A Local Collision Avoidance Method for Non-strictly Convex Polyhedra

Fumio Kanehiro*, Florent Lamiraux†, Oussama Kanoun†, Eiichi Yoshida* and Jean-Paul Laumond†
IS/AIST-ST2I/CNRS Joint Japanese-French Robotics Laboratory(JRL)
*Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology(AIST)
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 Japan
Email: {f-kanehiro, e.yoshida}@aist.go.jp
†LAAS-CNRS, University of Toulouse
7 Avenue du Colonel Roche, 31077 Toulouse, France
Email: {florent, okanoun, jpl}@laas.fr

***Abstract*— This paper proposes a local collision avoidance method for *non-strictly convex* polyhedra with *continuous* velocities. The main contribution of the method is that non-strictly convex polyhedra can be used as geometric models of the robot and the environment without any approximation. The problem of the continuous interaction generation between polyhedra is reduced to the continuous constraints generation between polygonal faces and the continuity of those constraints are managed by the combinatorics based on Voronoi regions of a face. A collision-free motion is obtained by solving an optimization problem defined by an objective function which describes a task and linear inequality constraints which do geometrical constraints to avoid collisions. The proposed method is examined using example cases of simple objects and also applied to a humanoid robot HRP-2.**

I. Introduction

Detecting and avoiding collisions is fundamental for the development of robots that can be safely operated in human environments. This issue has given rise to contributions in the 1980's in the context of robotic manipulators. One of the most famous and most used contribution in this domain is certainly [1]. [1] proposes a method which plans a motion by minimizing an error between a desired velocity and the planned velocity under inequality constraints to avoid collision. The robot and the environment must consists of strictly convex objects. [2] extends this method to avoid local minima by modifying the task description in heavily cluttered environment. [3] proposes another approach for avoiding collision by following the distance gradient. The robot is approximated by a set of strictly convex objects (ellipsoids).

Recent developments of humanoid robots have made the issue of collision detection and avoidance very critical again. [4] proposes a path planning method which computes dynamically stable and collision-free trajectories. It prepares a set of statically stable postures in advance and finds a path by exploring it with RRT-connect[5]. And the path is transformed into a dynamically stable trajectory by applying a dynamics filter[6]. [7] proposes a fast method to ensure that there is no self-collision in a trajectory. Shapes of the robot are approximated by convex hulls and the minimum distances between them are tracked using V-Clip[8]. [9] proposes a local collision avoidance method using repulsion fields defined by the minimum distance. [10] combines three methods to detect self-collisions online, (1) table look-up to detect self-collisions between adjacent joints and (2) reduction of pairs of links to be checked using heuristics, and (3) collision check using approximated shapes by convex hulls. [11] approximates shapes by spheres and swept sphere lines and used the minimum distances between them to avoid self-collision. [12] computes a strictly convex bounding volume called STP-BV by patching spheres and toruses for each body of a humanoid robot and builds collision-free postures.

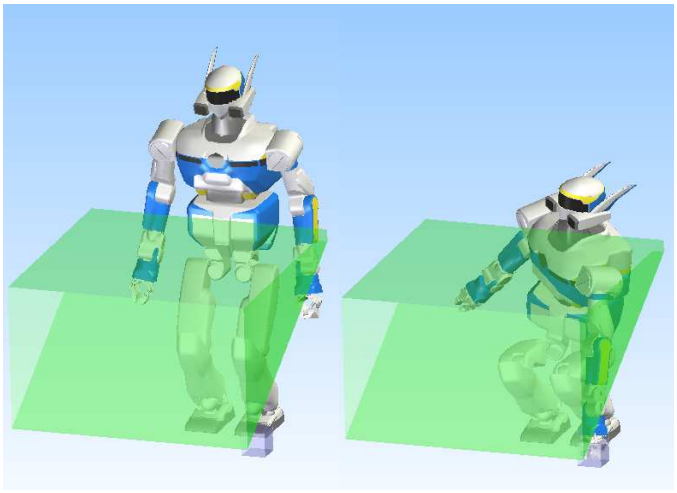

Fig. 1. "Pick up an object under the table" example

Many works have focused on approximating the robot by strictly convex objects but to our knowledge, very few has been done to deal with non-strictly convex objects without geometric approximation.

In this paper, we extend the method described in [1] to non-necessarily convex polyhedral objects, in such a way that the resulting velocity of the robot is continuous. The basic framework is same as in [1]. The robot velocity is computed by minimizing difference between the desired task velocity and the planned one under linear inequality constraints implied by pairs of objects close to each other. The main

difference is the way of generating constraints. In the case of strictly convex objects, one constraint over the velocity of the closest points between two objects is enough. The minimum distance and the closest points between non-strictly convex objects can be computed using efficient algorithms and robust implementations[13, 14, 15, 16]. But performing collision avoidance by applying a linear inequality constraint over the velocity between the closest points might result in discontinuous velocities for the objects. Because the closest points between two strictly convex objects move continuously whereas the closest points between non-strictly convex objects do not. If one object is a robot body, discontinuous velocity cannot be applied. So in the case of polyhedra, several constraints are generated for each pair of objects unlike the method in [1]. Our method reduces the problem of the continuous interaction generation between polyhedra to the continuous constraints generation between faces and manages the continuity of those constraints using the combinatorics based on Voronoi regions of a face.

The paper is organized as follows. In section II, we recall Faverjon and Tournassoud's method for strictly convex objects. In section III, we extend the method to make it possible to accept polyhedra. In section IV, the extended method is examined using example cases of simple objects and a humanoid robot HRP-2[17]. In section V, we summarize and conclude the paper.

II. A LOCAL METHOD FOR STRICTLY CONVEX OBJECTS

A. Strictly convex objects

First, let us recall the following definition.

