Robotica (2020) © Cambridge University Press 2020
doi:10.1017/xxxx

An Obstacle Avoidance Algorithm for Space Hyper-redundant Manipulators Using Combination of RRT and Shape Control Method

Xiaobo Zhang†,‡,§, Jinguo Liu†,‡* and Yangmin Li†,‡,††

†*State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China. E-mail: liujinguo@sia.cn*

‡*Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110169, China*

§*University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: zhangxiaobo@sia.cn*

††*Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hong Kong 999077, China. E-mail: yangmin.li@polyu.edu.hk*

(Accepted MONTH DAY, YEAR. First published online: MONTH DAY, YEAR)

SUMMARY

This paper proposes a kinematic obstacle avoidance algorithm for Space hyper-redundant manipulators, and its basic idea is to use a static and a dynamic curve to constrain the macro shape of the manipulators simultaneously. The static curve is constructed based on a traditional rapidly exploring random tree algorithm, and a backbone curve is utilized as the dynamic curve. For these two curves, two novel shape control methods are proposed to accomplish the shape constraining process, respectively. Finally, we verify the reliability and effectiveness of our algorithm through simulations.

KEYWORDS: Obstacle avoidance; RRT; Shape control; Space hyper-redundant manipulators.

* Corresponding author. E-mail: liujinguo@sia.cn

1. Introduction

Hyper-redundant manipulators have a large or infinite degree of kinematic redundancy, making them suited for many inspection and operation tasks in highly constrained environment,[1,2] such as search and rescue in earthquake-stricken areas,[3] undersea exploration,[4] capturing[5] and removing debris missions in space.[6,7] All these missions are inseparable from obstacle avoidance planning which is exactly what we discuss in this paper. In this study, our proposed obstacle avoidance algorithm is more suitable for space hyper-redundant manipulators which do not have the problem of insufficient joint torque as the number of the manipulators' links increase. Only kinematic planning problems are considered. As for studies of dynamic coupling and control methods, related works can be found in refs. [8, 9]. Hereafter we abbreviate the obstacle avoidance algorithm proposed in this paper as RRTSC (rapidly exploring random tree and shape control) algorithm. Fig. 1 shows two typical space application scenarios of the RRTSC algorithm. In the upper half of Fig. 1, the space hyper-redundant manipulator enters into a satellite for maintenance tasks, and in the lower half of Fig. 1, the manipulator passes through a truss for inspection tasks. For more on-orbit tasks in space, please refer to the study in ref. [9].

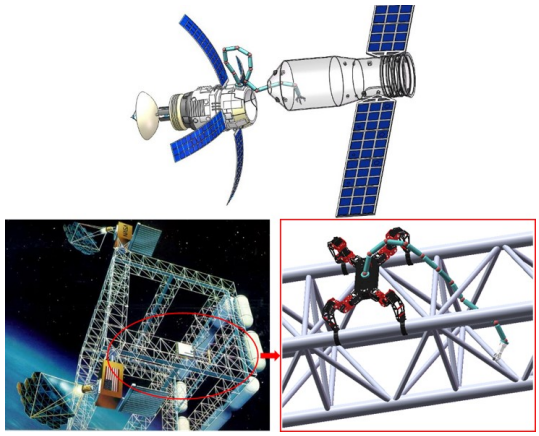

Fig. 1. Application scenarios of the RRTSC algorithm.

Many studies have been presented to solve the obstacle avoidance issue for redundant manipulators. Among these works, the first group of works involves the use of Jacobian pseudo-inverse.[10–15] In this kind of method, the inverse kinematics solution of manipulators is expressed as a sum of a minimum-norm special solution and a general solution that contains an arbitrary vector. Assigning this vector according to some performance functions will not affect the pose of the manipulator's end-effector (the main task[10]), but can adjust the configuration state of the intermediate links to

complete some secondary tasks such as obstacle avoidance,[11] singularity avoidance,[12] joint physical limitation avoidance,[13,14] and joint torque optimization.[15] In ref. [11], a minimum distance point is detected between all links and obstacles at every instant, and then an escaping velocity is applied to this critical point to ensure that all links do not collide with obstacles while the end-effector follows a given trajectory. However, the convergence rate of this pseudo-inverse-based approach will decrease rapidly owing to the fact that the increase of the number of links will lead to a huge calculation for Jacobian pseudo-inverse, which makes this kind of method unsuitable for hyper-redundant manipulators. The second group of works for obstacle avoidance planning of manipulators is developed by means of the artificial potential field.[16–18] In this kind of method, obstacles are represented by repulsive surfaces, and the target point is represented by an attractive pole. The end-effector of the manipulator is pulled by the potential field until it reaches the target point, and all links are repelled by obstacles to avoid collisions. This kind of method suffers from the problem that the manipulator may be trapped into local minima. In order to escape local minima, a randomized path planner is introduced to implement random steps in the local minima.[16,17] However, this planner is implemented in configuration space (C-space) and is computationally expensive when utilized in high-dimensional cases. Additionally, in a complex obstacle environment with only narrow passages, the planner can hardly find a solution to escape the local minima. In ref. [18], the harmonic function is utilized to construct a local minima-free potential field. The basic idea is to find a smooth path in the obstacle environment, and develop a point settling algorithm to keep the tip of each link on this path until the manipulator reaches the target point. However, as the authors mentioned, this work will become considerably slow when extending it to a 3D calculation, and it may fail to find a solution when an obstacle is located between the manipulator and the goal point. The third group of works for obstacle avoidance planning of manipulators involves searching in configuration space (C-space).[19–29] The concept of the C-space is first proposed in ref. [19]. Two processes are needed in this study: a mapping process (or a C-space constructing process) and a searching process. The mapping process maps obstacles in Cartesian space to C-space, and this process is achieved by randomly selecting enough configurations (or angle sequences) of the manipulator and detecting whether the manipulator collides with obstacles in those configurations. The searching process is responsible for finding a feasible collision-free trajectory in the obtained C-space. However, it is not suitable for hyper-redundant manipulators because the high C-space dimension will cause a huge computational burden in the mapping process. An example of building a C-space can be found in ref. [30].

The Probabilistic Roadmap Method (PRM), proposed in ref. [20], replaces the mapping process[19] with a learning process during which a graph is constructed. The graph is composed of nodes and edges, where the nodes correspond to collision-free configurations and the edges correspond to feasible trajectories. In ref. [21], a variational approach is presented to find a feasible collision-free solution after optimization, which takes the nodes of the graph in PRM as an initial guess. Indeed, PRM speeds up the searching process of feasible solutions to a certain extent. However, in order to improve the coverage of the graph in PRM to the C-space, considerable computer memory resources are needed as a guarantee. Some researchers use the Rapidly-exploring Random Tree (RRT) to find a feasible trajectory in C-space.[25–29] The RRT has been widely utilized in the obstacle avoidance planning of mobile robots.[31] When it is applied to manipulators, IK solver,[25] Jacobian pseudo-inverse[27–29] or Jacobian transpose[26] should be incorporated to improving the convergence rates of these RRT-based planners. In ref. [25], given a start point and a goal point in task space, corresponding to the start and goal configurations in C-space, obtained via an IK solver. The authors use an RRT-Connect path planner to establish a connection between the start and the goal configurations via a connect heuristic. If an IK solver is not available, the newly added node (or angle sequence) in C-space can be updated by an iterative process based on the Jacobian transpose[26] or Jacobian pseudo-inverse.[27–29] During this iterative process, collision detections are executed to make sure that the newly added node is collision-free. These RRT-based planners can be utilized for pick-and-place tasks[25–28] or end-effector trajectory tracking tasks.[29] It should be noted that the work in ref. [28] execute RRT in task space to guide the search of a feasible trajectory in C-space and that the work in ref. [29] execute RRT in both task space, guiding the end-effector to follow a collision-free path in task space, and C-space, guiding the links not to collide with obstacles. However, in the case of hyper-redundant manipulators, the IK-solver may not be available, and the calculations for Jacobian transpose or Jacobian pseudo-inverse will become inefficient. Some works utilize heuristic search algorithms such as genetic algorithm[22] and ant colony optimization algorithm,[23] or multi-pass sequential localized search technique[24] to find feasible solutions in C-space. However, the search processes in refs. [22, 23] are computationally inefficient and not suited for hyper-redundant manipulators. The work in ref. [24] involves a process of discretizing the C-space and arranging them in ascending order, and this process will consume a lot of calculation time if the discretization accuracy is increased. The fourth group of works for obstacle avoidance planning of manipulators employs neural networks.[32–34] In this kind of method, the inverse kinematics solution[35,36] and obstacle

avoidance planning[32–34] are incorporated as a quadratic programming (QP) formulation, which can be solved via neural networks. The inverse kinematics solution and obstacle avoidance are represented via a dynamic equality constraint and a dynamic inequality constraint respectively. However, whether this kind of method can be applied to obstacle avoidance of hyper-redundant manipulators in a multi-obstacle environment, with high computational efficiency, is not mentioned.

The fifth and sixth groups of works for obstacle avoidance planning of manipulators are tractrix-based method[37,38] and backbone-curve-based method[39–43] (or shape control method called in this paper) respectively. The key idea of the tractrix-based method is that given the motion of a single link, the motion of all links of the manipulators can be obtained via an iteration process and a kinematics transformation from the first link to the end link. The motion of a single link is obtained via a tractrix curve. A single link consists of a leading end and a trailing end, and the leading end can track any 2D or 3D path in task space while the trailing end follows the tractrix curve formulation, during which the link's length remains unchanged. The tractrix-based method can be utilized to simulate the motion of one-dimensional flexible objects such as cables, ropes, ribbons, hair, and hyper-redundant manipulators in free space.[44–46] When the tractrix-based method is used in obstacle avoidance planning, optimization techniques must be incorporated to guide the trailing end of each link to escape away from the obstacles with a minimal energy cost. In ref. [37], an optimization algorithm based on a calculus of variation is formulated where obstacles are modeled by smooth and differentiable super-ellipsoids. As for duct type obstacles, the work in ref. [37] suffers from a problem that analytical formulation of a duct is not always available. Therefore, the work in ref. [38] proposes three representations of ducts. The basic idea of the backbone-curve-based method (or shape control method called in this paper) is employing a backbone curve to constrain the macro shape of a manipulator by fitting the curve and the manipulator's shape as closely as possible.[47] In essence, this kind of method reduces the planning dimension. The degrees of freedom of the manipulator are reduced to the number of parameters of the backbone curve, thus leading to a low computation cost and making it suitable for online path planning and real-time control of hyper-redundant manipulators. In ref. [39], 'Tunnel' is defined in task space in which obstacles are present and then differential geometry is used to make the manipulator to be constrained to the tunnel. The tunnel (or backbone curve) is time-varying and collision-free with an ability to lead its endpoint to explore the obstacle space. However, although the tunnel (or backbone curve) is collision-free, it cannot guarantee that the manipulator is also collision-free

unless the number of the links of the manipulator is so large that the manipulator's shape can approximate the tunnel's shape. On the other hand, the work in ref. [39] does not prescribe any strategy for constructing the tunnel and the tunnel is manually constructed.[40] To solve this problem, in ref. [40], a follow-the-leader approach is proposed and it adopts a roadmap technique, termed Generalized Voronoi Graph (GVG), to construct the tunnel. In ref. [41], a harmonic potential function and a modified modal approach[1,48] are combined to construct the tunnel to constrain the macro shape of the manipulator. However, these approaches[39–41] suffer from a common problem that the link model of the manipulator may collide with obstacles. Another interesting backbone-curve-based method is presented in ref. [42], with a novel modified modal approach.[49] In this work, the macro shape of the manipulator is constrained by the modified backbone curve. The even joint points of the manipulator are located on the curve, and the odd ones are released to accomplish the secondary tasks such as obstacle avoidance, by adjusting two kinds of parameters (the equivalent link length and angle between the adjacent equivalent links). However, the two kinds of parameters increase the solution dimension of the manipulator's configuration, which deviates from the essence of the backbone-curve-based method. In other words, this work does not significantly reduce the planning dimension and will be computationally inefficient when dealing with obstacle avoidance of hyper-redundant manipulators in a highly constrained obstacle environment. Another interesting work is proposed in ref. [43], in which a serpenoid curve is considered as the backbone curve to constrain the manipulator and then an obstacle mapping process in posture space is implemented, similar to a C-space construction. Although the mapping dimension is reduced to the parameter of the backbone curve, the mapping process will still cause a huge computation burden. In addition to the above six groups of methods, two other interesting works are shown in ref. [50] and ref. [51]. In ref. [50], a hybrid method combing analytical and numerical methods is proposed, in which the analytical equations are responsible for determining the first two and the last three joint angles while the numerical technique is utilized for calculating the rest of the joints, and the obstacle avoidance is formulated as a constraint optimization problem and solved also by numerical techniques. In ref. [51], a Hyper-BumpSurface is constructed to capture both the free and the forbidden areas of the environment as one mathematical entity, and then a genetic algorithm is employed to find a time-optimal trajectory in this hyper-surface. In summary, among the existing works, the backbone-curve-based method (or shape control method called in this paper) may be a good choice for dealing with obstacle avoidance of hyper-redundant manipulators in a highly constrained obstacle environment, without

a huge computation burden. Our proposed RRTSC algorithm belongs to the category of the backbone-curve-based method.

The goal of this paper is to propose a kinematic obstacle avoidance planning algorithm for space hyper-redundant manipulators, in three-dimensional task space with static randomly distributed obstacles. Inspired by the works in refs. [40, 41] and aiming at solving the common problem (the backbone curve is collision-free, but the link model of the manipulator may collide with obstacles) existing in refs. [40, 41], we present the RRTSC algorithm. The successful implementation of the RRTSC algorithm is inseparable from a static curve, a dynamic time-varying backbone curve, and two novel shape control methods for these two curves. The task space is divided into two parts: obstacle space and free space. Correspondingly, the manipulator is divided into two parts: the inner part in the obstacle space and the outer part in the free space. The static curve and its corresponding shape control method are responsible for constraining the macro shape of the inner part of the manipulator and guiding it to move towards a goal point from a start point. The dynamic time-varying backbone curve and its corresponding shape control method are responsible for dynamically constraining the macro shape of the outer part of the manipulator. As time goes on, the inner part will be longer and the outer part will be shorter because the total number of links of the manipulator is constant. Please refer to section 2.5 for details of how these two shape control methods are achieved. The main contributions of this study are summarized by the following remarks:

1. Different from the existing backbone-curve-based works[40, 41] in which only a dynamic backbone curve is constructed in real-time for constraining the macro shape of the manipulator, our proposed RRTSC algorithm combines a static curve and a dynamic backbone curve for the same purpose. To our best knowledge, it is the first attempt to accomplish obstacle avoidance of manipulators with a combination of a static curve and a dynamic backbone curve among existing works.
2. The existing backbone-curve-based works[40, 41] suffer from a common problem that the backbone curve is collision-free but the link model of the manipulator may collide with obstacles. Our proposed RRTSC algorithm does not have this problem. For a detailed comparison between our algorithm and existing works, please refer to section 3.5.
3. The RRTSC obstacle avoidance algorithm architecture for hyper-redundant manipulators in three-dimensional space is proposed, including four components: a manual multi-sphere approximation for obstacles, an RRT algorithm, a dynamic backbone curve, and two novel shape control methods. Before the RRTSC algorithm is proposed, these four components are described. The implementation rules of the

manual multi-sphere approximation for obstacles are discussed in detail. The RRT algorithm is employed to construct a static curve.

