Robotics and Computer–Integrated Manufacturing 49 (2018) 98–104

Contents lists available at ScienceDirect

Robotics and Computer–Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Dynamic obstacle avoidance for manipulators using distance calculation and discrete detection

Dong Han[a,*], Hong Nie[a], Jinbao Chen[a], Meng Chen[b]

[a] State Key Laboratory of Mechanics and Control of Mechanical Structures, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, People's Republic of China

[b] Aerospace System Engineering Shanghai, Shanghai, People's Republic of China

ARTICLE INFO

Keywords:
Dynamic obstacle avoidance
Distance calculation
Collision detection
Redundant manipulators
Human-robot interaction

ABSTRACT

In order to avoid dynamic obstacle timely during manufacturing tasks performed by manipulators, a novel method based on distance calculation and discrete detection is proposed. The nearest distances between the links of a manipulator and the convex hull of an arbitrarily-shaped dynamic obstacle obtained from Kinect-V2 camera in real-time are calculated by Gilbert–Johnson–Keerthi algorithm, and the minimum one is defined as the closest distance between the manipulator and the obstacle. When the closest distance is less than a safe value, whether the dynamic obstacle is located in the global path of the manipulator is determined by improved discrete collision detection, which can adjust detection step-size adaptively for accuracy and efficiency. If the obstacle will collide with the manipulator, set a local goal and re-plan a local path for the manipulator. The proposed method is implemented in Robot Operating System (ROS) using C++. The experiments indicate that the proposed method can perform safe and timely dynamic avoidance for redundant manipulators in human-robot interaction.

1. Introduction

Avoiding dynamic obstacle accurately and timely for robots is one of the major issues of robotic intelligent manufacture in unstructured environment [1–3], and it can ensure safety in the scenario of human-robot interaction [4]. Hence much research has been carried out into collision avoidance beforehand. In [5], it presented a neural-network path planning algorithm based on reinforcement learning for avoiding obstacles in complex unknown environments. In [6], the authors proposed a Weighted Least Norm method to avoid collision with a moving obstacle for redundant manipulators. Luo et al. [7–8] adopted repulsive vectors to achieve active whole-arm collision avoidance for 7-DoF redundant manipulator. However, in these methods, the manipulators and obstacles are simplified as primary geometric elements like lines, points and spheres, which make the collision detection accuracy decrease.

In order to response more accurately to complex dynamic obstacles in advance, lots of dynamic collision detection algorithms have been proposed, which can be classified into two categories: discrete collision detection (DCD) and continuous collision detection (CCD). Generally speaking, DCD algorithms check for collisions by sampling several configurations in the planned trajectories and then adopting static-collision-detection algorithms at these configurations. Consequently, they have a high detection speed [9] and are applied widely in real-time operating, but they may miss a collision due to the gaps between two adjacent configurations, which is known as the tunneling problem. Li et al. [10] proposed optimum discrete interval according to the minimum distance between two objects for improving DCD algorithms.

Compared with DCD algorithms, CCD algorithms can completely overcome the undetected problem by interpolating a continuous motion between successive configurations and detecting collision along the whole of that motion [11]. However, the computation cost of CCD is obviously increased and many researchers have focused on improving the efficiency [12–13]. Redon et al. [14] presented a fast CCD method for articulated models that used an "arbitrary in-between motion" to interpolate motions between two successive time steps and checked collision on the generated path by OBB-trees. Tang et al. [15] also implemented the CCD of articulated models based on conservative advancement. These algorithms need multiple static detections over a time interval. Unlike the above methods, swept volume technology, which is the whole of all points that belong to the trace of an arbitrary moving object during a period, has been applied in more accurate collision detection [16–17], but creation of swept volumes is very complex, especially for redundant manipulators. Another approach of CCD is based on distance calculation. At a broad level, the closest distance algorithms of convex bodies can be categorized into two main classes: (1) Voronoi-regions-based methods, such as Lin–Canny (LC) algorithm [18], V-Clip [19] and SWIFT++ [20]; (2) simplex-based methods, like Gilbert–Johnson–Keerthi (GJK) algo-

* Corresponding author.
E-mail address: han_dongnuaa@126.com (D. Han).

http://dx.doi.org/10.1016/j.rcim.2017.05.013
Received 22 November 2016; Received in revised form 28 May 2017; Accepted 30 May 2017
Available online 20 June 2017
0736-5845/

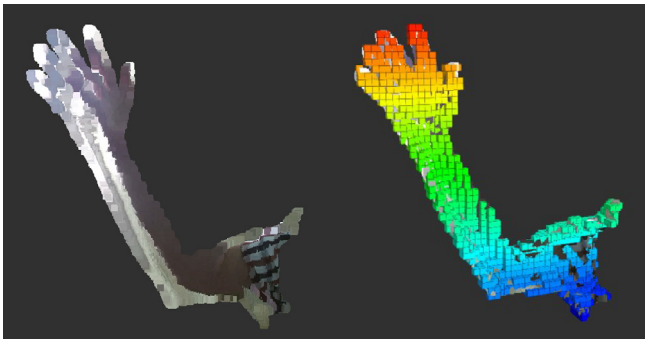

Fig. 1. The point cloud image (left) and the 3D octree map (right) of a human arm.

rithm [21]. The GJK-based algorithms, by contrast, are more robust than LC [22] and have simpler stored data structures [23]. Consequently, the GJK algorithm has been widely applied in distance calculation and fast collision detection [24–26], but is not for non-convex bodies.

Considering the pros and cons of the above algorithms, this paper presents a new dynamic- obstacle-avoidance method for redundant manipulators which is divided into two stages. The first stage utilizes the GJK algorithm to estimate the closest distance between a redundant manipulator and a dynamic obstacle, wherein the links of the manipulator are regarded as cylinders and the obstacle model from Kinect-V2 in real-time is replaced with its convex hull. Apparently, cylinders and convex hulls are much closer to the real manipulators and obstacles than those primary geometric elements. When the closest distance is less than a safe value, the second stage is started to accurately check whether or not the complex obstacle will collide with the manipulator through DCD algorithm. In order to overcome the tunneling problem of DCD, the step-size for checking can be adjusted adaptively according to previous results. Therefore, the proposed method is able to improve the accuracy and efficiency of dynamic obstacle avoidance aiming at redundant manipulators. Baxter robot is used to verify the performance of the proposed method in ROS platform, where Baxter is an industrial robot with two 7-DOF arms built by Rethink Robotics [27–28] and ROS is a collection of software frameworks for helping software developers create robot applications more efficiently [29].

