Exact collision detection along paths: Optimization and proof of convergence

Diane Bury, Joseph Mirabel, Florent Lamiraux, Marc Gouttefarde, Pierre-Elie Hervé

Exact collision detection along paths: Optimization and proof of convergence

Diane Bury, Joseph Mirabel, Florent Lamiraux, Marc Gouttefarde, and Pierre-Elie Hervé

***Abstract*—This paper presents an exact collision detection algorithm capable of determining the presence or absence of collisions along a path. For every pair of rigid bodies, using a known upper bound on their relative velocity along the path, if lower bounds on the distance between them can be computed at certain parameters along the path, portions of the path can then be validated. The algorithm proceeds by dichotomy until the whole path is validated or a collision is found. A proof of convergence of the algorithm is proposed, guaranteeing that any path can be validated or invalidated in finite time, except for one singular case. Three changes to this algorithm are then proposed, and are shown through experimental validation to reduce significantly the number of iterations needed to validate a path. The algorithm is also shown to perform better than a discretized collision detection method in terms of computation times, while missing no collision. Moreover, the method can be extended to take into accounts other types of constraints than those related to collision avoidance, and can be applied to various types of robots.**

***Index Terms*—Collision detection, continuous collision checking, motion planning**

I. Introduction

EXACT collision detection along paths is an important issue in robotics as well as in physics simulation [1]. The problem consists in determining whether along a given continuous path, a collision occurs between two links of the robot, or between a link of the robot and a static obstacle of the environment. Before actually executing a motion on a robot, this test is obviously critical. The word *exact* is used in opposition to *discretized* collision checking where only sample configurations along the path are checked for collision. Exact collision detection is an essential component of any motion planning algorithm, and especially algorithms derived from PRM (Probabilistic Roadmap) [2], [3], [4] or RRT (rapidly exploring random trees) [5], [3], [6]. In [7], the authors use a combination of discretized and exact collision checking. These algorithms develop graphs called *roadmap* in the free configuration space of the robot. Nodes are collision-free configurations and edges are collision-free paths. Moreover, discretized collision checking may validate a path and later on find a collision on a portion of this path simply because different sample configurations have been tested. This may make some iterative path optimization method like [8] fail since the latter method validates several times the same portion of a path that is iteratively shortened.

In the early 2000, Schwarzer et al. [9] proposed a method based on bisection that validates intervals for each pair of objects possibly colliding. Either the whole path interval is validated for each pair, or a collision is detected between a pair of objects somewhere along the path. Since then this algorithm has been widely used. The method is based on the computation of upper bounds of the velocities of the points of a rigid-body object in the reference frame of another object. If at a given time along the path, the two objects are at a positive distance to each other, this upper bound trivially defines a collision-free interval of time centered on the time where the positive distance has been computed. Intensively using our implementation of this algorithm [10], we noticed that for some pairs of objects, the computed intervals are very small although the distance between them is rather large. Investigating the underlying issue, it turns out that the used upper bound on the maximum of the velocities of the points of an object $\mathcal{A}$ in the reference frame of an object $\mathcal{B}$ is usually different from the upper bound on the maximum of the velocities of the points of object $\mathcal{B}$ in the reference frame of $\mathcal{A}$. We propose a change in the algorithm to overcome this issue.

The rest of the paper is organized as follows. Section II presents the related works. Section III describes the initial algorithm as proposed in [9]. Section V presents the improvements we propose to make the method more efficient in some common situations. Section IV introduces a convergence analysis of the method. Section VI provides some experimental results that show the importance of the proposed optimizations. Section VII concludes the paper.

II. Related works

Continuous collision detection (CDD) for articulated robots and freeflying objects has generated a lot of attention in the 2000s.

In [11], an algorithm to compute the time of collision of convex objects moving in a dynamic simulation setting is proposed. The motion of the objects are determined by physics laws. Almost at the same time as [9], [12] proposed a method to detect collisions between rigid objects moving along arbitrary discretized motions. As in [9], the motion between sample configurations is assumed to be a linear interpolation. Unlike in [9] however, interval arithmetic is used to compute collision times by bisection. They also speed up static collision tests by using a hierarchy of object oriented bounding boxes.

Diane Bury, Joseph Mirabel and Florent Lamiraux are with LAAS-CNRS, Université de Toulouse

Marc Gouttefarede is with LIRMM, Univ Montpellier, CNRS, Montpellier, France.

Pierre-Elie Hervé is with TECNALIA, Basque Research and Technology Alliance (BRTA)

Although the method is applied to pairs of rigid objects, it can be easily extended to kinematic chains. Recently, [13] refined the method of [9] in the specific case of spherical and revolute joints. In [14], Pan *et al* exploit a similar idea that they call *conservative advancement*.

[15] proposed a method of continuous collision detection for kinematic chains based on interval analysis and swept volume computation. This latter method however does not take into account collisions between the links of the kinematic chain. [16] improves [12] by replacing interval analysis by Taylor expansion where the rest is an interval. This method provides much better approximations of sine and cosine functions and therefore increases the bisection efficiency. Interval analysis is also used for CCD, in [17] and [18] for parallel robots, and in [19] for closed-loop robots.

[20] introduces FCL (Flexible Collision Library), a library which performs proximity queries based on hierarchical representations. Its flexible architecture and performance make FCL one of the most used libraries for collision detection. [21] applies continuous collision checking to the problem of trajectory smoothing by iteratively replacing random portions of an initial piecewise affine trajectory by cubic B splines. They extend continuous collision detection to spline motions, which is also implented in FCL. [22] proposed to apply linear transformations in the 3D space to optimize exact collision checking between rigid objects moving along a screw motion.

Recently, [23] proposed a differential cost for continuous collision avoidance that can be used with optimization-based trajectory planners. This cost is based on a harmonic potential field and uses [12] for CCD. Considering CDD for multiple robots, [24] tackles the problem of velocity uncertain motions, for which disjoints paths are required to consider the motion safe. Using a Lipschitz constant, they determine if two robot paths are disjoints, i.e. if their respective swept volumes are disjoints. Like the present paper, [24] is inspired by Conservative Advancement [25].

With respect to these previous works, the contribution of the present paper is twofold:

1) A simple yet important improvement of the initial CDD algorithm from [9] is proposed. It consists in choosing for each body pair the best upper bound for the relative velocity of the two bodies. Two other changes to the algorithm are also proposed.
2) The convergence of the algorithm is proved. Such a proof is not provided in [9].

III. Initial method

A. Algorithm

This section describes the basic CCD algorithm as proposed in [9]. Our formulation is slightly different – though equivalent – from this previous paper. In the rest of this paper, we consider a robot (or several robots) as a rigid-body system. The configuration space $\mathcal{CS}$ is the Cartesian product of the configurations space of each robot and object, which means that the robots and objects are considered as one single kinematic tree. A path is a function from an interval $[0, T] \in \mathbb{R}$ to the configuration space $\mathcal{CS}$, and describes the movement of a robot in the environment. T is the length (or duration) of the path. A collision occurs for a configuration $\mathbf{q} \in \mathcal{CS}$ when the distance between two bodies (among the bodies of the robots and the obstacles) is non-positive.