4. Novel shape control methods with the static curve and the dynamic backbone curve are proposed respectively. Shape control with the static curve ensures that the manipulator in obstacle space is collision-free, and guides the manipulator to move towards a goal point from a start point. Shape control with the dynamic backbone curve is responsible for constraining the macro shape of the manipulator in free space. It should be noted that the reason why we claim that the shape control method with the dynamic backbone curve is novel is based on the following two considerations. First, compared to the work in ref. [41], we propose a more general form of the dynamic backbone curve. Secondly, we adopt a different way to solve the joint points of the manipulator. In ref. [41], the joint points are solved by iteratively solving each joint point, and every iteration is an optimization process. In this study, we fix some of the joint points on the dynamic backbone curve, and the remaining joint points are obtained through optimization processes, reducing the calculation time.

The remainder of this paper is organized as follows. Section 2 describes the main components of the RRTSC algorithm architecture, including the manual multi-sphere approximation for obstacles, the RRT algorithm, the dynamic backbone curve, and the two novel shape control methods. Section 2.1 introduces an overview of these components to facilitate readers to have a conceptual and holistic understanding of these components. The manual multi-sphere approximation for obstacles is discussed in section 2.2. Section 2.3 reviews the RRT algorithm which will be later utilized to construct the static curve in obstacle space for the RRTSC algorithm. In section 2.4, the dynamic time-varying backbone curve is constructed in a more general form compared to the one in ref. [41]. Section 2.5 introduces the way to construct the static curve and proposes the two shape control methods (shape control with the static curve and shape control with the dynamic backbone curve). Shape control with the static curve is utilized to constraining the macro shape of the manipulator in obstacle space, while the shape control with the dynamic backbone curve is responsible for constraining the macro shape of the manipulator in free space. In section 3, the RRTSC algorithm architecture is proposed, and integrated simulations are carried out to demonstrate the feasibility of our algorithm. In addition, a comparison with existing works is also given in this section. Finally, the conclusions and future works are presented in section 4.

2. RRTSC Algorithm Architecture

In this section, we firstly introduce the obstacle avoidance system model, and then detail the components of the RRTSC algorithm separately.

2.1. System introduction

The whole obstacle avoidance system is illustrated in Fig. 2, and it is composed of obstacles, a rapidly exploring random tree (RRT), a RRT path, an environment boundary, a cubic spline interpolation curve, a backbone curve and a hyper-redundant manipulator. Let's introduce them one by one. To facilitate reading, in Appendix part, all the symbols utilized in this paper are concluded in Tables III-VI, and all the acronyms are summarized in Table VII.

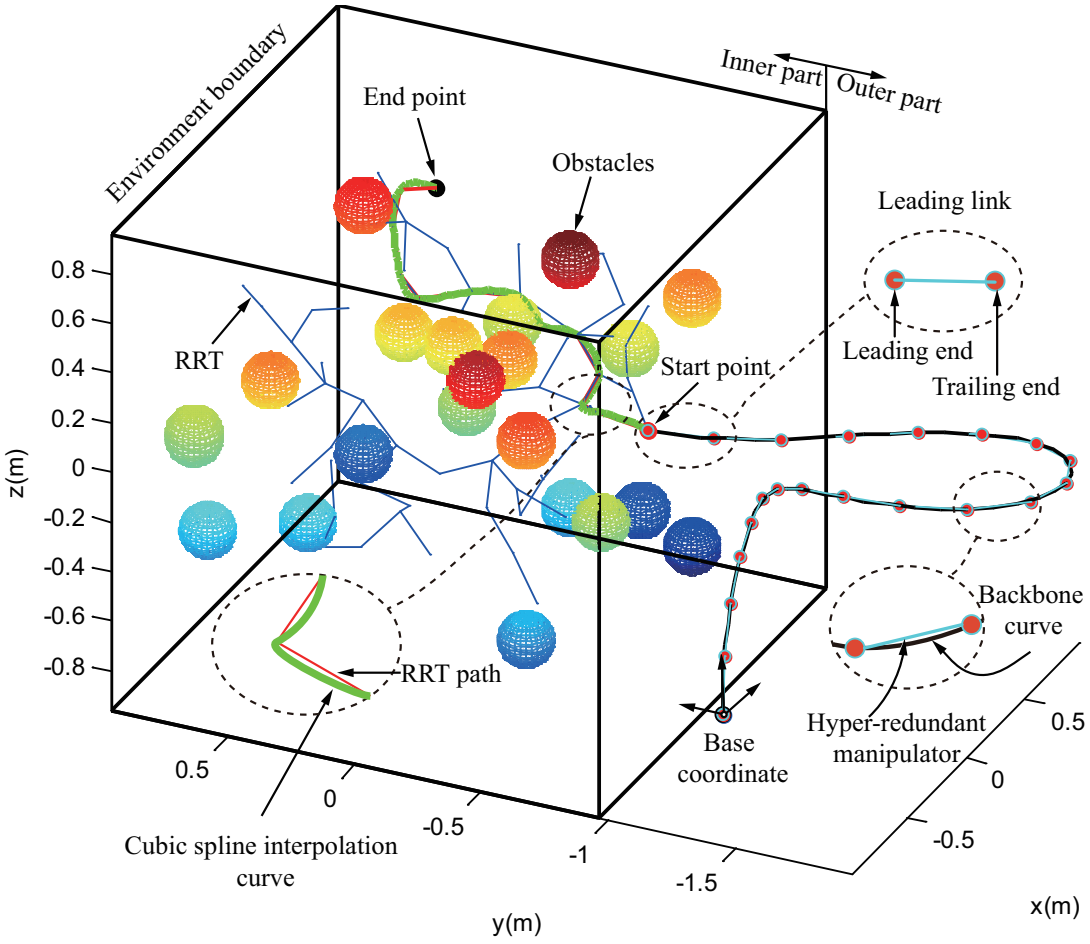

Fig. 2. Model of the whole obstacle avoidance system.

The obstacles are modeled by spheres in this case, and one sphere corresponds to one obstacle. However, an obstacle can't be simply modeled via a sphere when the obstacle is a truss or a hole. To solve this problem, we adopt a manual multi-sphere approximation for obstacles in subsequent chapters. RRT algorithm[25] is a traditional path planning

algorithm. The feature of this algorithm is that it can search high-dimensional space quickly and efficiently, and guide the search to a blank area through random sampling points in a state space, so as to find a collision-free path from a starting point to a target point. As shown in Fig. 2, we use blue line segments to represent the RRT, and red line segments to represent the collision-free RRT path. The start and end points of the RRT path are colored in red and black, respectively. The environment boundary is utilized to limit the sampling space of the RRT, and it is represented by a black rectangular external frame. It can be denoted by $En\_limit = (X_\mathrm{l},\ X_\mathrm{r},\ Y_\mathrm{l},\ Y_\mathrm{r},\ Z_\mathrm{l},\ Z_\mathrm{r})$, and the task space is divided into two parts: the inner part $En\_inner = \{P|P \in En\_limit\}$ (or obstacle space) and the outer part $En\_outer = \{P|P \notin En\_limit\}$ (or free space), where P is any point in a 3D task space, and X_l, X_r, Y_l, Y_r, Z_l, Z_r are left and right limit coordinate values of the environment boundary. Considering that the RRT path is not a smooth curve, the cubic spline interpolation curve (green curve in Fig. 2) is adopted to replace the original RRT path. Define RRT_path and CSI_path to denote the RRT path and the cubic spline interpolation curve, respectively, for the convenience of describing pseudo codes in subsequent sections. The CSI_path is a smooth curve through a series of shape-value points which are RRT nodes in this study. The backbone curve, colored in black in Fig. 2, is constructed based on a modal approach,[1,48] and in this paper it is extended to a more general form. The hyper-redundant manipulator is abstracted into cyan line segments connected end-to-end, and we refer to the line segments as the link model of the manipulator. The connection points represent joints colored in Indian red with three degrees of freedom. In this paper, we refer to the link where the end effector of the manipulator is located as the leading link. The leading link has two points. The end effector point is called the leading end, and the other is called the trailing end.

2.2. Modeling for Obstacles

The choice of an approach to model obstacles usually is closely related to many factors, such as modeling accuracy, specific operation tasks, and principles of collision detection algorithms. Traditional obstacle modeling method commonly makes use of a sphere to model an obstacle with an irregular shape, and the sphere is located at the obstacle's centroid with radius $r = d_\mathrm{max}$, where d_max denotes the maximum distance from the centroid to envelope boundary. Fig. 3a shows a cube obstacle, and Fig. 3b, 3c illustrate the corresponding envelope sphere and the radius, respectively. For scenarios where the operation tasks are simple and require low modeling accuracy, it is wise to choose this method. However, when encountering the following situations where the tasks are

passing through obstacles with holes (Fig. 3d, 3g) or truss type obstacles (Fig. 3j), the traditional method will no longer work. Considering that, an alternative method for the representation of obstacles must be selected. An interesting method can be found in ref. [37], in which obstacles are modeled using Minkowski sum[19] (a combination or union of geometric entities) of differentiable super-ellipsoids.[52] The formulation of the super-ellipsoids contains five parameters, and by adjusting these parameters a family of geometric entities, such as ellipses, rectangles, cylinders, spheres, etc., can be generated. Another interesting method[53,54] employs multi spheres to approximate geometric entities of arbitrary shape in an automatical way, and in our study, we refer to this method as the multi-sphere approximation method. Inspired by the works,[19,53,54] we use the multi-sphere approximation method to represent obstacles. The representation in our study is implemented in a manual form for some simple geometric entities, considering that the manual representation can achieve better approximation accuracy with fewer spheres compared with the automatic one. It should be noted that the manual multi-sphere approximation has two main disadvantages: 1) It is not suitable for handling obstacles with complex shapes, since the manual discretization process will become difficult; 2) It is not suitable for large obstacle environments, since manual discretization process will become tedious. In addition, a lot of computer memory is needed to store the obstacles' data. Therefore, for obstacles with complex shapes, the automatic multi-sphere approximation methods[53,54] are needed. For large obstacle environments, an algorithm can be developed to extract a local obstacle environment from a global one according to a specific task, and then use the automatic multi-sphere approximation methods[53,54] in the local obstacle environment to obtain the spheres' data. This algorithm essentially restricts the RRTSC algorithm to find a feasible solution in the local obstacle environment instead of the global one, which reduces the computational consumption to a certain extent. If this algorithm cannot be developed, we can also use other obstacle modeling methods. Our RRTSC algorithm does not rely on specific obstacle modeling methods. We model obstacles as multiple spheres to simplify the collision detection processes, in which only distance calculation from point to point is needed. In other words, other obstacle modeling methods, such as the super-ellipsoids method in ref. [37], can also be integrated into our algorithm, as long as the three collision detection sub-algorithms in the RRTSC algorithm are appropriately modified. The way to envelope a geometric entity with spheres is similar to the principle of a 3D printer, which is traversing the spheres along the obstacle's coordinate system. It is wise to choose different coordinate system types for different types of obstacles, as shown in Fig. 3f and 3i where the coordinate

systems are chosen as Cartesian and cylindrical ones, respectively. The final envelope results are shown in Fig. 3e and 3h. As for the envelope process, slice uniformly along the z -axis to obtain $x-y$ (Fig. 3f) or $\theta-r$ (Fig. 3i) cross sections firstly, and then envelope the spheres along the cross section curves until all cross sections along the z -axis are finished. In some special cases, the envelope method needs to be adjusted manually. Taking the truss type obstacles (Fig. 3j) for illustration, the ultimate envelope result is shown in Fig. 3k. The truss bars are considered as lines (Fig. 3l) instead of cylinders, therefore it is not necessary to slice along the z -axis. It should be noted that the number and radius of the spheres must satisfy conditions that there are no gaps, and at the same time guarantee suitable approximation accuracy. Fig. 4 shows an example of how to implement a suitable approximation process in one- and two-dimensional spaces (three-dimensional space has a similar enveloping principle). r_{env} represents the radius of a sphere. d_{adj} denotes the distance between the centers of adjacent spheres. α_{env} is the angle between two tangent lines at the intersection of adjacent circles (the cross-section of the sphere is a circle). If $d_{adj} > 2r_{env}$, as shown in Fig. 4a, there will be gaps among the spheres both in one- and two-dimensional spaces, which is a poor approximation case and will lead to a false obstacle avoidance detection. If $d_{adj} = 2r_{env}$, $\alpha_{env} = 0°$, as shown in Fig. 4b, there is no gap in one-dimensional space, but there is a gap in two-dimensional space. If $d_{adj} < 2r_{env}$, $\alpha_{env} = 90°$, as shown in Fig. 4c, there is no gap whether in one- or two-dimensional space and it is a perfect approximation process. If $d_{adj} < 2r_{env}$, $\alpha_{env} > 90°$, as shown in Fig. 4d, although the gap disappears, a phenomenon of over-approximation appears. For some complex geometry, this phenomenon is inevitable, and what we need to do is to avoid it as far as possible. Compared with Fig. 4c, r_{env} in Fig. 4d increases, which leads to the decrease of approximation accuracy. In summary, the approximation process is based on a coordinate system and should choose the proper radius and number of the spheres and consider modeling accuracy, without affecting specific tasks.

2.3. RRT Algorithm

RRT algorithm[25] is an efficient obstacle avoidance planning method. The basic RRT algorithm is shown in algorithm 1, and its relevant concept is shown in Fig. 5a.

