Efficient Local Planning using Connection Collision Query

Min Tang[1] Young J. Kim [1] Dinesh Manocha[2]
[1] Ewha Womans University, Korea
{tangmin|kimy}@ewha.ac.kr
[2] University of North Carolina at Chapel Hill, U.S.A.
dm@cs.unc.edu

May 15, 2010

Abstract

We introduce a novel proximity query, called *connection collision query* (CCQ), and use it for efficient and exact local planning in sampling-based motion planners. Given two collision-free configurations, CCQ checks whether these configurations can be connected by a given continuous path that either lies completely in the free space or penetrates any obstacle by at most ε, a given threshold. Our approach is general, robust, and can handle different continuous path formulations. We have integrated the CCQ algorithm with sampling-based motion planners and can perform reliable local planning queries with little performance degradation, as compared to prior methods. Moreover, the CCQ-based exact local planner is about an order of magnitude faster than prior exact local planning algorithms.

1 Introduction

Planning a collision-free motion for a robot amongst obstacles is an important problem in robotics, CAD/CAM, computer animation and bioinformatics. This problem is well studied and many approaches have been proposed. Over the last few decades, sampling-based motion planners such as probabilistic roadmaps [1] (PRMs) or rapidly-exploring random trees [2] (RRTs) have been shown to be successful in terms of solving challenging problems with high degrees-of-freedom (DoFs) robots. These planners attempt to capture the topology of the free space by generating random configurations and connect nearby configurations using local planning algorithms.

The main goal of a local planner is to check whether there exists a collision-free path between two free configurations It is important that the local planner should be reliable and does not miss any collisions with the obstacles [3, 4]. Moreover, it is known that a significant fraction of the overall running time of a sampling-based planner is spent in the local planning routines.

The simplest local planning algorithms compute a continuous interpolating path between the free configurations and check the path for collisions with the obstacles. These algorithms sample the continuous path at a fixed resolution and discretely check each of those resulting configurations for collisions. These *fixed-resolution local planning* algorithms are simple to implement, but suffer from two kinds of problems:

1. **Collision-miss:** It is possible that the planner may miss a collision due to insufficient sampling. This can happen in the narrow passages or when the path lies close to the obstacle boundary. This affects the accuracy of the planner.

2. **Collision-resolution:** Most planners tend to be conservative and tend to generate a very high number of samples. This results in a lot of discrete collision queries and affects the running time of the planner.

Overall, it is hard to compute the optimal resolution parameter that can guarantee collision-free motion and is fast. In order to overcome these problems, some local planners use exact methods such as continuous collision detection (CCD) [5, 6] or dynamic collision checking [7]. However, there is a widespread belief that these *exact local planning* methods are quite expensive and more than an order of magnitude slower than fixed-resolution local planner. Many well-known implementations of sampling-based planners such as OOPSMP[1] and MSL[2] only use fixed-resolution local planning, though MPK[3] performs exact collision checking for local planning.

Main Results: We introduce a novel proximity query, namely *connection collision query* (CCQ), for fast and exact local planning in sampling-based motion planners. At a high level, our CCQ algorithm can report two types of proximity results:

- **Boolean CCQ query**: Given two collision-free configurations of a moving robot in the configuration space, CCQ checks whether the configurations can be connected by a given path that lies in the free space, namely *Boolean CCQ_s query*. In addition, the CCQ query can also check whether the path lies partially inside obstacle region (C-obstacle) with at most ε-penetration, namely *Boolean CCQ_p query*. In this case, the robot may overlap with some obstacles and the extent of penetration is bounded above by ε.

- **Time of violation (ToV) query**: If the Boolean queries report `FALSE` (*i.e.* the path does not exist), the CCQ query reports the first parameter or the configuration along the continuous path that violates these path constraints

Moreover, our algorithm can easily check different types of continuous paths including a linear interpolating motion in the configuration space or a screw motion.

We have integrated our CCQ algorithm into well-known sampling-based motion planners and compared their performance with prior methods. In practice, we observe that an exact local planning algorithm based on the CCQ query can be at most two times slower than a fixed-resolution local planning, though the paths computed using CCQ queries are guaranteed to be collision-free. Finally, we also show that our CCQ algorithm outperforms prior exact local planners by an order of magnitude.

Paper Organization: The rest of this paper is organized as follows. In Sec. 2, we briefly survey the related work and formulate the CCQ problem in Sec. 3. Sections 4 and 5 describe the CCQ algorithms for rigid robots with separation and penetration constraints, respectively. We describe how our CCQ algorithm can be extended to articulated robots in Sec.6, and highlight the results for different motion planning benchmarks in Sec. 7 and conclude the paper in Sec. 8.

2 Previous Work

Our CCQ algorithm is related to continuous collision detection. In this section, we give a brief survey on these proximity queries and local planning.

[1]`http://www.kavrakilab.org/OOPSMP`
[2]`http://msl.cs.uiuc.edu/`
[3]`http://robotics.stanford.edu/~mitul/mpk/`

2.1 Continuous Collision Detection

The term, continuous collision detection, was first introduced by Redon *et al.* [5] in the context of rigid body dynamics, even though the earlier work on similar problems dates back to the late 1980s [8]. The main focus of CCD algorithms lies in finding the first time of contact for a fast moving object between two discrete collision-free configurations.

Many CCD algorithms for rigid models have been proposed [9]; these include algebraic equation solvers, swept volume formulations, adaptive bisection approach, kinetic data structures approach, Minkowski sum-based formulations and conservative advancement (CA).

For articulated models, Redon *et al.*[10] present a method based on the continuous OBB-tree test, and Zhang *et al.* [6] have extended the CA method to articulated models. In the context of motion planning, Schwarzer *et al.* [7] present a dynamic collision checking algorithm to guarantee a collision-free motion between two configurations.

These algorithms have been mainly used for rigid body dynamics and their application to sample-based planning has been limited [7]. In practice, the performance of these exact local planning methods is considered rather slow for motion planners. Moreover, current CCD algorithms are not versatile enough to handle different types of connection paths and queries, that have been used by different local planners.