In the following, Section 2 presents the method used to build dynamic obstacle models. Section 3 then describes the dynamic-collision-avoidance method using distance calculation and improved DCD and Section 4 describes the final experimental results. The results are discussed in Section 5. Finally, a conclusion is given in Section 6.

2. Dynamic obstacle models

Generally, a planning scene of manipulators includes static obstacle models for global planning and dynamic obstacle models for local planning. The real-time depth images of the scene are gathered by Kinect-V2 camera and then converted to 3D point cloud images. As for a dynamic obstacle within certain range, its point cloud is obtained by filtering out all points belonging to static obstacle models and manipulators from the totality and then converted to a 3D octree map [30], as shown in Fig. 1.

In Fig. 1, the scattered points are divided into different nodes of octree according to their positions. Therefore the occupied voxels of the 3D map are well-organized by octree structure, which can reduce memory requirement with lower accuracy. The Flexible Collision Library (FCL) proposed in [31] is used for carrying out fast static collision detection between octree maps and other rigid models in this paper.

Obviously, octree maps can not be directly used as the input of the GJK algorithm for computing distances, because GJK is designed for convex bodies and octree maps are not always convex. In order to solve this problem, a point set, which consists of the center points of each voxel in a octree map, is defined, and then the convex hull of this point set is calculated by the QuickHull algorithm [32]. Although this method may expand the volume of obstacles, it can greatly decrease the calculation time.

Before running the QuickHull algorithm, the inner voxels of octree map should be deleted for promoting the efficiency of convex hull calculation, because their center points will never be the vertices of convex hull. As mentioned above, the organization structure of the voxels is an octree, and thus each voxel has a unique index in octree, as follows:

$$\boldsymbol{Ind}_i = \left[Ind_{ix},\ Ind_{iy},\ Ind_{iz}\right]^{\mathrm{T}} \tag{1}$$

where Ind_{ix}, Ind_{iy}, Ind_{iz} are the index values of the ith voxel in X, Y, Z directions, respectively. Consequently, the key of deleting inner voxels is to classify all voxels in the map by index values:

(1) The voxels with the equal index values in X and Y directions are classified as a class, and only the voxels with the maximum and minimum index values in Z direction are retained;
(2) Similarly, the remaining voxels are categorized in turn according to the index values in Y and Z directions or X and Z directions, and as many inner voxels as possible are eliminated, as shown in Fig. 2.

In order to speed up the classification, the indexes are stored in a *multimap* which is an associative container based on red-black tree in Standard Template Library (STL). In a *multimap*, the elements are formed by a combination of a key value and a mapped value. The key values are generally used to sort and uniquely identify the elements, and the mapped values store the content associated to this key. It means a key value may have multiple mapped values that are stored in adjacent memory. According to this characteristic, the index values in two directions are combined into a key value and another one is the mapped value. Then insert them into a *multimap* for classifying the indexes automatically and the time complexity is $O(n\log n)$ due to red-black tree structure (n is the number of voxels) [33].

The performance of convex-hull computation with deleting the inner voxels is compared with that without deleting, as shown in Fig. 3. Therefore, by using the above method, the points input to the QuickHull algorithm can be reduced by up to 74%, and the total time of convex-hull calculation can also be efficiently decreased by 39% in maximum.

The results of constructing convex hulls are shown in Fig. 4, where the non-convex models are converted to convex models for distance calculation.

In summary, the dynamic obstacle obtained from Kinect-V2 is approximated as the convex hull of its octree map by mathematical modeling. It is essential for real-time dynamic collision avoidance in the next section.

3. Dynamic collision avoidance

3.1. Distance calculation based on GJK

The collision models of a redundant manipulator can be regarded as three cylinders, as shown in Fig. 5. Therefore, the closest distance from the manipulator to an obstacle is the minimum one among the distances between the cylinders and the obstacle. As mentioned in Section 2, convex hulls are used for describing complex obstacle models. So the following focuses on calculating the distance between a convex hull and a cylinder in three-dimensional space with the GJK algorithm.

The key of GJK is to get the support points of a convex set. The support point $\boldsymbol{s}$ in a given direction $\boldsymbol{d}$ satisfies the following equation:

$$\boldsymbol{d} \cdot \boldsymbol{s} = \max\{\boldsymbol{d} \cdot \boldsymbol{p} | \boldsymbol{p} \in C\} \tag{2}$$

where C is an arbitrary convex set.

The support points of a convex hull that is usually a convex polyhedron can be obtained by traversing all vertices of the convex hull. Hill climbing is capable of speeding up the search for the support points, especially for convex polyhedrons with many vertices, but this method needs to know all the adjacent vertices of each vertex [34].

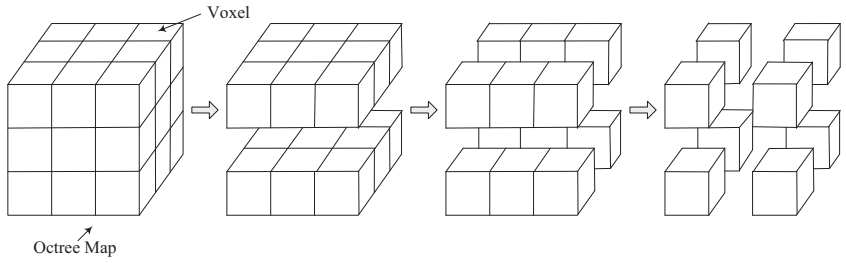

Fig. 2. Deleting the inner voxels in turn.