Algorithm 1 Validation of a path using the dichotomy method

```
1:  function ValidatePath(p)
2:      T ← Length(p)
3:      t ← T/2
4:      Δt ← +∞
5:      for each pair of bodies (a, b) do
6:          d ← ComputeDistance(a, b, t)
7:          V^max ← GetMaxRelativeVelocity(a, b)
8:          Δt_(a,b) ← d / V^max
9:          Δt ← min(Δt, Δt_(a,b))
10:     end for
11:     if Δt = 0 then
12:         return False
13:     else if Δt > T/2 then
14:         return True
15:     else
16:         p1_valid ← ValidatePath(p([0, T/2 − Δt]))
17:         p2_valid ← ValidatePath(p([T/2 + Δt, T]))
18:         return p1_valid ∧ p2_valid
19:     end if
20: end function
```

The presented method can determine if a path is continuously valid — i.e. there is no collision along the path — or if there is a collision somewhere on the path for a pair of bodies. This algorithm is exact and always finds a collision if one exists.

The algorithm is based on the following two requirements.

- For each joint of the robot, the norms of the joint linear and angular velocities must have known upper bounds. The latter are used to compute, for each pair of bodies, an upper bound on their relative velocity along the path — this corresponds to the function `GetMaxRelativeVelocity` in Algorithm 1.
- For each pair of bodies, at any configuration along the path, a lower bound on the distance between them can be computed. This corresponds to the function `ComputeDistance` in the algorithm. This function must return zero if and only if there is a collision between the two bodies.

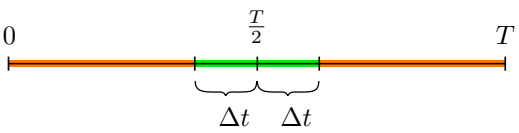


Fig. 1: Testing an interval of length T

The algorithm, detailed as pseudo-code in Algorithm 1, is dichotomous and recursive. To validate a path, it recursively validates an interval centered on the middle of the path for every pair of bodies. This is done by computing the distance between the two bodies at the time parameter $T/2$ (line 6

of Algorithm 1), and by dividing this distance by the known upper bound on their relative velocity (line 8 of Algorithm 1). For each pair of bodies (a, b), a value $\Delta t_{(a,b)}$ is thereby computed in such a way that the interval $[\frac{T}{2} - \Delta t_{(a,b)}, \frac{T}{2} + \Delta t_{(a,b)}]$ is guaranteed to be collision-free for this pair. If $\Delta t_{(a,b)} = 0$, then there is a collision for the pair (a, b) and the algorithm stops. Otherwise, Δt is obtained by taking the minimum of all the $\Delta t_{(a,b)}$ (line 9) and the interval $[\frac{T}{2} - \Delta t, \frac{T}{2} + \Delta t]$ is validated for all the body pairs. This means that the path to validate has been separated into three intervals (Figure 1): A validated interval centered on $\frac{T}{2}$ of length $2\Delta t$ and two untested intervals on either side, of equal lengths $\frac{T}{2} - \Delta t$. The algorithm is then called in a recursive manner on the two untested intervals (lines 16-17), until either a collision is found for one pair of bodies, or the whole path is validated. Section IV provides a convergence analysis of the algorithm.

B. Computation of V^{max}

Let us consider two links of the kinematic tree to test against collisions, $\mathcal{B}_a$ and $\mathcal{B}_b$. Together, they form the pair i. We want to compute $V_i^{\max}$, an upper bound on the relative velocity between $\mathcal{B}_a$ and $\mathcal{B}_b$ along the path. There exists a sequence of bodies, and a sequence of corresponding links, linking $\mathcal{B}_a$ to $\mathcal{B}_b$. We note $\mathcal{B}_0 = \mathcal{B}_a, \mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, \mathcal{B}_k = \mathcal{B}_b$ with $k \geq 1$ the successives bodies linking $\mathcal{B}_a$ and $\mathcal{B}_b$. J_0 is the frame fixed to $\mathcal{B}_a$, and for $1 \geq j \geq k$, we note J_j the frame linked to body B_j placed at the joint between bodies B_{j-1} and B_j, as (as shown in Fig. 2). We note $J_k = J_b$.

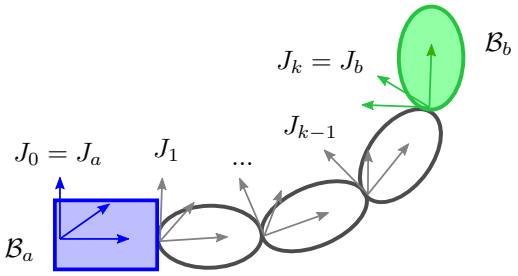


Fig. 2: Representation of the kinematic chain between two bodies

As stated in Section III-A, we suppose that for each joint n of the robot, an upper bound on the norm of the joint velocity is known. For each $n \in [1, k]$, we denote by

- ${}^n\mathbf{v}_{n-1}$ the linear velocity of the origin of J_{n-1} with respect to J_n, and ${}^n v_{n-1}$ an upper bound on $\|{}^n\mathbf{v}_{n-1}\|$,
- ${}^n\boldsymbol{\omega}_{n-1}$ the angular velocity of J_{n-1} with respect to J_n, and ${}^n\omega_{n-1}$ an upper bound on $\|{}^n\boldsymbol{\omega}_{n-1}\|$.

Let P_0 be a point of $\mathcal{B}_a$, which means P_0 is a fixed point in the frame $J_a = J_0$. The relation between the coordinate vector $\mathbf{m}_0$ of P_0 in J_0 and its coordinate vector ${}^k\mathbf{P}_0$ in J_k is given by

$$\begin{pmatrix} {}^k\mathbf{P}_0 \\ 1 \end{pmatrix} = {}^kM_{k-1}\,{}^{k-1}M_{k-2} \ \ldots \ {}^2M_1\,{}^1M_0 \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \quad (1)$$

where ${}^{n+1}M_n = \begin{pmatrix} {}^{n+1}R_n & {}^{n+1}\mathbf{T}_n \\ 0\ 0\ 0 & 1 \end{pmatrix}$ is the homogeneous matrix representing the position and orientation of J_n with respect to J_{n+1}, ${}^{n+1}R_n \in SO(3)$ is a rotation matrix and ${}^{n+1}\mathbf{T}_n \in \mathbb{R}^3$ is a translation vector.

${}^k\mathbf{P}_0$ is the coordinate vector of a point of body $\mathcal{B}_a$ in the frame of body $\mathcal{B}_b$, since the frame J_k is fixed to $\mathcal{B}_b$. To compute $V^{\max}$, we must find an upper bound on the norm of the time derivative of ${}^k\mathbf{P}_0$. By differentiating (1) with respect to time, we get:

$$\begin{aligned} \begin{pmatrix} {}^k\dot{\mathbf{P}}_0 \\ 0 \end{pmatrix} &= \begin{pmatrix} [{}^k\boldsymbol{\omega}_{k-1}]_\times\,{}^kR_{k-1} & {}^k\mathbf{v}_{k-1} \\ 0\ 0\ 0 & 0 \end{pmatrix} \cdots \\ &\quad \cdots {}^1M_0 \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \\ &+ {}^kM_{k-1} \begin{pmatrix} [{}^{k-1}\boldsymbol{\omega}_{k-2}]_\times\,{}^{k-1}R_{k-2} & {}^{k-1}\mathbf{v}_{k-2} \\ 0\ 0\ 0 & 0 \end{pmatrix} \cdots \\ &\quad \cdots {}^1M_0 \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \\ &+ \cdots \\ &+ {}^kM_{k-1} \cdots {}^2M_1 \begin{pmatrix} [{}^1\boldsymbol{\omega}_0]_\times\,{}^1R_0 & {}^1\mathbf{v}_0 \\ 0\ 0\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \end{aligned} \quad (2)$$

where $[\mathbf{x}]_\times$ is the 3x3 antisymmetric matrix representing the cross product with vector $\mathbf{x} \in \mathbb{R}^3$.

