Constrained Motion Interpolation with Distance Constraints

Liangjun Zhang and Dinesh Manocha

Department of Computer Science, University of North Carolina at Chapel Hill
{zlj,dm}@cs.unc.edu
http://gamma.cs.unc.edu/RRRT/CMI

Abstract: We present a novel constraint-based motion interpolation algorithm to improve the performance of local planners in sample-based motion planning. Given two free-space configurations of a robot, our algorithm computes a one-dimensional trajectory subject to distance constraints between the closest features of the robot and the obstacles. We derive simple and closed form solutions to compute a path that guarantees no collisions between these closest features. The resulting local planner is fast and can improve the performance of sample-based planners with no changes to the underlying sampling strategy. In practice, we observe speedups on benchmarks for rigid robots with narrow passages.

1 Introduction

The problem of computing an interpolating motion between two configurations arises in different applications including robot motion planning, kinematics, dynamic simulation, CAD/CAM, and keyframe animation. Given the initial and final configurations, the goal is to compute a one-dimensional function that interpolates the two configurations. Moreover, some applications impose constraints on the resulting trajectory such as smoothness or limits on its derivatives.

In this paper, we address the problem of computing a collision-free interpolating motion between two free-space configurations. The goal is to compute a trajectory that is less likely to intersect with any obstacles in the configuration space (C-space). The main motivation is the local planning step in sample-based planners, which attempts to connect two nearby free-space samples with a collision-free path. Typically, the local planners operate in two steps: computation of an interpolating path and checking that path for collisions with the obstacles. In practice, one of the most time-consuming steps in sample-based planners is checking whether the motion produced by the local planner is collision-free or not [3, 7, 15, 17, 18].

The performance of sample-based planners may degrade when the free space has narrow passages. The narrow passages are defined as small regions of free space whose removal or perturbation can change the connectivity of the free space. Most

of the prior work in improving the performance of planners has focused on increasing the probability of sampling in these regions [2, 6, 9, 22]. The underlying philosophy of these sampling strategies is that the planner would eventually generate a sufficient number of samples in and around the narrow passages, which can be easily connected using simple local planning algorithms (e.g. linear interpolation). However, the problem of generating sufficient number of samples in narrow passages is non-trivial. Moreover, the narrow passages may have poor visibility properties [9], which can result in high failure rates for the interpolation paths computed by the local planners. Some powerful local planners have been proposed to connect the samples [1, 7, 10], but they can either result in more expensive collision checking or do not take into account the position of the obstacles in the environment.

Main Results: We present a novel motion interpolation algorithm to improve the performance of local planners for sample-based motion planning. Given two free-space configurations, $\mathbf{q}_0$ and $\mathbf{q}_1$, our algorithm maintains distance constraints between the closest feature pairs at these configurations. We derive simple and closed form solutions to guarantee that the sign of the distance between each feature pair does not vary along the trajectory. As a result, there are no collisions between the closest features along that trajectory. Since a local planning algorithm is typically invoked between nearby configurations, the closest features are the most likely candidates for collisions. As a result, our motion interpolation algorithm is more likely to result in a collision-free path as compared to other interpolation schemes that ignore the position of the obstacles in the environment. Our local planner can be combined with sample-based planners with no changes to the sampling strategy or techniques used to compute nearest neighbors. We have combined our algorithm with a retraction-based planner that generates more samples near the contact space and narrow passages. We observe performance gains of the overall planner on benchmarks with narrow passages. Overall, our constrained interpolating motion offers the following benefits:

- **Simplicity:** We present simple and closed form solutions to compute the path based on the closest features.
- **Efficiency:** The main additional overhead of our interpolation algorithm is the computation of closest features at the initial and final configurations, which only takes up to a few milli-seconds.
- **Generality:** Our local planning algorithm is general to all rigid robots that can be represented as polygonal soups. It can also maintain constraints for compliant motion planning.
- **Improved performance:** Our local planning algorithm explicitly takes into account the position of the obstacles in the environment and is effective in connecting nodes in or near the narrow passages.

Organization: The rest of the paper is organized in the following manner. We survey related work on motion interpolation and local planning algorithms in Section 2. Section 3 introduces the notation and gives an overview of our approach. We present the constrained interpolation algorithm for a single distance constraint in Section 4 and extend it to handle multiple constraints in Section 5. We highlight the performance of our local planning algorithm on challenging benchmarks in Section 6. We discuss some properties and extensions of our interpolation scheme in Section 7.

2 Previous Work

In this section, we give a brief overview of previous work on motion interpolation and local planning algorithms.

2.1 Motion Interpolation

Many formulations have been proposed to interpolate the motion between two configurations. The simplest algorithms use straight-line linear interpolation or spherical linear interpolation [12]. Other algorithms tend to compute the minimal-length curve based on appropriate distance metrics, or maintain smoothness constraints [5,14,20]. However, these algorithms may not take into account the position of the obstacles in the environment. A related problem is to generate constrained motion that maintains a contact with the given surface or avoid obstacles [11]. Other more general variational-based interpolation schemes [8, 19] have also been proposed. However, the formulation and path computation using these approaches can be expensive as compared to other interpolation techniques.

2.2 Local Planning

The local planning algorithms generate an interpolating motion and check the resulting path for collision with the obstacles. The simplest local planners perform discrete collision detection along a finite number of samples on the continuous path. Discrete collision checking is easy and can be efficiently performed using bounding volume hierarchies. However, there is no guarantee that the portions on the path that are not sampled are collision-free. In order to overcome this issue, some continuous collision detection algorithms based on distance bounds or adaptive bisection have been proposed [15, 18, 23].

Many researchers have analyzed the performance of local planning algorithms and distance metrics, and suggested techniques to improve the performance of the overall motion planner [1,7,12]. Improved algorithms for local planning have been designed by combining them with potential field approaches [7] or path optimization [10]. However, these improved local planning algorithms are expensive and can have additional overhead in terms of collision checking.