Definition: *Strictly convex object. Let $\mathcal{O}$ be a closed subset of $\mathbb{R}^3$ and int($\mathcal{O}$) be the interior of (greater open subset of) $\mathcal{O}$. $\mathcal{O}$ is strictly convex if and only if*

$$\forall A \in \mathcal{O}, \forall B \in \mathcal{O}, \forall \lambda \in \mathbb{R}, 0 < \lambda < 1, \ \ \lambda A + (1-\lambda)B \in \textit{int}(\mathcal{O})$$

For instance, a convex polyhedron is not strictly convex. If two points on a facet are selected as A and B, the line segment linking them is not inside the interior of the polyhedron but on the facet.

B. Outline of the method

Let us recall the principle of Faverjon and Tournassoud's method [1]. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two strictly convex objects. For ease of explanation, let's consider $\mathcal{O}_1$ as a movable object and $\mathcal{O}_2$ as a static one. Let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ denote the closest points between $\mathcal{O}_1$ and $\mathcal{O}_2$ and d does the distance $\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|$ between them (Fig. 2). Since $\mathcal{O}_1$ and $\mathcal{O}_2$ are strictly convex objects, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ move continuously on $\mathcal{O}_1$ and $\mathcal{O}_2$ boundaries and d is continuously differentiable.

If d is smaller than a threshold called *influence distance* and denoted by d_i, the following constraint is defined for velocity of d:

$$\dot{d} \geq -\xi \frac{d - d_s}{d_i - d_s} \tag{1}$$

where ξ is a positive coefficient for adjusting convergence speed, $d_s (< d_i)$ is a positive value called *security distance*.

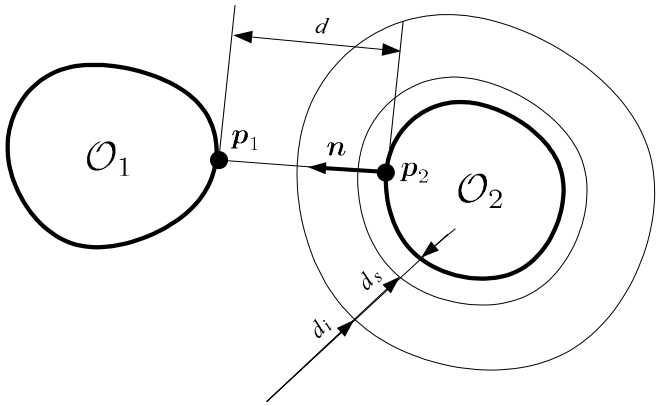

Fig. 2. Faverjon and Tournassoud's method

Inequality (1) is called *velocity damper* and it expresses that d must not decrease too fast when it is smaller than d_i. As the result, d never be smaller than d_s. The region where d is smaller than d_i is called *influence zone*. Note that $\dot{d}$ must satisfy Inequality (1) when $\boldsymbol{p}_1$ enters *influence zone*. If it doesn't, $\dot{d}$ is constrained discontinuously. Therefore, the value of ξ must be tuned according to applications.

$\dot{d}$ is computed by the following equation.

$$\dot{d} = (\dot{\boldsymbol{p}}_1 | \boldsymbol{n})$$

where $\boldsymbol{n}$ is the unit vector $(\boldsymbol{p}_1 - \boldsymbol{p}_2)/d$ and notation $(\boldsymbol{u}|\boldsymbol{v})$ refers to the inner product of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

Let $\boldsymbol{q}$ denote the configuration and $\dot{\boldsymbol{q}}$ the velocity of the robot. The velocity of $\boldsymbol{p}_1$ can be expressed as:

$$\dot{\boldsymbol{p}}_1 = \boldsymbol{J}(\boldsymbol{q}, \boldsymbol{p}_1)\dot{\boldsymbol{q}}$$

where $\boldsymbol{J}(\boldsymbol{q}, \boldsymbol{p}_1)$ is a Jacobian matrix of $\mathcal{O}_1$ at $\boldsymbol{p}_1$. Inequality (1) thus becomes a linear inequality constraint over the robot velocity $\dot{\boldsymbol{q}}$:

$$(\dot{\boldsymbol{q}} | \boldsymbol{J}(\boldsymbol{q}, \boldsymbol{p}_1)^T \boldsymbol{n}) \geq -\xi \frac{d - d_s}{d_i - d_s}$$

A task is described by the control of a measure of the problem, a vector $\boldsymbol{\tau}(\boldsymbol{q})$ in a task space. The task is achieved by finding $\boldsymbol{q}$ which satisfies $\boldsymbol{\tau}(\boldsymbol{q}) = \boldsymbol{0}$. Given a desired task velocity $\underline{\dot{\boldsymbol{\tau}}}$, the robot velocity to achieve the task while avoiding collision is computed by solving the following optimization problem over $\dot{\boldsymbol{q}}$:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} \quad & \|\boldsymbol{J}_\tau(\boldsymbol{q})\dot{\boldsymbol{q}} - \underline{\dot{\boldsymbol{\tau}}}\|^2 \\ \text{subject to} \quad & (\dot{\boldsymbol{q}} | \boldsymbol{J}(\boldsymbol{q}, \boldsymbol{p}_1)^T \boldsymbol{n}) \geq -\xi \frac{d - d_s}{d_i - d_s}. \end{aligned} \tag{2}$$

where $\boldsymbol{J}_\tau(\boldsymbol{q})$ is a Jacobian matrix of $\boldsymbol{\tau}(\boldsymbol{q})$ at $\boldsymbol{q}$.

Of course, considering each body of the robot and several obstacles yields several inequality constraints.