To bound the norm of ${}^k\dot{\mathbf{P}}_0$, we must first recall some properties of rigid-body transformations.

Lemma 1. *Consider the homogeneous matrices* $M_1 = \begin{pmatrix} R_1 & \mathbf{T}_1 \\ 0\ 0\ 0 & 1 \end{pmatrix}$, $M_2 = \begin{pmatrix} R_2 & \mathbf{T}_2 \\ 0\ 0\ 0 & 1 \end{pmatrix}$ *and* $M_3 = \begin{pmatrix} R_3 & \mathbf{T}_3 \\ 0\ 0\ 0 & 1 \end{pmatrix}$ *so that* $M_3 = M_1M_2$. *Then:*

$$\|\mathbf{T}_3\| \leq \|\mathbf{T}_1\| + \|\mathbf{T}_2\| \quad (3)$$

Proof. If $M_3 = M_1M_2$, then:

$$M_3 = \begin{pmatrix} R_1R_2 & R_1\mathbf{T}_2 + \mathbf{T}_1 \\ 0\ 0\ 0 & 1 \end{pmatrix}$$

This means $\mathbf{T}_3 = R_1\mathbf{T}_2 + \mathbf{T}_1$. Using the triangle inequality,

$$\begin{aligned} \|\mathbf{T}_3\| &\leq \|R_1\mathbf{T}_2\| + \|\mathbf{T}_1\| \\ &\leq \|\mathbf{T}_2\| + \|\mathbf{T}_1\| \end{aligned}$$

since R_1 is a rotation matrix. □

Lemma 2. *Consider the homogeneous matrix* $M_1 = \begin{pmatrix} R_1 & \mathbf{T}_1 \\ 0\ 0\ 0 & 1 \end{pmatrix}$, $\mathbf{m} \in \mathbb{R}^3$, *and* $\mathbf{p} \in \mathbb{R}^3$ *so that* $\begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = M_1 \begin{pmatrix} \mathbf{m} \\ 1 \end{pmatrix}$. *Then:*

$$\|\mathbf{p}\| \leq \|\mathbf{m}\| + \|\mathbf{T}_1\| \quad (4)$$

Proof.
$\begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = M_1 \begin{pmatrix} \mathbf{m} \\ 1 \end{pmatrix} = \begin{pmatrix} R_1\mathbf{m} + \mathbf{T}_1 \\ 1 \end{pmatrix} \Rightarrow p = R_1\mathbf{m} + \mathbf{T}_1$.
Using the triangle inequality and since R_1 is a rotation matrix, we obtain the property: $\|\mathbf{p}\| \leq \|\mathbf{m}\| + \|\mathbf{T}_1\|$. □

Using the lemmas 1 and 2 with (2), we obtain:

$$\begin{aligned}\|^k\dot{\mathbf{P}}_0\| \leq\ & \|^1\mathbf{v}_0\| + \|^1\boldsymbol{\omega}_0\|\|\mathbf{m}_0\| \\ & + \|^2\mathbf{v}_1\| + \|^2\boldsymbol{\omega}_1\|(\,\|\mathbf{m}_0\| + \|^1\mathbf{T}_0\|\,) \\ & + \|^3\mathbf{v}_2\| + \|^3\boldsymbol{\omega}_2\|(\,\|\mathbf{m}_0\| + \|^1\mathbf{T}_0\| + \|^2\mathbf{T}_1\|\,) \\ & + \dots \\ & + \|^k\mathbf{v}_{k-1}\| + \|^k\boldsymbol{\omega}_{k-1}\|(\,\|\mathbf{m}_0\| \\ & \qquad + \|^1\mathbf{T}_0\| + \dots + \|^{k-1}\mathbf{T}_{k-2}\|\,)\end{aligned} \tag{5}$$

Let us define the radius r_0 of the link $\mathcal{B}_a$ as the maximal distance between the origin of frame J_a and all points of the link:

$$r_0 = \max_{\mathbf{m}_0 \in \mathcal{B}_a} \|\mathbf{m}_0\| \tag{6}$$

We denote by D_j the cumulative length of J_j:

$$\begin{aligned} D_0 &= 0 \\ D_j &= \sum_{t=0}^{j-1} \|^{t+1}\mathbf{T}_t\| \quad \text{for } j \geq 1 \end{aligned} \tag{7}$$

Using (6) and (7) in (5), we obtain an upper bound on the velocity of any point of $\mathcal{B}_a$ with respect to frame $J_k = J_b$ for pair i, which gives us $V_i^{\max}$:

$$\|^k\dot{\mathbf{P}}_0\| \leq \sum_{t=0}^{k-1} {}^{t+1}v_t + {}^{t+1}\omega_t(\,r_0 + D_t\,) = V_i^{\max} \tag{8}$$

The variables written in red — ${}^{t+1}v_t$ and ${}^{t+1}\omega_t$ — depend on the path to be validated. The variables written in blue — r_0 and D_k — depend on the kinematic chain of the robot.

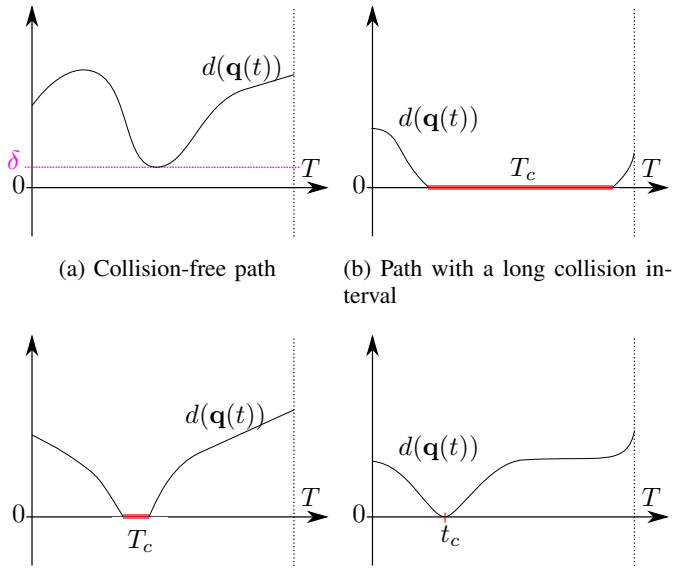
IV. Convergence Analysis

Let $\mathbf{p} : [0, T] \to \mathcal{CS}$ be a path of length T. For each pair of bodies i of the robot and the environment, we note $d_i : \mathcal{CS} \to \mathbb{R}$ the distance function between the two bodies. For a configuration $\mathbf{q} = \mathbf{p}(t)\ ,\ t \in [0, T]$, the bodies of the pair i are in collision if and only if $d_i(\mathbf{q}) = 0$. A configuration is valid if there is no collision for all the pair of bodies. We define two types of collisions:

1) A collision interval: For a given pair of bodies, there exists an interval T_c along the path so that $\forall t \in T_c$, $\mathbf{p}(t)$ is in collision,
2) a contact: For a given pair of bodies, there exists a time t_c such that the two bodies are in collision at configuration $p(t_c)$ but not in a neighborhoud of this configuration along the path, i.e., there exists $\delta_t > 0$ such that the only collision in the interval $[t_c - \delta_t, t_c + \delta_t]$ is at t_c.

