b \le t \le T}$ to get a shortcut trajectory. Otherwise, it needs an additional modification step to resolve collision where possible (Section 4.3).

We can also change the boundary conditions of the linear system to achieve $\mathcal{G}^2$ continuity instead, which is useful when we want to reduce any mechanical oscillations of the robot. According to Derose and Barsky (Barsky and DeRose 1989; DeRose and Barsky 1988), two curves $\mathbf{q}(t), t \in [t_0, t_1]$, and $\mathbf{s}(u), u \in [u_0, u_1]$, have $\mathcal{G}^2$ continuity at $\mathbf{q}(t_1) = \mathbf{s}(u_0)$ if and only if there exist real numbers $\eta_1 > 0$ and $\eta_2 \in \mathcal{R}$ so that

$$\begin{aligned}
\frac{d}{dt}\mathbf{s}\Big|_{u_0} &= \eta_1 \frac{d}{dt}\mathbf{q}\Big|_{t_1}\\
\frac{d^2}{dt^2}\mathbf{s}\Big|_{u_0} &= \eta_1^2 \frac{d^2}{dt^2}\mathbf{q}\Big|_{t_1} + \eta_2 \frac{d}{dt}\mathbf{q}\Big|_{t_1}
\end{aligned} \tag{9}$$

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

For the cubic B-spline in equation (5), the $\mathcal{G}^2$ boundary condition can be formalized as

$$\begin{aligned}
\frac{3}{\Delta_0}(\mathbf{d}_0 - \mathbf{d}_{-1}) &= \eta_1 \mathbf{q}'(t_a^-)\\
\frac{6}{\Delta_0^2}(\mathbf{d}_{-1} - 2\mathbf{d}_0 + \mathbf{d}_1) &= \eta_1^2 \mathbf{q}''(t_a^-) + \eta_2 \mathbf{q}'(t_a^-)\\
\frac{3}{\Delta_{m-1}}(\mathbf{d}_{m+1} - \mathbf{d}_m) &= \eta_3 \mathbf{q}'(t_b^+)\\
\frac{6}{\Delta_{m-1}^2}(\mathbf{d}_{m+1} - 2\mathbf{d}_m + \mathbf{d}_{m-1}) &= \eta_3^2 \mathbf{q}''(t_b^+) + \eta_4 \mathbf{q}'(t_b^+)
\end{aligned} \tag{10}$$

As a result, to achieve an interpolated spline with $\mathcal{G}^2$ continuity, we only need to construct a new algebraic system by replacing the $\mathcal{C}^1$ boundary condition in equation (5) with the new $\mathcal{G}^2$ boundary condition. Unlike the original linear system, the new system is: (a) non-linear due to $\eta_1^2$ and $\eta_3^2$ terms, and (b) overdetermined, because originally we have $2d$ degrees of freedom on the boundary and now we provide $4d-4$ constraints, where $d$, as mentioned before, is the dimension of the configuration space. To solve this problem, we use an approximation scheme instead of the interpolation scheme (Farin 1993), i.e. we try to find a curve $\mathbf{s}(u)$ that is close to the given points. Based on equation (3), we can formulate it as an optimization problem

$$\underset{\mathbf{d}_0,\ldots\mathbf{d}_m,\eta_1,\ldots,\eta_4}{\text{maximize}} \sum_{i=0}^{m} \left\| \mathbf{q}(t_i) - \sum_{k=-1}^{m+1} \mathbf{d}_k b_k(u_i) \right\|^2 \qquad (11)$$

subject to equation (10),

where $u_i$ is the correspondence of $t_i$ in the parameterization of curve $\mathbf{s}(u)$.

We can solve this non-linear optimization using non-linear solvers. However, an expectation-maximum (EM)-type method is also possible: we first fix $\eta_i$ and compute $\mathbf{d}_i$ using a linear optimization, next we fix $\mathbf{d}_i$ and solve $\eta_i$. This process repeats several times or until the solution does not improve. For initialization, we use $\eta_1 = \eta_3 = 1$ and $\eta_2 = \eta_4 = 0$.

Notice that the $\mathcal{G}^2$ interpolation is much simpler for articulated models made with 1-DOF joints (i.e. $d = 1$ for all joints) because $\mathcal{G}^2$ is equivalent to $\mathcal{C}^1$ in this case (i.e. we can set $\eta_1 = \eta_3 = 1$ to solve $\mathbf{d}_i$ and then solve the corresponding $\eta_2$ and $\eta_4$ to satisfy the $\mathcal{G}^2$ constraint).

Here 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 (Farin 1993; Kim et al. 1995). 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 as for 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 (Barr et al. 1992).

### 4.3. Recursive spline modification to resolve collisions

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 5. If a collision is found, we resolve it by modifying the spline curve. As shown in Figure 3, our solution is based on the locality of spline curves. Suppose that a collision between the B-spline curve and an obstacle 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 $\mathbf{q}(t)$, which is collision free. The modified spline will still be almost $\mathcal{C}^2$ if we use the $\mathcal{C}^1$ or $\mathcal{G}^2$ boundary conditions as before, though there will be two more isolated $\mathcal{C}^1$ or $\mathcal{G}^2$ points between $t_i$ and $t_j$.

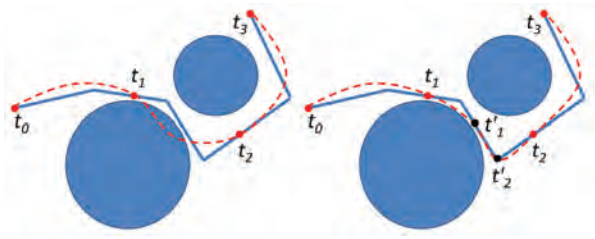


**Fig. 3.** For cubic B-spline $\mathbf{f}^{t_0,\ldots,t_3}(t), t \in [t_0, t_3]$, a collision happens during the interval $[t_1, t_2]$. We recursively add new knots within the interval and refine the spline $\mathbf{f}^{t_1,t'_1,t'_2,t_2}(t), t \in [t_1, t_2]$ (shown on the right). The splines defined 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 $[t_0, t_3]$.

The above procedure can be applied recursively whenever a collision is detected within any subinterval. 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 $\mathbf{q}(t)$ and repeats the process.

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 4.4. Therefore, we use the spline subdivision technique (Farin 1993) to compute the $m+3$ new de Boor points $\tilde{\mathbf{d}}_{-1}, \ldots, \tilde{\mathbf{d}}_{m+1}$ for each spline that is defined on the collision-free intervals. The final output is an almost $\mathcal{C}^2$ collision-free trajectory.

### 4.4. 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 3.1. It is possible to satisfy these constraints by formulating an optimization problem for the control points $\mathbf{d}_i$, as was shown in recent work (Lee et al. 2005; Lengagne et al. 2011). However, as we have already taken into account the velocity limits when determining the parameterization of the trajectory (refer to Section 4.2.1), usually a small adjustment is sufficient to make the spline satisfy these additional constraints. Therefore,

to avoid suffering the performance penalties of the heavyweight optimization framework, we use a simple control point adjustment technique.

Our solution exploits the properties of B-splines. It iteratively adjusts the de Boor control points successively so that the resulting spline curve can satisfy these constraints. For convenience, here we assume that the spline is a uniform B-spline; similar formulation can be derived for non-uniform splines.

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

$$\begin{aligned}\mathbf{f}'(t_i) &= \frac{\mathbf{d}_{i+2}-\mathbf{d}_i}{2\Delta}\\ \mathbf{f}'(\frac{t_i+t_{i+1}}{2}) &= \frac{\mathbf{d}_{i+3}+5\mathbf{d}_{i+2}-5\mathbf{d}_{i+1}-\mathbf{d}_i}{8\Delta}\end{aligned} \tag{12}$$

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

We define the limit velocity $\bar{\mathbf{v}}$ as the velocity at time $\bar{t}$ when $\mathbf{f}''(\bar{t}) = 0$, i.e. it is the maximum or minimum velocity in one interval $[t_a, t_b]$. The limit velocity within the interval $[t_i, t_{i+1}]$ is

$$\begin{aligned}\bar{\mathbf{v}}[t_i,t_{i+1}] &= \frac{1}{2\Delta}\left(\mathbf{d}_{i+2}-\mathbf{d}_i+\frac{(\mathbf{d}_{i+2}-2\mathbf{d}_{i+1}+\mathbf{d}_i)^2}{-\mathbf{d}_{i+3}+3\mathbf{d}_{i+2}-3\mathbf{d}_{i+1}+\mathbf{d}_i}\right)\\ &= \mathbf{f}'(t_i)+\frac{1}{8}\frac{\left(4\mathbf{f}'(\frac{t_i+t_{i+1}}{2})-\mathbf{f}'(t_{i+1})-3\mathbf{f}'(t_i)\right)^2}{2\mathbf{f}'\left(\frac{t_i+t_{i+1}}{2}\right)-2\mathbf{f}'(t_{i+1})-\mathbf{f}'(t_i)}\end{aligned} \tag{13}$$

From this equation, we find that $\bar{\mathbf{v}}[t_i, t_{i+1}]$ 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 $\{\mathbf{f}'((t_i + t_{i+1})/2)\}_{i=0}^{m-3}$, we can control the magnitude of the velocity of the spline $\mathbf{f}'(t)$, for all $t \in [t_0, t_{m-2}]$.