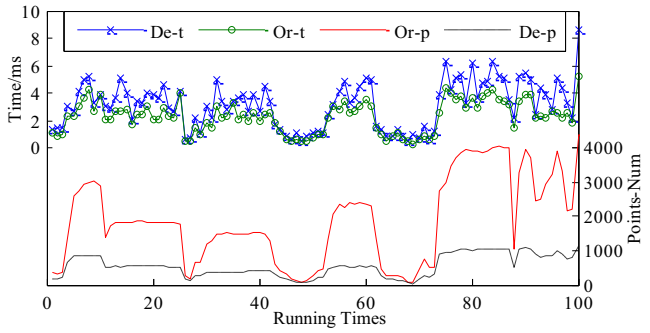

Fig. 3. The results of convex-hull computation, where Or-p represents the number of the initial points before deleting the inner voxels, De-p is the number of the points after deleting the inner voxels, Or-t is the computation time of convex hulls without deleting the inner voxels, De-t is the computation time with deleting the inner voxels (including the time cost of deleting the inner voxels).

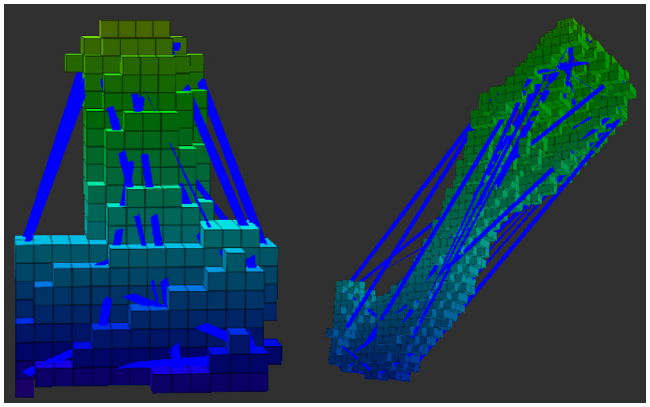

Fig. 4. Constructing the convex hulls of non-convex models.

In order to get the support points of a cylinder with arbitrary position and orientation in 3D space, a cylinder coordinate system $\{O_c\}$ is established, as shown in Fig. 6(a), where the origin O_c is the center point of the cylinder, Z_c-axis is the axis of the cylinder. In $\{O_c\}$, assuming an arbitrary direction vector ${}^c\boldsymbol{d}=({}^cd_x, {}^cd_y, {}^cd_z)$, the support point of the cylinder can be calculated by analyzing the geometric relationship in Fig. 6(b), as following:

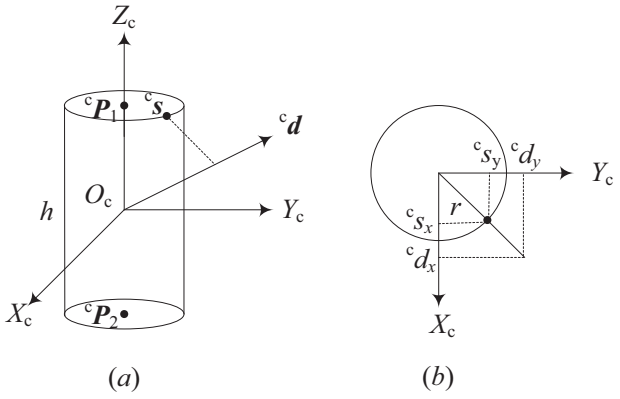

Fig. 6. The parameters of a cylinder in 3D space, where (a) is the cylinder coordinate system, (b) shows the geometric relationship between a direction vector and its support point.

$$S_C({}^c\boldsymbol{d}) = \left(r\frac{{}^cd_x}{\mu}, r\frac{{}^cd_y}{\mu}, \frac{h}{2}\mathrm{sgn}({}^cd_z)\right)^{\mathrm{T}} \tag{3}$$

where r is the radius, h is the height, $\mu = \sqrt{{}^cd_x{}^2 + {}^cd_y{}^2}$, $\mathrm{sgn}(x) = \begin{cases} -1 & x<0 \\ 0 & x=0. \\ 1 & x>0 \end{cases}$

The position-orientation and the size of a cylinder in the base coordinate system $\{O_b\}$ are solely determined on the radius (r) and the center points $({}^b\boldsymbol{P}_1, {}^b\boldsymbol{P}_2)$ of the two cylinder's undersides, as shown in Fig. 7. Given a direction vector ${}^b\boldsymbol{d}$ in $\{O_b\}$, the cylinder's support point in this direction can be calculated as follows:

(1) Transform the ${}^b\boldsymbol{d}$-direction vector from $\{O_b\}$ to $\{O_c\}$ and get ${}^c\boldsymbol{d}$ vector in $\{O_c\}$;
(2) Calculate the support point ${}^c\boldsymbol{s}$ of the cylinder along ${}^c\boldsymbol{d}$-direction in $\{O_c\}$ based on Eq. (3);

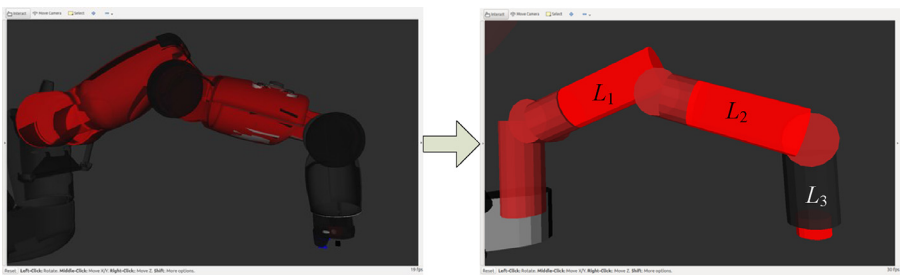

Fig. 5. The redundant manipulator (left) and its collision models (right).