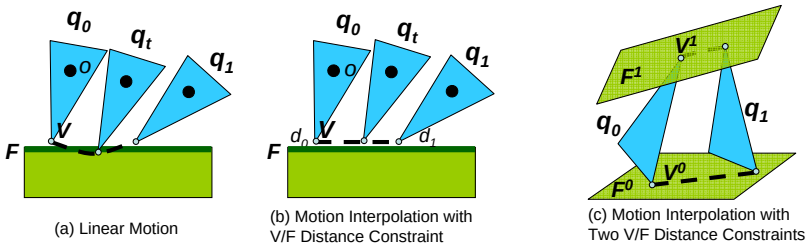


Fig. 1. *Motion Interpolation between two configurations* $\mathbf{q}_0$ *and* $\mathbf{q}_1$*: (a) There is collision at the intermediate configuration* $\mathbf{q}_t$ *if we use a linear interpolation; (b) Using our constrained interpolation algorithm, we obtain a collision-free trajectory for this case. (c) We take into account multiple closest feature pairs (*$(\mathbf{V}^0,\mathbf{F}^0)$ *and* $(\mathbf{V}^1,\mathbf{F}^1)$ *in this case) at the two configurations, and guarantee no collisions among these feature pairs along the trajectory.*

3 Overview

In this section, we give an overview of our constrained motion interpolation scheme. We further present our formulation for specifying distance constraints algebraically. The interpolation scheme discussed here is for rigid robots. Later in Section 7.4, we present a simple heuristic to extend it to articulated models.

3.1 Notation

We assume that the rigid robot and the obstacles are polyhedral models. We denote the features of vertices, edges, and faces on the boundary of the robot or obstacles as $\mathbf{V}$, $\mathbf{E}$, and $\mathbf{F}$, respectively. We use superscripts to enumerate the features, e.g. $\mathbf{V}^0$, $\mathbf{V}^1$, $\mathbf{V}^2$, and so forth. For the moving robot, we use subscripts to denote the position of its feature at time t, e.g. a specific vertex $\mathbf{V}$ at $t=0$, $t=1$ or t is denoted as $\mathbf{V}_0$, $\mathbf{V}_1$, or $\mathbf{V}_t$, respectively. A configuration $\mathbf{q}$ for a rigid robot is represented by using a vector $\mathbf{T}$ for translation and a rotation matrix $\mathbf{R}$, i.e. $\mathbf{q}=(\mathbf{R},\mathbf{T})$. In order to interpolate two given configurations $\mathbf{q}_0$ and $\mathbf{q}_1$, without loss of generality, we assume $\mathbf{q}_0=(\mathbf{I},[0,0,0]^t)$ and the rotational component of $\mathbf{q}_1$ is the rotation matrix about z-axis by θ.

The motion interpolation problem is to compute a one-dimensional trajectory - $\{\mathbf{M}_t=(\mathbf{R}_t,\mathbf{T}_t)|t\in[0,1]\}$ with $\mathbf{M}_0=\mathbf{q}_0$ and $\mathbf{M}_1=\mathbf{q}_1$. For example, undergoing a linear interpolating motion, the robot has constant angular and translation velocities with

$$\mathbf{R}_t=\begin{bmatrix} cos(\theta t) & -sin(\theta t) & 0 \\ sin(\theta t) & cos(\theta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

and $\mathbf{T}_t$ is $(1-t)\mathbf{T}_0+t\mathbf{T}_1$.

When the robot moves along an interpolating motion, the position of any point $\mathbf{x}$ on the model at time t can be computed:

$$\mathbf{x}_t=\mathbf{R}_t\mathbf{x}_0+\mathbf{T}_t, \tag{2}$$

where $\mathbf{x}_0$ is the position of $\mathbf{x}$ at time $t=0$.

3.2 Constrained Motion Interpolation

Most prior motion interpolation schemes do not take into account the position of the robot and the obstacles in the environment. The resulting interpolating motion

may not be collision-free when the robot is near an obstacle. Formally speaking, this corresponds to the situation when the two interpolated configurations $\mathbf{q}_0$ and $\mathbf{q}_1$ are close to the *contact space*, a subset of configurations in C-space at which the robot only touches one or more obstacles without any penetration. Fig. 1(a) shows one example where the robot is near the obstacle. A simple linear interpolation scheme will generate a motion where the robot's rotational center **o** undergoes a straight line motion. However, due to the affect of the rotation, the other points on the robot (e.g. **V**) follow a non-linear trajectory. Since the robot is near the obstacle in this example, the vertex **V** collides with the face **F** on the obstacle when the robot moves. The collision may happen even when the robot undergoes a small rotation only if the vertex **V** is far from **o**.

Our goal is to compute an interpolating motion that is less likely to result in a collision between the robot and the obstacles. In order to generate such a trajectory between $\mathbf{q}_0$ and $\mathbf{q}_1$, we compute the *closest feature pairs* between the robot at both configurations and the obstacles. Every feature corresponds to a vertex (**V**), an edge (**E**), or a face (**F**) on the boundary. Moreover, we impose constraints so that there is no collision among these closest features along the interpolated trajectory. This is also highlighted in Fig. 1(b), where the constrained motion ensures that the closest feature pair (**V**,**F**) does not collide when the robot moves. Intuitively, in many cases the closest features are the most likely candidates for a collision between the robot and the obstacles. By ensuring that there is no collision amongst these closest features, our interpolation algorithm is more likely to compute a collision-free trajectory for the entire robot. In this manner, our interpolation scheme takes into account the position of the robot and the obstacles in the environment. Moreover, we show that there is very little extra overhead of using our interpolation algorithm over prior methods.