2.2 Local Planning

There are two important issues related to our work in terms of local planning: the type of continuous interpolating path and the validity of the path in terms of collisions. The former is related to motion interpolation between collision-free samples, and the latter is related to collision checking.

2.2.1 Motion Interpolation

In the context of local planning, different types of motion interpolation methods have been used such as linear motion in C-space [7], spherical motion in C-space [11], screw motion [12], etc. These motion trajectories are rather simple to compute and cost-effective for local planning.

More sophisticated motion interpolation techniques have been introduced to find an effective local path by taking into account the robot/obstacle contacts [13, 14], variational schemes [15] and distance constraints [16]. Amato *et al.* [4] evaluate different distance metrics and local planners, and show that the translational distance becomes more important than the rotational distance in cluttered scenes.

2.2.2 Collision Checking

Given a path connecting two collision-free configurations, a conventional way of local planning is to sample the path at discrete intervals and perform static collision detection along the discrete path [17, 18]. Some exact collision checking method has been proposed for local planning such as [7, 19] using adaptive bisection.

Since collision checking can take more than 90% time in sample based planner, lazy collision evaluation techniques have been proposed [20, 21] to improve the overall performance of a planner. The main idea is to defer collision evaluation along the path until it is absolutely necessary. It turns out that these techniques help to greatly improve the performance of PRM-like algorithms, but they do not improve the reliability of resolution-based collision checker.

When narrow passages are present in the configuration space, it is hard to capture the connectivity of the free space by using simple collision checking, since it may report a lot of invalid local paths. However, some retraction-based planners [22, 23, 19, 24] allow slight penetration into the obstacle region based on penetration depth computation, which makes the local planning more effective.

3 Problem Formulation

We start this section by introducing our notation that is used throughout the paper. Next, we give a precise formulation of CCQ.

3.1 Notations and Assumptions

We use bold-faced letters to denote vector quantities (*e.g.* $\mathbf{o}$). Many other symbols used in the paper are given in Table 1. We assume that both the robot $\mathfrak{A}$ and obstacle $\mathfrak{B}$ are rigid and defined in $\mathbb{R}^3$ workspace. Moreover, the robot has 6 DoFs and the obstacle is fixed in space; thus, the C-space of $\mathfrak{A}$ is SE(3). We briefly discuss how to handle high DoF robots later in Sec. 6.

Notation	Meaning
$\mathfrak{A}, \mathfrak{B}, \partial\mathfrak{A}, \partial\mathfrak{B}$	robot, obstacle and their boundaries
$\mathscr{C}$	C-space of $\mathfrak{A}$
$\mathbf{q}, \mathbf{q}(t)$	a sample in C-space and a $1D$ curve in C-space
$\mathfrak{A}(\mathbf{q}), \mathfrak{A}(t)$	placements of the robot $\mathfrak{A}$ at $\mathbf{q}$ and $\mathbf{q}(t)$
$\mathscr{F}, \mathscr{O}$	C-free and C-obstacle region in $\mathscr{C}$ (*i.e.* $\mathscr{C} = \mathscr{F} \cup \mathscr{O}$)
$\|\cdot,\cdot\|$	Euclidean distance operator

Table 1: Notations.

3.2 Local Planning in Sampling-based Motion Planner

Given the starting $\mathbf{q}_0$ and goal $\mathbf{q}_1$ configurations in $\mathscr{F}$, most sampling-based randomized planners compute a search graph $\mathscr{G}$ to explore the C-space, where the vertex corresponds to a sample in $\mathscr{F}$ and the edge is a $1D$ curve in C-space connecting two collision-free samples. More specifically, the sample based planners works in the following manner :

1. **Sample Generation:** Sample a collision-free configuration $\mathbf{q}_1$ in $\mathscr{F}$.
2. **Local Planning:** Check whether $\mathbf{q}_1$ can be connected to a vertex $\mathbf{q}_0$ in $\mathscr{G}$ by some collision-free, continuous path $\mathbf{q}(t)$ in C-space. If so, a new edge connecting $\mathbf{q}_0, \mathbf{q}_1$ is created and added into $\mathscr{G}$ along with a vertex $\mathbf{q}_1$.
3. **Graph Search:** Use a graph search on $\mathscr{G}$ to find a path from $\mathbf{q}_0$ to $\mathbf{q}_1$. If such a path is found, the algorithm reports the path and is terminated; otherwise, go back to the step 1 and repeat.

In the local planning step, the choice of a continuous path $\mathbf{q}(t)$ interpolating $\mathbf{q}_0, \mathbf{q}_1$ may vary depending on the geometry and topology of $\mathscr{F}$. Once a specific path formulation is chosen, the algorithm needs to check whether that path is collision-free or not.

3.3 Connection Collision Query

Now we define the CCQ proximity query, the main problem to solve in this paper.

Let us assume that two collision-free samples $\mathbf{q}_0, \mathbf{q}_1 \in \mathscr{F}$ in $\mathscr{C}$ and a time-parameterized, continuous $1D$ curve $\mathbf{q}(t)$ in $\mathscr{C}$ connecting $\mathbf{q}_0$ and $\mathbf{q}_1$ are given for $t \in [0,1]$; *i.e.* $\mathbf{q}(0) = \mathbf{q}_0, \mathbf{q}(1) = \mathbf{q}_1$. Then, the CCQ with separation is formally defined as checking whether the following predicate CCQ_s is `TRUE`:

$$\text{CCQ}_s : \forall t \in [0,1] \Rightarrow \mathbf{q}(t) \in \mathscr{F}. \tag{1}$$

Moreover, if CCQ$_s$ is `FALSE`, we want to determine the minimum value of t that satisfies CCQ$_s$. We call such t as the time of violation (ToV) with separation, τ_s. More formally,

$$\tau_s \equiv \min_t \{\, s \in [0,t] \mid \mathbf{q}(s) \in \mathscr{F} \}. \tag{2}$$

The Boolean CCQ$_s$ query is useful for local planning in PRM and RRT, and the ToV CCQ$_s$ query is required by the local planning or expansion step in RRT.