Therefore, we check the bounds on $\mathbf{f}'(t_i)$ and $\mathbf{f}'((t_i + t_{i+1})/2)$ for $i$ from 1 to $m-3$. If $|\mathbf{f}'(t_i)|$ or $|\mathbf{f}'((t_i+t_{i+1})/2)|$ 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 $\bar{\mathbf{v}}$ 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 $\mathbf{f}''(t_i) = (\mathbf{d}_{i+2} - 2\mathbf{d}_{i+1} + \mathbf{d}_i)/\Delta^2$. We only need to adjust $\{\mathbf{d}_i\}_{i=3}^{m-3}$ to make sure that the acceleration is bounded by $\mathbf{a}_{max}$. We perform the adjustment in a greedy manner: if $\mathbf{f}''(t_i)$ is not bounded by $\mathbf{a}_{max}$, we perform a local search around the current value of $\mathbf{d}_{i+1}$ and find a suitable new $\mathbf{d}_{i+1}$. Such an adjustment will influence the accelerations at adjacent time points and therefore we need to repeat the adjustment for several rounds. Whenever the de Boor control points are adjusted, we need to perform a continuous collision query to ensure that the modified spline is also collision free. If the modified spline trajectory is collision free, we use these adjusted control points. Otherwise, we perform the adjustment step again using a smaller $\lambda$ (we used $0.5\lambda$ in our experiments). If the modified spline still collides with the obstacles, we discard the adjustment step and use the earlier path. Overall, our algorithm is designed to ensure computation of a collision-free path (i.e. a hard constraint). On the other hand, we treat velocity and acceleration as soft constraints.

Overall, our spline fitting and refinement schemes tend to maintain second-order continuity and ensure that the computed trajectories are almost $\mathcal{C}^2$. As a result, if our algorithm is able to compute a collision-free path, it would correspond to an almost $\mathcal{C}^2$ collision-free trajectory.

## 5. Exact collision detection

Collision checking is an integral component of any planning and smoothing algorithm. Many prior optimization-based smoothing algorithms cannot provide guarantees about collision-free path computation, especially when the environment has a high number of obstacles or narrow passages, because they usually approximate obstacles using potential fields or distance fields (Ratliff et al. 2009). At the same time, many shortcut algorithms tend to use rather simple or restricted smoothing functions so that they can easily perform collision checking. Our goal is to develop efficient algorithms that can be used for exact collision checking with spline-based smoothing functions.

The simplest algorithms for collision checking, compute discrete samples along a path or trajectory and check them for collisions. The formal definitions for discrete collision detection (DCD) and trajectory collision checking based on DCD (TDCD) are as follows:

**Definition** Given two objects $\mathcal{A}$ and $\mathcal{B}$, the discrete collision query returns a yes/no answer for whether the two objects are in collision or not, i.e. whether

$$\mathcal{A} \cap \mathcal{B} \neq \emptyset$$

Given the motion of the two objects $\mathcal{A}(t)$ and $\mathcal{B}(t)$, where $t \in [0, 1]$, trajectory collision checking based on DCD generates $n$ samples $\mathcal{A}(t_1)$, $\mathcal{A}(t_2)$, …, $\mathcal{A}(t_n)$, (or $\mathcal{B}(t_1)$, $\mathcal{B}(t_2)$, …, and $\mathcal{B}(t_n)$) along the path and returns a yes/no answer for whether the two objects are in collision within interval $[0, 1]$, i.e. whether

$$\exists i \in \{1, 2, \ldots, n\}, \quad \mathcal{A}(t_i) \cap \mathcal{B}(t_i) \neq \emptyset$$

If a collision occurs, it also returns an approximate *time of first contact* (TOC)

$$\texttt{toc}_{TDCD} = \min\{t_i : \mathcal{A}(t_i) \cap \mathcal{B}(t_i) \neq \emptyset\}$$

However, the resulting techniques can miss collisions due to poor sampling. Moreover, they cannot provide a high-precision TOC, which is useful in many applications such as planning and grasping. 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 collision missing problem of discrete collision detection. The formal definition for a CCD query is as follows:

**Definition** Given two objects $\mathcal{A}$ and $\mathcal{B}$ as well as their motion $\mathcal{A}(t)$ and $\mathcal{B}(t)$, where $t \in [0,1]$, the continuous collision query returns a yes/no answer for whether the two objects are in collision within interval $[0,1]$, i.e. whether

$$\exists t \in [0,1], \quad \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset$$

If a collision occurs, it also returns the first time of contact

$$\mathtt{toc}_{CCD} = \inf\{t : \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset\}$$

However, prior CCD algorithms are mainly limited to linear deformation (Govindaraju et al. 2005) or linear/screw/parabolic trajectories in the configuration space (Tang et al. 2011; Zhang et al. 2007), and no fast methods are known for CCD along arbitrary spline trajectories. We also define the motion bound in a new way, which can provide tighter motion bound estimation compared to those used in previous work.

## 5.1. Efficient spline collision detection algorithm

Our spline collision detection (SCD) algorithm is based on *conservative advancement* (CA) (Pan et al. 2011; Tang et al. 2011; Zhang et al. 2006). Figure 4 shows an overview of the CA method. Suppose we are given two convex objects $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ is moving and $\mathcal{B}$ is fixed. Let $\mathcal{A}(t)$ be $\mathcal{A}$ at time $t \in [0,1]$. The basic idea of CA is to incrementally advance $\mathcal{A}$ for a small time step $\Delta t_k$ toward $\mathcal{B}$ while avoiding a 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 $\mu$ on the motion of $\mathcal{A}(t)$ projected onto the direction $\mathbf{n}$ is estimated. Finally the advancing length at the step $k$ is calculated by

$$\Delta t_k = \frac{d(\mathcal{A}(t), \mathcal{B})}{\mu} \tag{14}$$

$\mathcal{A}$ advances by $\Delta t_k$ 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, $\tau$ is the first time of contact.

The CA technique can also be used to handle CCD between two moving objects $\mathcal{A}(t)$ and $\mathcal{B}(t)$. One way is to compute the relational motion between the two objects and then apply the original CA. However, this method is difficult to apply for articulated bodies. Another way is to estimate the motion bounds $\mu_{\mathcal{A}}$ and $\mu_{\mathcal{B}}$ for two objects respectively, and the advancing length computation is updated as

$$\Delta t_k = \frac{d(\mathcal{A}(t), \mathcal{B}(t))}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}}} \tag{15}$$

**Algorithm 2:** Spline continuous collision detection.

**Input** : Two objects $\mathcal{A}$ and $\mathcal{B}$; $\mathcal{A}$'s spline motion function $\mathbf{f}^{t_0,\ldots,t_{m-2}}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}(t)$
**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** $t \leq t_i$ **do**
      Compute current distance $d(\mathcal{A}(t), \mathcal{B})$
      Estimate the motion bound $\mu$
      Compute conservative advancement $\Delta t = d(\mathcal{A}(t), \mathcal{B})/\mu$
      $t \leftarrow t + \Delta t$
      Check collision between $\mathcal{A}(t)$ and $\mathcal{B}$
      **if** collision **then**
        **return** $\tau = t$
  **return** collision-free

We apply the CA method to B-spline trajectories. Algorithm 2 shows the overall algorithm SCD. Let us assume that $\mathcal{A}$'s trajectory is a B-spline with $m-2$ segments $\mathbf{f}^{t_0,\ldots,t_{m-2}}_{\mathbf{d}_0,\ldots,\mathbf{d}_m}(t)$. 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 $\mu$ for the given motion trajectory. Tang et al. (2011) present the motion bounds for linear and screw motion in the configuration 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 the exponential map, which is also a B-spline in $\mathcal{R}^3$.

## 5.2. Motion bound definition

As mentioned above, the motion bound $\mu$ is an upper bound on the motion of $\mathcal{A}(t)$ projected onto the direction $\mathbf{n}$. Previous work (Pan et al. 2011; Tang et al. 2011; Zhang et al. 2006) defined the motion bound as

$$\mu = \max_{\mathbf{p}_i \in \mathcal{A}} \int_0^1 |\dot{\mathbf{p}}_i(t) \cdot \mathbf{n}| dt \tag{16}$$

where $\mathbf{p}_i$ is the $i$-th point on the rigid body $\mathcal{A}$. However, this definition itself only gives an approximated bound. For example, if object $\mathcal{A}$ is moving away from a static object $\mathcal{B}$, an ideal motion bound definition should give a result less than or equal to zero (i.e. a collision will not happen during the time step), so that the CA iteration can stop and return collision free immediately. However, the bound definition in equation (16) will still give a positive motion bound, which may result in many more CA iterations, especially when the speed of $\mathcal{A}$ is high.

To avoid the above disadvantage, we define the motion bound in a tighter manner: it is the maximum distance along the direction $\mathbf{n}$ that one object moves toward the other objects

$$\mu = \max_{\tau\in[0,1]} \max_{\mathbf{p}_i\in\mathcal{A}} \int_0^\tau \dot{\mathbf{p}}_i(t) \cdot \mathbf{n} dt \tag{17}$$

Notice that if $\mathcal{A}$ moves away from $\mathcal{B}$, then $\dot{\mathbf{p}}_i(t) \cdot \mathbf{n} < 0$ for all $t \in [0,1]$ and therefore $\mu = 0 \leq 0$. Moreover, if $\mathcal{A}$ moves towards $\mathcal{B}$ within time interval $[0, t_0]$ and then moves away from $\mathcal{B}$ within $[t_0, 1]$, then the motion bound should be the distance $\mathcal{A}$ has moved at time $t_0$, which is exactly what is returned by our new motion bound definition: $\mu = \max_{\mathbf{p}_i\in\mathcal{A}} \int_0^{t_0} \dot{\mathbf{p}}_i(t) \cdot \mathbf{n} dt$. If $\mathcal{B}$ also moves, equation (17) can also be used to compute $\mathcal{B}$'s motion bound, but the projection direction should be $-\mathbf{n}$ instead.

Using the new motion bound definition, we can give a tighter motion bound for a rigid body or an articulated body. In Appendix B, we give the details for how to obtain a better motion bound using the new definition for a rigid body performing linear or screw motion. In Appendix C, we show how to apply the same technique to improve the motion bound of articulated bodies.