3.3 Distance Constraints

The main issue to formulate the constrained interpolation is the representation of non-collision constraints among the closest features. We use a sufficient condition given by the following lemma:

Lemma 1. *Let $\{P^i\}$ the set of the closest feature pairs between the robot at the configuration $\mathbf{q}_0$ and the configuration $\mathbf{q}_1$, and the obstacles. If the sign of the distance function d_t for each feature pair (formally defined in Section 3.4) dose not change when the robot moves along this trajectory, then there is no collision in each feature pair P^i.*

We highlight the use of distance constraints based on an example. Consider the case in Fig. 1(b), where there is only one closest feature pair (**V**, **F**) between the robot and the obstacles. Let $\mathbf{V}_0$ ($\mathbf{V}_1$) as the position of the vertex **V** at the configuration $\mathbf{q}_0$($\mathbf{q}_1$). The signed distance between $\mathbf{V}_0$ ($\mathbf{V}_1$) and the plane containing the face **F** is d_0 (d_1), respectively. Furthermore, we assume the signed distances d_0 and d_1 have the same sign. In this case, a constrained motion can be computed by imposing the distance d_t is a linear interplant of d_0 and d_1 with t. This ensures the sign of distance function does change along the motion. In this way, our constrained motion guarantees that

there is no collision between these features along the trajectory (as per Lemma 1). Given d_0 and d_1 with the same signs, in order to guarantee that the sign of the whole distance function does not change, a simple but sufficient way is to perform a linear interpolation on the signed distances. Quadratic or more complex interpolation functions may be chosen.

Multiple Distance Constraints: In many cases, taking multiple distance constraints into account increases the probability of computing a collision-free path for the local planner. Fig. 1(c) shows such an example. If we can guarantee that the signs of the distance functions between feature pairs $(\mathbf{V}^0, \mathbf{F}^0)$ and $(\mathbf{V}^1, \mathbf{F}^1)$ do not change along the trajectory, then there is no collision between these feature pairs throughout the motion.

Solving the Constrained System: Under multiple constraints, the motion interpolation is formulated as a constrained system:

$$\mathbf{C}(\mathbf{R}_t, \mathbf{T}_t) = 0, \tag{3}$$

where $\mathbf{C}$ denotes the collection of distance constraints.

The system is non-linear due to the rotational component $\mathbf{R}_t$. For simplicity, we choose a simple interpolation scheme for $\mathbf{R}_t$ that is independent of the position of the obstacles (e.g. a linear interpolation in Eq. (1)). By plugging $\mathbf{R}_t$, the system reduces to a linear one with three variables in the translational component $\mathbf{T}_t$. If the total number of closest features is less than three, this system is under-constrained and one has to choose a meaningful solution from the infinite solution set. If there are exactly three independent distance constraints, the system has a unique solution. We address these issues in more details in Sections 4 and 5.

3.4 Formulation of Distance Constraints

Given a pair of features from the robot and the obstacles, we want to impose a constraint that the signed distance between the pair of features varies linearly. We consider three possible types of closest feature pairs between the boundaries:

1. (**V**,**F**): the closest features are a vertex on the robot and a face of the obstacle,
2. (**F**,**V**): the closest features are a face of the robot and a vertex of the obstacle,
3. (**E**,**E**): the closest features are an edge of the robot and an edge of the obstacle.

To handle other types of closest feature pairs, we first decompose them. For example, a (**V**,**E**) pair can be decomposed into two (**V**,**F**) pairs.

(V,F) Distance Constraint: Suppose the equation of the plane containing the face $\mathbf{F}$ is $\{\mathbf{x} | \mathbf{N} \cdot \mathbf{x} + D = 0, \|\mathbf{N}\| = 1, \mathbf{x} \in \mathbb{R}^3\}$. Let d_0 as the signed distance between the vertex $\mathbf{V}_0$ ($\mathbf{V}$ at $\mathbf{q}_0$) and this plane. Similarly, d_1 is defined as the distance for $\mathbf{q}_1$. Given d_0 and d_1 with same signs, we want the sign of distance function between the vertex $\mathbf{V}$ and the plane does not change when the robot moves. This constraint can be satisfied by:

$$\mathbf{N} \cdot (\mathbf{R}_t \mathbf{V}_0 + \mathbf{T}_t) + D - d_t = 0, \tag{4}$$

with $d_t = (1-t)d_0 + t d_1$, a linear interpolant of the signed distances d_0 and d_1. This constraint can be reformulated as $\mathbf{N}_t \cdot \mathbf{T}_t + s_t = 0$, where:

$$\begin{cases} \mathbf{N}_t = \mathbf{N}, \\ s_t \ = \mathbf{N} \cdot \mathbf{R}_t \mathbf{V}_0 + D - d_t. \end{cases} \tag{5}$$

(F,V) Distance Constraint: Given a face on the robot at $\mathbf{q}_0$ with the plane equation $\{\mathbf{x}|\mathbf{N}_0 \cdot \mathbf{x} + D_0 = 0, \|\mathbf{N}_0\| = 1, \mathbf{x} \in \mathbb{R}^3\}$ and a vertex $\mathbf{V}$ on the obstacle, the sign of the distance function between the plane and the point should not change between $t = 0$ and $t = 1$ (Fig. 2(a)). This can be expressed as:

$$(\mathbf{R}_t\mathbf{N}_0) \cdot (\mathbf{V} - \mathbf{T}_t) + D_0 - d_t = 0, \tag{6}$$

where $d_t = (1-t)d_0 + td_1$.

We reformulate this constraint as $\mathbf{N}_t \cdot \mathbf{T}_t + s_t = 0$, where:

$$\begin{cases} \mathbf{N}_t &= -\mathbf{R}_t\mathbf{N}_0, \\ s_t &= (\mathbf{R}_t\mathbf{N}_0) \cdot \mathbf{V} + D - d_t. \end{cases} \tag{7}$$

(E,E) Distance Constraint: Given an edge on the robot at $\mathbf{q}_0$ with end points $\mathbf{a}_0^0$ and $\mathbf{a}_0^1$, and an edge on the obstacle with end points $\mathbf{b}^0$ and $\mathbf{b}^1$, the sign of the distance function between the lines containing the edges should not change between $t = 0$ and $t = 1$ (Fig. 2(b)). In this case, the normal of the plane that contains both the lines is given as:

$$\mathbf{N}_t = Normalize((\mathbf{b}^1 - \mathbf{b}^0) \times (\mathbf{R}_t(\mathbf{a}_0^1 - \mathbf{a}_0^0))) \tag{8}$$

This constraint can be expressed as:

$$\mathbf{N}_t \cdot (\mathbf{R}_t\mathbf{a}_0^0 + \mathbf{T}_t - \mathbf{b}^0) - d_t = 0, \tag{9}$$

where $d_t = (1-t)d_0 + td_1$.