We want to prove the convergence of the dichotomous collision detection algorithm introduced in Section III. There are three different possible cases, as shown in Fig. 3:

A. Collision-free: There is no collision between any pair of bodies along the path, which means the path is collision-free (Section IV-A, Fig. 3a),



(a) Collision-free path

(b) Path with a long collision interval

(c) Path with a short collision interval

(d) Path with a single collision

Fig. 3: Distance functions for four example paths

B. collision interval: There is at least one collision interval for at least one pair of bodies (Section IV-B, Fig. 3b and 3c),
C. single collision: There is at least one contact collision for one pair of bodies, and there is no collision interval along the path (Section IV-C, Fig. 3d).

We will show that the algorithm always validates or invalidates the path in a finite time, in Cases A and B, while the algorithm may fail to finish in Case C. When the algorithm finishes in finite time, we will provide an upper bound on the number of tests needed. Testing an interval consists in testing its middle configuration for collisions and determining the half-length Δt of a valid interval centered on this configuration.

A. If the path is collision-free

Let us consider that the path $\mathbf{p}$ is collision-free. This means that for every pair of bodies i, $\forall t \in [0, T]$, $d_i(\mathbf{p}(t)) > 0$. Since the functions d_i are continuous on $[0, T]$, using the extreme value theorem, there exists $\delta > 0$ so that

$$\forall t \in [0, T]\ , \forall i\ , d_i(\mathbf{p}(t)) \geq \delta \tag{9}$$

as shown in an example in Figure 3a.

For each pair, a maximum relative velocity along the path $V_i^{\max}$ is computed. If $V_i^{\max} = 0$, the case is trivial. If $V_i^{\max} > 0$, at each iteration k, the algorithm computes for each pair i

$$\Delta t_k^i = \frac{d_i(\mathbf{p}(t_k))}{V_i^{\max}} \tag{10}$$

Using (9) we have the following inequality:

$$\Delta t_k^i \geq \frac{\delta}{V_i^{\max}} > 0 \tag{11}$$

At each iteration k, the algorithm validates an interval of length at least $2 * \Delta t_k$ where Δt_k is the minimum of all Δt_k^i. Since $V_i^{\max}$ does not depend on k, we define Δt as

$$\Delta t = \frac{\delta}{V^{\max}} > 0 \tag{12}$$

with $V^{\max} = \max_i V_i^{\max}$. At each iteration k, the validated interval is at least of length $2 * \Delta t$. Let us consider a certain depth k of the recursive algorithm, as illustrated in Figure 4. The first call to the algorithm is the depth $k = 0$. The calls to validate the two untested intervals around the validated interval (lines 16-17 of Algorithm 1) correspond to the depth 1. We will prove the following statement by recursion.

Lemma 3. *At depth k, before testing, there are at most 2^k disjoint intervals needing to be tested, and each of these intervals has a length of at most T_k with:*

$$T_k = \frac{T}{2^k} - \sum_{j=0}^{k-1} \frac{\Delta t}{2^j} \tag{13}$$

Proof. At the start of the algorithm, for $k = 0$, we have the whole path to validate. We have $2^0 = 1$ interval and this interval is of length $T = \frac{T}{2^0} - \sum_{j=0}^{j=k-1} \frac{\Delta t}{2^j} = T_0$. The statement is thus true for $k = 0$.

Suppose the statement is true at k. We will prove that it is true at $k+1$. The statement is true at k means that at depth k, before testing, we have at most 2^k disjoint intervals. The tests at depth k consist in testing each interval and either validating it whole or dividing it into two smaller intervals to be tested. This means that after depth k, and before depth $k+1$, there is at most $2 * 2^k = 2^{k+1}$ intervals, thus validating the first part of the statement.

The statement being true for k also means that each interval at depth k before testing has a length of at most T_k. Each interval is tested and an interval of length at least $2 * \Delta t$ is validated at the middle of the interval. This means that the two smaller intervals left to be validated have lengths of at most $\frac{T_k}{2} - \Delta t$.

$$\frac{T_k}{2} - \Delta t = \frac{1}{2}\left(\frac{T}{2^k} - \sum_{j=0}^{k-1} \frac{\Delta t}{2^j}\right) - \Delta t \tag{14}$$

$$= \frac{T}{2^{k+1}} - \sum_{j=0}^{k-1} \left(\frac{\Delta t}{2^{j+1}}\right) - \Delta t \tag{15}$$

$$= \frac{T}{2^{k+1}} - \sum_{j=1}^{k} \left(\frac{\Delta t}{2^j}\right) - \Delta t \tag{16}$$

$$= \frac{T}{2^{k+1}} - \sum_{j=0}^{k} \frac{\Delta t}{2^j} \tag{17}$$

$$= T_{k+1} \tag{18}$$

The statement is proved. □

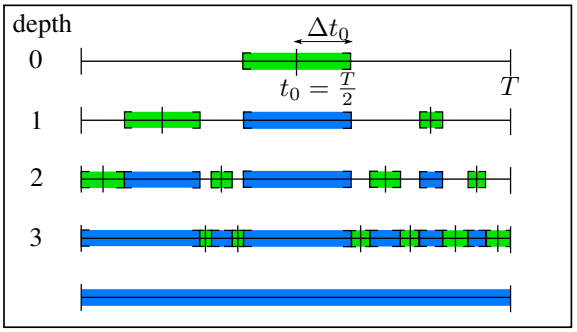


Fig. 4: Visualization of the validation of a collision-free path

We can reformulate T_k using the formula for the finite sum of a geometric series:

$$T_k = \frac{T}{2^k} - \left(2 - \frac{1}{2^{k-1}}\right)\Delta t \tag{19}$$

$$T_k = \frac{T + 2\Delta t}{2^k} - 2\Delta t \tag{20}$$

Using (20), we can determine the depth K at which point T_k reaches zero, which means that there is no interval left to test and the path has been entirely validated. Thus, the smallest integer K such that $T_K \leq 0$ is computed as follows:

$$K = \left\lceil \log_2\left(\frac{T}{2\Delta t} + 1\right)\right\rceil \tag{21}$$

At each depth k, 2^k tests are executed at most. The maximum number N of tests done from the start of the algorithm up to depth K is thus

$$N = \sum_{j=0}^{K-1} 2^j = 2^K - 1 \tag{22}$$

$$N = 2^{\left\lceil \log_2\left(\frac{T}{2\Delta t}+1\right)\right\rceil} - 1 \tag{23}$$

We proved that for a collision-free path, N is an upper bound on the number of tests required to validate it. According to (12) and (23), it depends on the minimum distance to collision δ and the maximum relative velocity between the pairs of bodies, $V^{\max}$. In Big-O notation, we can say that the algorithm is in $O(\frac{TV^{\max}}{\delta})$.

B. If the path has a collision interval for one pair of bodies

In this case, for a body pair i, there exists a collision interval $T_c = [t_1, t_2] \subset [0, T]$ with $t_2 > t_1$ so that $\forall t \in T_c,\ d_i(\mathbf{p}(t)) = 0$.