C. Lower bound of the distance between $\mathcal{O}_1$ and $\mathcal{O}_2$

If we denote by $d(t)$ the minimum distance between $\mathcal{O}_1$ and $\mathcal{O}_2$ along time, $d(t)$ is continuously differentiable. If a condition on the derivative $\dot{d}$ of d:

$$\forall t > 0, \dot{d}(t) \geq -\xi \frac{d(t) - d_s}{d_i - d_s}$$

and the initial condition $d(0) \geq d_s$ are satisfied, then the following condition is derived.

$$\forall t > 0,\ d(t) \geq d_s + (d(0) - d_s) e^{-\frac{\xi}{d_i - d_s} t} > d_s \qquad (3)$$

This proves that the distance between objects constrained by *velocity damper* never be smaller than d_s.

III. A Local Method for Non-strictly Convex Polyhedra

A. A discontinuous case of non-strictly convex polyhedra

If we apply Faverjon and Tournassoud's method to non-strictly convex polyhedra, the robot velocity changes discontinuously since the closest points between two objects move discontinuously. Fig. 3 and Fig. 4 show snapshots and results of an example case. In this example, a rectangle(0.2×0.8[m]) moves above an horizontal floor. The task of the rectangle is to move its center $\boldsymbol{p}_c(\boldsymbol{q})$ from $(0.0, 0.7)^T$[m] to $\boldsymbol{p}_g = (0.0, -1.0)^T$[m]. Parameters of *velocity damper*, d_i, d_s and ξ are set to 0.4[m], 0.2[m] and 0.5[m/s] respectively. $\dot{\underline{\boldsymbol{\tau}}}$ is given by:

$$\dot{\underline{\boldsymbol{\tau}}} = \delta\tau_{max} \frac{\boldsymbol{p}_g - \boldsymbol{p}_c(\boldsymbol{q})}{\|\boldsymbol{p}_g - \boldsymbol{p}_c(\boldsymbol{q})\|}$$

where $\delta\tau_{max}$ is set to 0.2.

Top left picture of Fig. 4 shows the minimum distance between the rectangle and the floor. Top right picture displays the vertical position of the rectangle. Bottom left and right pictures show the linear velocity along Y axis and the angular velocity respectively. One of vertices of the bottom edge of the rectangle, C_1 enters into *influence zone* at $t = 0.25$. But the velocity of C_1 is not affected since its velocity satisfies Inequality (1). From $t = 0.5$, the object velocity is affected by *velocity damper*. The center must move downward with the constant speed to achieve the task but the admissible speed of C_1 is limited. As a result, the object rotates. Around $t = 2.0$, the bottom edge of the rectangle becomes almost parallel to the ground, the closest points start to oscillate between C_1 and C_2. The object rotates clockwise to achieve the task when C_1 is constrained and does counterclockwise when C_2 is constrained. As a result, the minimum distance becomes smaller than d_s and the rectangle eventually collides with the floor at $t = 3.0$.

The collision-freeness is not assured anymore when the robot bodies and obstacles are not strictly convex as shown. The oscillation of the closest points causes discontinuous changes of $\boldsymbol{p}_1$ and $\boldsymbol{n}$ in Problem (2) and it leads to discontinuity in the solution of Problem (2) as we can see in Fig. 4.

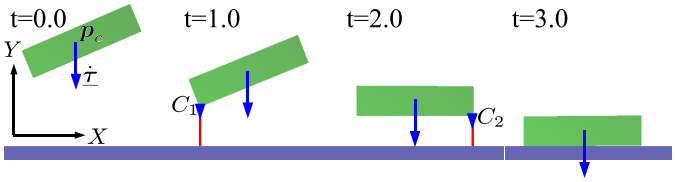

Fig. 3. Example of a discontinuous constraint

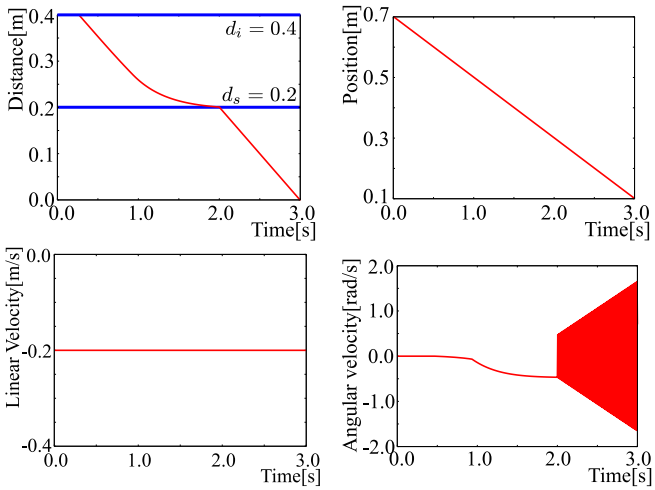

Fig. 4. Results of the example

B. Decomposition of interaction between polyhedra

The goal of this section is to define the inequality constraints in such a way that $\dot{\boldsymbol{q}}$ remains continuous. To cope with this issue, we propose to keep track of several pairs of points that move continuously on the facets of the polyhedra composing the obstacles and robot.

Discontinuity of constraints happens in the following cases.

1) a new constraint appears suddenly
2) a constraint disappears suddenly

The closest points jump discontinuously if these two cases happen at the same time.

In order to prevent these cases and generate a collision-free motion with continuous velocities, pairs of points must be selected by complying with the following rules.

1) the closest points between a robot body and an obstacle must be constrained to guarantee that the robot never collides.
2) the potential closest points must have been constrained before they become the closest points.
3) the closest points must continue to be constrained even if they are not closest anymore.

Let us decompose the interaction between polyhedra into a set of interactions between faces. Polygonal faces are assumed to be decomposed into triangles[1].

[1]In the following, *features* of a triangle are the triangular (open)face, the three edges and the three vertices.