The constraint then can be reformulated as $\mathbf{N}_t \cdot \mathbf{T}_t + s_t = 0$, with $s_t = \mathbf{N}_t \cdot (\mathbf{R}_t\mathbf{a}_0^0 - \mathbf{b}^0) - d_t$.

4 Motion Interpolation: Single Distance Constraint

In the previous section, we presented our formulation for specifying the distance constraints for different types of feature pairs. In order to compute the interpolating motion, we need to consider different combinations of closest feature pairs at the configuration $\mathbf{q}_0$ and $\mathbf{q}_1$. In this section, we present our motion interpolation algorithm for a single (**V**, **F**), (**F**, **V**), or (**E**,**E**) *distance constraint*. In other words, the closest feature pair at $\mathbf{q}_0$ and $\mathbf{q}_1$ is identical. With only one distance constraint, the system for computing the motion (Eq. (3)) is under-constrained even if the rotational component $\mathbf{R}(t)$ is given. Its solution set at any time t is a two-dimensional plane. We compute an interpolation motion using a simple geometric construction scheme. The resulting motion satisfies the distance constraint specified in Section 3.4.

4.1 Motion Interpolation with (V,F) Distance Constraint

We first consider the case when the closest features of the robot and the obstacle are a vertex **V** and a face **F**, respectively. In order to generate an interpolation motion that satisfies the distance constraint between features **V** and **F** at any time t, our algorithm rotates the robot around the vertex **V** at $\mathbf{q}_0$ (denoted as $\mathbf{V}_0$), instead of its origin **o** (Fig. 1). The robot is meanwhile translated along the vector from $\mathbf{V}_0$ to $\mathbf{V}_1$ (**V** at $\mathbf{q}_1$). More specifically, the equation to compute the coordinate of any point on the robot **x** at time t can be expressed as:

$$\mathbf{x}_t = \mathbf{R}_t(\mathbf{x}_0 - \mathbf{V}_0) + \mathbf{V}_0 + t(\mathbf{V}_1 - \mathbf{V}_0), \tag{10}$$

Therefore, we can represent this motion as:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{R}_t\mathbf{x}_0 + \mathbf{T}_t, \\ \mathbf{T}_t &= -\mathbf{R}_t\mathbf{V}_0 + \mathbf{V}_0 + t(\mathbf{V}_1 - \mathbf{V}_0), \end{aligned} \tag{11}$$

where $\mathbf{R}_t$ is any interpolant on the rotational component.

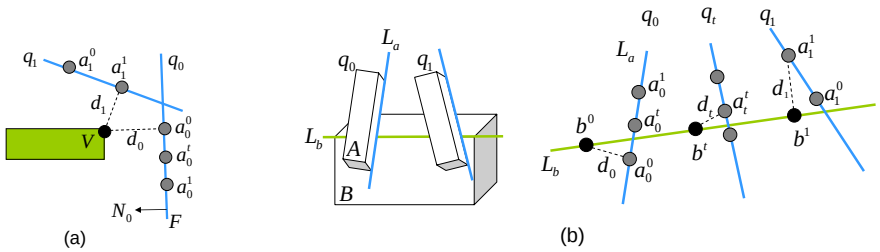


Fig. 2. *(a) Motion Interpolation with a single (F,V) distance constraint (Section 4.2). We highlight the closest points on the face F of the robot at $t = 0$, $\mathbf{a}_0^0$ and at $t = 1$, $\mathbf{a}_1^1$. The closest feature on the obstacle is the vertex* **V**. (b) *Motion Interpolation with a single (E,E) Distance Constraint (Section 4.3). These edges correspond to L_a on the robot and L_b on the obstacle. We also highlight the closest points on the edges at $t = 0$ and $t = 1$.*

4.2 Motion Interpolation with (F,V) Distance Constraint

Lets consider the case when the closest features of the robot and the obstacle are a face **F** and a vertex **V**, respectively. The goal is to generate an interpolating motion that satisfies the distance constraint for these features. Based on Fig. 2(a), we denote $\mathbf{a}_0^0$ and $\mathbf{a}_1^1$ as the projected points of the point **V** on the plane containing **F** at the configurations $\mathbf{q}_0$ and $\mathbf{q}_1$, respectively. We use the symbol $\mathbf{a}_0^1$ to denote the point $\mathbf{a}^1$ at time $t = 0$. At time t, we first compute the point $\mathbf{a}_0^t$, which is the linear interplant of t on the points $\mathbf{a}_0^0$ and $\mathbf{a}_0^1$; we then compute $\mathbf{a}_t^t$, which is the point $\mathbf{a}_0^t$ at time t by using Eq. (2). If we impose the motion such that $\mathbf{a}_t^t$ is the closest point between the point **V** and the plane containing **F** at time t, the distance constraint between the point **V** and this plane can be expressed as:

$$\mathbf{V} - d_t \mathbf{N}_t = \mathbf{R}_t \mathbf{a}_0^t + \mathbf{T}_t, \tag{12}$$

where d_t is a linear interpolant on the signed distances d_0 and d_1; $\mathbf{N}_t = \mathbf{R}_t \mathbf{N}_0$, where $\mathbf{N}_0$ as the normal of the plane at $\mathbf{q}_0$.