On the other hand, the notion of CCQ with ε-penetration is a less restrictive version of connection query than CCQ with separation, as it allows slight penetration (quantified by ε) into the C-obstacle region for the C-space curve $\mathbf{q}(t)$. Formally, we define CCQ with ε-penetration as checking whether the following predicate CCQ$_p$ is `TRUE`:

$$\begin{aligned} \mathrm{CCQ}_p : \forall t \in [0,1] \Rightarrow \{\mathbf{q}(t) \in \mathscr{F}\} \vee \\ \{\mathbf{q}(t) \in \mathscr{O} \wedge \forall \mathbf{p} \in \mathfrak{A}(t) \cap \mathfrak{B}, \|\mathbf{p} - \partial\mathfrak{B}\| \le \varepsilon\}. \end{aligned} \tag{3}$$

Furthermore, if CCQ$_p$ is `FALSE`, we also determine the minimum value of t that satisfies CCQ$_p$, called the ToV with ε-penetration, τ_p. More formally, τ_p is defined as:

$$\begin{aligned} \tau_p \equiv \min_t \{\, s \in [0,t] \mid \mathbf{q}(s) \in \mathscr{F} \vee \\ \{\mathbf{q}(s) \in \mathscr{O} \wedge \forall \mathbf{p} \in \mathfrak{A}(s) \cap \mathfrak{B}, \|\mathbf{p} - \partial\mathfrak{B}\| \le \varepsilon\}\}. \end{aligned} \tag{4}$$

The CCQ$_p$ query can be used for PRM and RRT when a small amount of penetration is allowed for a robot along the local path. Moreover, retraction-based planners also may use CCQ$_p$ to keep the samples with slight penetration [22, 23, 19].

4 CCQ with Separation Constraint

In this section, we present our algorithm to perform the CCQ$_s$ query. We start this section by explaining the conservative advancement (CA) technique upon which our CCQ algorithm is based. Next, we explain the procedure to compute the ToV information τ_s in Eq.2 along with CCQ$_s$. Finally, we provide a fast technique to solve the Boolean version of CCQ$_s$ (*i.e.* Eq.1).

4.1 Conservative Advancement

Our CCQ algorithm is based on the conservative advancement (CA) algorithm [25] for convex objects under continuous motion. In CA, the time of contact (ToC) τ between two convex objects $\mathfrak{A}$ and $\mathfrak{B}$ is obtained by iteratively advancing $\mathfrak{A}$ by Δt_s toward $\mathfrak{B}$ without generating collisions. Here, Δt_s can be calculated by:

$$\Delta t_s \le \frac{\|\mathfrak{A}(t), \mathfrak{B}\|}{\mu} \tag{5}$$

where μ is the bound of motion of $\mathfrak{A}(t)$ for $t \in [0,1]$ projected onto the closest direction from $\mathfrak{A}(t)$ to $\mathfrak{B}$, known as the directional motion bound. Then, the ToC is obtained as:

$$\tau = \sum_i \Delta t_s^i \tag{6}$$

where Δt_s^i denotes the ith CA iteration. The iteration continues until $\|\mathfrak{A}(\tau), \mathfrak{B}\| \approx 0$. This idea can be extended to non-convex models using bounding volume hierarchy [9].

4.2 Time of Violation Query for CCQ$_s$

In case of CCQ$_s$, the time of violation (ToV) is equivalent to the time of contact (ToC) in CA. Moreover, if the path $\mathbf{q}(t)$ is a linear motion in C-space, one can employ the C^{2}A algorithm [9] based on CA to compute τ_s for the robot $\mathfrak{A}$. We also show that we can devise a variant of C^{2}A algorithm that can handle the screw motion for $\mathbf{q}(t)$.

The screw motion consists of rotation about an axis ω in space by an angle of θ radians, followed by translation along the same axis by an amount of d as shown in Fig.1. The screw motion can be represented by using four

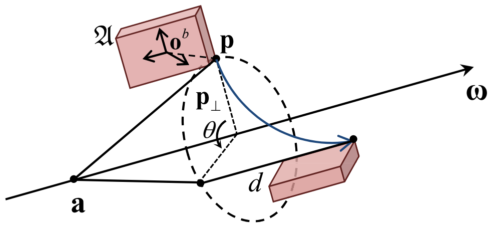


Figure 1: **Screw Motion.**

parameters $(\omega, \theta, \mathbf{a}, \mathbf{d})$, where $\mathbf{a}$ is any point on the axis ω. Given two configurations $\mathbf{q_0}$ and $\mathbf{q_1}$ in SE(3), the screw parameters can be easily computed [26].

The main challenge in computing τ_s under screw motion is to compute the directional motion bound μ for Eq.5. Let us assume that our robot $\mathfrak{A}$ is convex with the origin $\mathbf{o}^b$ of the body attached frame. Let $\mathbf{p}$ be any point on $\mathfrak{A}$ with $\mathbf{p}^b$ representing the same point but defined with respect to the body frame, $\mathbf{n}$ be the closest direction from $\mathfrak{A}$ to the obstacle $\mathfrak{B}$ at $t = 0$, $\mathbf{p}_\perp$ be a vector projected from $\mathbf{p}$ to the axis ω. Then, an upper bound μ of the motion of any point on $\mathfrak{A}$ under screwing, projected onto $\mathbf{n}$ is:

$$
\begin{aligned}
\mu &= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 (\dot{\mathbf{p}}(t)\cdot\mathbf{n})\,dt \right) \\
&= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 ((\mathbf{v}+\omega\times\mathbf{p}_\perp(t))\cdot\mathbf{n})\,dt \right) \\
&\le \max(\mathbf{v}\cdot\mathbf{n},0) + \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 |\omega\times\mathbf{n}\cdot\mathbf{p}_\perp(t)|\,dt \right) \\
&\le \max(\mathbf{v}\cdot\mathbf{n},0) + \|\omega\times\mathbf{n}\| \left(\max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \|\mathbf{p}_\perp(t)\|\,dt \right) \right) \\
&\le \max(d\omega\cdot\mathbf{n},0) + \|\omega\times\mathbf{n}\| \left(\left\|\left(\mathbf{o}^b-\mathbf{a}\right)\times\omega\right\| + \max_{\mathbf{p}\in\mathfrak{A}} \left\|\mathbf{p}^b\right\| \right).
\end{aligned} \tag{7}
$$

Note that $\max_{\mathbf{p}\in\mathfrak{A}} \left\|\mathbf{p}^b\right\|$ can be calculated as preprocessing since $\mathbf{p}^b$ is defined with respect to the body frame. A similar bound can be obtained for other motion trajectories such as the spherical motion [11].