First, let us remark that if T_c has a length equal to or greater than $T/2$ (Figure 3b), then the configuration $\mathbf{q} = \mathbf{p}(T/2)$ is guaranteed to correspond to a collision. Since the algorithm starts by testing this configuration in the middle of the path, the path is invalidated at the first iteration.

Let us now suppose that the collision is a collision interval T_c of any length (Figure 3c). As in the collision-free case, we consider the length of the different intervals to be tested at each depth of the recursive algorithm. Let T_k be an upper bound

on the length of an interval at depth k, before testing. We can re-use the proof of the previous section and the formula (13), but with $\delta = 0$, since there is a collision somewhere on the path.

$$T_k = \frac{T}{2^k} \tag{24}$$

T_k being an upper bound on the length of an interval at depth k, $\frac{T_c}{T_k}$ is a lower bound on the ratio of the length of the collision interval to the length of an interval at depth k, and can be written as

$$\frac{T_c}{T_k} = 2^k \frac{T_c}{T} \tag{25}$$

This lower bound is doubled at each iteration, i.e., the ratio of the length of the collision interval to the length of an interval at depth k is at least doubled at each iteration. At a certain depth K, this ratio becomes equal to or greater than $\frac{1}{2}$, which means that the collision interval has a length equal to or greater than half the length of the considered interval. The collision is then guaranteed to be found since the middle of the considered interval lies necessarily in the collision interval. K is thus the smallest integer such that

$$2^K \frac{T_c}{T} \geq \frac{1}{2} \tag{26}$$

$$K = \left\lceil \log_2 \frac{T}{2T_c} \right\rceil \tag{27}$$

This means that the collision is found after the tests at depth K. Similarly to (22), we can claim that the algorithm takes at most N iterations to find the collision, with

$$N = \sum_{j=0}^{K} 2^j = 2^{K+1} - 1 \tag{28}$$

$$N = 2^{\lceil \log_2 \frac{T}{2T_c} \rceil + 1} - 1 \tag{29}$$

This upper bound depends on the lengths of the path to be validated, and of the collision interval. As can be expected, small collision intervals may take more time to find than larger ones. In the case of a path with more than one collision interval, an upper bound for the maximum iterations can be written using (29) for the largest collision interval.

C. If the path has a single collision

In this case, there is at least one collision between two bodies, reduced to a single configuration $t_c \in [0, T]$ (Figure 3d), and there is no interval collision along the path. In this particular case, the computation may never end. If there exists a collision interval somewhere else along the path, then the path corresponds to Case B and the computation is guaranteed to end as proved in the previous section.

Let us consider the following 2-dimensional case with two triangles $A_1B_1C_1$ and $A_2B_2C_2$, as shown in Figure 5. Along the path, at parameter t, we denote by $\delta(t)$ the shortest distance between the two triangles, and by $L(t)$ the distance between A_1 and A_2. The angle α does not vary along the path (at the collision parameter t_c, α is not defined).

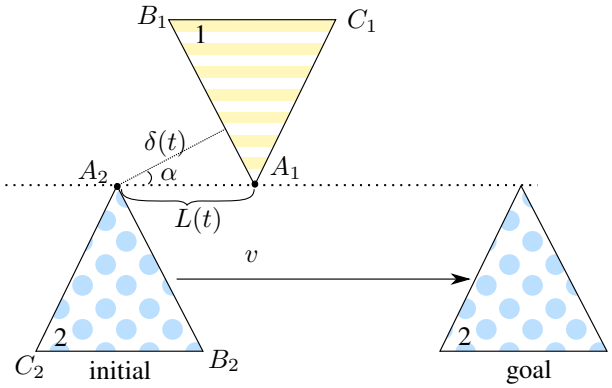


Fig. 5: Example of a single collision. Each body is a triangle $A_iB_iC_i$. Along the path, the triangle 1 (yellow stripes) is static. The triangle 2 (blue dots) moves from its initial position to its goal position with a constant velocity $\mathbf{v}$. Along the path, the lines (B_1C_1), (B_2C_2) and (A_1A_2) are parallel. There is a single collision during the motion, which the algorithm generally fails to find.

$$\cos\alpha = \frac{\delta(t)}{L(t)} \tag{30}$$

Since the relative velocity between both triangles is constant along the path, $V^{\max} = \|\mathbf{v}\|$. At each iteration, the validated interval is of length $\frac{\delta(t)}{V^{\max}}$. At parameter t, the remaining time before collision is $\frac{L(t)}{V^{\max}}$. Thus, at each iteration, the algorithm validates a portion of the remaining interval, which is $\frac{\delta(t)}{L(t)} = \cos\alpha < 1$. This ratio is constant along the path. The algorithm validates intervals of smaller and smaller lengths, and can only end if at one iteration, it considers an interval centered on t_c. Otherwise, the computation never ends.

V. Improvements on the initial method

As stated in the introduction, we propose a change to the algorithm, consisting in choosing for a pair of bodies the best upper bound on their relative velocity (Section V-A). Two other changes aiming to decrease the computation time are also proposed in Section V-B and Section V-C, respectively.

A. Choosing the best V^{max}

An expression of an upper bound $V^{\max}$ of the maximal relative velocity of all points of $\mathcal{B}_a$ in the reference frame of $\mathcal{B}_b$, is given by (8). However, this expression is not symmetric and switching $\mathcal{B}_a$ and $\mathcal{B}_b$ does not yield the same value of $V^{\max}$, even though both values are upper bounds on the relative velocity between the bodies. This can be shown by the following example, illustrated in Figure 6: A box ($\mathcal{B}_a$) lies immobile on the ground while a mobile robot moves at a linear velocity of norm v. The frame J_r is fixed on the robot at its virtual planar joint. B_b is a body linked to the mobile robot by a revolute joint and J_b is a frame attached to B_b which moves with an angular velocity of norm ω. We denote by r_a and r_b

the radii of $\mathcal{B}_a$ and $\mathcal{B}_b$, respectively. We denote by $d = \|{}^r\mathbf{T}_a\|$ the distance between the origins of J_a and J_r.

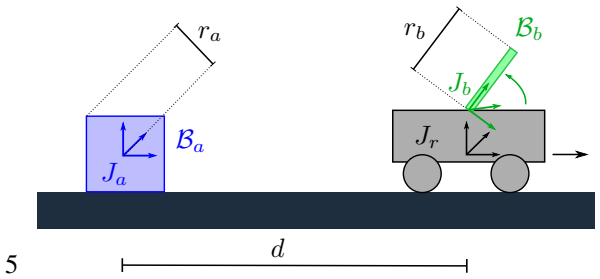


Fig. 6: Example of a situation where (8) gives two significantly different upper bounds on the relative velocity between two bodies $V^{\max}$

Using (8), we obtain $V^{\max}_{a\to b}$ and $V^{\max}_{b\to a}$, two upper bounds on the relative velocity of bodies $\mathcal{B}_a$ and $\mathcal{B}_b$ computed by considering the velocity of points of $\mathcal{B}_a$ in the reference frame of $\mathcal{B}_b$ and the velocity of points of $\mathcal{B}_b$ in the reference frame of $\mathcal{B}_a$, respectively.

$$\begin{aligned} V^{\max}_{a\to b} = {}& {}^r v_a + {}^r\omega_a(r_a) \\ & + {}^b v_r + {}^b\omega_r(r_a + \|{}^r\mathbf{T}_a\|) \\ = {}& v + \omega(r_a + d) \end{aligned} \tag{31}$$

$$\begin{aligned} V^{\max}_{b\to a} = {}& {}^r v_b + {}^r\omega_b(r_b) \\ & + {}^a v_r + {}^a\omega_r(r_b + \|{}^r\mathbf{T}_b\|) \\ = {}& \omega r_b + v \end{aligned} \tag{32}$$

When the distance d between the robot and the object is large, $V^{\max}_{a\to b}$ given by (31) can be much larger than $V^{\max}_{b\to a}$ given by (32). Thus, to improve the performance of the algorithm, for each body pair, we compute the two different relative velocity upper bounds and choose the smallest one. With a smaller $V^{\max}$, the intervals validated at each step are larger, and the algorithm needs fewer iterations and less time to finish.