In algorithm 1, the start tree $Tree$ contains only one node during initialization. First, check if the start point $Start\_node$ and the target point End_node can be directly connected (lines 4 and 5). This should satisfy two conditions: 1) Distance between these two points is less than a given step threshold $Step_{\text{tree}}$; 2) The line segment One_RRT_path

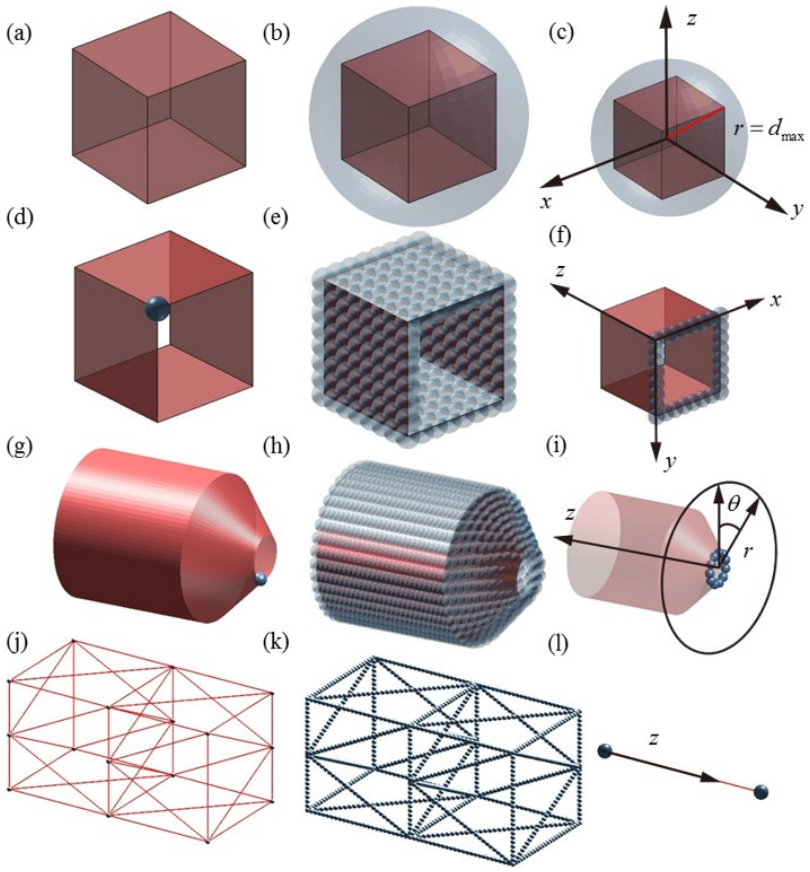

Fig. 3. Modeling for obstacles.

between these two points dose not collide with obstacles. If the conditions are met, add End_node to $Tree$, and if not repeat extending new nodes use function $Extend\_tree()$ (line 10) until $Tree$ reaches End_node. The procedure of function $Extend\_tree()$ is not detailed in the above pseudo code and we describes it as follows: 1) A sample function randomly selects a rand point $Rand\_node$ from the inner part of the obstacle environment En_inner; 2) A nearest function selects a node $Nearst\_node$ closest to $Rand\_node$; 3) Extend a distance $Step_{\text{tree}}$ from $Rand\_node$ to $Nearst\_node$ and obtain One_RRT_path. If a collision occurs, go to step 1, and if not add a new node New_node and check whether New_node could be directly connected to End_node; 5) Use flag variable $flag\_tree$ to denotes whether $Tree$ reaches the End_node. When a $Tree$ is available, use function $Find\_path()$ to find a collision free path RRT_path from the end point to the start point, with help of index relationship between a leaf node and its parent node. It should be noted that $Col\_det\_RRT()$ detects a collision every time a rand node is generated (lines 19-30). Principle of this function is easy to understand. First, discretize the line segment One_RRT_path via a coefficient vector $\delta = (0, 0.25, 0.5, 0.75, 1)$ to get point set $\{P_1, P_2, P_3, P_4, P_5\}$, and then calculate the distance between OB_i and P_j, for $i = 1, ..., n_{\text{ob}}$

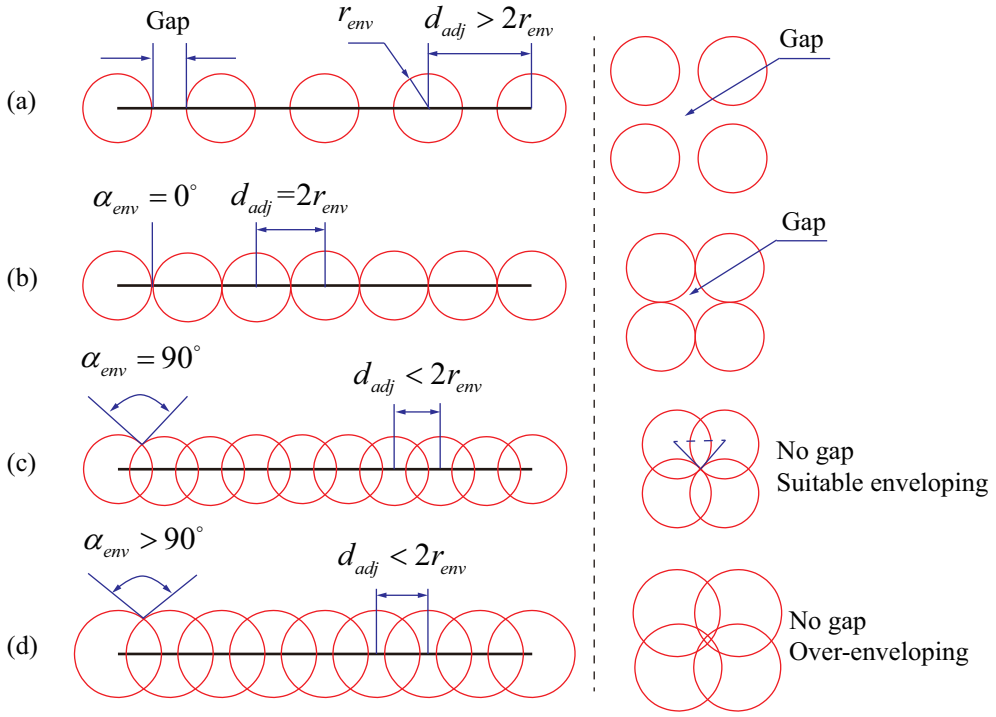

Fig. 4. Illustration example of how to implement a suitable approximation process.

where n_{ob} is the number of spheres and $j = 1, ..., 5$. If the distance is less than the sphere radius r, a collision has occurred and that means the *Rand_node* must be dropped.

Given a 3D task space where sphere obstacles are randomly distributed, initialize related parameters and execute the RRT algorithm, and we can obtain a tree shown in Fig. 5b. The specific parameters will be detailed in section 3. Different from traditional method which utilize RRT algorithm in C-space,[19] our obstacle avoidance algorithm applies it in task space. There is an example for illustrating the differences between them. Fig. 5c depicts a two-degree-of-freedom robotic arm, with two obstacles $OB1$ and $OB2$ surrounding it, in a plane task space. The corresponding representation of obstacles in C space, illustrated in Fig. 5d, is obtained by a mapping process. This process can be described as two steps: 1) Change the joint angles θ_1 and θ_2 to get every possible configuration via forward kinematics; 2) Detect collision between the links and the obstacles to divide the C-space into obstacle area and non-obstacle area. Obviously, this process will become more difficult as the degrees of freedom of manipulators and the number of obstacles increase. To avoid this problem, we propose the RRTSC algorithm, using the RRT algorithm in task space and two shape control methods to drive a hyper-redundant manipulator. The two shape control methods use shape control curve to constrain macro the shape of a manipulator as shown in Fig. 5e, and we will describe these two methods in section 2.5.

Algorithm 1 The procedure of generating collision-free RRT path

Input: OB_i, r, En_inner, $Start\_node$, End_node
Output: RRT_path

function $Generate\_RRT(\,)$
  Load obstacle envelope spheres' center data OB_i and radius r, and initialize environment boundary En_limit, start and end path points $Start\_node$, End_node
  $Tree \leftarrow Start\_node$ %Build a start tree
  if $(norm(Start\_node,\ End\_node) < Step_{\text{tree}})$ **and**
    $(Col\_det\_RRT(OB_i,\ r,\ One\_RRT\_path))$ **then**
    %The parameter One_RRT_path is a line segment constructed by one leaf node and its parent node
    $Tree \leftarrow Add\_node(Tree,\ End\_node)$
  else
    while $(flag\_tree)$ **do**
      $Tree \leftarrow Extend\_tree(Tree, OB_i, r, En\_inner, End\_node)$
      %Repeat extending the tree until it reaches the end node
    end while
  end if
  %Obtain the collision free RRT path via the tree
  $RRT\_path \leftarrow Find\_path(Tree)$
  return RRT_path
end function

Input: OB_i, r, One_RRT_path
Output: $flag\_RRT$

function $Col\_det\_RRT(\,)$
  $flag\_RRT \leftarrow 0$ %No collision
  for $\delta = 0,\ 0.25,\ \ldots,0.75,\ 1$ **do**
    $P \leftarrow \delta * One\_RRT\_path[1] + (1-\delta)One\_RRT\_path[2]$ %Discretization
    for $i = 1,\ \ldots,n_{ob}$ **do**
      if $norm(P,\ OB_i) < r$ **then**
        $flag\_RRT \leftarrow 1$ %Collison occurs
      end if
    end for
  end for
  return $flag\_RRT$
end function

2.4. Backbone curve

As mentioned before, the backbone curve is employed to constrain the macro shape of a manipulator in En_outer. This curve is a kind of spatial curve and its mathematical

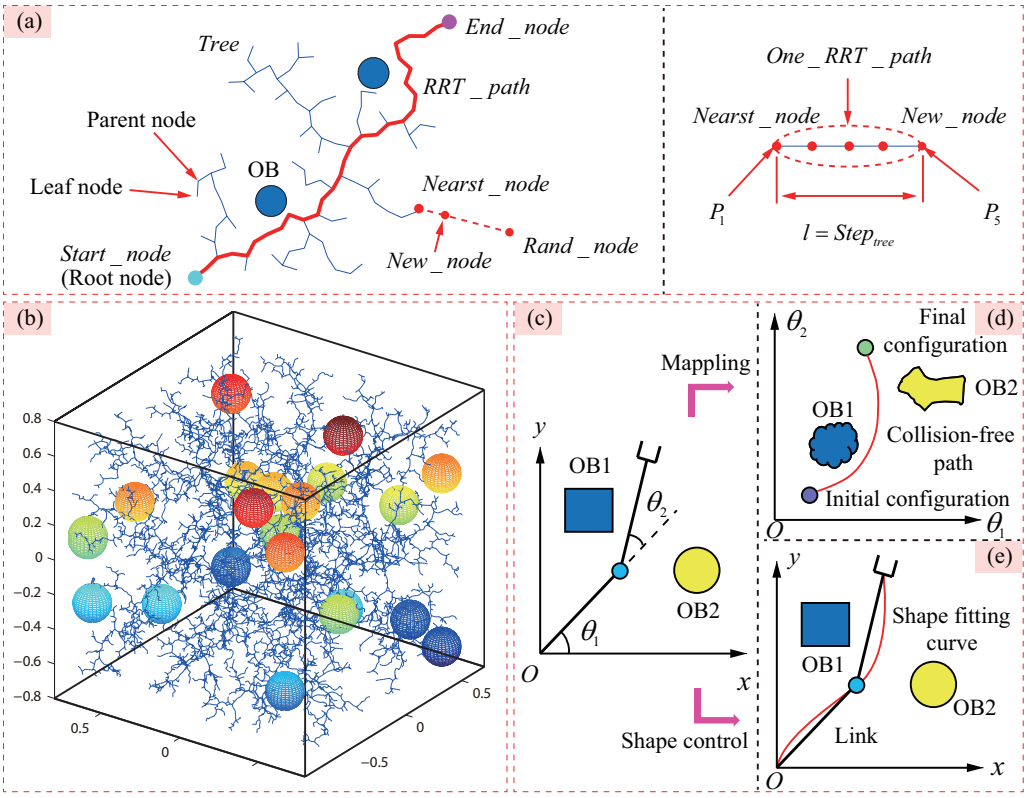

Fig. 5. Relevant illustration of RRT algorithm.

expressions are defined[48] as follows:

$$P(s,t) = P_{\text{int}}(t) + \int_0^s l(t)F(\sigma,t)d\sigma \tag{1}$$

where $P(s,t)$, shown in Fig. 6a, denotes an arbitrary point on the backbone curve. $P_{\text{int}}(t)$ is the initial point of the backbone curve in base frame, and it is defined as $P_{\text{int}}(t) = [x_{\text{int}}, y_{\text{int}}, z_{\text{int}}]^{\text{T}}$. s and t are independent variables, representing arc length of the backbone curve and time, respectively. $l(t)$ indicates the total length of the backbone curve at time t. $F(\sigma,t)$ represents the unit vector tangent to the backbone curve at $s = \sigma$. Detailed representation of the backbone curve can be defined as follows:

$$\begin{aligned} P(s,t) &= P_{\text{int}}(t) + [x(s,t)\ y(s,t)\ z(s,t)]^{\text{T}} \\ &= \begin{bmatrix} x_{\text{int}}(t) \\ y_{\text{int}}(t) \\ z_{\text{int}}(t) \end{bmatrix} + \begin{bmatrix} \int_0^s l(t)\sin K(\sigma,t)\cos T(\sigma,t)d\sigma \\ \int_0^s l(t)\cos K(\sigma,t)\cos T(\sigma,t)d\sigma \\ \int_0^s l(t)\sin T(\sigma,t)d\sigma \end{bmatrix} \end{aligned} \tag{2}$$

where $K(\sigma,t)$ is the angle between $F'(\sigma,t)$ and y aixs of the base frame at $s=\sigma$, and $T(\sigma,t)$ is the angle between $F'(\sigma,t)$ and $F(\sigma,t)$ at $s=\sigma$. Their geometric interpretation can be found in Fig. 6b. The $F'(\sigma,t)$ is the projection vector of $F(\sigma,t)$ on $x-y$ plane on the base frame. If functions $K(\cdot)$ and $T(\cdot)$ are specified, $P(s,t)$ can be obtained using Eq. (2). In ref. [1], $K(\cdot)$ and $T(\cdot)$ are formulated as follows:

$$\begin{aligned} K(s,t) &= \sum_{i=1}^{n_1} c_i(t) f_i(s) \\ T(s,t) &= \sum_{i=1}^{n_2} d_i(t) g_i(s) \end{aligned} \tag{3}$$