4.3 Boolean Version of CCQ$_s$

From the previous section, the CCQ$_s$ predicate in Eq.1 can be trivially determined by checking whether $\tau_s \ge 1$ (`TRUE`) or not (`FALSE`). However, one can devise a more efficient way to answer the CCQ$_s$ predicate without explicitly computing τ_s.

Given the starting $\mathbf{q}_0$ and goal $\mathbf{q}_1$ configurations, the main idea in evaluating CCQ$_s$ is to perform dual advancements from the both end-configurations $\mathbf{q}_0, \mathbf{q}_1$ with opposite velocities, and iterate this process until collision is found or the path turns out to be collision-free. The dual advancement is more effective than the normal advancement using a single end-configuration since the normal advancement is always conservative such that collision will be never identified until the final ToV value is obtained.

More specifically, as shown in Fig. 2, we perform a single CA iteration from $\mathbf{q}_0$ toward $\mathbf{q}_1$ as before and compute the first advancement time, Δt_0^+. Similarly, we perform another CA iteration but from $\mathbf{q}_1$ toward $\mathbf{q}_0$ with a negative velocity (*e.g.* $(-\mathbf{v}, -\omega)$) and compute the first advancement time, Δt_1^-.

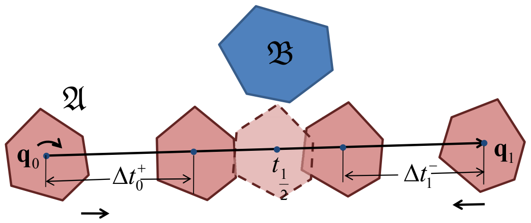


Figure 2: **A Single Step in the Boolean Query.** Dual advancements are performed from $\mathbf{q}_0$ toward $\mathbf{q}_1$ by Δt_0^+, and from $\mathbf{q}_1$ toward $\mathbf{q}_0$ by Δt_1^-. Then, collision is checked at $\mathbf{q}(\frac{1}{2})$.

If $(\Delta t_0^+ + \Delta t_1^-) \geq 0$, then the entire path $\mathbf{q}(t)$ is collision-free, thus the predicate is returned as TRUE; otherwise, we bisect the time interval at $t_{\frac{1}{2}} = \frac{t_0+t_1}{2}$ and perform collision detection at the configuration $\mathbf{q}(t_{\frac{1}{2}})$. If collision is detected at $\mathbf{q}(t_{\frac{1}{2}})$, CCQ$_s$ is reported as FALSE and the procedure is terminated. Otherwise, the same dual CA procedure is executed recursively on two sub-paths, $\{[\mathbf{q}(\Delta t_0^+), \mathbf{q}(t_{\frac{1}{2}})], [\mathbf{q}(t_{\frac{1}{2}}), \mathbf{q}(1-\Delta t_1^-)]\}$. Note that the rest of path segments $\{[\mathbf{q}(0), \mathbf{q}(\Delta t_0^+)], [\mathbf{q}(1-\Delta t_1^-), \mathbf{q}(1)]\}$ are collision-free because of conservative advance mechanism. This procedure is iterated until the separation condition is satisfied or a collision is detected. We provide a pseudo-code for CCQ$_s$ in Alg.1.

5 CCQ with Penetration Constraints

The CCQ$_s$ algorithm presented in Sec.4 strictly imposes that the interpolating path $\mathbf{q}(t)$ should lie entirely inside $\mathscr{F}$. However, this condition is rather restrictive since a slight overlap between the robot and the obstacles may be useful in practice and used by retraction-based planners [22, 19, 24]. For instance, often the curved surface model of a robot is tessellated with some surface deviation error ε and thus ε-penetration does not necessarily imply actual interference [19]. The notion of CCQ$_p$ is that we allow slight penetration for a robot along the path as long as the penetration amount is less than some threshold, ε.

5.1 Penetration Depth

To quantify the amount of penetration for a robot $\mathfrak{A}$, we need a suitable metric. The penetration depth (PD) is a proper metric to quantify the amount of overlap between $\mathfrak{A}$ and $\mathfrak{B}$. In the literature, different types of penetration depth are known and in our case, we adopt the pointwise penetration depth [9] since it is computationally more tractable than other penetration measures.

Algorithm 1 CCQ_s
Input: initial and goal configurations $\mathbf{q}_0$, $\mathbf{q}_1$, interpolating motion $\mathbf{q}(t)$
Output: whether Eq. 1 is `TRUE` or `FALSE`

```
1: {Initialize the queue with $[\mathbf{q}(0), \mathbf{q}(1)]$.}
2: while Queue $\neq \emptyset$ do
3:   Pop an element $[\mathbf{q}(t_a), \mathbf{q}(t_b)]$ from the queue;
4:   $t_{\frac{1}{2}} = \frac{t_a+t_b}{2}$;
5:   if $\mathbf{q}(t_{\frac{1}{2}})$ is in-collision then
6:     return FALSE;
7:   end if
8:   Perform CA from $\mathbf{q}(t_a)$ with a positive velocity and find the step size $\Delta t_a^+$;
9:   Perform CA from $\mathbf{q}(t_b)$ with a negative velocity and find the step size $\Delta t_b^-$;
10:  if $(\Delta t_a^+ + \Delta t_b^-) < (t_b - t_a)$ then
11:    Push $[\mathbf{q}(t_a + \Delta t_a^+), \mathbf{q}(t_{\frac{1}{2}})]$ and $[\mathbf{q}(t_{\frac{1}{2}}), \mathbf{q}(t_b - \Delta t_b^-)]$ onto the queue;
12:  end if
13: end while
14: return TRUE;
```

When $\mathfrak{A}$ and $\mathfrak{B}$ overlap, the pointwise penetration depth is defined as the point of deepest interpenetration of $\mathfrak{A}$ and $\mathfrak{B}$. Formally, the pointwise penetration depth (or PD for short) can be defined as:

$$PD \equiv \mathscr{H}\left(\mathfrak{A} \cap \partial(\mathfrak{A} \cap \mathfrak{B}), \mathfrak{B} \cap \partial(\mathfrak{A} \cap \mathfrak{B})\right) \tag{8}$$

where $\mathscr{H}(\cdot,\cdot)$ denotes the two-sided Hausdorff distance operator between surfaces.