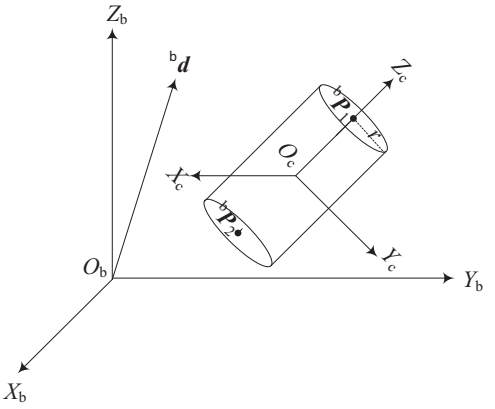

Fig. 7. The base coordinate system $\{O_b\}$ and the cylinder coordinate system $\{O_c\}$.

(3) Transform $^c\boldsymbol{s}$ from $\{O_c\}$ to $\{O_b\}$ for getting the support point $^b\boldsymbol{s}$ of the cylinder along $^b\boldsymbol{d}$- direction in $\{O_b\}$.

The transformation relationship between the base coordinate system $\{O_b\}$ and the cylinder coordinate system $\{O_c\}$ can be computed by the following equation:

$$^c\boldsymbol{P} = {}^c_b\boldsymbol{T} \cdot {}^b\boldsymbol{P} \tag{4}$$

where ${}^c_b\boldsymbol{T}$ represents the homogeneous transformation matrix, which can be divided into two parts: a translation matrix and a rotation matrix, as follows:

$${}^c_b\boldsymbol{T} = {}^c_{c'}\boldsymbol{R}(\alpha, \beta, \gamma)\,{}^{c'}_b\boldsymbol{D}(-{}^b\boldsymbol{P}_0) \tag{5}$$

where $^b\boldsymbol{P}_0$ is the position of the origin of $\{O_c\}$ in $\{O_b\}$, $^b\boldsymbol{P}_0 = (^b\boldsymbol{P}_1 + {}^b\boldsymbol{P}_2)/2$; α, β and γ are respectively the Euler angles rotating in X–Y–Z order. Rotating around Z_c-axis of $\{O_c\}$ has no effects on the position of the support points in $\{O_b\}$, because Z_c-axis is coincident with the cylinder's axis. Just let $\gamma = 0$. Eq. (5) can be expressed as:

$$\begin{aligned}
{}^c_b\boldsymbol{T} &= {}^c_{c'}\boldsymbol{R}(\alpha, \beta, 0)\,{}^{c'}_b\boldsymbol{D}(-{}^b\boldsymbol{P}_0) \\
&= \begin{bmatrix} c\beta & 0 & s\beta & 0 \\ s\alpha s\beta & c\alpha & -s\alpha c\beta & 0 \\ -c\alpha s\beta & s\alpha & c\alpha c\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -{}^bP_{0x} \\ 0 & 1 & 0 & -{}^bP_{0y} \\ 0 & 0 & 1 & -{}^bP_{0z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c\beta & 0 & s\beta & -c\beta\,{}^bP_{0x} - s\beta\,{}^bP_{0z} \\ s\alpha s\beta & c\alpha & -s\alpha c\beta & -s\alpha s\beta\,{}^bP_{0x} - c\alpha\,{}^bP_{0y} + s\alpha c\beta\,{}^bP_{0z} \\ -c\alpha s\beta & s\alpha & c\alpha c\beta & c\alpha s\beta\,{}^bP_{0x} - s\alpha\,{}^bP_{0y} - c\alpha c\beta\,{}^bP_{0z} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{6}$$

where $cx = \cos(x)$, $sx = \sin(x)$. Obviously, the coordinates of $^b\boldsymbol{P}_1 = (^bP_{1x}, {}^bP_{1y}, {}^bP_{1z})^T$ in $\{O_c\}$ are $^c\boldsymbol{P}_1 = (0, 0, h/2)^T$. Plug the coordinates of these two points ($^b\boldsymbol{P}_1$ and $^c\boldsymbol{P}_1$) into Eq. (6), and get the equations as follows:

$$\begin{cases} c\beta(^bP_{1x} - {}^bP_{0x}) + s\beta(^bP_{1z} - {}^bP_{0z}) = 0 \\ s\alpha s\beta(^bP_{1x} - {}^bP_{0x}) + c\alpha(^bP_{1y} - {}^bP_{0y}) - s\alpha c\beta(^bP_{1z} - {}^bP_{0z}) = 0 \end{cases} \tag{7}$$

Solve Eq. (7):

$$\beta = \arctan\left(-\frac{^bP_{1x} - {}^bP_{0x}}{^bP_{1z} - {}^bP_{0z}}\right) \tag{8}$$

$$\alpha = \arctan\left(-\frac{^bP_{1y} - {}^bP_{0y}}{s\beta(^bP_{1x} - {}^bP_{0x}) - c\beta(^bP_{1z} - {}^bP_{0z})}\right) \tag{9}$$

Now the support point of an arbitrary cylinder along a given direction can be calculated by the above method in 3D space.

One of the key factors that impact the iteration number of GJK is the initial simplex with $d+1$ vertices at most for d-dimensional space. Usually, the initial simplex is constructed by choosing several vertices from two convex bodies. However, unsuitable initial simplex may lead to the failure of distance calculation based on GJK. So this paper adopts the following initial method:

(1) Calculate the center points $\boldsymbol{c}_{10}$ and $\boldsymbol{c}_{20}$ of two convex bodies, and set the initial direction vector as $\boldsymbol{d}_0 = \boldsymbol{c}_{20} - \boldsymbol{c}_{10}$;
(2) According to the above support functions, calculate the support points $\boldsymbol{s}_{11}$, $\boldsymbol{s}_{21}$ of two convex bodies in $\boldsymbol{d}_0$ direction and the support points $\boldsymbol{s}_{12}$, $\boldsymbol{s}_{22}$ in $-\boldsymbol{d}_0$ direction respectively (Fig. 8(a)). Let $\boldsymbol{S}_1 = \boldsymbol{s}_{11} - \boldsymbol{s}_{21}$ and $\boldsymbol{S}_2 = \boldsymbol{s}_{12} - \boldsymbol{s}_{22}$. Then $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are the two vertices of the initial simplex.
(3) Based on the above two vertices, compute a new direction vector: $\boldsymbol{d}_1 = (\boldsymbol{S}_2 - \boldsymbol{S}_1) \times (-\boldsymbol{S}_1) \times (\boldsymbol{S}_2 - \boldsymbol{S}_1)$;
(4) Calculate the support points $\boldsymbol{s}_{13}$, $\boldsymbol{s}_{23}$ in $\boldsymbol{d}_1$ direction (Fig. 8(b)). Let $\boldsymbol{S}_1 = \boldsymbol{s}_{13} - \boldsymbol{s}_{23}$, which is the third vertex of the initial simplex.