The continuous motion between polyhedra, $\mathcal{O}_1$ and $\mathcal{O}_2$ can be achieved if each triangle of $\mathcal{O}_1$ moves with continuous velocity against each triangle of $\mathcal{O}_2$. Therefore we can focus on an interaction between triangles, $\mathcal{T}_1$ and $\mathcal{T}_2$. The continuous motion between $\mathcal{T}_i$ and $\mathcal{T}_j$ can be achieved if each edge of $\mathcal{T}_1$ moves with continuous velocity against $\mathcal{T}_2$ and each edge of $\mathcal{T}_2$ moves with continuous velocity against $\mathcal{T}_1$. Finally the interaction between polyhedra is decomposed into interactions between an edge and a triangle as shown in Fig. 5. It means that the continuous interaction between polyhedra can be achieved if we can find a method to realize a continuous interaction between an edge and a triangle. In the same way, the continuous interaction between a robot and the environment which consists of several polyhedra respectively can be achieved if each polyhedron of the robot moves in continuous way against each polyhedron of the environment.

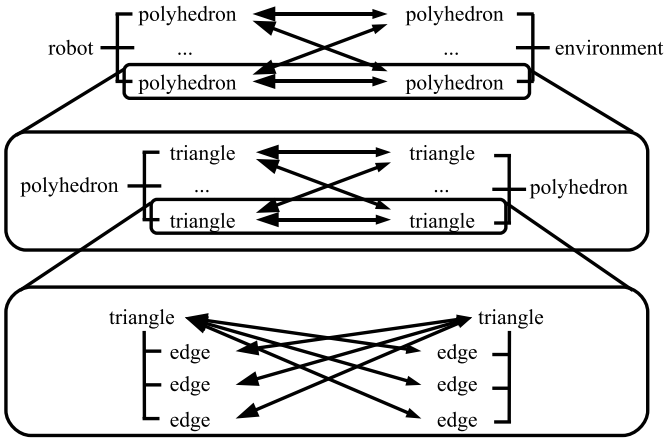

Fig. 5. Decomposition of interaction

C. Constraint generation using Voronoi regions

Next, let us find pairs of points to be constrained to realize the continuous interaction between an edge and a triangle. The pairs of points to be constrained depend on the Voronoi regions in which the edge lies. The Voronoi region is defined as follows.

Definition: *Voronoi region $\mathcal{VR}(X)$ for feature X. A Voronoi region associated with a feature X of a triangle is a set of points that are closer to X than any other feature.*

The Voronoi plane is also defined as follows.

Definition: *Voronoi plane $\mathcal{VP}(X,Y)$ between neighboring features X and Y. $\mathcal{VP}(X,Y)$ is the plane containing $\mathcal{VR}(X) \cap \mathcal{VR}(Y)$.*

Since a triangle consists of a face $\mathcal{F}$, three edges $\mathcal{E}_i (i = 1,2,3)$ and three vertices $\mathcal{V}_i (i = 1,2,3)$, 3D space around the triangle is separated into 7 Voronoi regions.

Case1 : The edge is in $\mathcal{VR}(\mathcal{F})$

The closest point jumps from one of end points of the edge to the other when the edge and the triangle are almost parallel. Therefore, two pairs, $(\mathcal{V}_1, \mathcal{V}_1')$ and $(\mathcal{V}_2, \mathcal{V}_2')$ are constrained, where $\mathcal{V}_1$ and $\mathcal{V}_2$ are end points of the edge. (a,b) denotes a pair of points, a and b and $\mathcal{V}_i' (i = 1,2)$ denotes a projection of $\mathcal{V}_i$ onto $\mathcal{F}$ along its normal vector. Any point on the edge can be the closest point when the edge and the triangle are parallel. But we don't need any additional pair since both end points are already constrained and they are also the closest points.

Case2 : The edge is in $\mathcal{VR}(\mathcal{E})$

The closest point jumps from one of end points of the edge to the other when the edge and $\mathcal{E}$ are almost parallel. So two pairs, $(\mathcal{V}_1, \mathcal{V}_1')$ and $(\mathcal{V}_2, \mathcal{V}_2')$ are constrained, where $\mathcal{V}_i' (i = 1,2)$ is a projected point of $\mathcal{V}_i$ onto $\mathcal{E}$. The closest points between the edge and $\mathcal{E}$ coincides with one of two pairs in some cases, but doesn't in other cases. Therefore, one more pair for the closest points is added. Three pairs are created as a consequence.

Case3 : The edge is in $\mathcal{VR}(\mathcal{V})$

The closest point moves continuously on the edge. So the pair of the closest points is constrained.

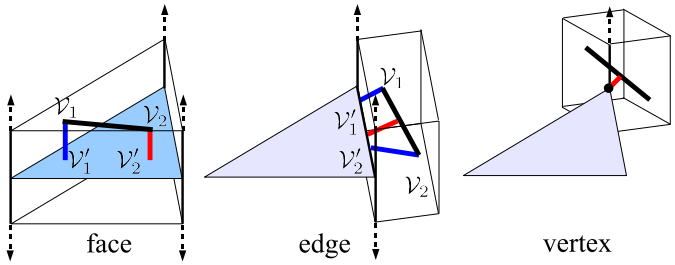

Fig. 6. Constraints generated between an edge and a triangle

So far, we considered cases the edge lies in one of the Voronoi regions of the triangle. But in most cases, the edge may move to another Voronoi region and lie in several Voronoi regions at the same time. Therefore, the edge is decomposed into several line segments again by clipping it with Voronoi planes.

Before this decomposition, we need to confirm that continuity of constraints is kept when an edge goes into another Voronoi region. When the edge moves between $\mathcal{VR}(\mathcal{F})$ and $\mathcal{VR}(\mathcal{E})$, continuity of constraints are maintained since end points of decomposed line segments are on $\mathcal{VP}(\mathcal{F},\mathcal{E})$ and they produce the same constraints. However, in other cases, constraints may appear or disappear suddenly. An example is shown in Fig. 7. The edge is moving from $\mathcal{VR}(\mathcal{V})$ to $\mathcal{VR}(\mathcal{E})$. When an end point of the edge touches $\mathcal{VP}(\mathcal{V},\mathcal{E})$ and a new constraint appears suddenly. In the reverse case, the constraint disappears suddenly.