5.2 Boolean Version of CCQ_p

We first explain how to evaluate the CCQ_p predicate in Eq.3. The main idea of our evaluation algorithm is to decompose the advancement step size Δt into two sub-steps Δt_s and Δt_p (*i.e.* $\Delta t = \Delta t_s + \Delta t_p$) such that collision-free motion is generated during Δt_s while Δt_p may induce penetration with the PD value being less than ε. Then, we perform dual CAs from the end-configurations $\mathbf{q}_0, \mathbf{q}_1$ like CCQ_s in Sec. 4.3.

Since Δt_s can be calculated just like in Eq. 5, computing Δt boils down to calculating Δt_p. In general, computing Δt_p can be quite challenging since one needs to search the entire C-space (both C-free and C-obstacle) where the placement of $\mathfrak{A}$ at $\mathbf{q}(t+\Delta t)$ may yield either collision-free or in-collision configuration. Thus, to find a feasible solution for Δt_p, we propose a conservative method.

The key idea is that, after the advancement of $\Delta t_s + \Delta t_p$ time step, we imposes that the robot still remains collision-free at $\mathbf{q}(t+\Delta t_s+\Delta t_p)$. Taking advantage of this constraint, we first move the robot to $\mathfrak{A}(t+\Delta t_s)$, and then calculate Δt_p that can bound the motion of $\mathfrak{A}$ by less than 2ε so that the possible PD between $\mathfrak{A}$ and $\mathfrak{B}$ can be less than ε, as shown in Fig. 3.

More precisely, an upper bound of the time step size Δt_p can be computed by observing the fact that the robot should not travel by more than 2ε; otherwise, the penetration depth can be greater than 2ε. Thus, assuming that the robot and obstacles are both convex, we have:

$$\Delta t_p \leq \frac{2\varepsilon}{\mu_u} \tag{9}$$

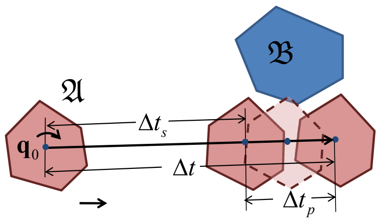


Figure 3: **Decomposition of the Time Step Δt into Δt_s and Δt_p for CCQ$_p$.** Δt_s corresponds to the collision-free time step and Δt_p to the time step that may have ε-penetration.

where μ_u is the maximum amount of motion that a point on $\mathfrak{A}$ can make between the time interval of $[0,1]$. Note that μ_u is an undirected motion bound unlike the directed one μ in Eq.5, since no closest direction will be defined for a robot in collision with obstacles. Essentially, μ_u depends on the underlying path. We present simple formulas to compute μ_u for both linear (Eq.10) and screw (Eq.11) motions as shown below. Here, $\mathbf{p}, \mathbf{p}^b, \mathbf{p}_\perp, \mathbf{o}^b$ have the same meanings as defined in Sec.4.2.

Linear Motion

$$\begin{aligned}\mu_u &= \max_{\mathbf{p}\in\mathfrak{A}}\left(\int_0^1 \|\dot{\mathbf{p}}_i(t)\|\,dt\right)\\ &= \max_{\mathbf{p}\in\mathfrak{A}}\left(\int_0^1 \left\|\mathbf{v}+\omega\times\mathbf{p}^b(t)\right\|dt\right)\\ &\le \|\mathbf{v}\| + \max_{\mathbf{p}\in\mathfrak{A}}\left(\int_0^1 \left\|\omega\times\mathbf{p}^b(t)\right\|dt\right)\\ &\le \|\mathbf{v}\| + \|\omega\|\max_{\mathbf{p}\in\mathfrak{A}}\left\|\mathbf{p}^b\right\|\end{aligned} \tag{10}$$

Screw Motion

$$\begin{aligned}\mu_u &= \max_{\mathbf{p}\in\mathfrak{A}}\left(\int_0^1 \|\dot{\mathbf{p}}_i(t)\|\,dt\right)\\ &= \max_{\mathbf{p}\in\mathfrak{A}}\left(\int_0^1 \|\mathbf{v}+\omega\times\mathbf{p}_\perp(t)\|\,dt\right)\\ &\le \|\mathbf{v}\| + \max_{\mathbf{p}\in\mathfrak{A}}\left(\int_0^1 \|\omega\times\mathbf{p}_\perp(t)\|\,dt\right)\\ &\le \|\mathbf{v}\| + \|\omega\|\left(\left\|\left(\mathbf{o}^b-\mathbf{a}\right)\times\omega\right\| + \max_{\mathbf{p}\in\mathfrak{A}}\left\|\mathbf{p}^b\right\|\right)\end{aligned} \tag{11}$$

The result of our algorithm is conservative in the sense that our algorithm does not report a false-positive result; *i.e.* if the algorithm reports `TRUE`, it guarantees that CCQ$_p$ is indeed `TRUE`.

5.3 Time of Violation Query for CCQ$_p$

A simple way to compute the ToV in Eq.4 can be devised similarly to evaluating CCQ$_p$ by decomposing the ToV into the one corresponding to collision-free motion τ_s (Eq.2) and one to ε-penetration $\Delta t'_p$: *i.e.*

$$\begin{aligned}\tau_{p1} &= \left(\sum_i \Delta t_s^i\right) + \Delta t'_p\\ &= \tau_s + \Delta t'_p.\end{aligned} \tag{12}$$

Moreover, in order to guarantee ε-penetration, $\Delta t'_p$ is calculated such that the motion of $\mathfrak{A}$ starting at $t = \tau_s$ should be bounded above by ε:

$$\Delta t'_p \leq \frac{\varepsilon}{\mu_u}. \tag{13}$$

Here, the undirected motion bound μ_u can be calculated similarly as in the previous section. However, there are two issues related to computing the ToV, as shown in Eq.12:

- τ_{p1} provides a lower bound of the ToV with ε-penetration, but this may be a loose bound since ε is typically much smaller than μ_u.
- The placement of the robot at $\mathfrak{A}(\tau_{p1})$ may correspond to an in-collision sample. This can be problematic for most sampling-based planners where only collision-free samples are permitted to represent the connectivity of the free C-space.