where $f_i(s)$ and $g_i(s)$ are mode functions, $c_i(t)$ and $d_i(t)$ are modal participation factors, n_1 is the number of mode functions for $K(s,t)$, and n_2 is the number of mode functions for $T(s,t)$. From the existing works,[1,41] trigonometric functions are often used as the mode functions, and the choice of them must meet two conditions. Firstly, the trigonometric functions must be linearly independent. Otherwise, some participation factors will be lost. Secondly, the trigonometric functions can not be all odd functions on the interval $s\in[0,1]$. Otherwise, the backbone will be degenerate because the values of x and z for $P(s,t)$ will always be equal to zero. According to these rules, the trigonometric functions can be chosen manually. It should be noted that n_1+n_2 is the total number of mode functions, and this number should equal or exceed the number of constraints imposed on the backbone curve. In a 3D task space without obstacles, a total of seven constraints need to be satisfied, which means n_1+n_2 is at least equal to seven. Three of the constraints are imposed by the position of the end point of the curve, and the remaining four constraints come from the intention of controlling the orientation at the start and end points of the curve. Now, we rewrite Eq. (3) into the following form:

$$\begin{aligned} K(s,t) &= a_1(t)f_1(s) + a_2(t)f_2(s) + b_1(t)f_3(s) + b_2(t)f_4(s) \\ T(s,t) &= a_3(t)g_1(s) + b_3(t)g_2(s) + b_4(t)g_3(x) \end{aligned} \tag{4}$$

where $a_1(t)$, $a_2(t)$, $a_3(t)$, $b_1(t)$, $b_2(t)$, $b_3(t)$ and $b_4(t)$ are participation factors. We hope that the orientation at the start and end points of the curve can be adjusted via intuitively adjusting one or a combination of the four parameters $b_1(t)$, $b_2(t)$, $b_3(t)$ and $b_4(t)$. Indeed, this can be achieved via a suitable selection of the mode functions. A specific example

can be found in ref. [41], where $K(s,t)$ and $T(s,t)$ are specified as follows:

$$\begin{aligned} K(s,t) &= a_1(t)\sin(2\pi s) + a_2(t)(1-\cos(2\pi s)) \\ &\quad + b_1(t)\sin(\pi s/2) + b_2(t)(1-\sin(\pi s/2)) \\ T(s,t) &= a_3(t)(1-\cos(2\pi s)) + b_3(t)\sin(\pi s/2) \\ &\quad + b_4(t)(1-\sin(\pi s/2)) \end{aligned} \tag{5}$$

where $\{b_1(t), b_3(t)\} = \{K(0,t), T(0,t)\}$ and $\{b_2(t), b_4(t)\} = \{K(1,t), T(1,t)\}$, specify the orientation at the start and end points of the backbone curve, respectively. In fact, the formulations for $K(s,t)$ and $T(s,t)$ are not unique and can be replaced by other ones. Considering that, we extend the formulations for $K(s,t)$ and $T(s,t)$ to a more general form. In other words, we give a selection rule for the mode functions. In Eq. (4), we make $\{b_1(t), b_3(t)\}$ equal to $\{K(0,t), T(0,t)\}$ and $\{b_2(t), b_4(t)\}$ equal to $\{K(1,t), T(1,t)\}$, and then the selection rule for the mode functions are given as follows:

$$\begin{cases} f_1(0) = f_2(0) = f_4(0) = 0, \quad f_3(0) = 1 \\ \quad g_1(0) = g_3(0) = 0, \quad g_2(0) = 1 \\ f_1(1) = f_2(1) = f_3(1) = 0, \quad f_4(1) = 1 \\ \quad g_1(1) = g_2(1) = 0, \quad g_3(1) = 1 \end{cases} \tag{6}$$

In addition, the $f_1(s)$, $f_2(s)$, $f_3(s)$ and $f_4(s)$ must be linearly independent trigonometric functions and not all odd functions, the same as the $g_1(s)$, $g_2(s)$ and $g_3(s)$. By now the backbone curve is determined via parameters P_{int}, $l(t)$, $a_1(t)$, $a_2(t)$, $a_3(t)$, $b_1(t)$, $b_2(t)$, $b_3(t)$ and $b_4(t)$, and we will put them into use in subsequent sections. It should be pointed out that in the follow-up simulations, we will use Eq. (5) to construct the dynamic backbone curve.

2.5. Shape control method

The shape control method is utilized to build rules of how to drive a hyper-redundant manipulator to move with a curve. We divide shape control into two categories: 1) Shape control with a static curve; 2) Shape control with a dynamic curve. As mentioned in section 2.3 and 2.4, we select the RRT path and the backbone curve to control the inner part and the outer part of a manipulator, respectively. The RRT path belongs to

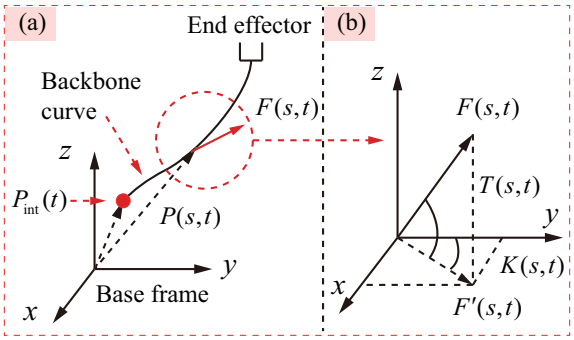

Fig. 6. The backbone curve.

the static curve, while the backbone curve is one of the dynamic curves. The successful implementation of the RRTSC algorithm will use these two shape control methods. Now we will describe them, respectively.

2.5.1. Shape control with the dynamic backbone curve. First, let's describe the shape control method with the dynamic backbone curve. Assuming that a hyper-redundant manipulator is composed of links with length l_{link} and number n_{link}. Then we can simplify the manipulator into line segments connected end to end, which is called the link model. The purpose of the shape control for the backbone curve is to calculate all connecting points (joint points), referred as P_i for $i = 0, ..., n_{link}$, of the link model. It can be formulated as the following optimization problem:

$$\text{Solve}: \ P_i, \quad i = 0, \ \dots, n_{\text{link}} \tag{7}$$

$$\text{Minimize}: \ \sum_{i=0}^{i=n_{\text{link}}} \parallel P_i - P'_i \parallel, \quad i = 0, \dots, n_{\text{link}} \tag{8}$$

$$\begin{aligned} \text{Subject to}: l_{\text{link}} - eps \leq \parallel P_i - P_{i-1} \parallel \leq l_{\text{link}} + eps, \\ i = 0, \dots, n_{\text{link}} \end{aligned} \tag{9}$$

where eps is a coefficient used to adjust the matching accuracy of the link length l_{link}; P_i' denotes a equally spaced point on the backbone curve and it satisfies Eq.(10).

$$\begin{aligned} P_i' &= P_{\text{int}}(t) + \int_0^{m_i} l(t)F(\sigma, t)d\sigma, \quad i = 0, \dots, n_{\text{link}} \\ m_i &= \frac{i}{n_{\text{link}}}, \quad i = 0, \dots, n_{\text{link}} \end{aligned} \tag{10}$$

$$\text{Solve}: \ P_i, \ \ i = 1, 3, \ \dots, n_{\text{link}} - 1 \ (\text{if } n_{\text{link}} \text{ is even}) \tag{11}$$

$$\text{Minimize}: \ \parallel P_i - P'_i \parallel, \ \ i = 1, 3, \ \dots, n_{\text{link}} - 1 \tag{12}$$

Subject to :

$$\begin{cases} P_i = P_i', \quad i = 0, 2, 4, \ \dots \ , n_{\text{link}} \\ l_{\text{link}} \ - eps \leq \parallel P_i - P_{i-1} \parallel \leq l_{\text{link}} + eps, \\ i = 1, 3, \ \dots \ , n_{\text{link}} - 1 \end{cases} \tag{13}$$

Equation (8) guarantees that P_i is as close as possible to the backbone curve. In order to solve this optimization problem, let some $P_i{}'$ equal $P_i{}'$ $(i = 0, ..., n_{\text{link}})$, then Eqs. (7)-(9) are reduced to Eqs. (11)-(13) if n_{link} is even, or Eqs. (14)-(16) if n_{link} is odd.

$$\text{Solve :} \ \ P_i, \quad i = 2, 4, \ \dots \ , n_{\text{link}} - 1 \ \text{(if } n_{\text{link}} \text{ is odd)} \tag{14}$$

$$\text{Minimize :} \ \parallel P_i - P'_i \parallel, \quad i = 2, 4, \ \dots \ , n_{\text{link}} - 1 \tag{15}$$

Subject to :

$$\begin{cases} P_i = P_i', \quad i = 0 \\ P_i = P_i', \quad i = 1, 3, \ \dots \ , n_{\text{link}} \\ l_{\text{link}} \ - eps \leq \parallel P_i - P_{i-1} \parallel \leq l_{\text{link}} + eps, \\ i = 2, 4, \ \dots \ , n_{\text{link}} - 1 \end{cases} \tag{16}$$

The first lines of Eq. (13) and Eq. (16) specify that $P_i|_{i=0}$ (manipulator's base) coincides with the start point of the backbone curve. The second line of Eq. (13) and the second line of Eq. (16) specify which points of P_i are equal to $P_i{}'$, for $i = 0, ..., n_{\text{link}}$, and make $P_i|_{i=n_{\text{link}}}$ (manipulator's end effector) coincides with the end point of the backbone curve. According to the Eqs. (11)-(16), employ mature optimization algorithms such as simulated annealing or particle swarm algorithm, and then we can easily get all the values of P_i, for $i = 0, ..., n_{\text{link}}$. Fig. 7a and 7b show two shape control examples where $n_{\text{link}} = 4$ and $n_{\text{link}} = 3$, respectively. It can be seen that no matter n_{link} is even or odd, the start and the end points always satisfy $P_0 = P'_0$ and $P_{n_{\text{link}}} = P'_{n_{\text{link}}}$, which is realistic.

In order to drive the manipulator to move, what we need to do is just changing the shape of the backbone curve by adjusting its parameters. Given a target point P_{target}, an initial point $P_{\text{int}}(t)$, total arc length $l(t)$, coefficients $b_1(t)$, $b_2(t)$, $b_3(t)$, $b_4(t)$, solve Eq.

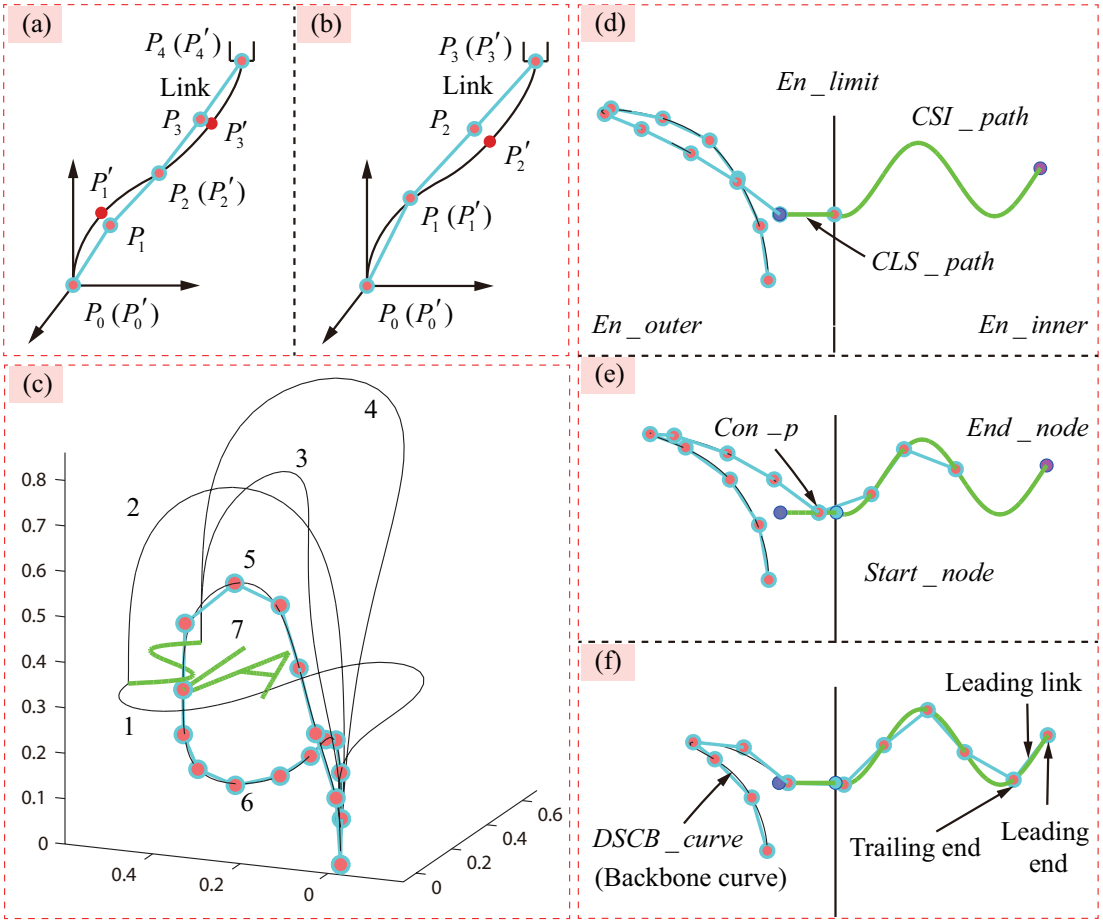

Fig. 7. Shape control method.

(17) via numerical approximation in Matlab, and then we can obtain parameters $a_1(t)$, $a_2(t)$, $a_3(t)$.

$$P_{\text{target}} = P_{\text{int}} + \int_0^1 l(t)F(\sigma,t)d\sigma \tag{17}$$

If we replace P_{target} with a continuous space curve, the manipulator will move continuously. Supplementary material shape_control_dscb_curve.mp4 shows an animation where the end effector of the manipulator and the corresponding end point of the backbone curve follow a given trajectory ("SIA"). Essentially, this is a process of constantly seeking inverse solutions. The label 7 in Fig. 7c is the "SIA" end trajectory and P_{target} is selected from it. Fig. 7c depicts six backbone curves with different parameters listed in Table I, and the initial points are set as $P_{\text{int}} = [0, 0, 0]^{\text{T}}$. For curves 1 and 2, target points are set as $P_{\text{target}} = [0.15, 0.54, 0.30]^{\text{T}}$. For curves 3 and 4, target points are set as $P_{\text{target}} = [0.50, 0.52, 0.30]^{\text{T}}$. For curves 5 and 6, target points are set as $P_{\text{target}} = [0.15, 0.42, 0.30]^{\text{T}}$. Comparing curves 1 and 2, it can be found that b_4 can affect

the tangent vector of the end point of the backbone curve. $b_1(t)$, $b_2(t)$ and $b_3(t)$ have the same function as $b_4(t)$. Comparing curves 3 and 4, it can be found that the total length of the backbone curve can be adjusted by changing $l(t)$. Curve 5 and 6 are the shape fitting results when n_{link} are set to seven and ten, respectively, which indicates that our shape control method can be adapted to different number of manipulator links.