Now the initial simplex with three vertices is obtained. The geometric parameters of the convex bodies have been considered in the above process. Thus the drawbacks of building an initial simplex randomly can be avoided.

After solving the above key problems of using GJK, the distances between convex hulls and cylinders can be calculated based on GJK for getting the closest distance between a redundant manipulator and a dynamic obstacle. The calculated closest distances between the Baxter's manipulator and a human arm are shown in Fig. 9.

3.2. Improved DCD method

When the computed closest distance reaches a safe value, whether the dynamic obstacle blocks the manipulator's way needs to be determined. An improved DCD method whose step-size can be changed adaptively is developed. In this paper, "step-size" means the number of trajectory points in joint space that are obtained by planning algorithms based on Bidirectional Rapidly-exploring Random Tree (Bi-RRT) [35] with a certain extended step size.

The manipulator's current state is defined as s_0 when the closest distance is up to the safe value. The following is for determining whether the obstacle is in the global path:

(1) Provide an initial step-size;
(2) Get a state s_1 of the manipulator according to the initial step-size and s_0-state;
(3) Check for collisions between the manipulator in s_1-state and the dynamic obstacle using the static-collision-detection algorithm in FCL. If there is a collision, a local path is planned by Bi-RRT-based algorithm for avoiding the obstacle, whose local initial state is set as s_0-state and local goal state is selected from the unexecuted global path in which the manipulator will not collide with the dynamic obstacle. If not, go to the next step;
(4) According to the position of the nearest point on the manipulator, a link L_i that is most likely to collide with the obstacle is selected. Then the distance d of this link's end from s_0-state to s_1-state is calculated by forward kinematics, which is relatively easy.
(5) If $d > \delta$ (δ is a small positive which is decided by multiple tests), there is a gap between s_0-state and s_1-state, as shown in Fig. 10(a). Consequently the initial step-size should be decreased for avoiding the tunneling problem of DCD. It should be noted that checking for collisions between the two states is not needed because of the safe value. If $d < -\delta$, there is an overlap between the two states. So the initial step-size should be increased for enlarging the detection region, as shown in Fig. 10(b). Otherwise the initial step-size remains unchanged.
(6) Get a new state s_2 of the manipulator based on s_1-state and the adjusted step-size in 5).
(7) Detect whether the manipulator in s_2-state collides with the dynamic obstacle. If there is a collision, a local path is planned for avoiding the obstacle. If not, the manipulator keeps moving along the global path.

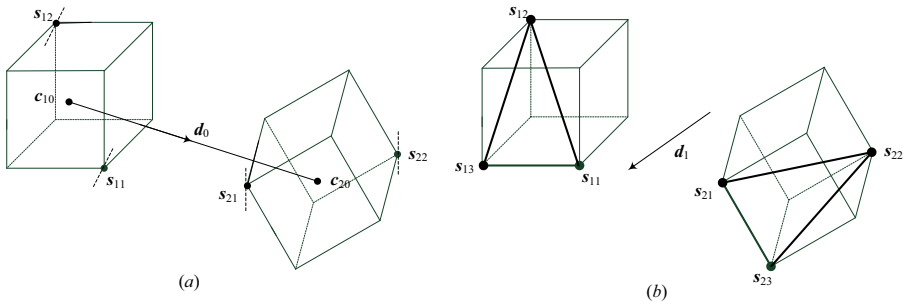

Fig. 8. The illustration of getting the initial simplex, where (a) shows the first two groups of support points (s_{11}, s_{21}) and (s_{12}, s_{22}) in d_0 direction and $-d_0$ direction respectively, and (b) shows the third group of support points (s_{13}, s_{23}) in d_1 direction, so the vertices of the initial simplex are s_{11}–s_{21}, s_{12}–s_{22} and s_{13}–s_{23}.

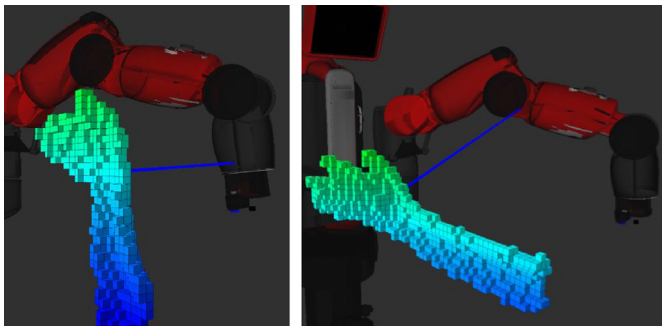

Fig. 9. The calculated closest distances based on GJK.

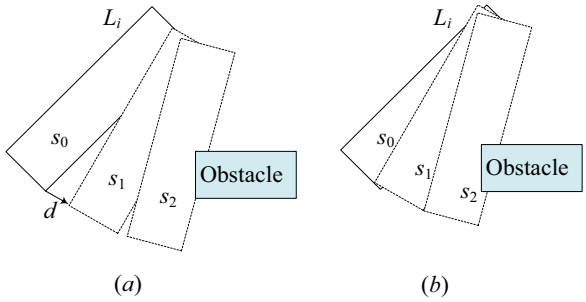

Fig. 10. The diagram of the improved DCD method, (a): the step-size is adjusted to smaller value due to the gap between s_0-state and s_1-state; (b): the step-size is adjusted to bigger value because of the overlap between s_0-state and s_1-state.