Therefore, the translational component $\mathbf{T}_t$ is given as:

$$\mathbf{T}_t = \mathbf{V} - d_t(\mathbf{R}_t \mathbf{N}_0) - \mathbf{R}_t \mathbf{a}_0^t. \tag{13}$$

4.3 Motion Interpolation with (E,E) Distance Constraint

Lets consider the case when the closest features are both edges. The line containing such edges on the robot and the obstacle are denoted as L_a and L_b, respectively (Fig. 2(a)). Moreover, we use the symbols $\mathbf{b}^0$ and $\mathbf{a}_0^0$ as the closest points on the lines L_b and L_a, respectively, at time $t = 0$, while $\mathbf{b}^1$ and $\mathbf{a}_1^1$ are the closest points between the lines at time $t = 1$. We further denote:

$$\begin{aligned} \mathbf{b}^t &= (1-t)\mathbf{b}^0 + t\mathbf{b}^1, \\ \mathbf{a}_0^t &= (1-t)\mathbf{a}_0^0 + t\mathbf{a}_0^1, \\ \mathbf{a}_t^t &= \mathbf{R}_t \mathbf{a}_0^t + \mathbf{T}_t, \\ \mathbf{N}_t &= Normalize(\mathbf{R}_t(\mathbf{a}_0^1 - \mathbf{a}_0^0) \times (\mathbf{b}^1 - \mathbf{b}^0)). \end{aligned} \tag{14}$$

If we impose the motion such that $\mathbf{a}_t^t$ and $\mathbf{b}^t$ are the closest points between the line L_a at time t and the line L_b, their distance constraint now can be expressed as:

$$\mathbf{R}_t \mathbf{a}_0^t + \mathbf{T}_t = \mathbf{b}^t + d_t \mathbf{N}_t, \tag{15}$$

where d_t is a linear interpolant on the signed distances of d_0 and d_1.

Therefore, $\mathbf{T}_t$ can be represented as:

$$\mathbf{T}_t = d_t\mathbf{N}_t + \mathbf{b}^t - \mathbf{R}_t\mathbf{a}_0^t. \tag{16}$$

One can show that our motion interpolation algorithm satisfies the distance constraint, as stated by the following lemma:

Lemma 2. *The motion interpolated by Eq. (11), (13), or (16) satisfies the input distance constraint between the given feature pairs specified by Eq. (5), (7), or (9), respectively.*

5 Motion Interpolation: Multiple Distance Constraints

In the previous section, we presented our motion interpolation algorithm for a single constraint. Our formulation had assumed that the closest features at configurations $\mathbf{q}_0$ and $\mathbf{q}_1$ are same and thereby handle a single distance constraint. In this section, we extend to the case of multiple distance constraints. We compute the locally closest feature pairs between the robot and obstacles. We derive closed forms for our constrained interpolation which can consider up to three locally closest feature pairs. By taking into account multiple constraints, the resulting interpolating motion conforms better to the local geometry of C-obstacles in the configuration space.

5.1 Two Distance Constraints

Similar to the earlier cases, we again assume that the rotational component of the motion, $\mathbf{R}_t$, is interpolated by a simple interpolant (e.g. linear interpolation). The goal of our motion interpolation is to compute so that the signed distances between each of two locally closest features vary according a given interpolant. This can be expressed as:

$$\begin{cases} \mathbf{N}_t^0 \cdot \mathbf{T}_t + s_t^0 = 0, \\ \mathbf{N}_t^1 \cdot \mathbf{T}_t + s_t^1 = 0, \end{cases} \tag{17}$$

where $\mathbf{N}^i$ and s^i is any type of distance constraint formulated in Section 3.4.

The goal is to compute $\mathbf{T}_t$ for this system. With three unknown variables and two equations, this is an under-constrained system. Furthermore, for a given value of time t, the system is linear since both $\mathbf{N}_t^i$ and s_t^i can be calculated according to our distance constraint formulation in Section 3.4. In general, the solution to this linear system is a one-dimensional set. In order to solve the system, we specify one more constraint explicitly as follows. The two equations at time t in Eq. (17) determine a line in $\mathbb{R}^3$. We first compute a base point ζ at any time t (denoted as ζ_t) on this line:

$$\begin{cases} \mathbf{N}_t^0 \cdot \zeta_t + s_t^0 = 0, \\ \mathbf{N}_t^1 \cdot \zeta_t + s_t^1 = 0, \\ \mathbf{N}_t^2 \cdot \zeta_t = 0. \end{cases} \tag{18}$$

Here $\mathbf{N}_t^2$ is given as:

$$\mathbf{N}_t^2 = Normalize(\mathbf{N}_t^0 \times \mathbf{N}_t^1). \tag{19}$$

ζ at time t can be easily computed by the following equation:

$$\zeta_t = \begin{bmatrix} \mathbf{N}_t^0 \\ \mathbf{N}_t^1 \\ \mathbf{N}_t^2 \end{bmatrix}^{-1} [-s_t^0, -s_t^1, 0]^T. \tag{20}$$

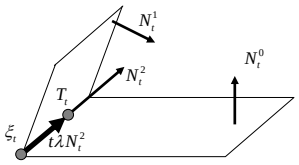


Fig. 3. *Computation of the translational component $\mathbf{T}_t$ of the interpolating motion with two distance constraints. We use the distance constraints for each type of feature pair defined in Section 3.4, and compute the vector $\mathbf{N}_t^2$ accordingly.*

Based on it, we can show that both ζ_0 and ζ_1 are 0.

Now, the base point ζ_t is displaced along the direction $\mathbf{N}_t^2$. The equation to compute $\mathbf{T}_t$ can be expressed as:

$$\begin{aligned}\mathbf{T}_t &= \zeta_t + t\lambda\mathbf{N}_t^2 \\ &= \begin{bmatrix}\mathbf{N}_t^0 \\ \mathbf{N}_t^1 \\ \mathbf{N}_t^2\end{bmatrix}^{-1} [-s_t^0, -s_t^1, 0]^T + t\lambda\mathbf{N}_t^2,\end{aligned} \tag{21}$$

where $\lambda = \mathbf{T}_1 \cdot \mathbf{N}_1^2$, so that the result of this equation at time $t = 1$ interpolates $\mathbf{T}_1$.