Table I . Parameters of the backbone curves in Fig. 6c.

Label	$b_1(t)$	$b_3(t)$	$b_2(t)$	$b_4(t)$	$a_1(t)$	$a_2(t)$	$a_3(t)$	n_{link}	$l(t)$
1	$\pi/2$	$\pi/2$	$-\pi/2$	π	0.697	-1.368	0.301	10	1.5
2	$\pi/2$	$\pi/2$	$-\pi/2$	$-\pi/2$	0.165	0.586	0.642	10	1.5
3	$\pi/2$	$\pi/2$	$-\pi/2$	$-\pi/2$	-0.478	0.924	0.643	10	1.5
4	$\pi/2$	$\pi/2$	$-\pi/2$	$-\pi/2$	0.299	0.860	0.576	10	2
5	$\pi/2$	$\pi/2$	$-\pi/2$	$-\pi/2$	-0.663	0.646	0.790	7	1
6	$\pi/2$	$\pi/2$	$-\pi/2$	$\pi/2$	0.099	0.605	-1.131	10	1

2.5.2. Shape control with the static RRT path. Next, let's describe the shape control method with the static RRT path. The purpose of the shape control with the RRT path is to calculate all connecting points P_i, for $i = 0, \ldots, n_{\text{link}}$. It can be formulated as

$$\begin{gathered} l_{\text{link}} - eps \leq \| P_i - P_{i-1} \| \leq l_{\text{link}} + eps, \\ i = 1, \ldots, n_{\text{link}} \end{gathered} \tag{18}$$

where P_i, for $i = 0, \ldots, n_{\text{link}}$, are points on the RRT path. First, set the value of $P_i|_{i=n_{\text{link}}}$, and then iteratively solve other points via Eq. (18). If we want to drive the link model of the manipulator to move, what we need to do is just changing the phase of $P_i|_{i=n_{\text{link}}}$. The so-called phase refers to a specific position of the $P_i|_{i=n_{\text{link}}}$ on the RRT path.

2.5.3. Shape control with SSC_curve and DSCB_curve. Given that the second derivative of the RRT path is not continuous, we replace it with a cubic spline interpolation curve, referred as *CSI_path*. To avoid collision between the end link constrained to the backbone curve and the environment boundary, we introduce an additional connecting line segment between *CSI_path* and the backbone curve. We call it as *CLS_path*. The *CSI_path* and *CLS_path* belong to the static curve, having the same shape control principle as the RRT path, and they are collectively referred

to as *SSC_curve* (static shape control curve). Accordingly, we call the backbone curve as *DSCB_curve* (dynamic shape control backbone curve). When *SSC_curve* and *DSCB_curve* are available, we could use them to drive a manipulator as depicted in Fig. 7d-7f. Fig. 7d-7f show initial, intermediate and final stages, respectively, of the manipulator's movement. The essence of the whole movement is a process of continuously solving the connecting points, referred as $P^j_{\mathrm{hrm}(m)}$, of the manipulator. We summarize this process into algorithm 2, and for the convenience of descriptions, some symbolic representations are defined. Compared to the previous definition of the connecting points, we add the upper right corner mark j to indicate the phase of the manipulator, and add the lower right corner mark $\mathrm{hrm}(m)$ to indicate which connecting point of the manipulator is solved. If $m = 0$, $P^j_{\mathrm{hrm}(m)}|_{m=0}$ corresponds to the base of the manipulator which is fixed. If $m = n_{\mathrm{link}}$, $P^j_{\mathrm{hrm}(m)}|_{m=n_{\mathrm{link}}}$ represents the end point of the manipulator. Considering that $P^j_{\mathrm{hrm}(m)}$ is partly calculated by *SSC_curve* and partly calculated by *DSCB_curve*, we define the lower right corner marks $\mathrm{hrm\_ssc}(i)$ and $\mathrm{hrm\_dscb}(k)$ to distinguish them. $n_{\mathrm{link\_ssc}}$ and $n_{\mathrm{link\_dscb}}$ denote number of links constrained by *SSC_curve* and *DSCB_curve*, respectively. The point connecting *SSC_curve* and *DSCB_curve*, shown in Fig. 7e, is denoted as *Con_p*.

Algorithm 2 describes the whole process of the shape control method with *SSC_curve* and *DSCB_curve*. Before introducing algorithm 2, we need to clarify the principle of how the manipulator's configuration is updated. During a shape control cycle, the configuration of the manipulator is updated by following steps: 1) Change the position of the end effector ($P^j_{\mathrm{hrm}(m)}|_{m=0}$) on *CSI_path*; 2) Calculate all positions of the connecting points ($P^j_{\mathrm{hrm}(m)}$) via the two shape control methods; 3) Repeat the steps 1 and 2 until the end effector reaches a goal point. In other words, if we want to update the configuration of the manipulator, we should update the position of the end effector first. We use the superscript $j = 1,\ 2,\ \ldots,\ n_{\mathrm{config}}$ or $j = 1,\ 2,\ \ldots,\ n_{\mathrm{phase}}$ to indicate that the manipulator is in a different configuration. The n_{config} represents the total number of configurations of the manipulator during a shape control cycle. The n_{phase} represents the total number of phases during the shape control cycle, and the phase, in this study, means a specific position of the end effector ($P^j_{\mathrm{hrm}(m)}|_{m=0}$, $P^j_{\mathrm{hrm\_ssc}(i)}|_{i=n_{\mathrm{link\_ssc}}}$, or the leading end of the leading link) on *CSI_path*. n_{config} and n_{phase} are equal, because the configuration of the manipulator and the position of the end effector have a one-to-one correspondence. Given a specific task that the manipulator passes into an obstacle space from a start point to a goal point, the end effector should be placed at the start point in the first phase (Fig. 7d) and the goal point in the last phase (Fig. 7f).In our study, *CSI_path* is a path (or curve)

from the start point to the goal point (Fig. 7d), and it is stored in the form of discrete points in the computer. Consider that the end effector is always placed on CSI_path, what we need to do is find a set of indexes of the discrete points of CSI_path for updating the position of the end effector. This can be achieved by following steps: 1) Suppose that CSI_path is stored as discrete points, with index number $Ind_{\text{p\_dis\_csi}}$=1, 2, ... , $n_{\text{p\_dis\_csi}}$ where $n_{\text{p\_dis\_csi}}$ is the total number of the discrete points; 2) Select a proper value of n_{phase} (or n_{config}), and then calculate an index increment $\Delta Ind_{\text{p\_dis\_csi}}$ via Eq. (19). If it is not divisible, we can adjust the value of the $\Delta Ind_{\text{p\_dis\_csi}}$ in the last phase; 3) A set of indexes of the discrete points for updating the position of the end effector, denoted by $Ind_{\text{p\_dis\_csi\_ee}}$, can be expressed to a form of Eq. (20). Eqs. (19) and (20) are as follows:

$$\Delta Ind_{\text{p\_dis\_csi}}=\frac{n_{\text{p\_dis\_csi}}-1}{n_{\text{phase}}} \tag{19}$$

$$Ind_{\text{p\_dis\_csi\_ee}} = 1,\ 1+\Delta Ind_{\text{p\_dis\_csi}},\ \dots,\ 1+n_{\text{phase}}\Delta Ind_{\text{p\_dis\_csi}} \tag{20}$$

Using $Ind_{\text{p\_dis\_csi\_ee}}$, all positions of the end effector ($P^{j}_{\text{hrm\_ssc}(i)}|_{i=n_{\text{link\_ssc}}}$), during a shape control cycle ($j = 1,\ 2,\ \dots,\ n_{\text{phase}}$), can be obtained. Now let's introduce algorithm 2. First, use Eqs. (19) and (20) to obtain $Ind_{\text{p\_dis\_csi\_ee}}$, and then use funtion $Cal\_SSC\_p()$, which is based on Eq. (18), to calculate all the points $P^{j}_{\text{hrm\_ssc}(i)}$, for $i = 0, 1, ..., n_{\text{link\_ssc}}$ and $j = 1, 2, ..., n_{\text{phase}}$, via SSC_curve (line 5). During the shape control cycle, in the first phase, $P^{j}_{\text{hrm\_ssc}(i)}|_{j=1,i=n_{\text{link\_ssc}}}$ coincides with $Start\_node$, as shown in Fig. 7d, and in the final phase, $P^{j}_{\text{hrm\_ssc}(i)}|_{j=n_{\text{phase}},i=n_{\text{link\_ssc}}}$ coincides with End_node, as shown in Fig. 7f. Next calculate all the points $P^{j}_{\text{hrm\_dscb}(k)}$, for $i = 0, 1, ..., n_{\text{link\_dscb}}$ and $j = 1, 2, ..., n_{\text{phase}}$, via $DSCB\_curve$ (lines 9-13). It should be noted that function $Build\_DSCB\_curve()$ is programmed via Eqs. (2) and (5), and function $Cal\_DSCB\_p()$ is based on Eqs. (11)-(16). Finally, employ function $Combine\_p()$ to combine $P^{j}_{\text{hrm\_ssc}(i)}$ and $P^{j}_{\text{hrm\_dscb}(k)}$ to obtain the complete connecting points $P^{j}_{\text{hrm}(m)}$, of the manipulator.

3. Integrated Simulations of the RRTSC Algorithm

In this section, we further refine the RRTSC algorithm by adding two collision detection sub-algorithms in addition to the existing RRT collision detection sub-algorithm, and then propose the RRTSC algorithm frame. After that, the feasibility and effectiveness of the RRTSC algorithm are verified through integrated simulations utilizing the abstract link model of the hyper-redundant manipulator. When applying our algorithm to a specific real hyper-redundant manipulator, the mapping relationship between the link

Algorithm 2 Shape control of the hyper-redundant manipulator

Input: l_{link}, n_{link}, CSI_path, CLS_path

Output: $P^{j}_{\text{hrm}(m)}$

function *Shape_control*( )
    Initialize link length and number parameters l_{link} and n_{link}, and load path data CSI_path, CLS_path
    Connect CSI_path and CLS_path, forming static shape control curve SSC_curve
    %Calculate the connecting points $P^{j}_{\text{hrm\_ssc}(i)}$ of the hyper-redundant manipulator links via SSC_curve
    $P^{j}_{\text{hrm\_ssc}(i)} \leftarrow Cal\_SSC\_p(SSC\_curve,\ l_{\text{link}},\ n_{\text{link}})$
    for $j = 1,\ \ldots,\ n_{\text{phase}}$ **do**
        $Con\_p \leftarrow P^{j}_{\text{hrm\_ssc(i)}}|_{i=n_{\text{link\_ssc}}}$ %Find the point Con_p
        $n_{\text{link\_dscb}} \leftarrow n_{\text{link}} - n_{\text{link\_ssc}}$
        %Calculate the number of links which are obtained by dynamic shape control backbone (DSCB) curve
        $L_{\text{dscb}} \leftarrow n_{\text{link\_dscb}} * l_{\text{link}}$ %Length of the DSCB curve
        $DSCB\_curve \leftarrow Build\_DSCB\_curve(Base\_p, Con\_p, L_{dscb})$
        %Build the DSCB curve, given base position
        $P^{j}_{\text{hrm\_dscb(k)}} \leftarrow Cal\_DSCB\_p(DSCB\_curve, n^{j}_{\text{link\_dscb}}, l_{\text{link}})$
        %Calculate the connecting points $P^{j}_{\text{hrm\_dscb(k)}}$ of the hyper-redundant manipulator links via $DSCB\_curve$
    end for
    %Combing $P^{j}_{\text{hrm\_ssc}(i)}$ and the $P^{j}_{\text{hrm\_dscb(k)}}$ to obtain the complete connecting points $P^{j}_{\text{hrm(m)}}$
    $P^{j}_{\text{hrm(m)}} \leftarrow Combine\_p(P^{j}_{\text{hrm\_ssc(i)}},\ P^{j}_{\text{hrm\_dscb(k)}})$
    return $P^{j}_{\text{hrm}(m)}$
end function

model and the robot must be established, and it depends on a specific joint configuration of the robot. We give an example of how to build this relationship.

3.1. RRTSC Algorithm Frame

RRT algorithm guarantees that the RRT path and obstacles do not collide, but it can't guarantee that the cubic spline interpolation curve and obstacles do not collide. In addition, even if the cubic spline interpolation curve and obstacles do not collide, the link model of the manipulator may collide with obstacles. Therefore, additional two collision detection sub-algorithms are necessary. Fig. 8 is a schematic diagram of the collision detection sub-algorithms. All the collision detections are implemented in *En_inner*. Fig. 8a shows all the components, including obstacles, link model of the manipulator, the cubic spline interpolations curve, the RRT and the RRT path, inside an obstacle environment.

Fig. 8b is a local enlarged drawing of Fig. 8a. Fig. 8c-8e show three cases: Case 1 indicates that no collision occurred; Case 2 indicates that the cubic spline curve collides with the obstacle; Case 3 indicates that the link model collides with the obstacle. For the latter two collision situations, Fig. 8g and 8h show the corresponding collision detection sub-algorithms. Their principles are similar to function $Col\_det\_RRT()$, which is defined in section 2.3 and used to detect collision between One_RRT_path and obstacles (Fig. 8f). First, discretize the detected objects and then detect whether the distance, between each discrete point and each sphere's center, is smaller than the radius of the sphere. It should be noted that only all the positions of the first link (or the leading link shown in Fig. 2) of the manipulator in different phases are required for collision detection in Fig. 8h, because the other links of the manipulator will follow the trajectory of the first link. The first link is obtained via Eq. (18).