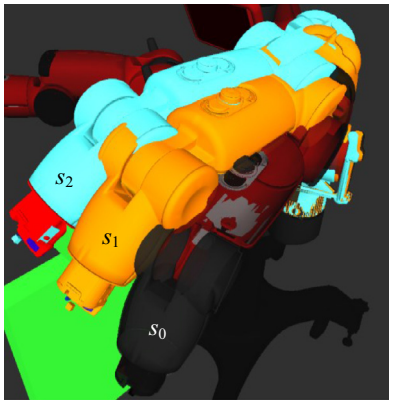

Fig. 11. The effect of the improved DCD method applied in Baxter robot, where s_0 is the current state, and s_1, s_2 are the interpolation states for static collision detection.

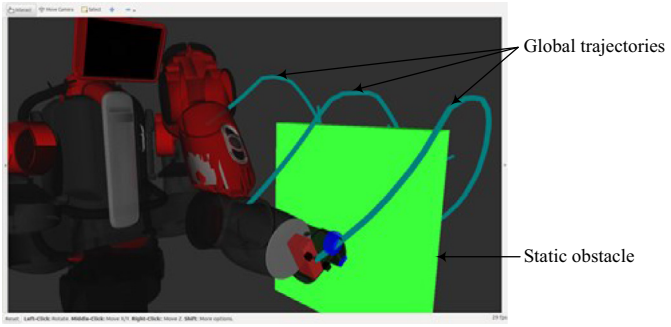

Fig. 12. The planned global trajectories of the Baxter manipulator in static environment.

It needs to be explained that: the static-collision-detection algorithm only runs up to two times for DCD because the diameter of the manipulator's collision models is 0.12 m and the safe value is set as 0.15 m in this paper, as shown in Fig. 11. From this figure, it can be seen that the static-collision-detection times are as few as possible with the prerequisite of no missing detection by the improved DCD method.

The above method is able to reduce the frequency of static collision detection and efficiently prevent missing detection for fast and accurate dynamic collision detection.

4. Experiments of dynamic obstacle avoidance

The proposed algorithm is implemented in C++ based on Moveit! Packages for demonstrating the feasibility, where Moveit! is the most widely used open-source software for manipulation planning relying on ROS. A Baxter robot model with two 7-DOF manipulators is used in the following experiments. A Kinect-V2 camera is mounted on a table for getting the information of dynamic obstacles in the Baxter's workspace. The results are visualized by RViz that is a 3D visualization tool for ROS. All of the experiments are performed on an Intel(R) Core(TM) i7-3770 CPU and ROS-Indigo (Ubuntu 14.04 ×64).

Firstly, the trajectories of each joint of the Baxter manipulator are planned based on Bi-RRT algorithm in static environment, namely the global trajectories, as shown in Figs. 12 and 13.

Then a human arm as a dynamic obstacle occurs randomly in the planning scene during the manipulator's motion. In this experiment, we assume that the human arm is in stationary positions in the process of DCD and avoiding it if necessary. The real-time point cloud images of the human arm from Kinect-V2 are converted to octree maps with resolution of 0.01 m.

The re-planned joint trajectories of the manipulator are obtained by the proposed method for avoiding the dynamic human arm, as shown in Fig. 14, wherein the local planned trajectories are between the two dotted lines. Compared with the global trajectories in Fig. 13, the local path cost is increased for obstacle avoidance.

In order to further prove the flexibility of the proposed method, more re-planned results are shown in Fig. 15 when the human arm is in different states. In Fig. 15(a)–(c), the human arm is in the global trajectories of

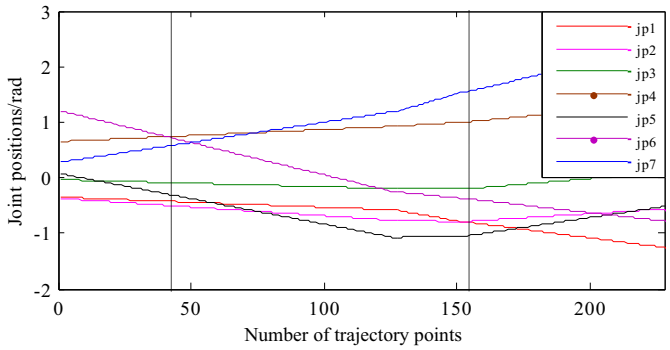

Fig. 13. The global joint trajectories of the Baxter manipulator in static environment.

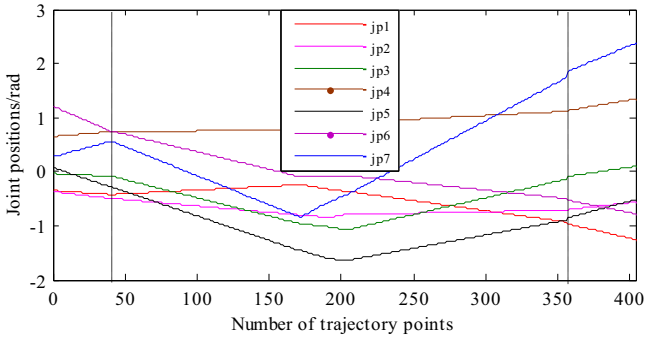

Fig. 14. The re-planned joint trajectories for avoiding the dynamic obstacle, where the parts between the two dotted lines are the local trajectories.

the manipulator, in which case the trajectories are re-planned; however, in Fig. 15(d), the human arm is not, therefore the motion trajectories are unchanged though the nearest distance between the manipulator and the human arm is less than the safe value.

In the above process of avoiding obstacles, the closest distance between the manipulator and the dynamic obstacle is calculated in each control cycle, moreover whether a collision will occur should be determined when the closest distance reaches the safe value. Therefore, the longest control cycle of the manipulator mainly includes the followings:

(1) The closest distance calculation
   (a) Calculating the convex hull of the octree map: once;
   (b) Calculating the closest distances between the three links of the manipulator and the convex hull: three times;
(2) Checking whether there will be a collision
   (a) Static collision detection: twice;

After 30 runs, the average time of calculating the human-arm's convex hull is about 1.501 ms; the average time of calculating a closest distance based on GJK is about 0.050 ms; the average time of static collision detection is about 1.543 ms. Therefore the longest control cycle is about 4.737 ms; the shortest is about 1.651 ms. These data illustrate that the proposed method can meet the request of real-time dynamic obstacle avoidance for redundant manipulators.