This discontinuity can be solved by adding two more constraints on both end points of the edge when it is in $\mathcal{VR}(\mathcal{V})$.

Finally, the algorithm to pick up pairs of points to be constrained is described as in Algorithm 1-4.

Algorithm 1 decomposes interaction between polyhedra, $\mathcal{O}_1$ and $\mathcal{O}_2$ into interactions between triangles. Function DISTANCE_BOUND($\mathcal{O}_1$, $\mathcal{O}_2$, d_i) filters out such pairs of triangles that distances between triangles are bigger than d_i. This function is very important to improve efficiency and it can

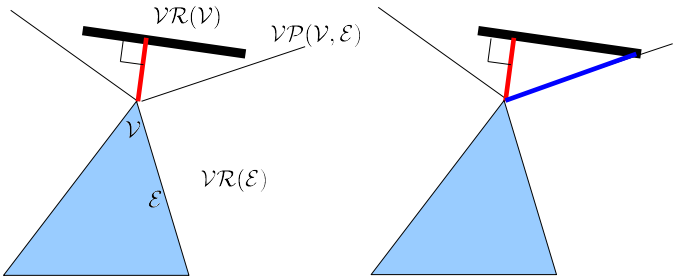

Fig. 7. Appearance/disappearance of a constraint

Algorithm 1 PAIRS_POLYHEDRA($\mathcal{O}_1$, $\mathcal{O}_2$)

```
pairs ← ∅
triangle_pairs ←DISTANCE_BOUND(O_1, O_2, d_i)
for all (T_1, T_2) ∈ triangle_pairs do
  pairs ← pairs ∪ PAIRS_TRIANGLES(T_1, T_2)
end for
return pairs
```

be implemented easily using techniques for collision detection such as OBB-Tree.

Algorithm 2 PAIRS_TRIANGLES($\mathcal{T}_1$, $\mathcal{T}_2$)

```
pairs ← ∅
for all E_2 ∈EDGES(T_2) do
  pairs ← pairs ∪ PAIRS_EDGE_TRIANGLE(E_2, T_1)
end for
for all E_1 ∈EDGES(T_1) do
  pairs ← pairs ∪ PAIRS_EDGE_TRIANGLE(E_1, T_2)
end for
return pairs
```

Algorithm 2 decomposes interaction between triangles, $\mathcal{T}_1$ and $\mathcal{T}_2$ into interactions between an edge and a triangle. Function EDGE($\mathcal{T}$) returns the set of edges that compose $\mathcal{T}$.

Algorithm 3 decomposes interaction between an edge and a triangle into interactions between a line segment and a triangle. A function VORONOI_CLIP($\mathcal{E}$, $\mathcal{VR}(f)$) clips a part of $\mathcal{E}$ which is in $\mathcal{VR}(f)$.

Algorithm 4 generates pairs of points to be constrained. Function VERTICES($\mathcal{S}$) returns the set of end points of $\mathcal{S}$, a function PARALLEL($\mathcal{S}_1$, $\mathcal{S}_2$) checks two segments are parallel or not, and a function CLOSEST_PAIR(A, B) computes the closest points between geometric elements A and B.

After $pairs$ are computed, *velocity damper* is inserted between each pair of points if the distance between those points is smaller than d_i.

In this procedure, constraints are generated on all end points of line segments. Since end points are shared by several line segments, duplicated pairs are generated. A duplicated pair is also generated when the closet point coincides with one of the end points. Therefore, we need to check duplication before adding a new pair.

Algorithm 3 PAIRS_EDGE_TRIANGLE($\mathcal{E}$, $\mathcal{T}$)

```
pairs ← ∅
for all f ∈ {F, E_1, E_2, E_3, V_1, V_2, V_3} do
  S ←VORONOI_CLIP(E, VR(f))
  if S then
    pairs ← pairs ∪ PAIRS_SEGMENT_FEATURE(S, f)
  end if
end for
return pairs
```

Algorithm 4 PAIRS_SEGMENT_FEATURE($\mathcal{S}$, f)

```
pairs ← ∅
for all V ∈VERTICES(S) do
  pairs ← pairs ∪ {CLOSEST_PAIR(V, f)}
end for
if f ∈ {V_1, V_2, V_3} then
  pairs ← pairs ∪ {CLOSEST_PAIR(S, f)}
else if f ∈ {E_1, E_2, E_3} then
  if not PARALLEL(S, f) then
    pairs ← pairs ∪ {CLOSEST_PAIR(S, f)}
  end if
end if
return pairs
```

D. Sources of discontinuous robot velocities

Solving Problem 2 is equivalent to finding the closest point between the desired task velocity $\underline{\dot{\tau}}$ and the space of admissible task velocities $\mathcal{S}_{\dot{\tau}}$(light blue region in Fig.8). The robot velocities which satisfy all the linear inequality constraints(yellow regions in Fig.8) exist in the convex subspace(orange region in Fig.8). It is projected into $\mathcal{S}_{\dot{\tau}}$ by $\boldsymbol{J}_\tau$. And when the task is defined in the lower dimensional space than the robot velocity space, a point in the task velocity space corresponds to the subspace in the robot velocity space $\mathcal{S}_{\dot{q}}$(red region in Fig.8).

There are three kinds of sources of discontinuous robot velocities.