Note that the second issue is more severe than the first one in practice. We introduce an alternative way to compute τ_p to circumvent the above issues.

The main idea is that, instead of accumulating the collision-free time steps first (*i.e.* τ_s), we intertwine collision-free and in-collision motions for every time step, just like the Boolean query in the previous section. Thus, the new ToV τ_{p2} is:

$$\tau_{p2} = \sum_i \left(\Delta t_s^i + \Delta t_p^i\right). \tag{14}$$

Here, $\Delta t_s^i, \Delta t_p^i$ for the ith iteration are calculated using Eq. 5 and Eq. 9, respectively. The above iteration continues until the ith iteration yields a collision. Thus, by construction, $\mathfrak{A}(\tau_{p2})$ is collision-free. Moreover, in general, $\tau_{p1} \leq \tau_{p2}$; however this is not always true but less likely to happen in practice since Eq. 14 continues to iterate until collision is found unlike Eq. 12, as illustrated in Fig.4.

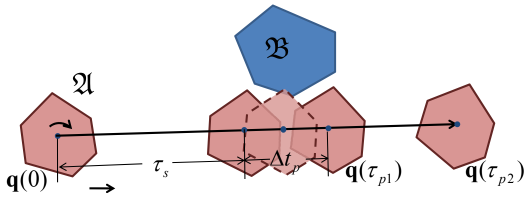


Figure 4: **Comparison between τ_{p1} and τ_{p2}.** In general, $\tau_{p2} > \tau_{p1}$ since more iterations will be performed for τ_{p2} until collision is found at $\mathbf{q}(\tau_{p2})$.

6 Extension to Articulated Robots

Our CCQ algorithms for rigid robots can be extended to articulated robots. The basic equations that support CCQ algorithms such as Eqs. 6 or 12 can be reused as long as the directed and undirected motion bounds μ, μ_u can be calculated. However, this turns out to be relatively straightforward. For instance, in case of linear motion, the directed motion bound μ for an articulated robot can be obtained using the same motion bound presented by Zhang *et al.* [6]. Moreover, the spatial and temporal culling techniques proposed in the paper to accelerate the query performance are also reusable for CCQ query between articulated models.

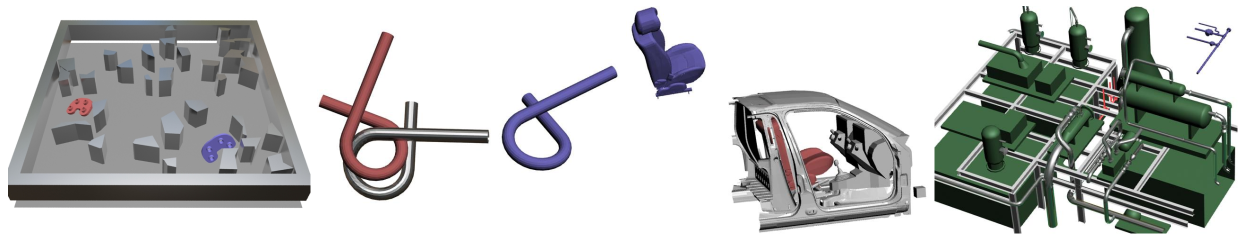
7 Results and Discussion

In this section, we describe the implementation results of our CCQ algorithms, and benchmark the performance of the algorithms by plugging them into well-known, sampling-based planners. Finally, we compare our algorithm against prior exact local planning techniques.

7.1 Implementation Details

We have implemented our CCQ algorithm using C++ on a PC running Windows Vista, equipped with Intel Dual CPU 2.40GHz and 5GB main memory. We have extended public-domain collision libraries such as PQP [27] and C^{2}A. Note that these collision libraries are designed only for static proximity computation or ToV computation (similar to τ_s) under a linear motion. Throughout the experiments reported in the paper, we set the penetration threshold ε for CCQ$_p$ and τ_p as one tenth of the radius of the smallest enclosing sphere of $\mathfrak{A}$.

To measure the performance of our algorithms, we have used the benchmarking models and planning scenarios as shown in Table 2 and Fig.5 with sampling-based motion planners including PRM and RRT. These benchmarking models consist of 1K $\sim$ 30K triangles, and the test scenarios have narrow passages for the solution path. Typical query time for our CCQ algorithms takes a few milli-seconds; for instance, the most complicated benchmark, the car seat, takes 21.2 msec and 28.3 msec for ToV and Boolean queries, respectively.



(a) Maze (b) Alpha Puzzle (c) Car Seat (d) Pipe

Figure 5: **Benchmarking Scenes.** For each benchmark scene, the starting and goal configurations of the robot are colored in red and blue, respectively.

Benchmarks	$\mathfrak{A}$	$\mathfrak{B}$	# of tri ($\mathfrak{A}$)	# of tri ($\mathfrak{B}$)
Maze	CAD piece	Maze	2572	922
Alpha Puzzle	Alpha	Alpha	1008	1008
Car seat	Seat	Car Body	15197	30790
Pipe	Pipe	Machinery	10352	38146

Table 2: Benchmarking Model Complexities.

7.2 Probabilistic Roadmap with CCQ

In Sec.3.2, we have explained the basic steps of sampling-based planners. These planners use a different local planning step (the step 2 in Sec. 3.2). In case of PRM, the local planning step corresponds to checking whether two collision-free samples can be connected by some collision-free path $\mathbf{q}(t)$ or not (*i.e.* Boolean query) [1].