Computing the motion bound directly based on equations (16) or (17) is slow, because we need to compute the bound for each point on the object and then select the maximum one. Moreover, it cannot be applied to non-convex objects. To solve these difficulties, we use the swept sphere volume (SSV) hierarchy technique to accelerate motion bound computation and extend it to non-convex objects (Tang et al. 2011). As a preprocess, we compute the SSV hierarchy for both objects. At runtime, we apply CA to the nodes in the SSV hierarchy in an iterative manner. As for the convex case in equations (14) or (15), we require the closest distance between SSVs and the motion bound for SSVs. The distance is obtained as a byproduct of the SSV algorithm and we need to estimate a tight motion bound for each SSV node $\alpha$, which is denoted as $\mu_\alpha$.

## 5.3. Motion bound for cubic B-splines

To apply the CA technique for the spline motion, the main challenge is to compute a tight motion bound for the moving object $\mathcal{A}$. The proofs of the following theorems are shown in Appendix D.

We first assume $\mathcal{A}$ is convex and the following theorem calculates its motion bound:

**Theorem 1.** *Suppose $\mathbf{p}_i$ is a point on the convex object $\mathcal{A}$ and $\mathbf{r}_i$ is its local coordinate with respect to the origin of $\mathcal{A}$'s local frame. $\mathcal{A}$ moves according to spline motion $\mathbf{p}_i(t) = \mathbf{T}(t) + \mathbf{R}(t)\mathbf{r}_i$. Then the spline motion bound $\mu$ with respect to the closest direction $\mathbf{n}$ is given by*

$$\mu \leq 2 \max_i(|\mathbf{r}_i \cdot \mathbf{n}| + \|\mathbf{r}_i \times \mathbf{n}\|) \min(1, \max_{t\in[0,1]} \|\mathbf{w}'(t)\|) + \frac{1}{6}(A\tilde{\tau}^3 + B\tilde{\tau}^2 + C\tilde{\tau}) \tag{18}$$

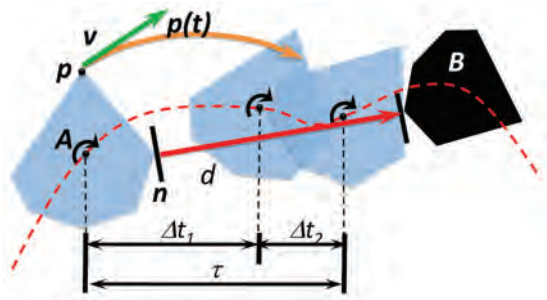


**Fig. 4.** Continuous collision detection between $\mathcal{A}(t)$ and $\mathcal{B}$. $\mathcal{A}(t)$ is a spline motion. $\mathbf{p}$ is a point on $\mathcal{A}$. $\mathbf{p}$'s motion function is $\mathbf{p}(t)$ and the velocity is $\mathbf{v}(t) = \mathbf{p}'(t)$. $d$ is the closest distance between the two objects and $\mathbf{n}$ is the direction of closest distance. $\tau = \Delta t_1 + \Delta t_2$ is the first time of contact. $\Delta t_1$ and $\Delta t_2$ correspond to the advances during two successive steps.

*where $\mathbf{w}(t)$ is the preimage of $\mathbf{R}(t)$ under the exponential map; A, B and C are constants depending on $\mathbf{T}(t)$ and $\tilde{\tau} = \text{argmax}_{\tau\in[0,1]} A\tau^3 + B\tau^2 + C\tau$.*

We then extend the spline motion bound to non-convex objects by using the swept sphere volume hierarchy technique. 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 the following motion bound result for the SSV:

**Theorem 2.** *The spline motion bound for one SSV $\alpha$ is given as*

$$\mu_\alpha = 2(3r + \max_k |\mathbf{c}_k \cdot \mathbf{n}| + \max_k |\mathbf{c}_k \times \mathbf{n}| \max_k \|\mathbf{c}_k\|) \cdot \min(1, \max_{t\in[0,1]} \|\mathbf{w}'(t)\|) + \frac{1}{6}(A\tilde{\tau}^3 + B\tilde{\tau}^2 + C\tilde{\tau}) \tag{19}$$

*where the symbols are the same as in Theorem 1.*

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.

Finally, we can also compute the spline motion bound for articulated bodies. Suppose the spline motion for articulated models is as defined in Section 3.2.2. Using our new motion bound definition and the technique described in Zhang et al. (2007) and assuming that all the links are convex objects, we have:

**Table 1:** Geometric complexity of our benchmarks.

| | Piano | Gear | Wiper | CarSeat | AlphaPuzzle | Bridge |
|---|---|---|---|---|---|---|
| No of DOFs | 6 | 3 | 6 | 6 | 6 | 40 |
| No of faces | 952 | 7188 | 26,766 | 245,127 | 2088 | 31,718 |

**Theorem 3.** *The spline motion bound for the i-th link of the articulated model is*

$$\mu^i \leq \mathbf{n} \cdot \left( \max_{t\in[0,1]} {}^0\mathbf{T}_1(t) - {}^0\mathbf{T}_1(0) \right) + \max_{t\in[0,1]} \|{}^0\mathbf{w}_1'(t)\| \sum_{j=1}^{i} \|^{j-1}\mathbf{L}_j\|_\mu + \sum_{j=2}^{i} \left( \max_{t\in[0,1]} \|^{j-1}\mathbf{T}_j'(t)\| + \max_{t\in[0,1]} \|^{j-1}\mathbf{w}_j'(t)\| \sum_{k=j}^{i} \|^{k-1}\mathbf{L}_k\|_\mu \right) \quad (20)$$

*where* ${}^{j-1}\mathbf{w}_j$ *is the preimage of* ${}^{j-1}\mathbf{R}_j$ *under the exponential map and* ${}^j\mathbf{L}_i$ *denotes the vector from the origin of frame* $\{i\}$ *to that of frame* $\{j\}$. $\|^{j-1}\mathbf{L}_j\|_\mu$ *is the motion bound for* ${}^{j-1}\mathbf{L}_j$ *and when* $j = i$ *we have* $\|^{i-1}\mathbf{L}_i\|_\mu = \max_{\mathbf{p}\in\mathcal{A}_i} \|^{i-1}\mathbf{r}_i\|$ *where* ${}^{i-1}\mathbf{r}_i$ *is each point* $\mathbf{p}$*'s local coordinate in frame* $\{i-1\}$.

We can also extend the above bound to articulated models containing non-convex objects using the swept sphere volume hierarchy technique and we have:

**Theorem 4.** *The spline motion bound for SSV node* $\alpha$ *of the i-th link of the articulated model is*

$$\mu_\alpha^i \leq \mathbf{n} \cdot \left( \max_{t\in[0,1]} {}^0\mathbf{T}_1(t) - {}^0\mathbf{T}_1(0) \right) + \max_{t\in[0,1]} \|{}^0\mathbf{w}_1'(t)\| \sum_{j=1}^{i} \|^{j-1}\mathbf{L}_j\|_\mu + \sum_{j=2}^{i} \left( \max_{t\in[0,1]} \|^{j-1}\mathbf{T}_j'(t)\| + \max_{t\in[0,1]} \|^{j-1}\mathbf{w}_j'(t)\| \sum_{k=j}^{i} \|^{k-1}\mathbf{L}_k\|_\mu \right) \quad (21)$$

*where* $\|^{i-1}\mathbf{L}_i\|_\mu = r + \max_k \|\mathbf{c}_k\|$ *and the other symbols have the same meaning as in Theorem 3.* $r$ *and* $\mathbf{c}_k$ *are the parameters of the SSV.*

## 6. Results

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

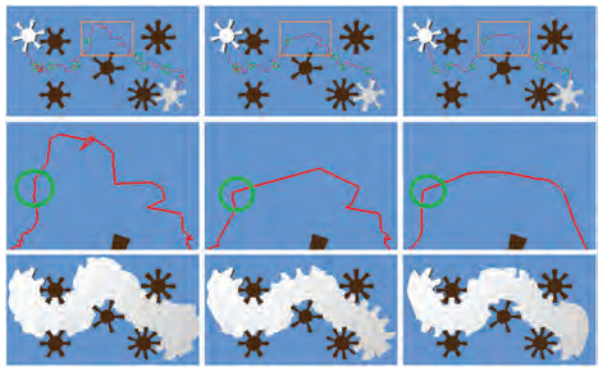

**Fig. 5.** Results for the gear benchmark. The three columns show results computed by the planner, the linear shortcut algorithm and our B-spline smoothing algorithm, respectively. The first row shows the trajectories traversed by the origin of the robot; the second row shows a zoomed view of the trajectories within the orange box in the first row; the third row shows the swept volumes for the trajectories computed by the three 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.

### 6.1. Rigid robots

The results of five different benchmarks for rigid robots are shown in Figures 5, 6 and 7. Table 1 summarizes the geometric complexity and DOFs for each benchmark. Among the five benchmarks, only the piano benchmark does not have narrow passages. The rest of the benchmarks are challenging for sample-based motion planners and smoothing algorithms. We use a variant of a sample-based motion planner (Pan et al. 2010) 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, the smoothed trajectory computed using the linear shortcut smoothing algorithm (Yamane et al. 2004), and the smoothed trajectory computed using our spline-based algorithm. In particular, we visualize these trajectories for the local coordinate origin of each rigid robot (Figures 5, 6 and 7). Compared to the linear shortcut algorithm, our spline-based smoothing algorithm computes almost $\mathcal{C}^2$ trajectories. Moreover, for the gear benchmark (Figure 5), we show the swept volume of the robot generated using the trajectories computed by the different algorithms.