5.2 Three Distance Constraints

Next we consider the case when there are three pairs of closest features between the robot and obstacles. In this case, we obtain a linear system with three constraints. In general, $\mathbf{T}_t$ can be calculated by solving this linear system:

$$\begin{cases}\mathbf{N}_t^0 \cdot \mathbf{T}_t + s_t^0 = 0, \\ \mathbf{N}_t^1 \cdot \mathbf{T}_t + s_t^1 = 0, \\ \mathbf{N}_t^2 \cdot \mathbf{T}_t + s_t^2 = 0,\end{cases} \tag{22}$$

where $\mathbf{N}^i$ and s^i is any type of distance constraint formulated in Section 3.4, as shown in Eqs. (6), (8) or (10).

Therefore:

$$\mathbf{T}_t = \begin{bmatrix}\mathbf{N}_t^0 \\ \mathbf{N}_t^1 \\ \mathbf{N}_t^2\end{bmatrix}^{-1} [-s_t^0, -s_t^1, -s_t^2]^T. \tag{23}$$

Lemma 3. *The motion computed by Eq. (21) or (23) satisfies the input two or three distance constraints between the given feature pairs.*

5.3 Degenerate Situations

Our algorithm to compute the interpolating motion with multiple distance constraints is general. Depending on the specific pairs of closest features, it uses the appropriate formulations derived in Section 3.4. However, the formulation can result in degenerate situations, which can happen during the normalization operation in Eq. (19) or the matrix inverse operation in Eqs. (21) and (23). Conceptually, the degeneracy happens when $\mathbf{N}_t$ for two specific constraints becomes parallel at some time t. We can easily detection these situations by computing a bound on dot-product of $\mathbf{N}_t$ for the whole interval $t \in [0,1]$, either using discrete sampling or a continuous scheme based on interval arithmetic computation. Once a degenerate case is detected, our algorithm generates a new motion interpolation by only considering a subset of the given distance constraints (e.g. only one constraint instead of two constraints).

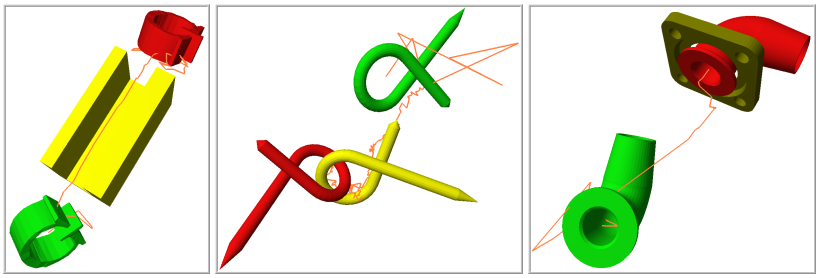


Fig. 4. *Benchmarks: We highlight the collision-free paths computed for Notch-g, Alpha Puzzle and Flange benchmarks. Each benchmark has narrow passages. The performance improvement using our constrained motion interpolation algorithm is summarized in Table 2.*

6 Implementations and Performance

We have implemented our constrained motion interpolation algorithm and used it for local planning step in sample-based motion planning. All the timings reported in this section were taken on a 3.6GHz Xeon PC.

6.1 Implementation and Performance

Our implementation makes no assumption about the models or connectivity and is applicable to all general rigid models that can be represented as polygonal soups.

Distance Computation for Polygonal Soup Models: In order to compute multiple closest feature pairs between the robot and the obstacles, we extend a distance computation algorithm in library - *PQP* [13]. More specifically, we determine all feature pairs between the robot at $\mathbf{q}_0$ and $\mathbf{q}_1$, and obstacles, whose distances are less than a user-specified tolerance. This can be efficiently performed by making use of bounding volume hierarchy in *PQP*. We then use a simple heuristic of clustering to choose a set of representative feature pairs, i.e. the locally closest feature pairs between the models [21]. As shown in Fig. 5, we compute two locally closest feature pairs in (a) and three pairs in (b). Compared with other distance computation algorithms (such as Lin-Canny or GJK algorithms), our implementation is able to handle polygonal soup models and compute a set of locally closest feature pairs.

Constrained Motion Interpolation: We use the set of locally closest feature pairs to setup the distance constraints for computing the interpolation motion. Our formulation can take up to three feature pairs. If any feature pair results in a degeneracy situation, we ignore that pair. Table 1 highlights the performance of our constrained interpolation algorithm, showing a breakdown of timing in different steps of the algorithm on various benchmarks. There is very little extra overhead of using our algorithm to compute the motion.

Collision Checking: Though our algorithm guarantees no collisions between the closest features, we still need to check whether there are collision between other features of the robot and obstacles. Currently, we use the simplest method by generating a finite number of discrete samples and performing discrete collision detection at those samples.

	Notch-g	Alpha Puzzle	Flange
Compute closest features (ms)	1.315	2.045	18.619
Formulate distance constraints (ms)	0.046	0.137	0.179
Collision checking (ms)	9.418	26.193	69.452

Table 1. *This table gives a breakdown of the timing among different steps of a local planner that uses our constrained interpolation algorithm. This table presents the average timings per query in milli-seconds on different benchmarks. Most of the time is spent is spent in collision checking, which is similar to other local planners.*

	Notch-g	Alpha Puzzle	Flange
Constrained Motion (timing)	56.4s	1,043.0s	33.4s
Constrained Motion (# samples)	2,659	53,535	80
Linear Interpolation (timing)	272.1s	1,861.4s	46.3s
Linear Interpolation (# samples)	11,762	94,283	117
Slerp Interpolation (timing)	295.9s	2,029.8s	50.6s
Slerp Interpolation (# samples)	12,568	113,298	113