1) If a constraint changes discontinuously, the shape of $\mathcal{S}_{\dot{\tau}}$ also does. As the result, the robot velocity might change discontinuously. This source can be removed using the constraint generation method described in this section.
2) Even all the constraints move continuously in the robot velocity space, discontinuous robot velocities might be generated when the task is defined in the lower dimensional space than the robot velocity space. This situation is similar with solving inverse kinematics at singular postures. SR-Inverse[18] is proposed to prevent the robot velocity from going to infinity. The same thing can be realized by modifying the objective function of Problem 2 as follows:

$$\|\boldsymbol{J}_\tau(\boldsymbol{q})\dot{\boldsymbol{q}} - \underline{\dot{\boldsymbol{\tau}}}\|^2 + \lambda\|\dot{\boldsymbol{q}}\|^2 \tag{4}$$

where λ is a positive coefficient for adjusting strength

of the penalty for big velocities. The added second term put a damping effect to the robot velocity and remove discontinuities.

3) Every point in $\mathcal{S}_{\dot{q}}$ is the optimal solution of Problem 2. A point in $\mathcal{S}_{\dot{q}}$ might be chosen discontinuously by the optimization algorithm. The modified objective function changes $\mathcal{S}_{\dot{q}}$ into a point and this discontinuity is also removed.

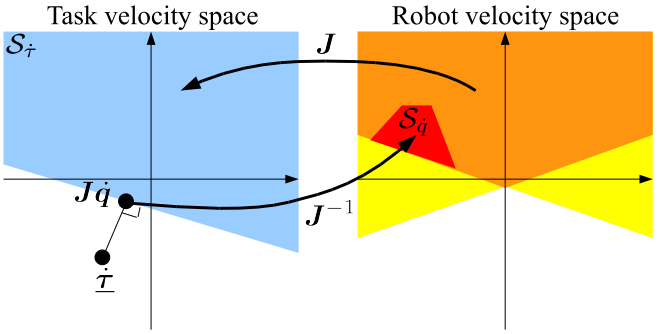

Fig. 8. Mappings between velocity spaces

IV. Examples

A. Collision Avoidance of a Single Object

The proposed method is applied to the example shown in Section III-A to check that a collision avoidance motion with continuous velocity can be generated. Fig. 9 shows snapshots of the generated motion and Fig. 10 shows results corresponding to Fig. 4. The minimum distance converges to d_s and there is no collision. And linear velocity along Y axis and angular velocity changes in continuous way. In this case, a constraint is generated when C_1 enters into *influence zone* and one more constraint is added when C_2 does.

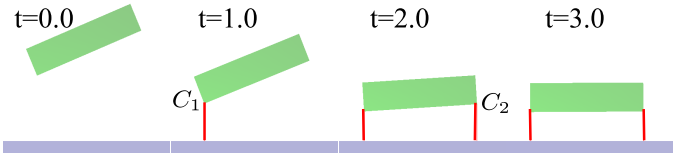

Fig. 9. Snapshots of interaction between a rectangle and the ground

Fig. 11 shows another example which includes a concave shape and a shape with a hole. A "L" shape object which consists of 12 triangles passes through a torus which consists of 512 triangles. It is impossible for existing methods to plan a collision free motion between these kinds of shapes without approximations.

Top left of Fig. 12 shows the minimum distance. The distance between non-strictly convex objects is continuous and piecewise smooth. Since the closest points are constrained at any time, the condition in Eq.(3) holds and the minimum distance between objects never become smaller than d_s. Top right of Fig. 12 shows the number of pairs of triangles which are found by DISTANCE_BOUND(). The total number of

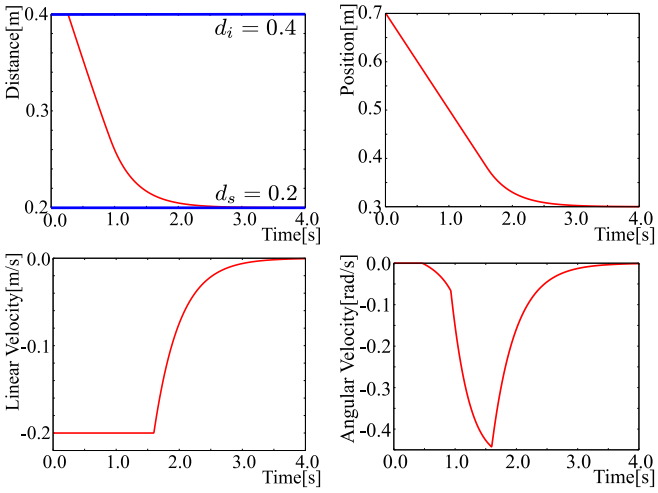

Fig. 10. Improved results of an example #1

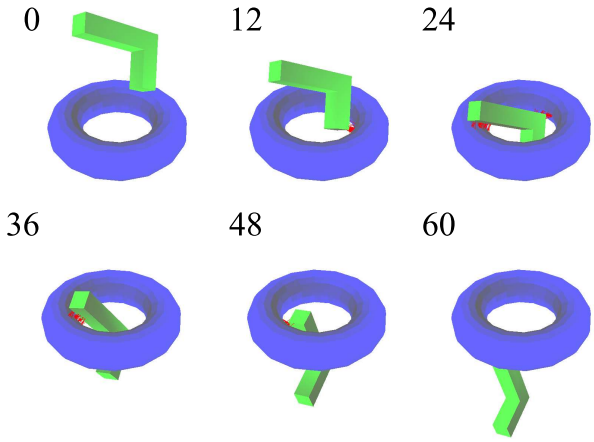

Fig. 11. Snapshots of an interaction between a concave shape and a torus

pairs of triangles is $12 \times 512 = 6144$ whereas it is less than 100. Bottom left of Fig. 12 shows the number of constraints. It changes along time since they are activated only if the distance between points is smaller than d_i. In this example, 117 constraints are generated at a maximum. Bottom right of Fig. 12 shows the computational time. It is measured on a PC equipped with Intel Core 2 Duo 2.13[GHz]. It is almost proportional to the number of pairs of triangles.