Now let's put all the components of the RRTSC obstacle avoidance algorithm together and organize them, forming the RRTSC algorithm frame described briefly in algorithm 3. The RRTSC algorithm is implemented according to following steps: 1) Utilize the multi-sphere envelope method (section 2.2) to model obstacles, and obtain the spheres' center data OB_i and radius r; 2) Build an obstacle environment and determine environment boundary En_limit; 3) In En_inner space, give an $Start\_node$ and an End_node and then run a loop (lines 4-13 in algorithm 3). In the loop, keep running the function $Generate\_RRT$ (algorithm 1 in section 2.3) to generate a RRT_path until no collision occurs. Each time a RRT_path is generated, two collision detection sub-algorithms (line 10 and line 12 in algorithm 3) need to be executed. Use function $Generate\_CSI()$ to obtain a cubic spline interpolation curve CSI_path and run function $Col\_det\_CSI()$ to detect whether CSI_path collides with obstacles (collision detection 2 in Fig. 8g). Use the function $Col\_det\_L()$ to detect whether the first link in each phase collides with obstacles (collision detection 3 in Fig. 8h); 4) Generate a CLS_path with function $Generate\_CLS()$; 5) Determine n_{phase}, and in each phase execute function $Shape\_control()$ to obtain $P^j_{\text{hrm}(m)}$ (shape control method, algorithm 2 in section 2.5). Utilize $P^j_{\text{hrm}(m)}$ to construct the link model of the manipulator, and with the phase shift, the manipulator, constrained via SSC_curve and $DSCB\_curve$, moves to End_node from $Start\_node$ without collision with obstacles. There is a failure condition for the RRTSC algorithm where path planning might fail, and it can be avoided. In the RRTSC algorithm, we attach the leading link to the static shape control curve (SSC_curve) and move continuously to detect if the leading link collides with obstacles, during which the solution of the trailing end of the leading link may not exist due to a poor discretization process of SSC_curve. As long as

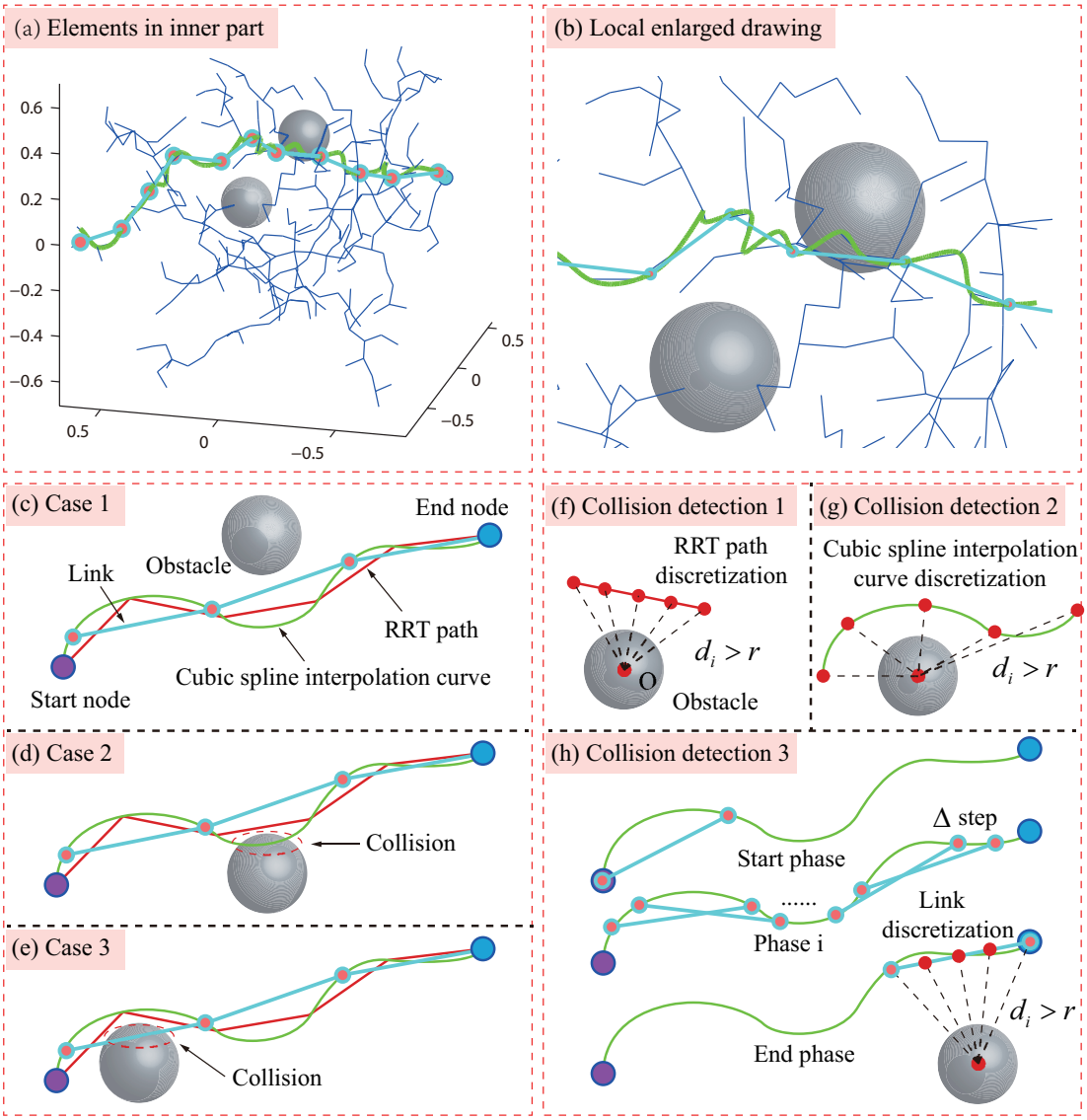

Fig. 8. Collision detection sub-algorithms.

we improve the discretization accuracy of *SSC_curve* through interpolation, this problem can be avoided.

3.2. Base-movable hyper-redundant manipulators simulation

Now the RRTSC algorithm is available, and it can be used for base-movable hyper-redundant manipulators. A simulation is carried out in this section, as shown in Fig. 10. Table II provide all the simulation-related data. Obstacles are modeled by the traditional single-sphere enveloping method, and spherical center coordinates are set randomly within *En_limit* and listed in supplementary material fig10_ob.xlsx. Without loss of generality, we assume that all the spheres' radius r are the same, and here we set them as 0.06. Given an initial point (*Start_node*), a target point (*End_node*) and

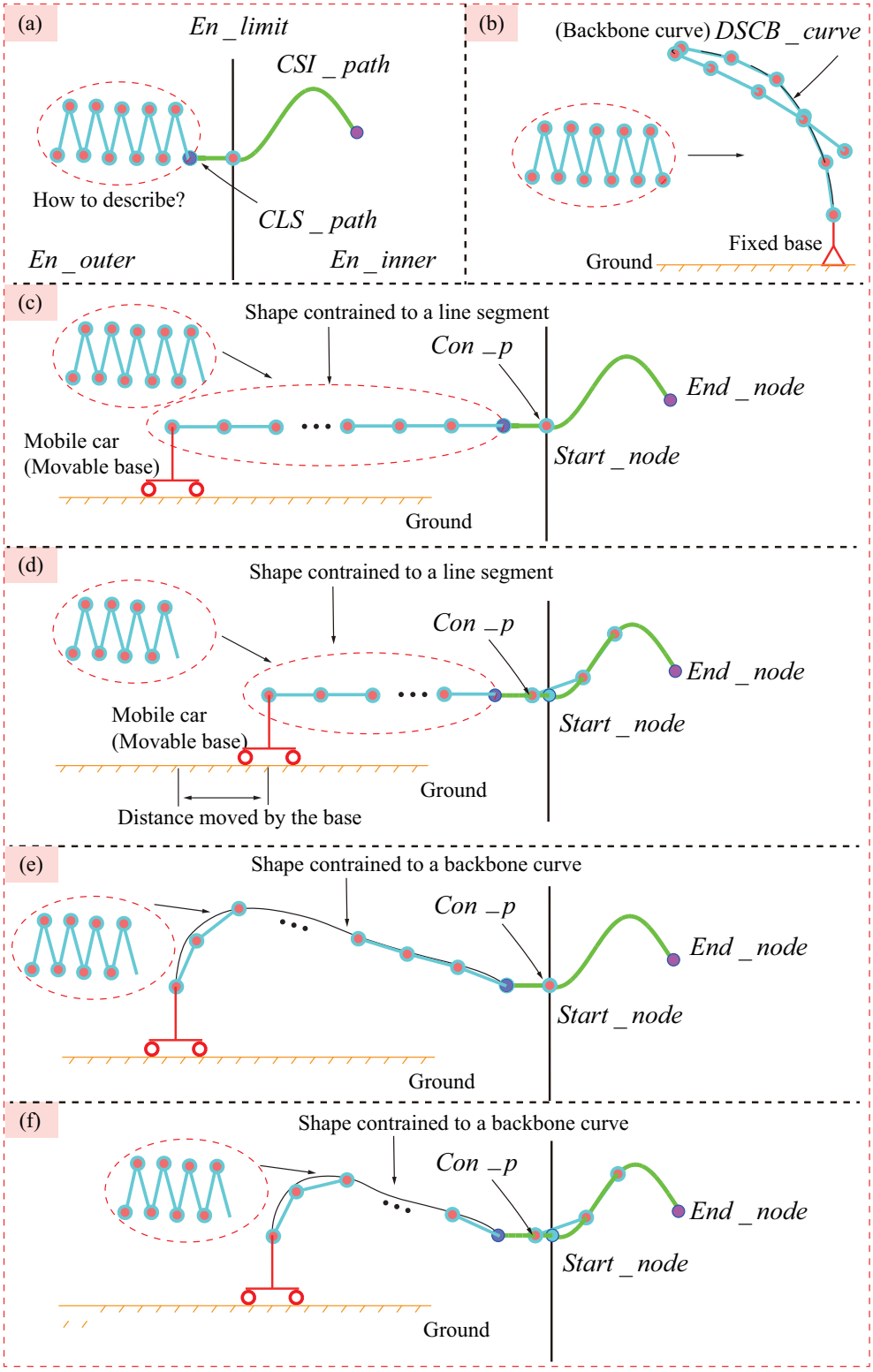

Fig. 9. Illustration for the reason why the line segment and the backbone curve are selected for the base-movable and base-fixed manipulators respectively.

a sample step length $Step_{\text{tree}}$, execute the RRT algorithm and then a *RRT_path* is obtained. The *RRT_path* is smoothed by cubic spline interpolation to get a *CSI_path*.

Algorithm 3 Obstacle avoidance based on RRT and shape control method (RRTSC)

Input: OB_i, r, l_{link}, n_{link}, n_{ob}, En_limit, $Start\_node$, End_node

Output: $P^j_{\text{hrm(m)}}$

function RRTSC
  Determine obstacle envelope spheres' center data OB_i and radius r, and initialize link length l_{link}, link number n_{link} and environment boundary En_limit
  for $j \leftarrow 1, \ \ldots \ , \ n_{\text{phase}}$ **do**
    while $(flag\_path = 1)$ **or** $(flag\_link = 1)$ **do**
      %Generate RRT path in inner part of environment boundary
      $RRT\_path \leftarrow Generate\_RRT(Start\_node, End\_node, OB_i, r, En\_limit)$
      %Cubic spline interpolation of a RRT path
      $CSI\_path \leftarrow Generate\_CSI(RRT\_path)$
      %Collision detection of the cubic spline interpolation curve
      $flag\_path \leftarrow Col\_det\_CSI(CSI\_path, OB_i, r)$
      %Collision detection of links
      $flag\_link \leftarrow Col\_det\_L(l_{\text{link}}, CSI\_path, OB_i, r)$
    end while
    %Generate a connecting line segment between cubic spline interpolation curve and backbone curve
    $CLS\_path \leftarrow Generate\_CLS(CSI\_path, \ l_{\text{link}})$
    %Shape control method
    $P^j_{\text{hrm(m)}} \leftarrow Shape\_control(l_{\text{link}}, n_{\text{link}}, CSI\_path, CLS\_path)$
  end for
  return $P^j_{\text{hrm(m)}}$
end function

Combining the CSI_path and a CLS_path we can obtain static shape control curve SSC_curve. The RRT_path, the CSI_path and the CLS_path are listed in supplementary materials fig10_rrt_path.xlsx, fig10_csi_path.xlsx and fig10_cls_path.xlsx, respectively. In this study, for the base-movable manipulator, a straight line segment of varying length is employed to constrain the macro shape of the manipulator in En_outer, and for base-fixed manipulators, the dynamic backbone curve is utilized for the same purpose. If the manipulator's base is fixed, the straight line segment will be no longer applicable, and it is necessary to choose the dynamic backbone curve. Fig. 9 is shown for detailed explanations. As shown in Fig. 9a, the working environment (or task space) is divided into two parts: obstacle space (En_inner) and free space (En_outer). When the manipulator passes into En_inner, the manipulator in En_outer will be a free state if we do not add constraints. We want to control the movement of the whole manipulator, so the free state must be avoided. It is wise to choose a straight line segment to constrain the shape

of the manipulator in *En_outer* when the base of the manipulator is movable, which is exactly what we did in our study. However, when the base of the manipulator is fixed with the ground as shown in Fig. 9b, the straight line segment will no longer work. That is determined by the essence of our RRTSC algorithm. In our RRTSC algorithm, all the joint positions of the manipulator are calculated from the end-effector to the base iteratively. As depicted in Fig. 9c-9d, when the manipulator's end-effector moves from the starting point to the target point, its base must move passively. Therefore, the straight line segment can not be applied to a base-fixed manipulator. It is necessary to choose a curve with variable length to ensure that the position of the base of the manipulator remains unchanged, and we choose the backbone curve in our study. We characterize the backbone curve as a dynamic curve because the length of the backbone curve changes with time, which is described in Fig. 7d-7f. When the manipulator moves from the starting point to the target point, the number of links exposed in *En_outer* will be reduced, and the length of the backbone curve should be reduced accordingly. If the manipulator's base is movable, it is not a good choice to use the backbone curve as the dynamic shape control curve, because adopting a straight line segment will make the process of solving $P^j_{\text{hrm\_dscb}(k)}$ more efficient. As shown in Fig. 9e-9f, if the manipulator is movable, it is feasible to use the backbone curve to constrain the manipulator in *En_outer*. When adopting the backbone curve, there exist three cases for updating $P^j_{\text{hrm\_dscb}(k)}$ if $P^j_{\text{hrm\_ssc}(i)}$ is updated: 1) Only $P_{\text{int}}(t)$ and $l(t)$ are updated; 2) Only $a_1(t)$, $a_2(t)$, $a_3(t)$ and $l(t)$ are updated; 3) Both $P_{\text{int}}(t)$, $a_1(t)$, $a_2(t)$, $a_3(t)$ and $l(t)$ are updated. Cases 2 and 3 have low computational efficiency compared to case 1. When the line segment are adopted, only $P_{\text{int}}(t)$ and $l(t)$ are needed to update (case 4). Compared with the corresponding case 1 where the backbone curve is utilized to calculate $P^j_{\text{hrm\_dscb}(k)}$ numerically, it will more computationally efficient to use the straight line segment (case 4) since analytical solutions for solving $P^j_{\text{hrm\_dscb}(k)}$ are available. We stipulate that manipulator's links in *En_outer* are arranged in a straight line, and the first two links are perpendicular to the ground. Then $P^j_{\text{hrm\_dscb}(k)}$ can be calculated easily. Set values of parameters: link number n_{link}, link length l_{link}, total number of phases n_{phase}, and coefficient *eps*. Then run function $Cal\_SSC\_p()$, thus getting $P^j_{\text{hrm\_ssc}(i)}$ in *En_inner*. Finally, combine $P^j_{\text{hrm\_dscb}(k)}$ and $P^j_{\text{hrm\_ssc}(i)}$ to obtain $P^j_{\text{hrm(m)}}$. The $P^j_{\text{hrm\_ssc}(i)}$, $P^j_{\text{hrm\_dscb}(k)}$, $P^j_{\text{hrm}(m)}$ are listed in corresponding excel files. The $n_{\text{link\_ssc}}$ and $n_{\text{link\_dscb}}$ are also constantly changing with phase j, and they are collected in fig10_n.xlsx. Fig. 10a shows the obstacle environment, the RRT, the *RRT_path*, the *CSI_path*, and the *CLS_path*. Fig. 10b depicts the initial configuration of the manipulator in the first phase ($j = 1$), and Fig. 10c represents the

final configuration of the manipulator in the last phase ($j = 90$). In order to visually show all phases of the obstacle avoidance movement of the manipulator, we provide an animation in supplementary material fig10_ani.mp4. Through animation, it can be found that the manipulator successfully avoids all obstacles and reaches a target point from an initial point.