B. Memory of the validated intervals

The algorithm presented in Section III-A is recursive. To avoid unnecessary computations, at each iteration, the algorithm keeps a memory of every validated intervals for each pair of bodies. When searching for a valid interval around a configuration along the path, the algorithm finds a valid interval for each body pair and keeps the intersection of the valid intervals. If a pair has in memory a validated interval larger than the interval the algorithm is testing, then the distance computation is not performed for this pair. With this improvement, we limit the number of calls to the distance computation function. For example, if at the first iteration, the whole path is validated for one pair of bodies, then this pair will never be tested again at any future iteration.

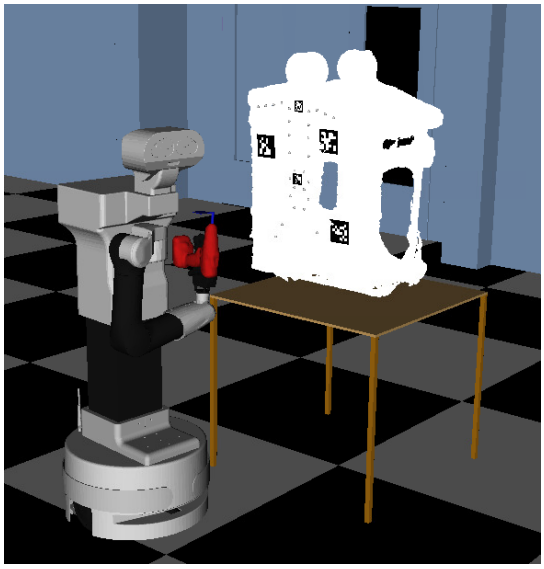


Fig. 7: Visualization of Tiago holding a drill next to an airplane part

C. Order of validation of the collision pairs

At each test of an interval, every collision pair is tested to determine the half-length $\Delta t_{(a,b)}$ of a valid interval for this pair (Algorithm 1, line 8). The order in which all the pairs are tested can be arbitrary or chosen to optimize the tests. We optimize the order of validation as follows. Once every pair has been tested, the pair which yielded the smallest $\Delta t_{(a,b)}$ is relatively close to a collision since it will probably yield a small $\Delta t_{(a,b)}$ at the next iteration. This pair is put at the beginning of the list of pairs, so that it will be the first one tested at the next iteration and yield a small interval $\Delta t_{(a,b)}$. Paired with the fact that each pair retains a memory of the previously validated intervals, no test will be needed for the pairs which have already validated a larger interval containing this small interval.

VI. Experimental results

A. Impact of the proposed improvements

The continuous validation method presented in this paper has been applied to the generation of movements of the robot Tiago [26]. Tiago is a modular robot with a differential-drive mobile base. In our case, Tiago has one 7-DOF arm and a 5-finger actuated hand. The test is divided in two parts. First, full-body random paths for Tiago are generated and tested with each of five variants of the continuous validation method (Random Test Case). Then, we solve an industrial test case (Deburring Test Case) using a planning algorithm and the same variants of the continuous validation method. This industrial test case is representative of the typical use of a robot like Tiago in an industrial setting. The performance of the algorithm is measured using the number of iterations performed for the validation of each path. One iteration corresponds to one computation of the distance between one pair of bodies. This indicator is independent of the computational power of

the machine used. The computation time of the algorithm is discussed in Section VI-B.

We define five variants of the continuous validation method corresponding to the different improvements presented in Section V. First, the "Basic" variant, corresponding to the initial method presented in Section III. Then, one variant corresponding to the initial method with the addition of one of the three proposed improvements:

- "Vmax" variant for the improvement proposed in Section V-A, where two possible values of $V^{\max}$ are computed and the smallest one is kept.
- "Memory" for the improvement from Section V-B, where each pair of bodies keeps a memory of the parts of the path already validated.
- "Sorting" for the improvement from Section V-C, where the pair of bodies which produced the smallest valid interval is put at the beginning of the list of pairs to be tested at the next iteration.

Finally, the "Complete" variant corresponds to the initial method with the addition of each of the three improvements.

For the Deburring Test Case, as shown in Fig. 7, the robot Tiago holds an electric drill in its hand, and next to the robot is a table with an aircraft part lying on it. Tiago must reach each screw hole of the airplane part and perform a deburring task by inserting the tool into the hole. We assume that the positions of the holes are known. For the application in real life, AprilTags markers are placed on the part and next to the end-effector of the drill, and used to re-plan in real-time the movements for a better accuracy.

A RRT-like motion planning algorithm is used to generate a path resulting in Tiago reaching every screw hole. The algorithm generates random paths, for which an upper bound on the relative velocity is known for every pair of bodies. Every path the algorithm generates must be checked against collisions — self-collisions between Tiago's bodies as well as collisions between Tiago and the environment. The room in which Tiago moves is fully modeled and is also taken into account for the collision avoidance. The algorithm discards the non-valid paths and keeps only the valid paths in order to generate the final global path. A traveling salesman problem is formulated and solved to determine the order in which Tiago reaches the holes.

Collision detection between pairs of bodies is performed by `hpp-fcl`, a variant of FCL that provides a lower bound on the distance between two objects when the objects are not in collision, at no additional cost.

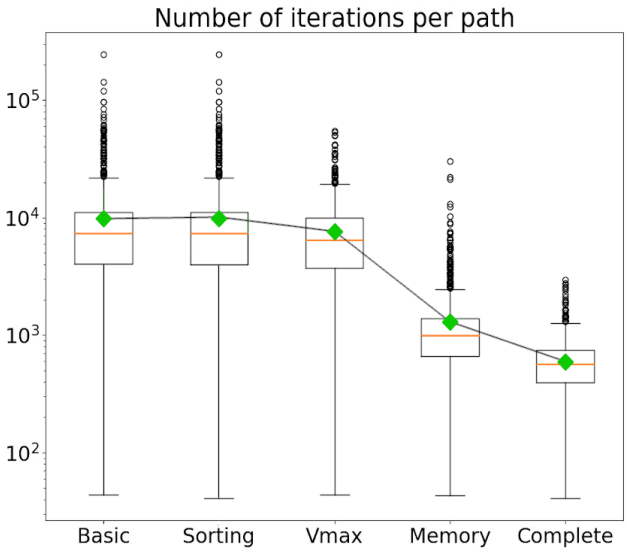
We use each variant as the validation method to test the random paths and to solve the test case. For each variant, we record the number of iterations of the validation algorithm and use it as an indicator of the efficiency of the algorithm. The results are shown in Fig. 8. For both cases, we test for both self-collisions and collisions with the environment. Figures 8a and 8b show the results when 1000 paths are randomly generated and each path is tested with every variant of the continuous validation algorithm. To generate each path, two valid configurations are sampled in the workspace and a linear interpolation is made between the two configurations to obtain a path. The same airplane part on a table is present in the environment as an obstacle. Figures 8c and 8d show the results for one run of the path planning algorithm to solve the deburring test case.