B. Collision Avoidance of a Humanoid Robot

The proposed method is also applied to a humanoid robot HRP-2. The task of the robot is to move its left hand to the specified position using its whole body. In addition to constraints for collision avoidance, three kinds of constraints are added. (1) A relative transformation of feet is kept while reaching since the robot stands on both legs. (2) A horizontal position of the center of mass is kept to keep static balance(We assume the center of mass is above the support polygon at the initial configuration.). (3) Joint angles are kept in their movable ranges. As a result, this collision-free reaching task

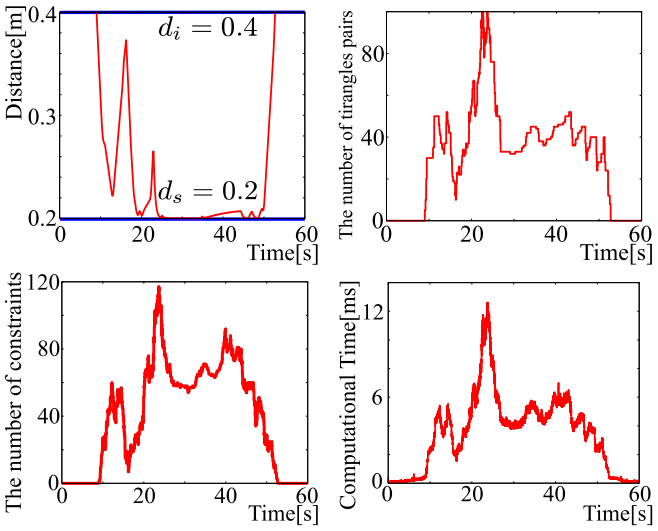

Fig. 12. Results of an example #2

can be achieved by solving the following QP problem:

$$\text{minimize} \quad \|\boldsymbol{J}_{hand}\dot{\boldsymbol{q}} - \underline{\dot{\boldsymbol{p}}}\|^2 + \lambda\|\dot{\boldsymbol{q}}\|^2 \tag{5a}$$

$$\text{subject to} \quad (\dot{\boldsymbol{q}}|\boldsymbol{J}_{dist_j}^T\boldsymbol{n}) \geq -\xi\frac{d_j - d_s}{d_i - d_s}, \quad \text{for } j \in \{1, \cdots, n_c\}, \tag{5b}$$

$$\boldsymbol{J}_{feet}\dot{\boldsymbol{q}} = \boldsymbol{0}, \tag{5c}$$

$$\boldsymbol{J}_{com}\dot{\boldsymbol{q}} = \boldsymbol{0}, \tag{5d}$$

$$vmax_j(q_j) \geq \dot{q}_j \geq vmin_j(q_j), \quad \text{for } j \in \{1, ..., n_{dof}\}. \tag{5e}$$

$\boldsymbol{J}_{hand}, \boldsymbol{J}_{com}, \boldsymbol{J}_{feet}$ and $\boldsymbol{J}_{dist_j}$ are Jacobian matrices for the hand position(3DOF), for the horizontal position of the center of mass(2DOF), for the relative transformation between feet(6DOF) and for the distance between jth pair of points(1DOF) respectively. Inequality (5b) defines geometric constraints to avoid collision where n_c is the number of pairs of points to be constrained. In this example, d_i, d_s and ξ are set to 0.05[m], 0.03[m] and 0.5[m/s] respectively. Equality (5c) defines a kinematic constraint to keep the relative transformation between feet and Equality (5d) does a dynamic one to keep the center of mass on a vertical line. Inequality (5e) defines kinematic constraints for joint limits, where n_{dof} is the number of joints. The joint velocity is limited when the joint angle comes close to its limit. The limit is also computed by *velocity damper*:

$$vmax_j(q_j) = \begin{cases} \xi\dfrac{(q_j^+ - q_j) - q_s}{q_i - q_s} & \text{if } q_j^+ - q_j \leq q_i, \\ v_j^+ & \text{otherwise} \end{cases} \tag{6}$$

$$vmin_j(q_j) = \begin{cases} -\xi\dfrac{(q_j - q_j^-) - q_s}{q_i - q_s} & \text{if } q_j - q_j^- \leq q_i, \\ v_j^- & \text{otherwise} \end{cases} \tag{7}$$

where q_j^+ and q_j^- are physical upper bound and lower bound of joint angle and v_j^+ and v_j^- are those of joint velocity of jth joint respectively. In this example, q_i, q_s and ξ are set to 0.2[rad], 0.02[rad] and 0.3[rad/s] respectively.

Fig. 13 shows snapshots of the generated motion. The frame 1 shows an initial configuration. A trapezoid in front of the robot is an obstacle and a small box close to the robot foot indicates the target position of the hand. At the frame 2, the left upper arm comes close to the obstacle, and constraints for collision avoidance become active. The left shoulder avoids the obstacle from the frame 2 to 4 and the head directs upward to avoid collision in the frame 5. We can see that the whole body is fully used to avoid collision and achieve the task simultaneously. If we don't include Inequality (5b), the left shoulder collides with the table as shown in the frame 3'.