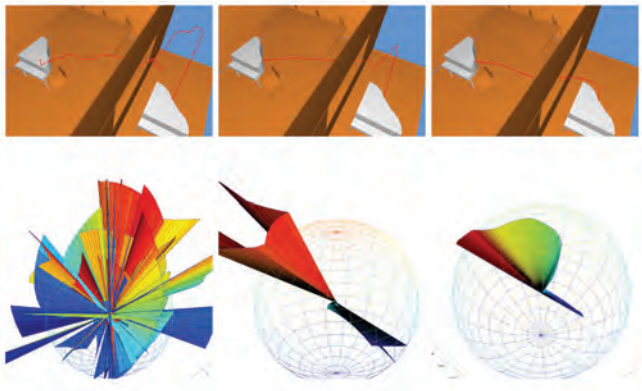

**Fig. 6.** Results for piano benchmark. The three columns show the paths computed by the sample-based planner, the linear shortcut algorithm and our method, respectively. The first row shows the trajectory traversed by the origin of the robot. The second row visualizes the rotational motion of the robot obtained by the three methods. Each vector originating from the sphere center represents a rotation: the magnitude is for the angular velocity and the direction for the rotation axis. Our method can compute a smooth rotational motion.

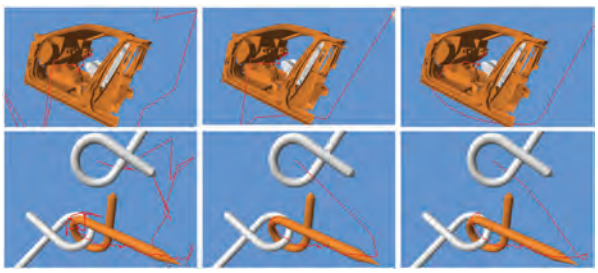

**Fig. 7.** Rigid robot benchmarks: car seat (top) and alpha puzzle (bottom). The three columns show results computed by a sample-based planner, the smooth result computed using the linear shortcut algorithm and the smooth trajectory generated by our algorithm, respectively. We show the paths in the workspace traversed by the origin of the rigid robot.

For the piano benchmark (Figure 6), we also visualize the rotation curves using quaternions. In this case, the distance to the sphere is the angle and the direction from 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, while our spline-based smoothing algorithm results in a smooth trajectory.

We show the timing results in Table 2. For each benchmark, we generated 50 trajectories using the randomized motion planner. Then, all methods performed 500 shortcut operations on different trajectories. This process was performed 10 times for each method with different random seed settings and we measured the average timing for each method for the different benchmarks. For each shortcut operation in the linear shortcut and the spline shortcut, we checked collisions using $30(N|t_b - t_a|)/T$ discrete samples, where $N, t_b, t_a$ and $T$ were 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 used our SCD algorithm and integrated it with the spline shortcut framework. The conservative advancement-based SCD algorithm is significantly faster than prior methods used, which follows from the efficiency of the conservative advancement method. The overall smoothing algorithm with SCD is comparable to (or faster than) prior methods. Notice that the result indicates that to check the collision state for one spline segment, it is faster to use one SCD operation than to perform multiple DCD checks on multiple samples along the curve. However, a single DCD operation is still faster than a single SCD operation. We also compare the SCD algorithm when using the original motion bound in our previous work (Pan et al. 2011) and when using the more compact motion bound presented in this paper. We find that the new motion bound can provide 6%-10% acceleration on our benchmarks. The differences among resulting trajectories of different algorithms are better illustrated by the video, Extension 1.

## 6.2. *Articulated models*

We now apply our spline-based smoothing method to articulated robots. Figure 8 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 to access the brackets of a bridge and perform an inspection. We used a variant of the sample-based planner to compute an initial path and apply our spline-based algorithm to smooth the trajectory. Figure 8 shows the intermediate configurations for the trajectory of the robot. Compared to the linear shortcut method, our method can compute a shorter and smoother trajectory.

We also implement our spline-based smoothing algorithm for a PR2 robot from Willow Garage for different benchmarks, as shown in Figure 9. For each benchmark, we generated 50 trajectories using the randomized motion planner in the robot operating system (ROS) and then filtered them using our B-spline smoothing algorithm or using the default trajectory filter in ROS. The default trajectory filter in ROS depends on the cubic spline, which models each segment of the linear trajectory as a cubic spline and therefore cannot provide any guarantee for the continuity of the trajectory nodes. To compare the qualities of the trajectories generated using the two different smoothing techniques, we use three criteria: (1) the length of the trajectories; (2) the maximum torque generated when executing the smoothed trajectories; (3) the integration of absolute torque when executing the smoothed trajectories. The results are compared in Table 3. From the results, we can see

**Table 2.** Timing of shortcut algorithms in seconds. The first two rows correspond to algorithms that perform discrete collision checking and tend to use a high number of discrete samples. The third row uses the original spline collision detection (SCD) in Pan et al. (2011). The last row corresponds to the improved SCD algorithm presented in Section 5.

| | Piano | Gear | Wiper | CarSeat | AlphaPuzzle | Bridge |
|---|---|---|---|---|---|---|
| Linear shortcut | 59.9 | 8.53 | 24.1 | 20.0 | 10.5 | 163 |
| Spline shortcut + TDCD | 43.5 | 9.78 | 33.1 | 51.4 | 15.8 | 185 |
| Spline shortcut + SCD (Pan et al. 2011) | 24.5 | 5.45 | 4.69 | 28.3 | 17.2 | 87.8 |
| Spline shortcut + SCD | 22.9 | 5.10 | 4.32 | 26.2 | 15.3 | 80.7 |

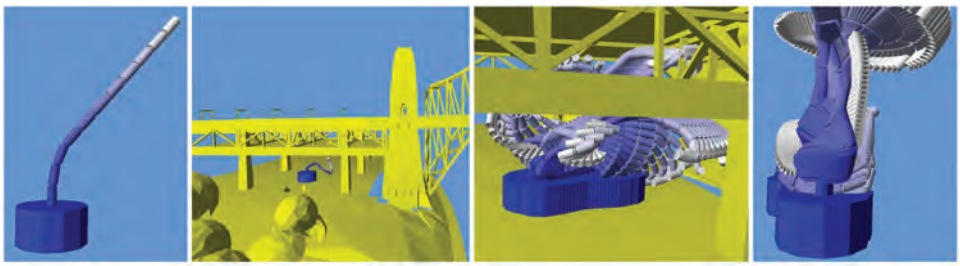

**Fig. 8.** Results for the 40-DOF hyper-redundant articulated model. The left two sub-figures show the robot and the bridge inspection scenario. In the right two sub-figures, we show the results computed by a sample-based planner and the smooth result computed by our algorithm.

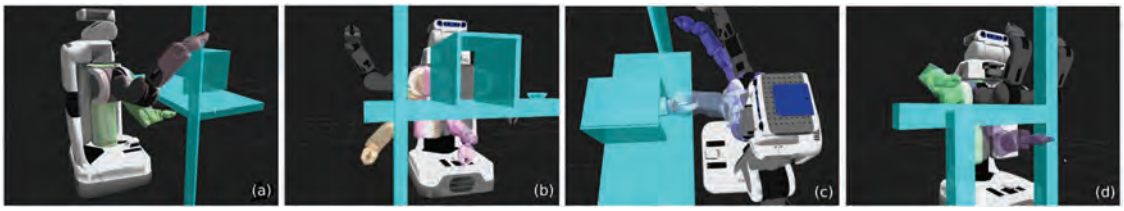


**Fig. 9.** PR2 planning benchmarks: robot arms with different colors show the initial and goal configurations. The first three benchmarks correspond to the same environment, but the robot's arm is in different places: (a) the arm moves from under the desk (initial) to over the desk (final); (b) the arm moves from under the desk to another position under the desk and (c) the arm moves from inside the box to outside the box. In the rightmost benchmark, the robot tries to move the arm from under a shelf to above it.

that the smoothed trajectories given by the B-spline smoothing algorithm can be a bit longer than the smoothing results provided by ROS's default filter. While the maximum torque is about the same, the overall torque along the trajectory is much lower with our new method.

### 6.3. *Time of contact*

Compared to performing discrete collision detection on multiple samples in the trajectory, the spline continuous collision detection provides a more accurate time of contact (TOC), which is useful for many applications, like planning and grasping. We compare the TOC convergence speeds of SCD and TDCD, as shown in Figure 10. To reduce the error under 1%, TDCD requires about 100 samples. To obtain a precision with comparable SCD requires more than 100,000 samples, which will be much slower than SCD.

## 7. Analysis and comparison

There is extensive work on path smoothing in robotics, control and related areas, as highlighted in Section 1. We use the cubic B-spline method for trajectory computation as it provides sufficient degrees of freedom to compute almost $\mathcal{C}^2$ trajectories and handle cluttered environments. Furthermore, we are able to perform fast and reliable continuous collision checking. None of the prior methods provides all these features or capabilities. Most of the prior shortcut algorithms use linear shortcuts or parabolic curves (Hauser and Ng-Thow-Hing 2010; Yamane et al. 2004), but cannot provide high-order smoothness. 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 (Yang and Sukkarieh 2010). This approach uses Bézier curves to construct $\mathcal{G}^2$ smooth, and the formulation appears to be limited to a point robot navigating in a 2D or 3D workspace.