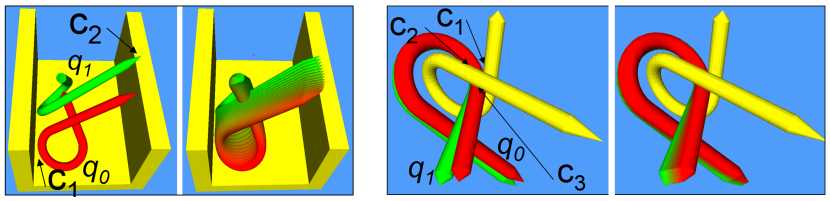
Table 2. *Benefit of our constrained motion interpolation algorithm: We compare our local planner based on constrained motion interpolation with other interpolation schemes. All these local interpolation algorithms were integrated with a retraction-based RRT planner and applied to different benchmarks. This table presents the average timings on computing a collision-free path on different benchmarks. In models with complex narrow passages, like Notch-g and Alpha Puzzle, we observe significant speedups due to our planner. Moreover, the resulting planner needs to generate fewer samples.*

6.2 Integration with Sample-based Planners

Sample-based planners randomly generate samples and connect nearby free-space samples using local planning algorithms. To improve the overall performance of planners, many sampling strategies have been proposed to increase the probability of sampling in narrow passages. Instead, our motion interpolation algorithm is used to improve the local planning step. In general, our interpolation algorithm is more useful for sampling strategies which tend to generate more samples near contact space [2, 4, 15, 16, 22]. For samples in open free space, simple interpolation schemes such linear interpolation can work well. However, when samples are closer to the C-obstacle boundary or lie in narrow passages, they are more difficult to be connected. To address this issue, our constrained motion interpolation formulation takes into account the position of C-obstacle.

In our experiment, we integrate our interpolation algorithm with a variant RRT planner, namely RRRT [22]. The RRRT planner uses a retraction-based sampling strategy to generate more samples near the contact space and bias the exploration towards difficult or narrow regions. We use our constrained motion interpolation to connect samples near contact space. For the other samples (e.g. in the open free space), we use a simpler interpolation scheme, such as linear interpolation. In our current implementation, we perform discrete collision checking on a finite number of samples along the interpolated motion.

Table 2 highlights the performance improvement in our new planner on different benchmarks. The total timing and the number of nodes in the resulting RRT tree are



(a) (b)

Fig. 5. *Motion Interpolation With Distance Constraints: given two configurations* $\mathbf{q}_0$ *and* $\mathbf{q}_1$*, our algorithm computes a constrained interpolating motion (two constraints in (a) and three constraints in (b)). The resulting motion is more likely collision-free.*

reported. We observe the overall performance improvement on these benchmarks. In Fig. 7, we highlight the performance on a more complex benchmark for part disassembly application.

7 Properties and Extensions

In this section, we first highlight some properties of our constrained motion interpolation scheme. We further show the extensions to compliant motion planner and articulated robots.

7.1 Coordinate-Invariance Property

Our formulation of constrained motion interpolation is coordinate-invariant. We outline the proof as follows and omit the detailed proof due to the space limitation. First, the formulation is left-invariant w.r.t the choice of the inertial frame of the robot. This can be proved in a similar way as [20] by replacing the exponential map with the transformation matrix for representing rotational components. Secondly, the formulation is also right-invariant w.r.t the choice of the body-fixed frame of the robot. Consider the simplest $(\mathbf{V}, \mathbf{F})$ closest feature case. In this motion equation Eq. (11), the terms $\mathbf{x}_0$, $\mathbf{V}_0$ and $\mathbf{V}_1$ do not vary w.r.t the change of the body frame. Therefore, the position $\mathbf{x}_t$ does not change as well. This guarantees right-invariance. For the other cases, we can prove this property in a similar manner.

7.2 Visibility Function Formulation

Our interpolation scheme can improve the visibility between samples near contact space and in narrow passages. In many ways, our constrained motion interpolation algorithm can be interpreted as defining a new visibility function between nearby configuration that is more effective for sample-based planners. This is shown by Table 3. We generate a set of samples near the contact space of the robot and obstacles. Then we performs link queries among adjacent samples by using different interpolation schemes in each experiment. We compute the failure ratio of link queries, which is defined as the number of link queries reporting collisions divided by the total number of link queries performed between the samples near the contact space. Table. 3 highlights the improved ratios obtained using constrained motion interpolation algorithm.

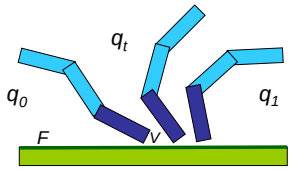


Fig. 6. *Articulated Models: our interpolation scheme can be potentially applied to articulated models.*

	Notch-g	Alpha Puzzle	Flange
Constrained Motion	27.62%	13.22%	16.41%
Linear Interpolation	47.02%	27.82%	22.19%
Slerp Interpolation	46.60%	28.63%	26.57%

Table 3. *Improved Visibility Formulation near Contact Space: Our constrained motion interpolation algorithm is more effective in connecting the samples near the contact space.*

7.3 Compliant Motion Generation

Given two contact configurations $\mathbf{q}_0$ and $\mathbf{q}_1$, our algorithm can be used to interpolate a motion, which maintains a contact among the closest features pairs at $\mathbf{q}_0$ and $\mathbf{q}_1$. In order to generate such a motion, the signed distances d_0 and d_1 in our distance constraint formulations are set as zero, since $\mathbf{q}_0$ and $\mathbf{q}_1$ are contact configurations. Fig. 5(b) shows the motion that maintains the contact among the given feature pairs along the trajectory. We still need to check whether any other features of the robot have penetrated into the obstacle. This formulation can be combined with sample-based compliant motion planners [11].