5. Discussion

Compared with the traditional dynamic-obstacle-avoidance algorithms, the great advantage of the proposed algorithm is high real-time achieved by simplifying obstacle models (Section 2), fast distance calculation (Section 3.1) and improved DCD (Section 3.2), so that the manipulator can interact with humans or other dynamic obstacles timely. It should be noted that the accuracy of collision detection isn't influenced though the complex model is simplified from non-convex body to convex body before calculating the closest distances. This is because the high-precision static-collision-detection algorithm is adopted in the improved DCD method after reaching the safe distance for detecting collision finally, and the main purpose of distance calculation is just to minimize the number of high-precision static collision detection in actuality.

As aforementioned in Section 1, Luo's method can generate intuitive repulsive reaction for manipulators when obstacles occur nearby the

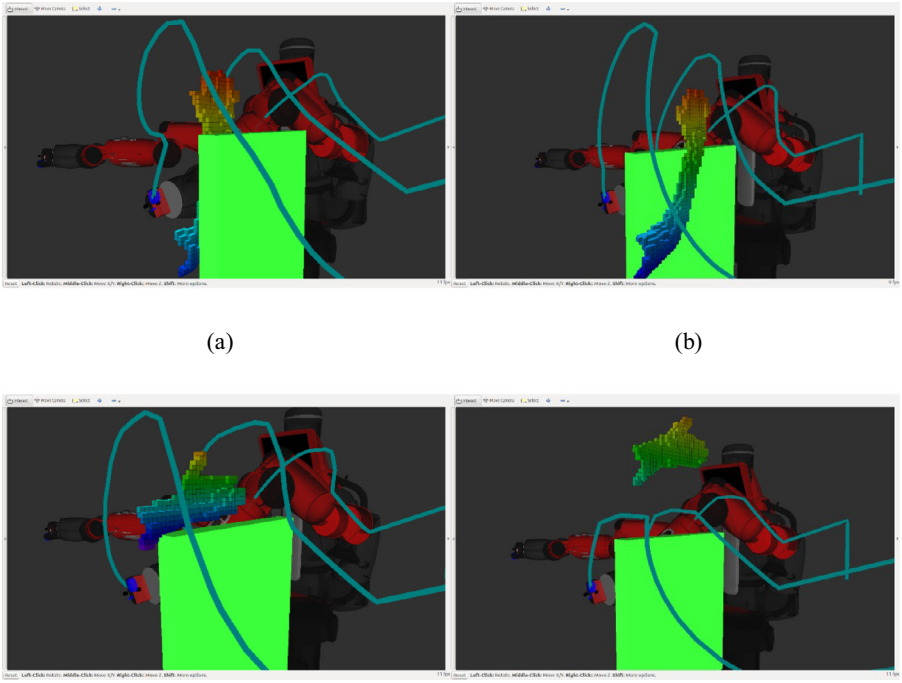

Fig. 15. The motion trajectories of the Baxter manipulator when the human arm is in different states.

manipulator. The comparison between Luo's method and this paper's method is shown below:

(1) Luo's method can't further determine whether or not obstacles are in the global trajectories of the manipulator. So in the case of Fig. 14(d), the manipulator by Luo's method may take a detour, or fail to pass. This paper's method can, however, solve this problem, which is one of the advantages over Luo's method.
(2) Luo's method only used several representative points to describe the manipulator. However, this paper's method adopts cylinders that are more accurate for checking collision.
(3) Unlike the simple obstacles in Luo's method, the obstacles in this paper are much more complex and practical such as the above human arm.

Therefore, this paper's method can achieve more accurate collision detection and is more suitable for redundant manipulators to accurately avoid complex dynamic obstacles like non-convex bodies than previous methods.

6. Conclusion

A novel method combined distance calculation with DCD is proposed for improving the dynamic obstacle avoidance of redundant manipulators in both real-time property and accuracy. In order to decrease the time cost of distance calculation between the manipulator and the dynamic obstacle, the complex models in the planning scene are simplified: the manipulator's links are replaced with cylinders; the real-time octree maps of dynamic obstacles are converted to the corresponding convex hulls. In terms of accuracy, an improved DCD method based on step-size adaptive-adjustable strategy is adopted when the calculated closest distance reaches a given safe value, which can minimize the number of static collision detection and overcome the tunneling problem of DCD. These improvements can effectively shorten the manipulator's control cycle, enhance the real-time performance and be beneficial to the timely and accurate response to sudden obstacles for manipulators. It is of important significances to the safe human-robot interaction.

However, the case that obstacles are always moving isn't considered in this paper. In this case, the velocity of an obstacle should be estimated according to image information for predicting the next state, then the possibility of collision is calculated by the proposed method to determine whether to avoid obstacles. It will be researched further in future work.

Acknowledgement

The authors would like to thank the anonymous reviewers for their critical and constructive review of the manuscript. This study was supported by Funding of Jiangsu Innovation Program for Graduate Education (no. KYLX16_0388), the Fundamental Research Funds for the Central Universities, National Natural Science Foundation of China (no. 51675264) and Open Foundation of Shanghai Key Laboratory of Spacecraft Mechanism.