**Table 3.** Comparison of the qualities of linear trajectories generated by the randomized motion planner, smoothed trajectories generated using the default option in ROS and smoothed trajectories generated using our B-spline smoothing algorithm described in Section 4. While the maximum torque is about the same, we observe considerable improvements in terms of the overall torque with our method.

| | Linear trajectory (randomized planner) | | | Cubic spline trajectory (ROS) | | | Cubic B-spline trajectory (Section 4) | | |
|---|---|---|---|---|---|---|---|---|---|
| | Length | Max. torque | Overall torque | Length | Max. torque | Overall torque | Length | Max. torque | Overall torque |
| PR2 benchmark (a) | 7.18 | 41.32 | 2944 | 5.99 | 42.9 | 499 | 6.08 | 42.6 | 177.6 |
| PR2 benchmark (b) | 5.70 | 37.53 | 2271 | 4.46 | 37.1 | 332 | 4.46 | 36.7 | 130.6 |
| PR2 benchmark (c) | 8.79 | 47.65 | 4755 | 6.89 | 47.7 | 845 | 7.04 | 47.6 | 291.8 |
| PR2 benchmark (d) | 8.55 | 47.13 | 3707 | 6.77 | 47.3 | 626 | 6.94 | 47.2 | 230.6 |

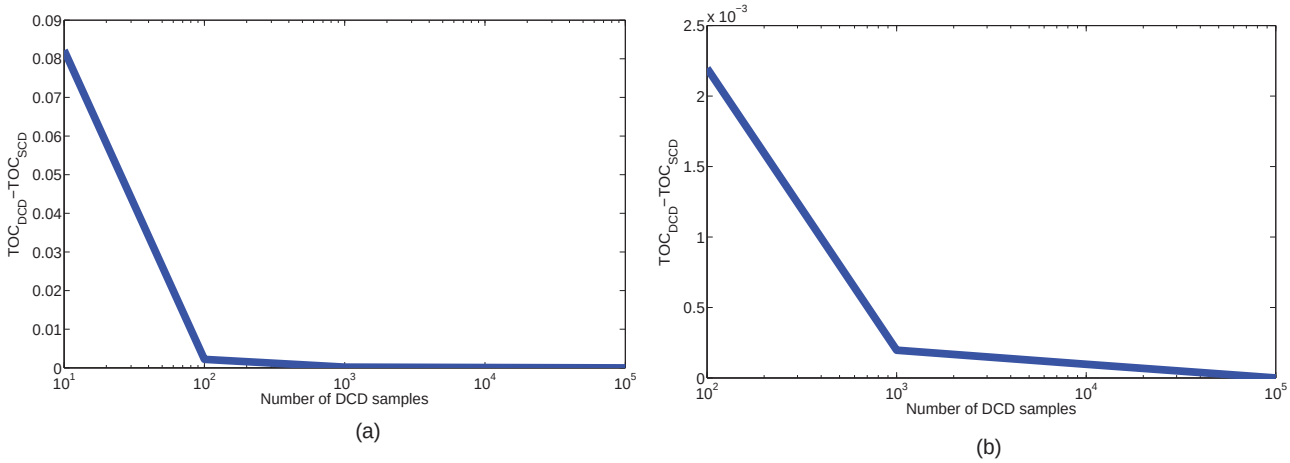


**Fig. 10.** Comparison of TOC convergence for SCD and TDCD with the piano benchmark. The *x*-axis is the number of DCD samples along the curve and the *y*-axis shows the difference between TOCs obtained by TDCD and SCD. We sampled five points along the *x* axis: 10, 100, and 1000, 10, 000 and 100, 000. (b) shows the detail of (a) when the difference is small. TDCD needs more than 100,000 samples to reach an accuracy similar to that of SCD.

There are many optimization-based methods, such as CHOMP (Ratliff et al. 2009), which have many useful properties and work well on robotic systems. Some of these methods do not require 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.

## 8. Conclusions and future work

We present a trajectory smoothing algorithm using cubic B-splines. Our formulation can compute almost $C^2$ 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 (Ratliff et al. 2009), 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. Moreover, we can also design fast and reliable collision detection algorithms, similar to SCD, for

other trajectory formulations or optimization-based smoothing algorithms. Finally, one main limitation of our method is that it cannot achieve global optimality. An interesting topic is to combine our method with critical cycle-based methods (e.g. Marble and Bekris (2011)) in order to obtain provable bounds on (asymptotically) sub-optimality.

## Funding

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. Liangjun Zhang was supported in part by the NSF Computing Innovation Fellowship (grant number 0937060) to the Computing Research Association.

## References

Barr AH, Currin B, Gabriel S and Hughes JF (1992) Smooth interpolation of orientations with angular velocity constraints using quaternions. In: *Proceedings of international conference on computer graphics and interactive techniques*, pp. 313–320.

Barsky BA and DeRose TD (1989) Geometric continuity of parametric curves: Three equivalent characterizations. *IEEE Computer Graphics and Applications* 9(6): 60–68.

Brock O and Khatib O (2002) Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research* 18(6): 1031–1052.

Chang H (1995) Motion planning in virtual prototyping: Practical considerations. In: *Proceedings of IEEE international symposium on assembly and task planning*, pp. 427–428.

DeRose TD and Barsky BA (1988) Geometric continuity, shape parameters, and geometric constructions for Catmull-Rom splines. *ACM Transactions on Graphics*, 1–41.

Fang Y, Hsieh C, Kim M, Chang J and Woo T (1998) Real time motion fairing with unit quaternions. *Computer-Aided Design* 30(3): 191–198.

Farin G (1993) *Curves and Surfaces for Computer Aided Geometric Design (3rd ed.): A Practical Guide*. Academic Press Professional, Inc.: San Diego, CA, USA.

Froissart C and Mechler P (1993) On-line polynomial path planning in Cartesian space for robot manipulators. *Robotica* 11(03): 245–251.

Geraerts R and Overmars MH (2007) Creating high-quality paths for motion planning. *International Journal of Robotics Research* 26(8) 845–863.

Govindaraju NK, Knott D, Jain N, Kabul I, Tamstorf R, Gayle R, Lin MC and Manocha D (2005) Interactive collision detection between deformable models using chromatic decomposition. *ACM Transactions on Graphics* 24: 991–999.

Hauser K and Ng-Thow-Hing V (2010) Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In: *Proceedings of international conference on robotics and automation*.

Jaillet L and Simeon T (2008) Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *International Journal of Robotics Research* 27(11–12): 1175–1188.

Kallmann M, Aubel A, Abaci T and Thalmann D (2003) Planning collision-free reaching motions for interactive object manipulation and grasping. *Computer Graphics Forum* 22(3): 313–322.

Kanehara M, Kagami S, Kuffner J, Thompson S and Mizoguhi H (2007) Path shortening and smoothing of grid-based path planning with consideration of obstacles. In: *Proceedings of IEEE international conference on systems, man and cybernetics*, pp. 991–996.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7): 846–894.

Kavraki L, Svestka P, Latombe JC and Overmars M (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

Kim M-J, Kim M-S and Shin SY (1995) A general construction scheme for unit quaternion curves with simple high order derivatives. In: *Proceedings of international conference on computer graphics and interactive techniques*, pp. 369–376.

Krebs HI, Hogan N, Aisen ML and Volpe BT (1998) Robot-aided neurorehabilitation. *IEEE Transactions on Rehabilitation Engineering* 6(1): 75–87.

Kuffner J and LaValle S (2000) RRT-connect: An efficient approach to single-query path planning. In: *Proceedings of international conference on robotics and automation*.

LaValle SM (2006) *Planning Algorithms*. Cambridge University Press.

Lee S-H, Kim J, Park F, Kim M and Bobrow JE (2005) Newton-type algorithms for dynamics-based robot movement optimization. *IEEE Transactions on Robotics* 21(4): 657–667.

Lengagne S, Kheddar A and Yoshida E (2011) Generation of optimal dynamic multi-contact motions: Application to humanoid robots. *IEEE Transactions on Robotics*. Submitted.

Marble JD and Bekris KE (2011) Asymptotically near-optimal is good enough for motion planning. In: *Proceedings of international symposium on robotics research*.

Murray R, Li Z and Sastry S (1994) *A Mathematical Introduction to Robotic Manipulation*. CRC Press.

Najfeld I and Havel TF (1995) Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics* 16: 321–375.

Nielson GM (2004) $\nu$-quaternion splines for the smooth interpolation of orientations. *IEEE Transactions on Visualization and Computer Graphics* 10(2): 224–229.

Nieuwenhuisen D and Overmars MH (2004) Useful cycles in probabilistic roadmap graphs. In: *Proceedings of international conference on robotics and automation*, pp. 446–452.

Pan J, Chitta S and Manocha D (2012) FCL: A general purpose library for collision and proximity queries. In: *Proceedings of international conference on robotics and automation*, to appear.

Pan J, Zhang L and Manocha D (2010) Retraction-based RRT planner for articulated models. In: *Proceedings of international conference on robotics and automation*, pp. 2529–2536.

Pan J, Zhang L and Manocha D (2011) Collision-free and curvature-continuous path smoothing in cluttered environments. In: *Robotics: Science and Systems*.

Pettre J, Simeon T and Laumond J (2002) Planning human walk in virtual environments. In: *Proceedings of international conference on intelligent robots and systems*, vol. 3, pp. 3048–3053.

Powell A and Rossignac J (2008) Screwbender: Smoothing piecewise helical motions. *IEEE Computer Graphics and Applications* 28(1): 56–63.

Ratliff N, Zucker M, Bagnell JAD and Srinivasa S (2009) CHOMP: Gradient optimization techniques for efficient motion planning. In: *Proceedings of international conference on robotics and automation*, pp. 489–494.

Redon S, Kim YJ, Lin MC and Manocha D (2004) Fast continuous collision detection for articulated models. In: *Proceedings of ACM symposium on solid modeling and applications*, pp. 145–156.

Shoemake K (1985) Animating rotation with quaternion curves. In: *Proceedings of international conference on computer graphics and interactive techniques*, pp. 245–254.

Tang M, Kim Y and Manocha D (2011) CCQ: Efficient local planning using connection collision query. In: *Algorithmic Foundations of Robotics IX*, vol. 68 of *Springer Tracts in Advanced Robotics*, pp. 229–247.

Toussaint M, Gienger M and Goerick C (2007) Optimization of sequential attractor-based movement for compact behavior generation. In: *Proceedings of international conference on humanoid robots*, pp. 122–129.

Yamane K, Kuffner JJ and Hodgins JK (2004) Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics* 23(3): 532–539.

Yang K and Sukkarieh S (2010) An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics* 26(3): 561–568.

Zhang L, Huang X, Kim YJ and Manocha D (2008) D-Plan: Efficient collision-free path computation for part removal and disassembly. *Journal of Computer-Aided Design and Applications* 5(6): 774–786.

Zhang X, Lee M and Kim YJ (2006) Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer* 22(9): 749–760.