7.4 Articulated Models

In Sections 4 and 5, we described our algorithm for rigid robots. In this section, we present a simple heuristic to extend the approach to articulated models. Fig. 6 illustrates a simple scheme for a serial chain robot with movable base. We first compute the link of the robot that is closest to the obstacles. When the robot is near the obstacle, this link would be a likely candidate for collisions with the obstacle. We treat this link as a rigid robot and compute a constrained interpolating motion for it between the two configurations based on the algorithm described in Section 5. Given the motion of this link, we use inverse kinematics to compute the interpolating path for the other links. Intuitively speaking, we are imposing the geometric constraint on the motion of the link which is most likely to result in collisions with the obstacles.

8 Conclusion and Future work

We present a simple and general algorithm for motion interpolation and local planning. Based on the closest features at the two configurations, we derive closed formulations of the trajectories. The main additional overhead is the computation of closest features. Our local planning algorithm can be combined with sample-based planners. The main benefit of our approach arises in computing collision-free paths in narrow passages, especially when the samples are very close to the boundary of the contact space. In that case, the paths computed using straight-line linear interpolation, spherical linear interpolation or screw motion may overlap with the obstacles.

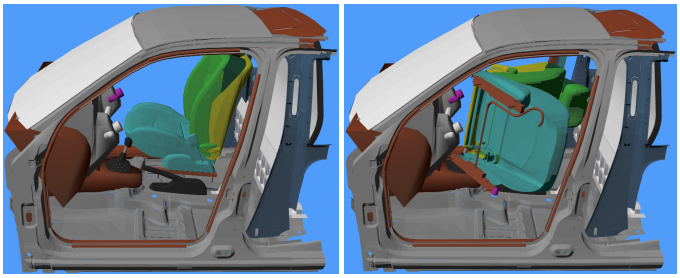


Fig. 7. *Disassembly of a Seat outside a Car Body: for this complex benchmark, our planner using constrained motion takes* 181.2*s while the same planner using linear motion takes* 242.7*s.*

On the other hand, our formulation adapts to the boundary of the contact space and can generate collision-free paths more likely. As a result, we observe speedups in the performance of overall planner.

Limitations: Our approach has a few limitations. The closed-form formulas for motion interpolation are more complex as compared to other interpolation schemes (e.g. linear interpolation). As a result, there is some additional overhead of computing feature pairs. Furthermore, the parameterization of the constrained motion is not uniform, and we may need to reparameterize in order to compute appropriate discrete samples for collision checking. Most of the prior algorithms for local planning and exact collision checking [18] work well in relatively open space. Therefore, our algorithm is applied only for the cases when the robot becomes nearer any obstacle within a user-defined parameter. This introduces a new parameter that has to be optimized.

Future Work: There are many avenues for future work. We would like to design efficient continuous collision detection algorithm based on motion bound computation. One difficulty is to compute the bounds for the constrained motion with multiple constraints. Furthermore, we would like to perform a detailed performance evaluation on articulated robots with serial and parallel joints. Finally, it may be possible to further improve the performance of the planner by designing appropriate sampling strategies and nearest neighbor selection algorithms that can take into account some of the properties of our motion interpolation algorithm.

Acknowledgment: This project was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel.

References

1. N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *Proceedings of ICRA*, pages 630–637, 1998.

2. N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based prm for 3d workspaces. *Proceedings of WAFR*, pages 197–204, 1998.
3. R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proceedings of International Conference on Robotics and Automation*, pages 521–528, 2000.
4. H.-L. Cheng, D. Hsu, J.-C. Latombe, and G. Sánchez-Ante. Multi-level free-space dilation for sampling narrow passages in PRM planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1255–1260, 2006.
5. G. S. Chirikjian and A. B. Kyatkin. *Engineering Applications of Noncommutative Harmonic Analysis*. CRC, 1 edition, Sep 2000.
6. M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.
7. R. Geraerts and M. H. Overmars. Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems*, 55:824–836, 2007.
8. M. Hofer, H. Pottman, and B. Ravani. Geometric design of motions constrained by a contacting surface pair. *Computer-Aided Geometric Design*, 20:523–547, 2003.
9. D. Hsu, J. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robotics Research*, 25(7):627–643, 2006.
10. P. Isto. Constructing probabilitic roadmaps with powerful local planning and path optimization. In *Proc. of IROS*, pages 2323–2328, 2002.
11. X. Ji and J. Xiao. Planning motion compliant to complex contact states. In *Proc. of IROS*, pages 1512–1517, 2001.
12. J. Kuffner. Effective sampling and distance metrics for 3d rigid body path planning. In *IEEE Int'l Conf. on Robotics and Automation*, 2004.
13. E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*, pages 3719–3726, 2000.
14. F. Park. Distance metrics on the rigid-body motions with applications to mechanism design. *ASME J. Mechanical Design*, 117(1):48–54, March 1995.
15. S. Redon and M. Lin. Practical local planning in the contact space. *Proc. of IEEE ICRA*, 2005.
16. M. Saha, J. Latombe, Y. Chang, Lin, and F. Prinz. Finding narrow passages with probabilistic roadmaps: the small step retraction method. *Intelligent Robots and Systems*, 19(3):301–319, Dec 2005.
17. G. Sanchez and J. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Int. Symposium on Robotics Research*, 2001.
18. F. Schwarzer, M. Saha, and J. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics*, 21(3):338–353, June 2005.
19. M. Zefran and V. Kumar. A variational calculus framework for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 415–420, 1997.
20. M. Zefran and V. Kumar. Interpolation schemes for rigid body motions. *Computer-aided Design*, 30(3):179–189, 1998.
21. L. Zhang, X. Huang, Y. Kim, and D. Manocha. D-plan: Efficient collision-free path computation for part removal and disassembly. In *Journal of Computer-Aided Design and Applications*, 2008.
22. L. Zhang and D. Manocha. A retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3750, 2008.
23. X. Zhang, M. Lee, and Y. Kim. Interactive continuous collision detection for non-convex polyhedra. In *Pacific Graphics 2006 (Visual Computer)*, 2006.