When tested on the same path, the different variants of the continuous validation method all return the same result concerning the validity of the path. However, when the path is not valid, the variants may find the collision at different parameters on the path. The first valid interval of the non-valid path is used in the RRT algorithm, and may differ for each variant. It would be very difficult to obtain the exact same run of the RRT algorithm for the different variants, even when using the same seed in the random generator. This explains the different numbers of paths validated for each method during the Deburring Test Case in Fig. 8.

For every method, we observe the presence of some paths with very high numbers of iterations compared to the mean (represented by black dots in Fig. 8a and Fig. 8c). This means that some paths are more "difficult" and need considerably more iterations than others to be validated: Paths that are valid but close, or almost tangent, to collisions. Each improvement in the algorithm (variants Sorting, Vmax, Memory) corresponds to an improvement of the mean of the number of iterations. The Complete variant has the lowest mean of all the variants. The maximum number of iterations for the Complete variant is reduced by a factor 100 compared to the Basic variant, while the mean is reduced by a factor of almost 20. The different improvements reduce the standard deviation of the data. Since the improvements deal better with those "difficult" path, the data is more packed.

During the Deburring Test Case, the planning algorithm finds paths that avoid the obstacles while staying close to them. Those paths are "difficult" to test, and this explains the higher standard deviation for the Basic method when compared to the Random Test Case. The randomly generated paths are in average farther away from the obstacles than the paths generated in the Deburring Test Case, and thus are "easier" to validate. This also explains the difference in efficiency for the Sorting variant between the two tests. Considering the results of Figures 8a and 8b, the Sorting variant does not improve much from the Basic variant for the Random Test Case. However, when solving the Deburring Test Case, the Sorting variant reduces significantly the number of iterations per path. Indeed, there are more "difficult" paths in the Deburring Test Case for which at least one pair of bodies is close to a collision. By putting the pair closest to collision in the first place of the test list, the Sorting variant accelerates the validation of these "difficult" paths. Because there are not many "difficult" paths in the Random Test Case, the Sorting algorithm does not have such a significant impact as for the Deburring Test Case.

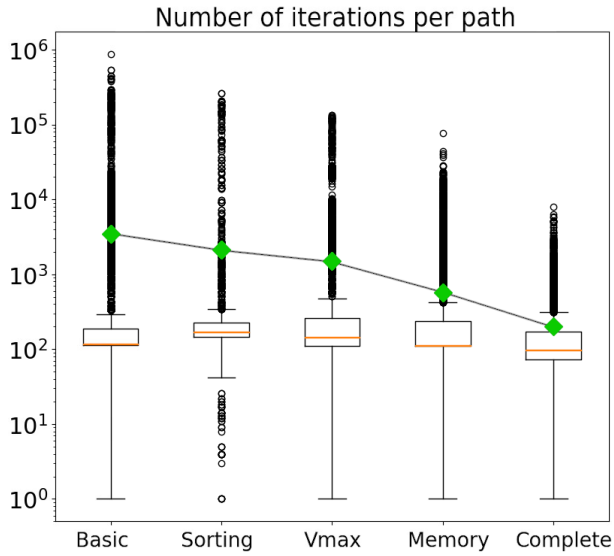
The results show that the Complete variant is the most efficient in terms of number of iterations per path. In terms of computation time, solving the Deburring Test Case takes in average almost 2 hours using the Basic variant, and around 20 minutes using the Complete variant. It should be noted that for those results, the whole room is modeled and used for the collision detection, as well as the airplane part, the table, the screwdriver and every part of the robot. Without the room model, which is relatively complex, the computations



(a) Random Test Case: Box plot (log scale) of the number of iterations per path

Method	**Basic**	**Sorting**	**Vmax**	**Memory**	**Complete**
Nb of paths	1 000	1 000	1 000	1 000	1 000
Mean	9 852	9 850	7 572	1 291	592
StDev	13 849	13 848	6 495	1 771	357
Minimum	44	41	44	43	41
Median	7 360	7 360	6 480	1 005	569
Maximum	244 118	244 081	54 840	30 273	2 948

(b) Random Test Case: Table of the number of iterations per path for five variants of the CCD algorithm. The mean, standard deviation (StDev), minimum, median and maximum number of iterations are indicated for the data set of each variant.



(c) Deburring Test Case: Box plot (log scale) of the number of iterations per path

Method	**Basic**	**Sorting**	**Vmax**	**Memory**	**Complete**
Nb of paths	23 297	5 240	10 110	54 718	20 722
Mean	3 441	2 113	1 493	565	201
StDev	23 893	15 837	8 263	1 564	323
Minimum	1	1	1	1	1
Median	116	168	144	111	96
Maximum	871 676	261 144	132 480	76 984	7 986

(d) Deburring Test Case: Table of the number of iterations per path for five variants of the CCD algorithm. The mean, standard deviation (StDev), minimum, median and maximum number of iterations are indicated for the data set of each variant.

Fig. 8: Number of iterations needed to validate a path using the different variants of the CCD algorithm for the Random Test Case and the Deburring Test Case. For the Random Test Case (Figures 8a and 8b), 1000 random full-body paths are generated for Tiago and tested. The Deburring Test Case (Figures 8c and 8d) corresponds to the solving of the industrial deburring test case with Tiago. For both tests, all five variants of the validation algorithm are used. Each of the variants Sorting, Vmax and Memory consists of the basic algorithm with one improvement. The Complete variant has all three improvements. We record the number of iterations necessary to validate each path. One iteration corresponds to one computation of the distance between one pair of bodies (Algorithm 1, line 6). In Figures 8a and 8c, the orange line represents the median, while the green diamond marker represents the mean and the black dots represent the outliers values.

are faster by a factor of 10.

The test case has been tested in real life and a video of the movement is available [27], with the use of the AprilTag markers to improve the precision of the position of the end-effector.

B. Comparison with a discretized method

The Random Test Case presented in the previous section has also been performed using a discretized collision detection method. When given a time step and a path to be tested, the discretized method samples the path with the given time step and tests each sampled configuration by performing an exact

Method	**Complete**	**Discretized,** $\tau_c = 0.12$
Nb of paths	1 000	1 000
Mean	0.20	0.23
StDev	0.26	9.8×10^{-2}
Minimum	9.4×10^{-3}	8.9×10^{-3}
Median	0.14	0.22
Maximum	3.6	0.64
Paths found valid	**756**	**760**

TABLE I: Random Test Case: Table of the computation time (in seconds) per path for the Complete variant of the CDD algorithm and the discretized method with the corresponding time step $\tau_c = 0.12$ for 1000 random paths. The mean, standard deviation (StDev), minimum, median and maximum computation times are indicated for the data set of each method.

collision test for each pair of bodies. For such a discretized method, the average computation time is loosely proportional to the inverse of the time step.

We compare the computation times obtained with the discretized method and the Complete CCD method. The computer used to run the software HPP is an "Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz" with 4,096 KB of cache memory and 16 GB of RAM.