Zhang X, Redon S, Lee M and Kim YJ (2007) Continuous collision detection for articulated models using Taylor models and temporal culling. *ACM Transactions on Graphics* 26(3): 15:1–15:10.

## A. Index to multimedia extensions

The multimedia extensions to this article are at: http://www.ijrr.org.

**Table of Multimedia Extensions**

| Extension | Type | Description |
|---|---|---|
| 1 | Video | Demo for rigid robots in Section 6.1 |

## B. Improved motion bound for linear and screw motion

Here we use our new definition in equation (17) to give tighter motion bounds for linear motion and screw motion.

### *B.1. Motion bound for linear motion*

For linear motion, we assume the object rotates around a given point $\mathbf{o}$ fixed along its local frame at a constant angular velocity $\omega$, while the local frame originated at $\mathbf{o}$ translates at a constant velocity $\mathbf{v}$. As a result, for a given point $\mathbf{p}$ on the object, its velocity is $\dot{\mathbf{p}}(t) = \mathbf{v} + \omega \times \mathbf{r}(t)$ where $\mathbf{r}(t) = \mathbf{p}(t) - \mathbf{o}(t)$ is the curve of the point's local coordinate. We have $\mathbf{r}(t) = \mathbf{R}(t)\,\mathbf{r}(0) = e^{\hat{\omega}t}\mathbf{R}_0\mathbf{r}$ where $R_0$ is the object's initial orientation and $\mathbf{r}$ is the point's coordinate in its local frame.

Now suppose the projection direction is $\mathbf{n}$ and $\{\mathbf{p}_i\}$ are points belonging to the object. According to equation (17), the motion bound can be computed as

$$\begin{aligned}\mu &= \max_{\tau\in[0,1]} \max_i \int_0^\tau (\dot{\mathbf{p}}_i(t)\cdot\mathbf{n})\,dt \\ &= \max_{\tau\in[0,1]} \max_i \int_0^\tau ((\mathbf{v}+\omega\times\mathbf{r}_i(t))\cdot\mathbf{n})\,dt \\ &\le \max_{\tau\in[0,1]} \left((\mathbf{v}\cdot\mathbf{n})\tau + \int_0^\tau \max_i \left|(\omega\times\mathbf{r}_i(t))\cdot\mathbf{n}\right| dt\right)\end{aligned} \tag{22}$$

Recall $\mathbf{R}(t) = e^{\hat{\omega}t}\mathbf{R}_0$, then we have

$$\begin{aligned}\left|(\omega\times\mathbf{r}_i(t))\cdot\mathbf{n}\right| &= \left|\left(\omega\times e^{\hat{\omega}t}\left(\mathbf{r}_i(0)\times\frac{\omega}{\|\omega\|}\right)\right)\cdot\mathbf{n}\right| \\ &= \left|(\mathbf{R}(t)(\omega\times\mathbf{n}))\cdot\left(\mathbf{r}_i(0)\times\frac{\omega}{\|\omega\|}\right)\right| \\ &= \|\omega\times\mathbf{n}\|\left\|\frac{\omega}{\|\omega\|}\times\mathbf{R}_0\mathbf{r}_i\right\|\end{aligned} \tag{23}$$

The final result for the motion bound is

$$\mu \le \max\left(0, \mathbf{v}\cdot\mathbf{n} + \|\omega\times\mathbf{n}\| \max_i \left\|\frac{\omega}{\|\omega\|}\times\mathbf{R}_0\mathbf{r}_i\right\|\right) \tag{24}$$

Suppose $r$, $\mathbf{c}_k$ are parameters of SSV $\alpha$ and let

$$\mathbf{c}_i^\perp = \mathbf{R}_0(\mathbf{c}_i - \mathbf{o}) \times \frac{\omega}{\|\omega\|}$$

then the motion bound for the SSV is

$$\begin{aligned}\mu_\alpha &\le \max_{\tau\in[0,1]} \left((\mathbf{v}\cdot\mathbf{n})\tau + \int_0^\tau \|\mathbf{n}\times\omega\|(r + \max_i \|\mathbf{c}_i^\perp\|)\,dt\right) \\ &= \max_{\tau\in[0,1]} \left((\mathbf{v}\cdot\mathbf{n}) + \|\mathbf{n}\times\omega\|(r + \max_i \|\mathbf{c}_i^\perp\|)\right)\tau \\ &= \max\left(0, \mathbf{v}\cdot\mathbf{n} + \|\mathbf{n}\times\omega\|(r + \max_i \|\mathbf{c}_i^\perp\|)\right)\end{aligned} \tag{25}$$

### B.2. Motion bound for screw motion

The screw motion is of the form $\mathbf{p}(t) = \mathbf{a} + e^{\hat{\omega}t}(\mathbf{p}(0)-\mathbf{a})+\mathbf{v}t$ where $\mathbf{p}(0) = \mathbf{R}_0\mathbf{p} + \mathbf{T}_0$ is the initial position and the translation $\mathbf{v} = v\omega/\|\omega\|$ is along the rotation axis. $\mathbf{a}$ is the reference point of the screw motion. Then the velocity for the screw motion is

$$\begin{aligned}\dot{\mathbf{p}}(t) &= \omega \times \left(e^{\hat{\omega}t}(\mathbf{p}(0)-\mathbf{a})\right) + \mathbf{v} \\ &= \omega \times \left(e^{\hat{\omega}t}\left((\mathbf{p}(0)-\mathbf{a}) \times \frac{\omega}{\|\omega\|}\right)\right) + \mathbf{v}\end{aligned}$$

where

$$(\mathbf{p}(0)-\mathbf{a}) \times \frac{\omega}{\|\omega\|}$$

projects $\mathbf{p}(0)-\mathbf{a}$ onto the plane perpendicular to $\omega$.

Now suppose the projection direction is $\mathbf{n}$ and $\{\mathbf{p}_i\}$ are points belonging to the object. According to equation (17), the motion bound can be computed as

$$\begin{aligned}\mu &= \max_{\tau\in[0,1]} \max_i \int_0^\tau (\dot{\mathbf{p}}_i(t) \cdot \mathbf{n})\, dt \\ &= \max_{\tau\in[0,1]} \max_i \int_0^\tau \left(\omega \times \left(e^{\hat{\omega}t}\left((\mathbf{p}_i(0)-\mathbf{a}) \times \frac{\omega}{\|\omega\|}\right)\right) + \mathbf{v}\right) \cdot \mathbf{n}\, dt \\ &\le \max_{\tau\in[0,1]} \left((\mathbf{v} \cdot \mathbf{n})\tau + \int_0^\tau \max_i \left|\left(\omega \times \left(e^{\hat{\omega}t}\left((\mathbf{p}_i(0)-\mathbf{a}) \times \frac{\omega}{\|\omega\|}\right)\right)\right) \cdot \mathbf{n}\right| dt\right)\end{aligned} \tag{26}$$

For the item within the integration, we have

$$\begin{aligned}&\left|\left(\omega \times \left(e^{\hat{\omega}t}\left((\mathbf{p}_i(0)-\mathbf{a}) \times \frac{\omega}{\|\omega\|}\right)\right)\right) \cdot \mathbf{n}\right| \\ &= \|e^{\hat{\omega}t}(\omega \times \mathbf{n})\| \left\|(\mathbf{p}_i(0)-\mathbf{a}) \times \frac{\omega}{\|\omega\|}\right\| \\ &= \|\omega \times \mathbf{n}\| \left\|(\mathbf{R}_0\mathbf{p}_i + \mathbf{T}_0 - \mathbf{a}) \times \frac{\omega}{\|\omega\|}\right\|\end{aligned} \tag{27}$$

and the final result for the motion bound is

$$\mu \le \max\left(0, \mathbf{v} \cdot \mathbf{n} + \|\omega \times \mathbf{n}\| \max_i \left\|(\mathbf{R}_0\mathbf{p}_i + \mathbf{T}_0 - \mathbf{a}) \times \frac{\omega}{\|\omega\|}\right\|\right) \tag{28}$$

Similarly, the motion bound for SSV $\alpha$ is

$$\mu_\alpha \le \max\left(0, \mathbf{v} \cdot \mathbf{n} + \|\omega \times \mathbf{n}\| \cdot \left(\frac{\|(\mathbf{T}_0 - \mathbf{a}) \times \omega\|}{\|\omega\|} + r + \max_i \frac{\|\mathbf{R}_0\mathbf{c}_i \times \omega\|}{\|\omega\|}\right)\right) \tag{29}$$

## C. Improved motion bound for articulated body

We assume the articulated model is as described in Section 3.2.2 and ${}^j\mathbf{L}_i$ denotes the vector from the origin of frame $\{i\}$ to that of frame $\{j\}$. Suppose the motion of the $j$-th link is described by two parts: the translational velocity ${}^{j-1}\mathbf{v}_j$ and the rotational velocity ${}^{j-1}\omega_j$, both with respect to its parent link $j-1$. The motion bound for such a general motion is given in equation (33), which can be further improved given a specific type of motion.