In conventional PRM-based planners, this Boolean checking is implemented by performing fixed-resolution collision detection along the path, namely fixed-resolution local planning (DCD). In Table 3, we show the performance of PRM with DCD with varying resolution parameters and a linear path. Here, the resolution parameter means the average number of collision checks performed for each local path. We have used the OOPSMP implementation of PRM, and only the maze and pipe benchmarks was solvable by OOPSMP within a reasonable amount of time. The optimal performance is obtained when the resolution is 23, and as the resolution parameter becomes less than 23, the OOPSMP may not be able to compute a collision-free path. In any case, the DCD local planner still does not guarantee the correctness of the path in terms of collision-free motion.

Avg. Collision Resolution	23	40	47	80	128
PRM with DCD (Boolean)	12.70s	15.88s	18.76s	39.49s	44.75s

Table 3: The performance of PRM in seconds based on fixed-resolution local planning (DCD) with different resolutions for the maze benchmark.

However, exact local planning is made possible by running the Boolean version of our CCQ algorithm on the path. In Table 4, we highlight the performance of CCQ-based local planning algorithms (CCQ$_s$ and CCQ$_p$) with PRM, and compare it against that of the DCD local planning method with the optimal resolution parameter. In case of the pipe benchmark, the PRM performance using our algorithm is similar to that of the DCD. In case of the maze benchmark, our CCQ-based local planner is about 1.8 times slower than DCD local planner. Even for this benchmark, when the resolution parameter becomes higher than 80, our CCQ algorithm performs faster than DCD, even though the DCD local planner still cannot guarantee the correctness of the solution path. Also notice that CCQ$_p$ takes less time than CCQ$_s$ since the former is a less restrictive query than the latter.

Benchmark	DCD	Boolean Query	
		CCQ$_s$	CCQ$_p$
Maze	12.70s	36.34s	24.09s
Pipe	8425.09s	9610.13s	8535.60s

Table 4: The performance of PRM using DCD local planner and CCQ-based local planner. The CCQ-based local planner can guarantee collision-free motion while the other cannot give such guarantees.

7.3 Rapidly-Exploring Random Tree with CCQ

In RRT, the local planning step is, after sampling a new collision-free configuration, used to extend from the closest node in the search graph to the new node [2]. In this case, both ToV and Boolean CCQ can be employed to implement exact local planning. Specifically, when the new node is to be extended along some path, if the path is not collision-free, the path can be entirely abandoned (Boolean query) or the partial collision-free segment of the path before the ToV can be still kept (ToV query).

In Fig. 6, we show the performance of RRT planner with our CCQ algorithms and DCD local planner with the optimal resolution parameter. Also, different types of motion paths such as linear and screw motion have been tested. We also have used the OOPSMP implementation of RRT for this experiment. To find the optimal resolution parameter for DCD local planner, we test different resolution parameters ranging between $[3,15]$; for instance, see Table 5 for the alpha puzzle benchmark using the ToV query based on DCD local planner. Similar to PRM, the variation in performance depends on the resolution parameter, but it does not show the linear relationship between the resolution and performance unlike PRM since computing more accurate ToV using a higher resolution requires a lot more collision checks. Thus, picking a right value for the resolution parameter is even more difficult in case of RRT.

In our benchmarks, the RRT with CCQ-based local planner is roughly two times slower than the one with DCD local planner with the optimal resolution. However, in some cases such as the Maze (BS), Alpha puzzle (BL) and pipe (BL) benchmarks in Fig.6, the RRT with CCQ-based local planner is even faster than the one with DCD local planner since the number of collision checks can be kept minimal.

Avg. Collision Resolution	4.21	5.96	6.01	6.97
RRT with DCD (ToV)	25.60s	0.25s	2.08s	39.65s

Table 5: The performance of RRT in seconds based on fixed-resolution local planning (DCD) with different resolutions for the alpha puzzle benchmark. In this case, RRT uses the ToV query. When the resolution is less than 4, RRT cannot find a path

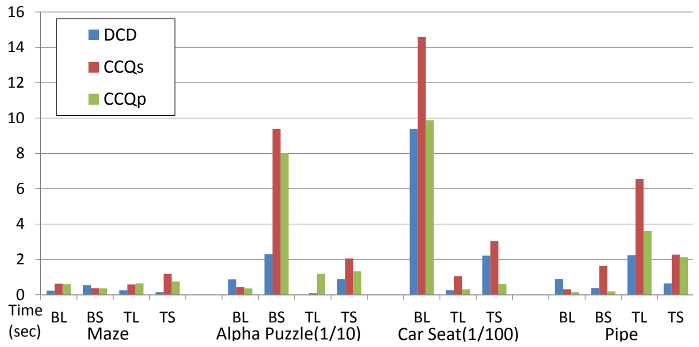


Figure 6: **The Performance of RRT using DCD and CCQ-based Local Planner.** The x-axis represents different benchmarking scenes with different queries such as BL (Boolean query with a linear motion), BS (Boolean query with a screw motion), TL (ToV query with a linear motion), and TS (ToV query with a screw motion) for each benchmark. The y-axis denotes the planning time in seconds for the maze and pipe benchmark, in tens of seconds for the alpha puzzle, and in hundreds of seconds for the car seat. The blue, red and green bars denote the planning time using DCD, CCQ$_s$-based, and CCQ$_p$-based local planners, respectively.

7.4 Comparisons with Prior Approaches

We also compare the performance of our CCQ-based local planning algorithm with the prior exact local planning algorithms such as the dynamic collision checking method (DCC) [7] implemented in MPK. To the best of our knowledge, the dynamic collision checking algorithm is the only public-domain exact local planner that has been integrated into sampling-based motion planner.

Since DCC supports only a Boolean query with a linear motion and separation constraints, we compare the performance of the Boolean version of our CCQ$_s$ against DCC by plugging CCQ$_s$ into the MPK planner, as shown in Table 6. For benchmarks, we use the same pipe model in Fig.5-(d), but shrink the robot a bit to enable MPK to find a solution path. We also use another benchmark model as shown in Fig. 7, the alpha-shape with two holes. In this case, we plan a path for an alpha-shape tunnelling through two holes, and measure the average performance of DCC and CCQ$_s$-based local planner.