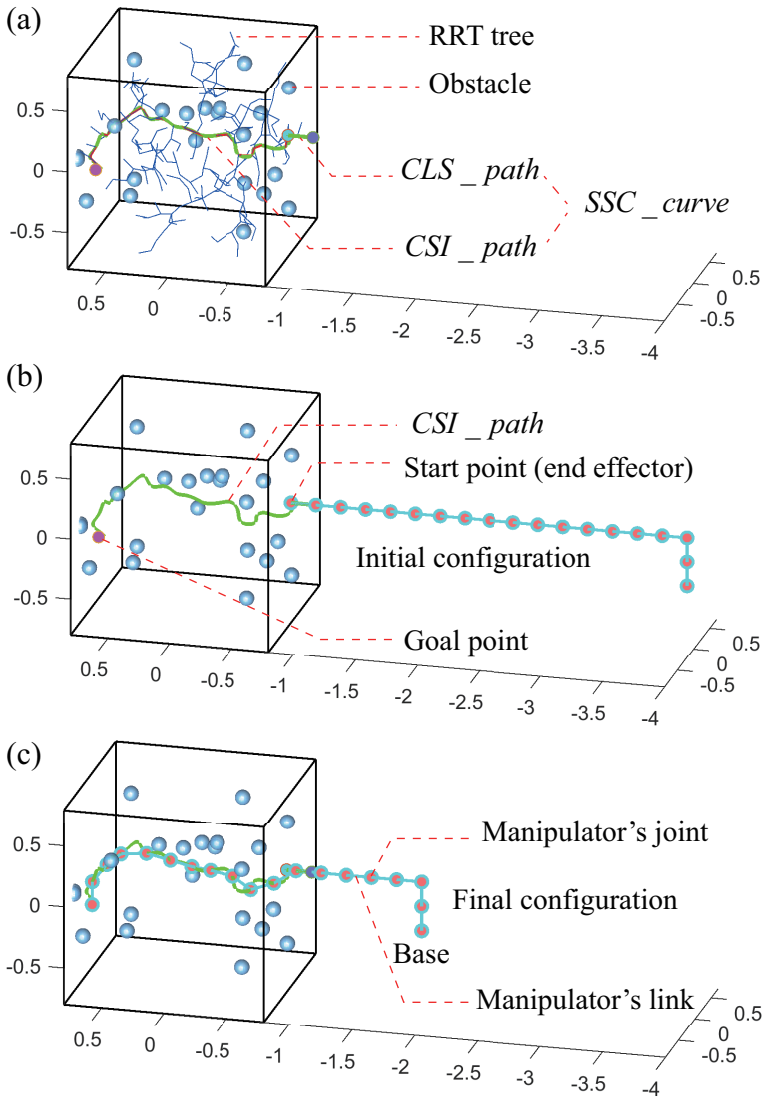

Fig. 10. Simulation of the base-moving manipulator in sphere obstacle environment.

3.3. Base-fixed hyper-redundant manipulators simulation

The RRTSC algorithm is proposed mainly for base-fixed hyper-redundant manipulators, and three simulations are carried out in this section, as depicted in Figs. 11-13. Table II show all the simulation-related data. When the RRTSC algorithm is employed for a base-movable manipulator, the $DSCB\_curve$ is a line segment of varying length. When RRTSC algorithm is utilized for a base-fixed manipulator, the backbone curve is selected

as *DSCB_curve*. Therefore, compared to the simulation-related data in Fig. 10, Figs. 11-13 add the parameters of the backbone curve: $P_{\text{int}}(t)$, $a_1(t)$, $a_2(t)$, $a_3(t)$, $l(t)$, $b_1(t)$, $b_2(t)$, $b_3(t)$, and b_4. To verify generality of the RRTSC algorithm, in this part, three obstacle environments are selected. In Fig. 11, the obstacle environment is the same as in Fig. 10. In Fig. 12, the obstacle is a hollow cabin and it is simplified to a combination of a cylindrical surface and a tapered surface. Fig. 13 shows a truss obstacle composed of many slender rods. Using manual multi-sphere approximation method, the cabin and the truss are modelled as shown in Fig. 3h and Fig. 3k, respectively. For display convenience, Figs. 12 and 13 only show half of the envelope spheres here. Obstacle environment, RRT, *RRT_path*, *CSI_path* and *CLS_path* are shown in Figs. 11a-13a. Figs. 11b-13b, 11c-13c, and 11d-13d depict initial, intermediate and final configurations, respectively. For complete movement animations, refer to corresponding supplementary materials.

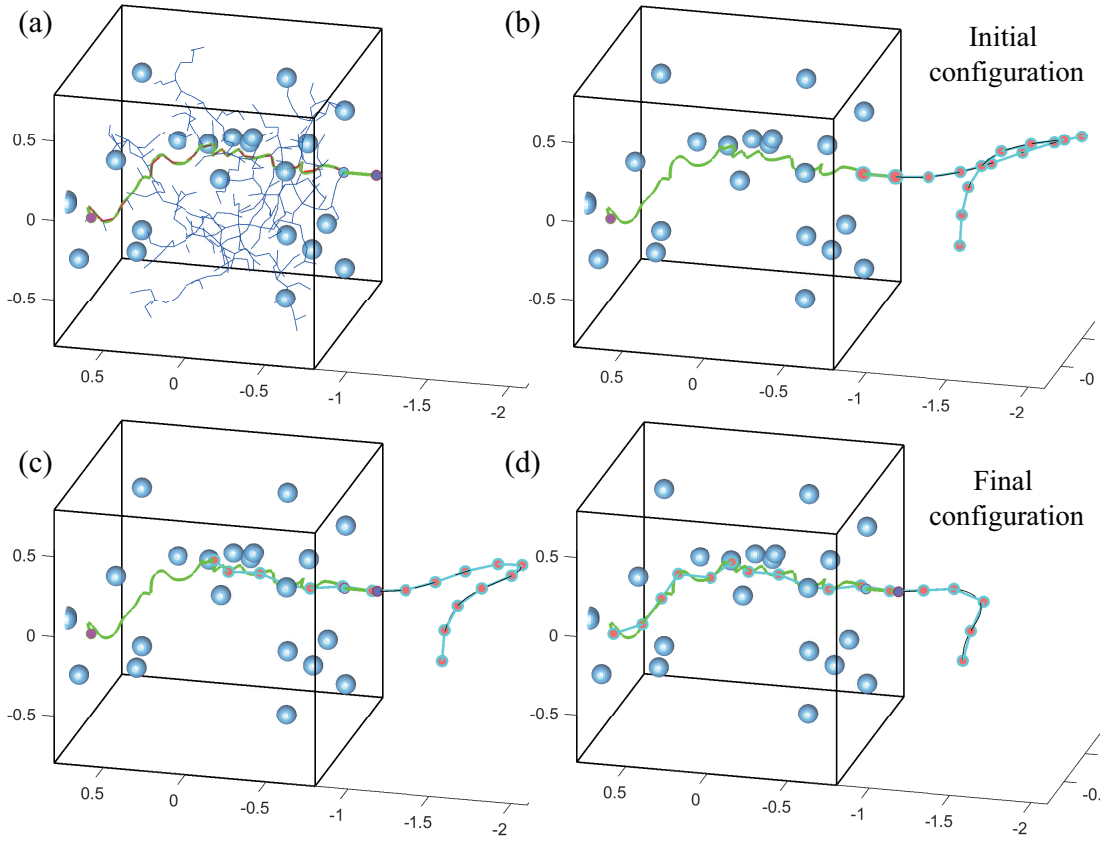

Fig. 11. Simulation of base-fixed manipulators in sphere obstacle environment.

3.4. Mapping relationship between a simplified prototype model and its link model

Simulations in Figs. 10-13 are carried out through the abstract link model of the manipulator. When the RRTSC algorithm is applied on a real manipulator, the link model must have enough information to drive the robot, which needs specific joint angle

Table II . Simulation-related data.

Object	Fig. 10	Fig. 11	Fig. 12	Fig. 13
OB_i	fig10_ob.xlsx	fig11_ob.xlsx	fig12_ob.xlsx	fig13_ob.xlsx
r	0.06	0.06	150	2
En_limit	(-0.9,0.7, -0.8,0.8, -0.8,0.8)	(-0.9,0.7, -0.8,0.8, -0.8,0.8)	(-100,4600, -1800,1800, -1800,1800)	(-60,60, -60,160, -20,120)
$Start\_node$	(-0.2,-0.8,0.2)	(-0.2,-0.8,0.2)	(0,0,10)	(0,-50,50)
End_node	(-0.8,0.6,0)	(-0.8,0.6,0)	(2000,700,700)	(20,80,80)
$Step_{\text{tree}}$	0.1067	0.08	300	13
RRT_path	fig10_rrt_path.xlsx	fig11_rrt_path.xlsx	fig12_rrt_path.xlsx	fig13_rrt_path.xlsx
CSI_path	fig10_csi_path.xlsx	fig11_csi_path.xlsx	fig12_csi_path.xlsx	fig13_csi_path.xlsx
CLS_path	fig10_cls_path.xlsx	fig11_cls_path.xlsx	fig12_cls_path.xlsx	fig13_cls_path.xlsx
n_{link}	18	16	22	33
l_{link}	0.2	0.2	300	10
n_{phase}	90	87	93	97
eps	0.002	0.02	3	0.1
$n_{\text{link\_ssc}}$ and $n_{\text{link\_dscb}}$	fig10_n.xlsx	fig11_n.xlsx	fig12_n.xlsx	fig13_n.xlsx
Con_p	fig10_cp.xlsx	fig11_cp.xlsx	fig12_cp.xlsx	fig13_cp.xlsx
$P^j_{\text{hrm\_ssc}(i)}$	fig10_p_ssc.xlsx	fig11_p_ssc.xlsx	fig12_p_ssc.xlsx	fig13_p_ssc.xlsx
$P^j_{\text{hrm\_dscb}(k)}$	fig10_p_dscb.xlsx	fig11_p_dscb.xlsx	fig12_p_dscb.xlsx	fig13_p_dscb.xlsx
$P^j_{\text{hrm}(m)}$	fig10_p.xlsx	fig11_p.xlsx	fig12_p.xlsx	fig13_p.xlsx
Animation	fig10_ani.mp4	fig11_ani.mp4	fig12_ani.mp4	fig13_ani.mp4
$P_{\text{int}}(t)$	–	[-0.2;-1.4;-0.2]	[-600;0;-1190]	[0;-70;0]
$a_1(t)$,$a_2(t)$ and $a_3(t)$	–	fig11_a.xlsx	fig12_a.xlsx	fig13_a.xlsx
$b_1(t)$,$b_2(t)$,$b_3(t)$ and $b_4(t)$	–	(0.51,-3.15,1.56,3.2)	(0.51,-1.58,1.56,3.2)	(0.51,-3.15,1.56,3.2)

sequences. To fulfill that, a coordinate system must be established on the link model. Fig. 14 shows a simplified prototype model and its link model with coordinates, and it is convenient for us to understand their mapping relationship. The prototype models (Figs. 14a, 14c, 14e, and 14g) are composed of same modules and each module has a joint

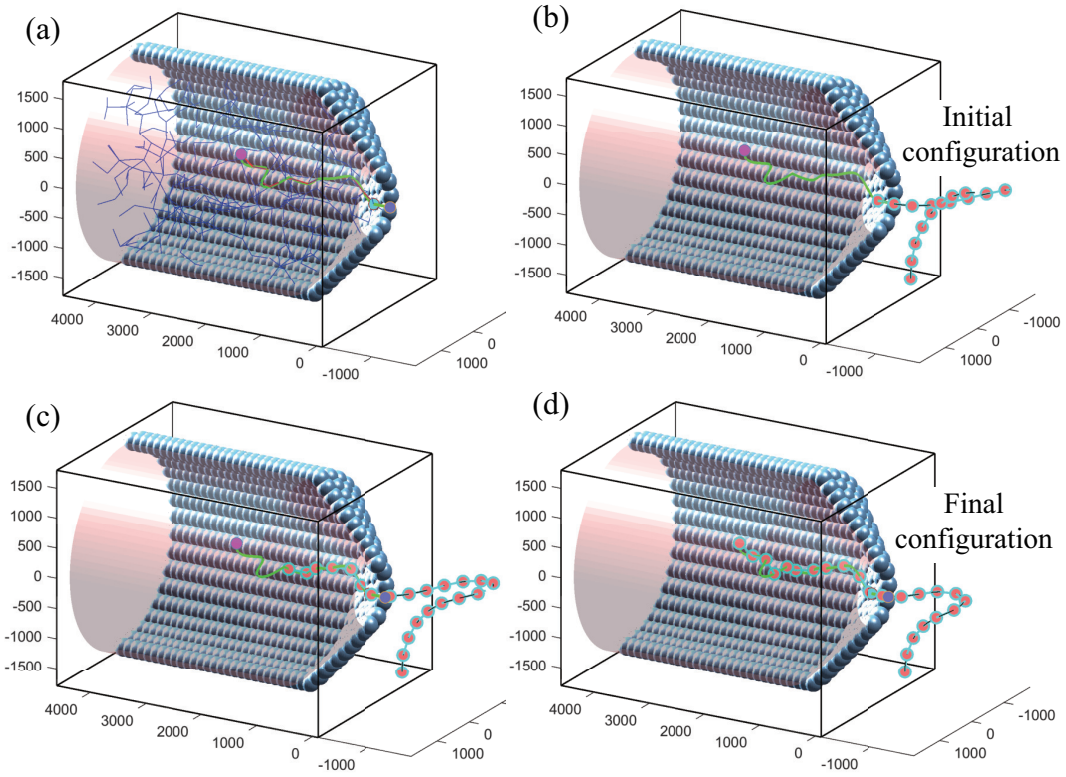

Fig. 12. Simulation of base-fixed manipulators in cabin obstacle environment.