Now we use linear motion as an example. For linear motion, we have ${}^{i-1}\mathbf{v}_i$ and ${}^{i-1}\omega_i$ are constants. For the $i$-th joint, we have

$$\max_{\mathbf{p}\in\mathcal{A}_i} \|{}^{i-1}\mathbf{L}_i(t)\| = \max_{\mathbf{p}\in\mathcal{A}_i} \|{}^{i-1}\mathbf{r}_i\| \equiv \|{}^{i-1}\mathbf{L}_i(t)\|_\mu \tag{30}$$

and for the $j$-th joint ($j < i$), we have

$$\begin{aligned}\|{}^{j-1}\mathbf{L}_j(t)\| &= \|{}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{v}_j t\| \\ &\le \max({}^{j-1}\mathbf{L}_j(0), {}^{j-1}\mathbf{L}_j(1)) \equiv \|{}^{j-1}\mathbf{L}_j(t)\|_\mu\end{aligned} \tag{31}$$

Therefore the motion bound for linear motion is

$$\begin{aligned}\mu \le {}^0\mathbf{v}_1 \cdot \mathbf{n} + \|\mathbf{n} \times {}^0\omega_1\| \sum_{j=1}^{i} \|{}^{j-1}\mathbf{L}_j\|_\mu \\ + \sum_{j=2}^{i} \left(\|{}^{j-1}\mathbf{v}_j\| + \|{}^{j-1}\omega_j\| \sum_{k=j}^{i} \|{}^{k-1}\mathbf{L}_k\|_\mu\right)\end{aligned} \tag{32}$$

## D. Motion bound for spline motion

Here we prove the theorems presented in Section 5.3.

$$
\begin{aligned}
\mu &= \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau {}^0\mathbf{v}_i \cdot \mathbf{n}dt \\
&= \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau \Big(\sum_{j=1}^{i}\big({}^0_{j-1}\mathbf{R}^{j-1}\mathbf{v}_j+(\sum_{k=1}^{j}{}^0_{j-1}\mathbf{R}^{k-1}\omega_k)\times{}^0_{j-1}\mathbf{R}^{j-1}\mathbf{L}_j\big)\Big)\cdot\mathbf{n}dt \\
&= \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau \Big({}^0\mathbf{v}_1+{}^0\omega_1\times{}^0\mathbf{L}_1+\sum_{j=2}^{i}\big({}^0_{j-1}\mathbf{R}^{j-1}\mathbf{v}_j+(\sum_{k=1}^{j}{}^0_{j-1}\mathbf{R}^{k-1}\omega_k)\times{}^0_{j-1}\mathbf{R}^{j-1}\mathbf{L}_j\big)\Big)\cdot\mathbf{n}dt \\
&= \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau {}^0\mathbf{v}_1\cdot\mathbf{n}+(\mathbf{n}\times{}^0\omega_1)\cdot{}^0\mathbf{L}_1+\sum_{j=2}^{i}\Big({}^0_{j-1}\mathbf{R}^{j-1}\mathbf{v}_j\cdot\mathbf{n}+\big((\sum_{k=1}^{j}{}^0_{j-1}\mathbf{R}^{k-1}\omega_k)\times{}^0_{j-1}\mathbf{R}^{j-1}\mathbf{L}_j\big)\cdot\mathbf{n}\Big)dt \\
&\le \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau {}^0\mathbf{v}_1\cdot\mathbf{n}+(\mathbf{n}\times{}^0\omega_1)\cdot{}^0\mathbf{L}_1+\sum_{j=2}^{i}\Big(\|^{j-1}\mathbf{v}_j\|+(\mathbf{n}\times\sum_{k=1}^{j}{}^0_{j-1}\mathbf{R}^{k-1}\omega_k)\cdot{}^0_{j-1}\mathbf{R}^{j-1}\mathbf{L}_j\Big)dt \\
&\le \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau {}^0\mathbf{v}_1\cdot\mathbf{n}+(\mathbf{n}\times{}^0\omega_1)\cdot{}^0\mathbf{L}_1+\sum_{j=2}^{i}\Big(\|^{j-1}\mathbf{v}_j\|+\|\mathbf{n}\times{}^0\omega_1+\sum_{k=2}^{j}{}^0_{j-1}\mathbf{R}^{k-1}\omega_k\|\|{}^0_{j-1}\mathbf{R}^{j-1}\mathbf{L}_j\|\Big)dt \\
&\le \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau {}^0\mathbf{v}_1\cdot\mathbf{n}+(\mathbf{n}\times{}^0\omega_1)\cdot{}^0\mathbf{L}_1+\sum_{j=2}^{i}\Big(\|^{j-1}\mathbf{v}_j\|+(\|\mathbf{n}\times{}^0\omega_1\|+\|\mathbf{n}\times\sum_{k=2}^{j}{}^0_{j-1}\mathbf{R}^{k-1}\omega_k\|)\|^{j-1}\mathbf{L}_j\|\Big)dt \\
&\le \max_{\tau\in[0,1]}\max_{\mathbf{p}\in\mathcal{A}_i}\int_0^\tau {}^0\mathbf{v}_1\cdot\mathbf{n}+(\mathbf{n}\times{}^0\omega_1)\cdot{}^0\mathbf{L}_1+\sum_{j=2}^{i}\Big(\|^{j-1}\mathbf{v}_j\|+(\|\mathbf{n}\times{}^0\omega_1\|+\sum_{k=2}^{j}\|^{k-1}\omega_k\|)\|^{j-1}\mathbf{L}_j\|\Big)dt \\
&= \max_{\tau\in[0,1]}\int_0^\tau {}^0\mathbf{v}_1\cdot\mathbf{n}dt+\int_0^\tau(\mathbf{n}\times{}^0\omega_1)\cdot{}^0\mathbf{L}_1dt+\int_0^\tau\sum_{j=2}^{i}\|^{j-1}\mathbf{v}_j\|dt+\int_0^\tau\sum_{j=2}^{i-1}(\|\mathbf{n}\times{}^0\omega_1\|+\sum_{k=2}^{j}\|^{k-1}\omega_k\|)\|^{j-1}\mathbf{L}_j\|dt \\
&\quad+\int_0^\tau(\|\mathbf{n}\times{}^0\omega_1\|+\sum_{k=2}^{i}\|^{k-1}\omega_k\|)\max_{\mathbf{p}\in\mathcal{A}_i}\|^{i-1}\mathbf{L}_i\|dt
\end{aligned}
\tag{33}
$$

## *D.1. Spline motion bound for a rigid body*

The velocity for spline motion is $\dot{\mathbf{p}}(t) = \dot{\mathbf{T}}(t)+(q(t)\mathbf{r}q(t)^{-1})'$, where $q(t)=(\cos\theta(t),\mathbf{u}\sin\theta(t))$ is the rotation quaternion and $\mathbf{r}$ is the local coordinate. Then according to equation (17), the motion bound is

$$
\begin{aligned}
\mu &= \max_{\tau\in[0,1]}\max_i\int_0^\tau(\dot{\mathbf{p}}_i(t)\cdot\mathbf{n})\,dt \\
&= \max_{\tau\in[0,1]}\int_0^\tau(\dot{\mathbf{T}}(t)\cdot\mathbf{n})\,dt+\max_i\int_0^\tau \\
&\quad((q(t)\mathbf{r}_iq(t)^{-1})'\cdot\mathbf{n})\,dt
\end{aligned}
\tag{34}
$$

We first compute the second term

$$
\begin{aligned}
&\int_0^\tau((q(t)\mathbf{r}_iq(t)^{-1})'\cdot\mathbf{n})\,dt \\
&=\mathbf{n}\cdot(q(t)\mathbf{r}_iq(t)^{-1})\big|_0^\tau \\
&=\mathbf{n}\cdot(\mathbf{r}_i\cos2\theta+(\mathbf{u}\times\mathbf{r}_i)\sin2\theta+\mathbf{u}(\mathbf{u}\cdot\mathbf{r}_i) \\
&\quad(1-\cos2\theta))\big|_0^\tau \\
&\le(|\mathbf{r}_i\cdot\mathbf{n}|+\|\mathbf{r}_i\|)|\cos2\theta(\tau)-\cos2\theta(0)| \\
&\quad+\|\mathbf{r}_i\times\mathbf{n}\||\sin2\theta(\tau)-\sin2\theta(0)|
\end{aligned}
\tag{35}
$$

To further simplify the result, we need the following lemma

**Lemma 5.** *If $f(\cdot)$ is continuous on $[a,b]$, then*

$$
\left|\frac{f(b)-f(a)}{b-a}\right|\le|f'(c)|\le\max_{t\in[a,b]}|f'(t)| \tag{36}
$$

*where $c$ is one point in $[a,b]$.*

Moreover, as $\mathbf{w}=\mathbf{u}\theta$, we have $\mathbf{w}'=\mathbf{u}'\theta+\mathbf{u}\theta'$. Multiplying both sides by $\mathbf{u}$, we have $\theta'=\mathbf{w}'\cdot\mathbf{u}-\mathbf{u}'\cdot\mathbf{u}=\mathbf{w}'\cdot\mathbf{u}$ because $\mathbf{u}\cdot\mathbf{u}=1$ and thus $\mathbf{u}\cdot\mathbf{u}'=0$. Based on these properties, the second term can be further represented as

$$
\begin{aligned}
&\int_0^\tau((q(t)\mathbf{r}_iq(t)^{-1})'\cdot\mathbf{n})\,dt \\
&\le(|\mathbf{r}_i\cdot\mathbf{n}|+\|\mathbf{r}_i\|)\,\tau\max_{t\in[0,\tau]}2|\theta'\sin2\theta| \\
&\quad+\|\mathbf{r}_i\times\mathbf{n}\|\tau\max_{t\in[0,\tau]}2|\theta'\cos2\theta| \\
&\le(|\mathbf{r}_i\cdot\mathbf{n}|+\|\mathbf{r}_i\|)\,\tau\max_{t\in[0,\tau]}2|\mathbf{w}'\cdot\mathbf{u}| \\
&\quad+\|\mathbf{r}_i\times\mathbf{n}\|\tau\max_{t\in[0,\tau]}2|\mathbf{w}'\cdot\mathbf{u}| \\
&\le\big(2(|\mathbf{r}_i\cdot\mathbf{n}|+\|\mathbf{r}_i\|+\|\mathbf{r}_i\times\mathbf{n}\|)\max_{t\in[0,\tau]}\|\mathbf{w}'\|\big)\tau
\end{aligned}
\tag{37}
$$

As $\mathbf{w}(t)$ is a cubic polynomial, then $\mathbf{w}'$ is a quadratic polynomial and therefore $\max_{t\in[0,\tau]}\|\mathbf{w}'\|$ can be obtained by

solving a cubic equation, which can be computed efficiently by comparing the cubic equation's analytical solutions and the boundary values.