References

[1] C. Fulgenzi, A. Spalanzani, C. Laugier, Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid, in: IEEE International Conference on Robotics & Automation, 2007, pp. 1610–1616.
[2] G.S. Chyan, S.G. Ponnambalam, Obstacle avoidance control of redundant robots using variants of particle swarm optimization, Rob. Comput. Integr. Manuf. 28 (2) (2012) 147–153.
[3] M. Deng, A. Inoue, K. Sekiguchi, L. Jiang, Two-wheeled mobile robot motion control in dynamic environments, Rob. Comput. Integr. Manuf. 26 (3) (2010) 268–272.
[4] R.C. Luo, C.W Kuo, Intelligent seven-DoF robot with dynamic obstacle avoidance and 3-D object recognition for industrial cyber–physical systems in manufacturing automation, Proc. IEEE 104 (5) (2016) 1102–1113.
[5] M. Duguleana, F.G. Barbuceanu, A. Teirelbar, G. Mogan, Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning, Rob. Comput. Integr. Manuf. 28 (2) (2012) 132–146.
[6] P. Chen, C. Shan, J. Xiang, W. Wei, Moving obstacle avoidance for redundant manipulator via weighted least norm method, in: IEEE Control and Decision Conference, 2015, pp. 6181–6186.
[7] R.C. Luo, Y.T. Chung, Dynamic multi-obstacles avoidance of a robot manipulator based on repulsive vector summation for human-robot co-works, in: IEEE International Conference on Advanced Intelligent Mechatronics, 2015, pp. 1676–1681.
[8] R.C. Luo, M.C. Ko, Y.T. Chung, R. Chatila, Repulsive reaction vector generator for whole-arm collision avoidance of 7-DoF redundant robot manipulator, in: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2014, pp. 1036–1041.
[9] H.A. Sulaiman, A. Bade, N.M. Suaib, Balanced hierarchical construction in collision detection for rigid bodies, in: International Conference on Science & Social Research, 2010, pp. 1132–1136.
[10] J.G. Li, G.W. Xie, Y.X. Yao, P.J. Xia, Dynamic collision detection with optimum discrete interval, Key Eng. Mater. 431 (2010) 166–169.
[11] X. Zhang, S. Redon, M. Lee, Y.J. Kim, Continuous collision detection for articulated models using Taylor models and temporal culling, ACM Trans. Graphics 26 (3) (2007) (15-1)–(15-10).
[12] X. Zhang, Y. Liu, A simple filtering algorithm for continuous collision detection using Taylor models, in: International Conference on Computer-Aided Design and Computer Graphics, 2015, pp. 1–7.
[13] R. Stéphane, K. Abderrahmane, C. Sabine, Fast continuous collision detection between rigid bodies, Comput. Graphics Forum 21 (3) (2002) 1370–1378.
[14] S. Redon, M.C. Lin, D. Manocha, Y.J. Kim, Fast continuous collision detection for articulated models, J. Comput. Inf. Sci. Eng. 5 (2) (2005) 126–137.
[15] M. Tang, D. Manocha, Y.J. Kim, Hierarchical and controlled advancement for continuous collision detection of rigid and articulated models, IEEE Trans. Visual. Comput. Graphics 20 (5) (2013) 755–766.
[16] P.G. Xavier, Fast swept-volume distance for robust collision detection, in: IEEE International Conference on Robotics and Automation, 1997, pp. 1162–1169.
[17] K. Abdelmalek, J. Yang, D. Blackmore, K. Joy, Swept volumes: fundation, perspectives, and applications, Int. J. Shape Model. 12 (1) (2011) 87–127.
[18] M.C. Lin, J.F. Canny, A fast algorithm for incremental distance calculation, in: IEEE International Conference on Robotics and Automation, 1991, pp. 1008–1014.
[19] B. Mirtich, V-Clip: fast and robust polyhedral collision detection, ACM Trans. Graphics 17 (3) (1998) 177–208.
[20] S.A. Ehmann, M.C. Lin, Accurate and fast proximity queries between polyhedra using convex surface decomposition, Comput. Graphics Forum 20 (3) (2001) 500–511.
[21] E.G. Gilbert, D.W. Johnson, S.S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, IEEE J. Rob. Autom. 4 (2) (1988) 193–203.
[22] E. Dyllong, W. Luther, The GJK distance algorithm: an interval version for incremental motions, Numer. Algorithms 37 (37) (2004) 127–136.
[23] J.O. Chong, E.G. Gilbert, Fast versions of the Gilbert-Johnson-Keerthi distance algorithm: additional results and comparisons, IEEE Trans. Rob. Autom. 17 (4) (2001) 531–539.
[24] W. Zhao, Y. Lan, A fast collision detection algorithm based on distance calculations between NURBS surfaces, in: IEEE International Conference on Computer Science and Electronics Engineering, 2012, pp. 534–537.
[25] S. Oh, S. Hwang, A GJK based real-time collision detection algorithm for moving objects, in: Advances in Cognitive Neurodynamics, Springer, Netherlands, 2008, pp. 817–820.
[26] Z. Dong, S. Liu, H. Li, Y. Zheng, GKJ and clustering based algorithm for collision detection on deformable body, in: International Conference on Multimedia Technology (ICMT), 2011, pp. 3539–3542.
[27] Baxter robot. http://www.rethinkrobotics.com/baxter/.
[28] H. Reddivari, C. Yang, Z. Ju, P. Liang, Z. Li, B. Xu, Teleoperation control of Baxter robot using body motion tracking, in: IEEE International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014, pp. 1–6.
[29] ROS. http://wiki.ros.org/.
[30] A. Hornung, M. Kai, M. Bennewitz, C. Stachniss, W. Burgard, OctoMap: an efficient probabilistic 3D mapping framework based on octrees, Auton. Robots 34 (3) (2013) 189–206.
[31] J. Pan, S. Chitta, D. Manocha, FCL: A general purpose library for collision and proximity queries, in: IEEE International Conference on Robotics and Automation (ICRA), 2012, pp. 3859–3866.
[32] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. 22 (4) (1996) 469–483.
[33] J. Shen, Multimap and Multiset Data Structures in Stapl, Texas A&M University, 2014.
[34] S. Cameron, Enhancing GJK: computing minimum and penetration distances between convex polyhedral, in: IEEE International Conference on Robotics and Automation, 1997, pp. S674–S675.
[35] J.J. Kuffner, S.M. Lavalle, RRT-connect: an efficient approach to single-query path planning, in: Proc IEEE Intl Conf on Robotics & Automation, 2000, pp. 995–1001.