The Random Test Case is performed for different time steps, in order to obtain a linear regression between the time step and the average computation time per path. Using this regression, we obtain the time step τ_c which gives an average computation time per path close to the one obtained with the Complete continuous validation method. We obtain $\tau_c = 0.12$.

The results are shown in Table I. As shown in the last line of the table, for the same set of 1000 random paths, the discretized method finds more valid paths than the continuous method. Indeed, for 4 paths out of 1000, the discretized method fails to detect collisions that are found by the continuous method. In order for the discretized method to find those collisions, the time step would have to be smaller than τ_c, and consequently the discretized method would have a higher average computation time than the Complete method. This means that the Complete CCD method outperforms the discretized method in this test case.

In summary, compared to the non-continuous discretized method, the CCD method presented in this paper has the following advantages:

- The CCD method provides a validity certificate that a non-continuous method cannot offer.
- The CCD method eliminates the need to determine an appropriate time step for every robot and situation.
- The CCD method performs better in terms of computation times than the discretized method.

The presented algorithm also has the advantage of versatility. Although this paper has presented only the case of collision avoidance, any other type of constraint can be taken into account. To be acceptable, a constraint must be separable into several elements — like the pair of bodies for the collision avoidance — and it must be possible to validate each element around a given configuration on the path. For example, the algorithm has been extended to take into account collision avoidance for cable-driven parallel robots [28], as well as the constraints associated with the tensions of the cables of such robots [29]. For the cable-driven-parallel robot CoGiRo in [28], the continuous validation algorithm applied to collision detection also performs significantly better than the discretized method. Indeed, the collisions that include cables are difficult to find using the discretized method due to the small radius of the cables, and the CCD method validates the paths faster while missing no collision.

VII. Conclusion

This paper has reformulated an existing continuous collision validation method able to exactly validate or invalidate a path. Unlike discretized validation methods, this method guarantees to find a collision along the path if any exists. The algorithm proceeds in a dichotomous manner, considering successive configurations along the path and finding valid intervals centered on these configurations. Every pair of bodies is considered, and for each, a known upper bound on the relative velocity between the bodies is used to compute the valid intervals. This paper has also presented a proof of the convergence of the algorithm. In every case, except the case of a collision reduced to a single configuration, the algorithm is able to determine the validity of the path in finite time. Expressions of upper bounds on the number of iterations needed to validate a path have also been introduced.

Three changes can be easily made to the algorithm to reduce the number of iterations per validation. The first change consists in choosing the smallest of two possible values of the upper bound on the relative velocity between two bodies. The second change consists in keeping in memory, for every pair of bodies, every interval already validated, to avoid redundant computations. The third change consists in putting in first position, in the list of the pair of bodies to be tested, the pair most prone to being the closest to collision.

The different variants have been used to solve an industrial test case where the robot Tiago must perform a deburring task on an airplane part. Through this experimental validation, we show that each of these changes improve the algorithm by reducing the average number of iterations needed to validate a path. The final method incorporates all three improvements and constitutes a complete and versatile validation method, which can take into account pairs of bodies for collision detection, but may also take into account other types of constraints for more complex robots or different situations. This final method is exact and performs better in terms of computation times than a discretized method.

Acknowledgment

This work has been partially supported by Airbus S.A.S. in the framework of the common laboratory Rob4Fam.

References

[1] D. Flickinger, J. Williams, and J. J. Trinkle, "What's wrong with collision detection in multibody dynamics simulation?" in *IEEE International Conference on Robotics and Automation*, 05 2013, pp. 959–964.

[2] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for fast path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[3] S. K. Sertac and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, L. Kavraki, and S.Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[6] S. M. Lavalle, *Planning algorithms*. Cambridge University Press, 2006.

[7] N. Vahrenkamp, P. Kaiser, T. Asfour, and R. Dillmann, "Rdt+: A parameter-free algorithm for exact motion planning," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 715–722. [Online]. Available: https://h2t.anthropomatik.kit.edu/pdf/Vahrenkamp2011.pdf

[8] M. Campana, F. Lamiraux, and J.-P. Laumond, "A gradient-based path optimization method for motion planning," *Advanced Robotics*, vol. 30, no. 17-18, pp. pp.1126–1144, 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01301233

[9] 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.

[10] "Implementation of continous collision detection." [Online]. Available: https://gepettoweb.laas.fr/hpp/hpp-core/doxygen-html/classhpp_1_1core_1_1ContinuousValidation.html

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

[12] S. Redon, A. Kheddar, and S. Coquillart, "Fast Continuous Collision Detection between Rigid Bodies," *Computer Graphics Forum*, vol. 21, no. 3, pp. 279–287, 2002, the definitive version is available at www.blackwell-synergy.com. [Online]. Available: https://hal.inria.fr/inria-00390356

[13] S. Tarbouriech and W. Suleiman, "On bisection continuous collision checking method: Spherical joints and minimum distance to obstacles," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7613–7619.

[14] M. Tang, Y. J. Kim, and D. Manocha, *CCQ: Efficient Local Planning Using Connection Collision Query*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 229–247. [Online]. Available: https://gamma-web.iacs.umd.edu/CSPlanning/wafr10.pdf

[15] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *ACM Symposium on solid modeling and applications*, 2004.

[16] 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*, 2007.

[17] J.-P. Merlet, "A generic trajectory verifier for the motion planning of parallel robots," *J. Mech. Des.*, vol. 123, no. 4, pp. 510–515, 2001.

[18] J.-P. Merlet and D. Daney, "Legs interference checking of parallel robots over a given workspace or trajectory," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 757–762.

[19] J.-P. Merlet, "A local motion planner for closed-loop robots," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 3088–3093.

[20] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *IEEE International Conference on Robotics and Automation*, 2012.

[21] J. Pan, L. Zhang, and D. Manocha, "Collision-free and smooth trajectory computation in cluttered environments," *International Journal of Robotics Research*, vol. 31, no. 10, 2012.

[22] L. He and J. van den Berg, "Efficient exact collision-checking of 3-D rigid body motions using linear transformations and distance computations in workspace," in *IEEE International Conference on Robotics and Automation*, Hong Kong, China, June 2014.

[23] W. Merkt, V. Ivan, and S. Vijayakumar, "Continuous-time collision avoidance for trajectory optimization in dynamic environments," in *Intelligent Robots and Systems (IROS), 2019 IEEE/RSJ International Conference on*, 6 2019.

[24] E. Åblad, D. Spensieri, R. Bohlin, and A.-B. Strömberg, "Continuous collision detection of pairs of robot motions under velocity uncertainty," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1780–1791, 2021.

[25] R. Culley and K. Kempf, "A collision detection algorithm based on velocity and distance bounds," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, 1986, pp. 1064–1069.

[26] J. Pages, L. Marchionni, and F. Ferro, "Tiago: the modular robot that adapts to different research needs," in *International workshop on robot modularity, IROS*, 2016.

[27] "Video "Tiago mobile manipulator performs deburring tasks on an aircraft part"," https://peertube.laas.fr/videos/watch/6f40ea79-abcd-490e-a616-3a67bf297d93, 2021.

[28] D. Bury, J.-B. Izard, M. Gouttefarde, and F. Lamiraux, "Continuous collision detection for a robotic arm mounted on a cable-driven parallel robot," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 8097–8102.

[29] ——, "Continuous tension validation for cable-driven parallel robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6558–6563.