We can also apply $|\cos 2\theta(\tau) - \cos 2\theta(0)| \leq 2$ and $|\sin 2\theta(\tau) - \sin 2\theta(0)| \leq 2$ in equation (35) to obtain a (usually) looser bound for the rotation

$$\begin{aligned} &\int_0^\tau ((q(t)\,\mathbf{r}_i q(t)^{-1})' \cdot \mathbf{n})\,dt \\ &\leq 2(|\mathbf{r}_i \cdot \mathbf{n}| + \|\mathbf{r}_i\| + \|\mathbf{r}_i \times \mathbf{n}\|) \end{aligned} \tag{38}$$

The bound is looser because even when there is no motion, i.e. $\mathbf{w}'(t) = 0$, it will still give a positive bound.

Combining equations (37) and (38), we can obtain a tight bound for the second term as

$$\begin{aligned} &\int_0^\tau ((q(t)\,\mathbf{r}_i q(t)^{-1})' \cdot \mathbf{n})\,dt \\ &\leq 2(|\mathbf{r}_i \cdot \mathbf{n}| + \|\mathbf{r}_i\| + \|\mathbf{r}_i \times \mathbf{n}\|)\min(\tau, \max_{t\in[0,\tau]} \|\mathbf{w}'\|) \end{aligned} \tag{39}$$

To estimate a bound for the first term, we need the explicit spline form for the translational part. If we assume the spline is uniform, then

$$\begin{aligned} \mathbf{T}(t) = \mathbf{d}_0 \frac{1-3t+3t^2-t^3}{6} + \mathbf{d}_1 \frac{4-6t^2+3t^3}{6} \\ + \mathbf{d}_2 \frac{1+3t+3t^2-3t^3}{6} + \mathbf{d}_3 \frac{t^3}{6} \end{aligned}$$

where $\mathbf{d}_k$ are the control points for the current spline segment. The motion bound for the translation part is

$$\begin{aligned} &\int_0^\tau \mathbf{T}'(t) \cdot \mathbf{n} dt \\ &= \frac{1}{6}\big(\tau^3(3\mathbf{d}_1 - 3\mathbf{d}_2 + \mathbf{d}_3 - \mathbf{d}_0) \cdot \mathbf{n} \\ &\quad + 3\tau^2(\mathbf{d}_0 - 2\mathbf{d}_1 + \mathbf{d}_2)\,\mathbf{n} + 3\tau(\mathbf{d}_2 - \mathbf{d}_0) \cdot \mathbf{n}\big) \\ &= \frac{1}{6}(A\tau^3 + B\tau^2 + C\tau) \end{aligned} \tag{40}$$

Similar results can be obtained for the non-uniform spline.

By combining equations (39) and (40), we obtain the motion bound for the spline motion

$$\begin{aligned} \mu &\leq \max_{\tau\in[0,1]} \left( \frac{1}{6}(A\tau^3 + B\tau^2 + C\tau) + 2\max_i(|\mathbf{r}_i \cdot \mathbf{n}| + \|\mathbf{r}_i\| \right. \\ &\quad \left. + \|\mathbf{r}_i \times \mathbf{n}\|)\min(\tau, \max_{t\in[0,\tau]} \|\mathbf{w}'\|) \right) \\ &\leq 2\max_i(|\mathbf{r}_i \cdot \mathbf{n}| + \|\mathbf{r}_i\| + \|\mathbf{r}_i \times \mathbf{n}\|)\min(1, \max_{t\in[0,1]} \|\mathbf{w}'\|) \\ &\quad + \frac{1}{6}\max_{\tau\in[0,1]}(A\tau^3 + B\tau^2 + C\tau) \end{aligned} \tag{41}$$

The maximum value of $(A\tau^3 + B\tau^2 + C\tau)$, when $\tau \in [0,1]$ can be reached at $\tilde{\tau}$, which can be 0, 1 or the root(s) of $3A\tau^2 + 2B\tau + C = 0$. Therefore the final form for the motion bound is

$$\begin{aligned} \mu &\leq 2\max_i(|\mathbf{r}_i \cdot \mathbf{n}| + \|\mathbf{r}_i\| + \|\mathbf{r}_i \times \mathbf{n}\|)\min(1, \max_{t\in[0,1]} \|\mathbf{w}'\|) \\ &\quad + \frac{1}{6}(A\tilde{\tau}^3 + B\tilde{\tau}^2 + C\tilde{\tau}) \end{aligned} \tag{42}$$

and Theorem 1 is proved.

Each point within one SSV $\alpha$ 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 the 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 $\alpha$'s medial structure. Then for points within $\alpha$, we have

$$\begin{aligned} |\mathbf{r}_i \cdot \mathbf{n}| &= |(\mathbf{r}^i + \mathbf{k}) \cdot \mathbf{n}| \leq r + \max_k |\mathbf{c}_k \cdot \mathbf{n}| \\ |\mathbf{r}_i \times \mathbf{n}| &= |(\mathbf{r}^i + \mathbf{k}) \times \mathbf{n}| \leq r + \max_k |\mathbf{c}_k \times \mathbf{n}| \end{aligned}$$

and

$$\|\mathbf{r}_i\| = \|\mathbf{r}^i + \mathbf{k}\| \leq r + \max_k \|\mathbf{c}_k\|$$

By putting these results into equation (42), we can prove Theorem 2.

## D.2. Spline motion bound for the articulated model

For articulated models, similar to the rigid body case, we now assume ${}^{j-1}\mathbf{T}_j(t)$ and ${}^j\mathbf{w}_j(t)$ are cubic B-splines, where ${}^{j-1}\mathbf{T}_j'(t) = {}^{j-1}\mathbf{v}_j(t)$ and ${}^{j-1}\mathbf{w}_j'(t) = {}^{j-1}\omega_j(t)$. Then using equation (33), it is easy to prove Theorem 3 and Theorem 4. Notice that, as with the rigid body case, $\max_{t\in[0,1]} {}^0\mathbf{T}_1(t)$, $\max_{t\in[0,1]} \|{}^{j-1}\mathbf{T}_j'(t)\|$ and $\max_{t\in[0,1]} \|{}^{j-1}\mathbf{w}_j'(t)\|$ can all be computed by solving a quadratic or a cubic equation. However, the bound computation for $\|{}^{j-1}\mathbf{L}_j(t)\|$ $(j < i)$ is different from that of linear motion in equation (31). Here we have

$$\begin{aligned} \|{}^{j-1}\mathbf{L}_j(t)\| &= \|{}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{T}_j(t)\| \\ &\leq \max_{t\in[0,1]} \|{}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{T}_j(t)\| \end{aligned} \tag{43}$$

Of course we can directly compute $\max_{t\in[0,1]} \|{}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{T}_j(t)\|$ by solving a degree-5 polynomial equation because ${}^{j-1}\mathbf{T}_j(t)$ is a cubic spline polynomial. However, degree-5 polynomials do not have analytic solutions, so we have to use numerical methods. To avoid finding numerical roots, here we estimate an upper bound for $\max_{t\in[0,1]} \|{}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{T}_j(t)\|$ based on the convex hull property of the B-spline. Suppose $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2$ and $\mathbf{d}_3$ are the four control points of the spline segment ${}^{j-1}\mathbf{T}_j(t)$,

then ${}^{j-1}\mathbf{T}_j(t)$ is completely within tetrahedron $\Delta_{\mathbf{d}_0,\mathbf{d}_1,\mathbf{d}_2,\mathbf{d}_3}$, which is convex. Therefore we have

$$\max_{t\in[0,1]} \|{}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{T}_j(t)\| \le \max_{l\in\{0,1,2,3\}} \|{}^{j-1}\mathbf{L}_j(0) + \mathbf{d}_l\| \equiv \|{}^{j-1}\mathbf{L}_j(t)\|_\mu \tag{44}$$

Moreover, for a special case in which each joint of the articulated model has one DOF (i.e. either a prismatic joint or a revolute joint), we can provide a better bound. For a prismatic joint, we have

$${}^{j-1}\mathbf{v}_j(t) = {}^{j-1}\mathbf{l}_j\, {}^{j-1}s_j'(t) \tag{45}$$

and for a revolute joint we have

$${}^{j-1}\omega_j(t) = {}^{j-1}\mathbf{u}_j\, {}^{j-1}\theta_j'(t) \tag{46}$$

where unit vectors ${}^{j-1}\mathbf{l}_j$ and ${}^{j-1}\mathbf{u}_j$ are the translational axis and the rotational axis, respectively. As a result, we can provide a tighter bound for this type of special articulated body

$$\begin{aligned}\mu \le {} & (\mathbf{n} \times {}^0\mathbf{u}_1)\left(\|{}^0\mathbf{L}_1\|_\mu \max_{t\in[0,1]} |{}^0\theta_1'(t)| \right.\\ & \left. + \|{}^0\mathbf{L}_1\|_\mu \max_{t\in[0,1]} |{}^0\theta_1'(t)|\right)\\ & + \sum_{j=2}^{i}\left(\max_{t\in[0,1]} |{}^{j-1}s_j'(t)| + (\|\mathbf{n}\times{}^0\mathbf{u}_1\| \max_{t\in[0,1]} |{}^0\theta_1'(t)| \right.\\ & \left. + \sum_{k=2}^{j} \max_{t\in[0,1]} |{}^{k-1}\theta_k'(t)|)\,\|{}^{j-1}\mathbf{L}_j\|_\mu\right)\end{aligned} \tag{47}$$

where $\max_{t\in[0,1]} |{}^{j-1}\theta_j'(t)|$ and $\max_{t\in[0,1]} |{}^{j-1}s_j'(t)|$ can be computed by solving a cubic equation. $\|{}^{j-1}\mathbf{L}_j(t)\|_\mu$ $(j < i)$ can also be estimated based on a B-spline's convex hull property.