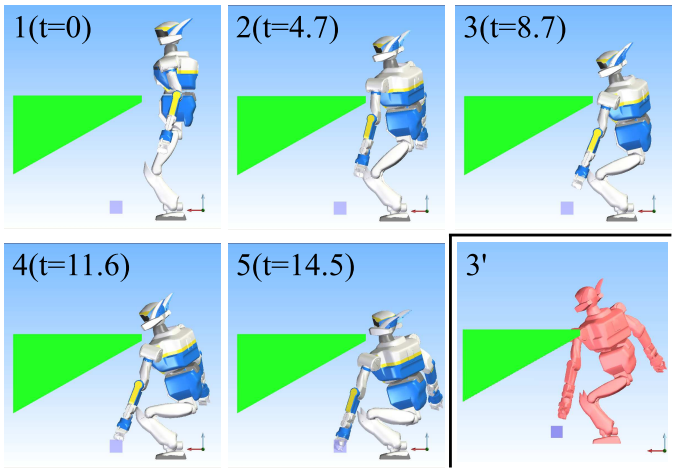

Fig. 13. Snapshots of "pick up an object under the table" motion

Fig. 14 shows the number of constraints(left) and the computational time(right). The average computational time for one step is about 100[ms] on the same PC with previous examples. We can get this motion(the total duration is 14[s]) in 28[s] when we select 50[ms] as the time step. The computational time is not so long in this case since shape of the obstacle is very simple. It is expected that it becomes longer drastically if the number of obstacles and complexity of shapes increase. If the number of pairs of triangles is too many to get a solution within reasonable time, we can reduce the number of pairs by using thin *influence zone* or simplified shapes. Even in the latter case, simplified shapes can be non-strictly convex.

V. Conclusion

In this paper, we proposed a local method for collision avoidance between non-strictly convex polyhedra with continuous velocities. The continuity is achieved by decomposing the interaction between polyhedra into a set of interactions between line segments clipped by Voronoi regions and triangles and constraining several pairs of points on those geometrical elements. These pairs of points can be used to define constraints in other collision avoidance methods like [9].

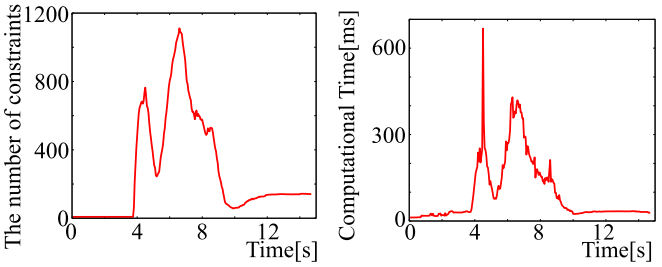

Fig. 14. Results of a picking up motion example

In case of a humanoid robot, dynamic stability of the robot is a very critical issue. But it is not guaranteed by our method since ZMP [19] is not constrained directly. A generated motion can be stable at least if it is executed with sufficiently small speed since the center of mass is constrained above its support polygon. In order to get a fast and dynamically stable motion, we are trying to use the motion as an initial path of an optimization method.

ACKNOWLEDGMENT

Researchers from LAAS-CNRS are partly supported by the project ANR RNTL PerfRV2. We gratefully acknowledge AEM Design, developer of a powerful QP solver, C-FSQP[20].

REFERENCES

[1] B. Faverjon and P. Tournassoud, "A Local Based Approach for Path Planning of Manipulators With a High Number of Degrees of Freedom," in *Proc. of IEEE International Conference on Robotics and Automation*, 1987, pp. 1152–1159.

[2] C.Helguera and S.Zeghloul, "A Local-based Method for Manipulators Path Planning in Heavy Cluttered Environments," in *Proc. of International Conference on Robotics & Automation*, 2000, pp. 3467–3472.

[3] M. Schlemmer and G. Gruebel, "A Distance Function and its Gradient for Manipulator On-Line Obstacle Detection and Avoidance," in *Proc. of International Conference on Advanced Robotics*, 1997, pp. 427–432.

[4] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion Planning for Humanoid Robots Under Obstacle and Dynamic Balance Constraints," in *Proc. of International Conference on Robotics & Automation*, 2001, pp. 692–698.

[5] J. James J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *In Proc. IEEE International Conference on Robotics & Automation*, 2000, pp. 995–1001.

[6] S.Kagami, F.Kanehiro, Y.Tamiya, M.Inaba, and H.Inoue, "AutoBalancer: An Online Dynamic Balance Compensation Scheme for Humanoid Robots," in *Proc. of the Fourth International Workshop on the Algorithmic Foundations on Robotics(WAFR'00)*, 2000.

[7] J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Self-Collision Detection and Prevention for Humanoid Robots," in *Proc. of International Conference on Robotics and Automation*, 2002, pp. 2265–2270.

[8] M.Mirtich, "VClip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177–208, 1998.

[9] L. Sentis and O. Khatib, "A Whole-Body Control Framework for Humanoids Operating in Human Environments," in *Proc. of International Conference on Robotics and Automation*, 2006, pp. 2641–2648.

[10] K. Okada and M. Inaba, "A Hybrid Approach to Practical Self Collision Detection System of Humanoid Robot," in *Proc. of International Conference on Intelligent Robots and Systems*, 2006, pp. 3952–3957.

[11] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-Time Collision Avoidance with Whole Body Motion Control for Humanoid Robots," in *Proc. of International Conference on Intelligent Robots and Systems*, 2007, pp. 2053–2058.

[12] A. Escande, S. Miossec, and A. Kheddar, "Continuous gradient proximity distance for humanoids free-collision optimized-postures," in *Proc. of International Conference on Humanoid Robots*, 2007.

[13] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[14] M. C. Lin and J. F. Canny, "A Fast Algorithm for Incremental Distance Calculation," in *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, 1991, pp. 1008–1014.

[15] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast Proximity Queries with Swept Sphere Volumes," in *Proc. of International Conference on Robotics and Automation*, 2000, pp. 3719–3726.

[16] S. A. Ehmann and M. C. Lin, "Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching," in *Proc. of International Conference on Intelligent Robots and Systems*, 2000, pp. 2101–2106.

[17] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi, "Humanoid Robot HRP-2," in *Proc. of IEEE International Conference on Robotics and Automation*, 2004, pp. 1083–1090.

[18] Y.Nakamura and H.Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *ASME, Transactions, Journal of Dynamic Systems, Measurement, and Control*, vol. 108, pp. 163–171, 1986.

[19] M.Vukobratović and B.Borovac, "Zero-moment point –thirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004.

[20] C. Lawrence, J. L. Zhou, and A. L. Tits, *User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints*.