Benchmarks	# of triangles	CCQ$_s$	DCC
Pipe	48K	0.24s	2.28s
Alpha-shape with two Holes	1K	14.90s	91.24s

Table 6: Performance Comparisons between dynamic collision checking (DCC) and CCQ$_s$-based local planner. The timings are the total collision checking time in seconds used for local planning.

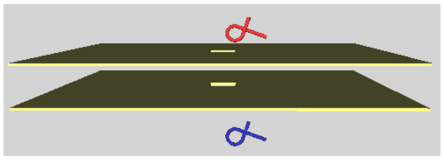


Figure 7: **Alpha-Shape Through Two Holes.** The red and blue alpha shapes represent the starting and goal configurations, respectively.

In our experiment, CCQ$_s$-based local planner is about an order of magnitude faster than DCC local planner mainly because CCQ uses a tighter, directional motion bound than DCC relying on undirectional motion bound. A similar explanation was also provided in [28] why the directional bound is superior to the undirectional one. Another reason is because of the dual advancement mechanism in CCQ-based local planner.

The ToV version of our CCQ$_s$ algorithm has a similar objective as continuous collision detection algorithms. Since our algorithm is based on the known fastest CCD algorithm C^{2}A [9], it shows a similar performance of that of C^{2}A. However, C^{2}A is not optimized for a Boolean query and does not support CCQ with penetration constraints. Ferre and Laumond's work [19] supports a penetration query, but their work is essentially similar to DCC [7].

8 Conclusion

We have presented a novel proximity query, CCQ, with separation and penetration constraints. It can be used for efficient and exact local planning in sampling-based planner. In practice, we have shown that the CCQ-based local planner is only two times slower or sometimes even faster than the fixed-resolution local planner. Moreover, CCQ-based local planners outperform the state-of-the-art exact local planners by almost an order of magnitude.

There are a few limitations in our CCQ algorithm. Both CCQ$_s$ and CCQ$_p$ algorithms are sensitive to threshold values; *e.g.* the termination condition threshold for CA or CCQ$_s$ and penetration threshold ε for CCQ$_p$. The motion bound calculation such as μ or μ_u depends on the underlying path.

For future work, it may be possible for a planner to try different types of paths and automatically choose the suitable or optimal one. We would like to extend our CCQ framework to deformable robots. We are also interested in applying our CCQ technique to other applications such as dynamics simulation where the ToV computation is required. In particular, the use of CCQ$_p$ may also provide a direction for contact dynamics where slight penetration is allowed (*e.g.* penalty-based method).

References

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics & Automation*, vol. 12, no. 4, pp. 566–580,

June 1996.

[2] S. M. LaValle and J. J. Kuffner, “Rapidly-exploring random trees: Progress and prospects,” in *Proceedings Workshop on the Algorithmic Foundations of Robotics*, 2000.

[3] P. Isto, “Constructing probabilistic roadmaps with powerful local planning and path optimization,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2323–2328.

[4] N. Amato, O. Bayazit, C. Jones, and D. Vallejo, “Choosing good distance metrics and local planners for probabilistic roadmap methods,” *IEEE Transactions on Robotics and Automation*, vol. 16, no. 4, pp. 442–447, 2000.

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

[6] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, “Continuous collision detection for articulated models using taylor models and temporal culling,” *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, vol. 26, no. 3, p. 15, 2007.

[7] F. Schwarzer, M. Saha, and J.-C. Latombe, “Adaptive dynamic collision checking for single and multiple articulated robots in complex environments,” *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 338–353, 2005.

[8] J. F. Canny, “Collision detection for moving polyhedra,” *IEEE Trans. PAMI*, vol. 8, pp. 200–209, 1986.

[9] M. Tang, Y. J. Kim, and D. Manocha, “C^2A: Controlled conservative advancement for continuous collision detection of polygonal models,” *Proc. of IEEE Conference on Robotics and Automation*, 2009.

[10] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, “Fast continuous collision detection for articulated models,” in *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004.

[11] J. J. Kuffner, “Effective sampling and distance metrics for 3D rigid body path planning,” in *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[12] S. Redon and M. Lin, “Practical local planning in the contact space,” in *Proceedings IEEE International Conference on Robotics & Automation*, 2005, pp. 4200– 4205.

[13] X. Ji and J. Xiao, “Planning motion compliant to complex contact states,” *International Journal of Robotics Research*, vol. 20, no. 6, pp. 446–465, 2001.

[14] M. Hofer, H. Pottmann, and B. Ravani, “Geometric design of motions constrained by a contacting surface pair,” *Comput. Aided Geom. Des.*, vol. 20, no. 8-9, pp. 523–547, 2003.

[15] M. Zefran and V. Kumar, “A variational calculus framework for motion planning,” in *Proceedings IEEE International Conference on Robotics & Automation*, 1997, pp. 415– 420.

[16] L. Zhang and D. Manocha, “Constrained motion interpolation with distance constraints,” in *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008, pp. 269–284.

[17] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[18] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[19] E. Ferre and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," in *Proceedings IEEE International Conference on Robotics & Automation*, 2004, pp. 3149– 3154.

[20] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," 2003, pp. 403–417.

[21] R. Bohlin and L. Kavraki, "Path planning using Lazy PRM," in *Proceedings IEEE International Conference on Robotics & Automation*, 2000.

[22] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 1998.

[23] D. Hsu, "Randomized single-query motion planning in expansive spaces," Ph.D. dissertation, 2000.

[24] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 3743–3750.

[25] B. V. Mirtich, "Impulse-based dynamic simulation of rigid body systems," Ph.D. dissertation, University of California, Berkeley, 1996.

[26] J. R. Rossignac and J. J. Kim, "Computing and visualizing pose-interpolating 3d motions," *Computer-Aided Design*, vol. 33, no. 4, pp. 279–291, 2001.

[27] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Department of Computer Science, University of North Carolina, Tech. Rep. TR99-018, 1999.

[28] X. Zhang, M. Lee, and Y. J. Kim, "Interactive continuous collision detection for non-convex polyhedra," *The Visual Computer*, pp. 749–760, 2006.