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/

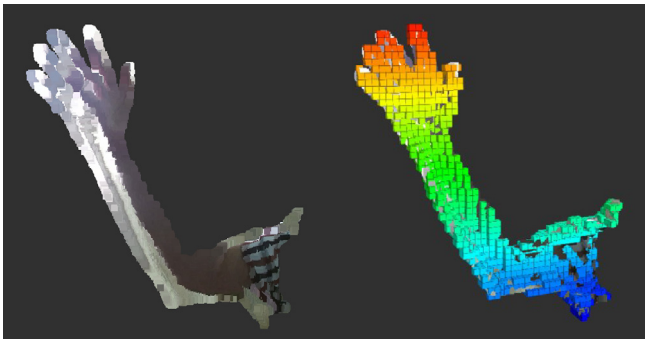

Fig. 1. The point cloud image (left) and the 3D octree map (right) of a human arm.

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].