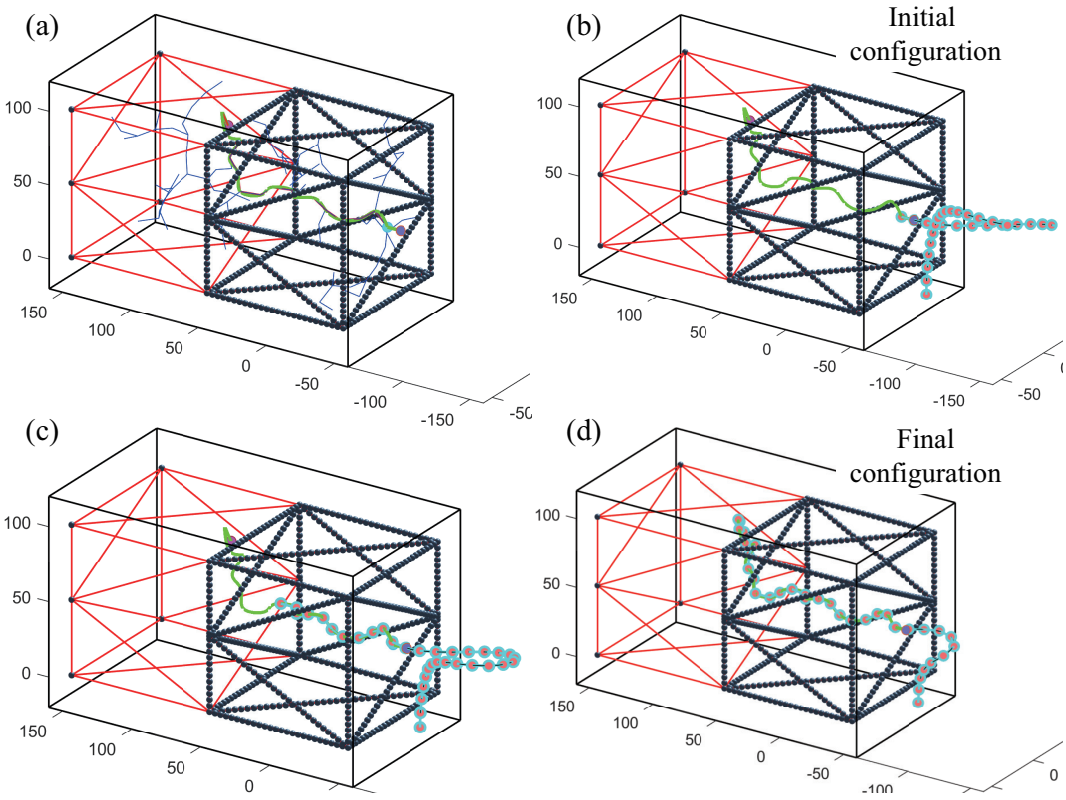

Fig. 13. Simulation of base-fixed manipulators in truss obstacle environment.

with three degrees of freedom (roll angle γ, yaw angle α, and pitch angle β). The link models (Figs. 14b, 14d, 14f, and 14h) are obtained by the RRTSC algorithm. The local coordinates are obtained via following steps: 1) every origin of the local coordinates is located on ${}^{j}P_{\mathrm{hrm}(m)}$; 2) y axis is the unit vector from ${}^{j}P_{\mathrm{hrm}(m)}$ to $P_{\mathrm{hrm}(m+1)}$; 3) x axis is selected by satisfying two conditions. One is that it is vertical to the y axis, and the other is that it is on a plane parallel to the ground plane; 4) z axis is calculated via a right hand rule. When the local coordinates are available, the angle sequences can be solved by matrix transformation, and the detailed solution process can be found in ref. [47]. It should be pointed out that different rotating sequences or joint configurations lead to different matrix transformation processes. On the other hand, the link model must take the enveloping diameter into consideration when applying the RRTSC algorithm on a real manipulator. This problem can be easily solved by adjusting the threshold value of distance detection when executing algorithm 1.

3.5. Comparision with existing works

The RRTSC algorithm belongs to the backbone-curve-based method, the same as the existing works in refs. [40, 41]. In this section, a comparison with these two works is conducted, and the similarities and differences are summarized.

3.5.1. Similarity.

1. Task space is divided into two parts: obstacle space and free space.
2. In obstacle space, a collision-free path from a given start point to a goal point, denoted by CF_path, is constructed based on a Generalized Voronoi Graph (GVG) in ref. [40], a harmonic potential function in ref. [41] or an RRT algorithm in our study.
3. A dynamic shape control backbone curve in free space, denoted by $DSCBFS\_curve$, is constructed in ref. [41] and our study.

3.5.2. Difference.

1. The usage of CF_path is different. In refs. [40, 41], CF_path is utilized to form a dynamic shape control backbone curve in obstacle space, denoted by $DSCBOS\_curve$ and $DSCBOS\_curve$ to obtain a new dynamic shape control curve in the entire task space, denoted by $DSCBETS\_curve$. In our study, CF_path is utilized to form a static shape control curve, denoted by SSC_curve.
2. Different from the works in refs. [40, 41] in which only a dynamic shape control backbone curve in the entire task space ($DSCBETS\_curve$) is constructed in real-time for constraining the macro shape of the manipulator, our proposed RRTSC algorithm

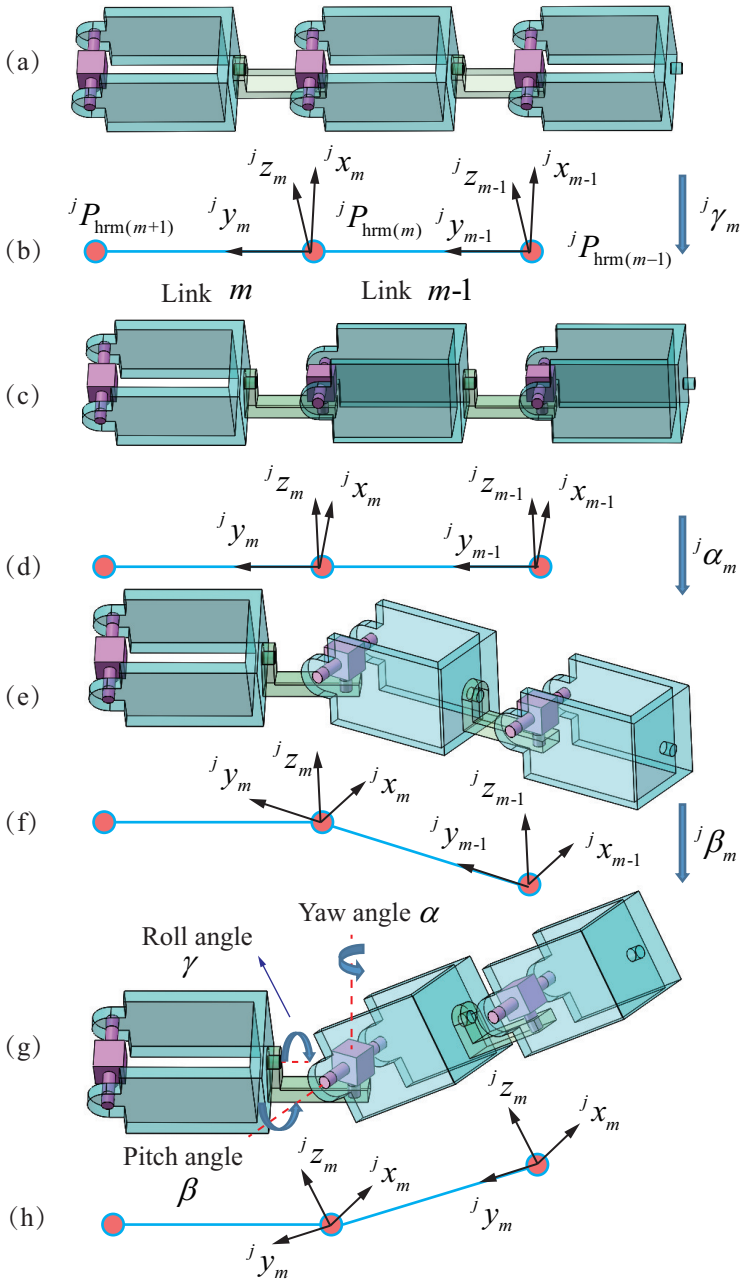

Fig. 14. Mapping relationship between a simplified prototype model and its link model.

combines a static shape control curve (SSC_curve) and a dynamic shape control backbone curve in free space ($DSCBFS\_curve$) for the same purpose. It should be noted that $DSCBFS\_curve$ is abbreviated as $DSCB\_curve$ in our study. This form is written here to distinguish between $DSCBFS\_curve$ and $DSCBOS\_curve$ more clearly. To our best knowledge, it is the first attempt to accomplish obstacle avoidance of manipulators with a combination of a static shape control curve and a dynamic shape control backbone curve among existing works. How to achieve this combination is described in detail in our study.

3. Regarding how to dynamically generate $DSCBETS\_curve$, the work in ref. [40] only provides conceptual descriptions, and no specific mathematical formulations or algorithms are given. In ref. [41], these conceptual descriptions in ref. [40] come to a realization mathematically. However, how to use CF_path to determine $DSCBOS\_curve$, and how to choose the specific length of $DSCBFS\_curve$ are not mentioned clearly. In our study, detailed information about the generating process of the SSC_curve and the $DSCBFS\_curve$ are provided via the RRTSC algorithm.
4. In ref. [41], $DSCBFS\_curve$ is given with a specific mathematical form, and how this form is constructed is not mentioned. In our study, we detail the construction process of this form and extend it to a more general form. In other words, the mathematical form is not unique and can be replaced by other ones if proper mode functions are selected via our proposed general form.
5. The most important difference is that our proposed RRTSC algorithm can avoid a common problem existing in the works in refs. [40, 41]. Based on the Generalized Voronoi Graph,[40] or the harmonic potential function,[41] the works in refs. [40, 41] ensure that $DSCBETS\_curve$ is collision-free with obstacles. However, when utilizing $DSCBETS\_curve$ to constraining the macro shape of the manipulator, the links of the manipulator may collide with obstacles, because the links can not fully approximate the shape of $DSCBETS\_curve$ unless the number of the links is large enough (this is not realistic). Our proposed RRTSC algorithm does not have this problem. In order to avoid this problem, an RRT algorithm and a cubic spline interpolation are adopted to construct a smooth collision-free path (CF_path) from a start point to a goal point in obstacle space, and then two ends (a leading end and a trailing end) of a leading link of the manipulator are attached to this collision-free path and move continuously until the leading end of the leading link reaches a given goal point, during which collision detections are executed between the leading link and obstacles. If a collision occurs between the leading link and obstacles, a new collision-free path (CF_path) is generated until no collision occurs. The motion of the remaining links follows the motion of the leading link in turn, so all links will not collide with obstacles (this phenomenon is called 'follow-the-leading-link' in this study). The essence of our RRTSC algorithm is to reduce the multi-link collision detection to single-link one, which provides the algorithm with high computational efficiency. It should be noted that the phenomenon of the 'follow-the-leading-link' in this study is different from a similar phenomenon of the 'follow-the-leader' in refs. [40, 41]. The phenomenon of the 'follow-the-leading-link' means that the motion of the remaining links follow

the motion of the leading link in turn, while the phenomenon of the 'follow-the-leader' means that the entire manipulator is advanced into a goal point by the 'leader' (head, end-effector, or leading end of a leading link). Let us explain in more detail how the phenomenon of the 'follow-the-leader' is produced. In refs. [40,41], the manipulator is advanced by three steps: 1) Add an δs increment via the GVG[40] or the harmonic potential function[41] to form $DSCBOS\_curve$; 2) Connect $DSCBFS\_curve$ and $DSCBOS\_curve$ to form $DSCBETS\_curve$; 3) Utilize an optimization algorithm to fit the macro shape of the manipulator to $DSCBETS\_curve$ as closely as possible, during which the 'leader' (head, end-effector, or leading end of a leading link) is always put on the δs while the remaining joint positions are calculated via the optimization algorithm; 4) Repeat the steps 1-3 continuously until the 'leader' reaches a given goal point. Steps 1-4 make the movement of the manipulator seem to be driven by the 'leader', and this phenomenon is denoted as 'follow-the-leader'.

4. Conclusions and Future Work

In this study, a kinematic obstacle avoidance algorithm, referred to as the RRTSC algorithm, mainly for space hyper-redundant manipulators is proposed. Different from the existing backbone-curve-based works in which only a dynamic backbone curve is utilized to constraining the macro shape of the manipulator, our algorithm successfully combines a static curve and a dynamic backbone curve for the same purpose. In addition, our algorithm solves the common problem (the backbone curve is collision but the manipulator may collide with obstacles) existing in the previous backbone-curve-based works. Our algorithm can be applied to both base-movable and base-fixed manipulators, and four simulations are conducted to verify the effectiveness of the RRTSC algorithm. Space hyper-redundant manipulator plays an important role in space station maintenance, because it has more flexible obstacle avoidance ability compared to other conventional manipulators. Therefore, this proposed algorithm is meaningful.

However, the RRTSC algorithm has drawbacks which can be further improved in our future works. Firstly, the collision-free path is not optimal. Some optimization algorithms can be combined to shorten the path length as much as possible. Secondly, this planning method is designed for static obstacles in this study. If obstacles' positions are time-varying, the RRT path should be time-varying accordingly. In order to obtain the time-varying RRT path, the original RRT algorithm may be modified by integrating multi-level RRT in series, rules of obstacles' motion and other detection sub-algorithms. It will be the main direction of our future works.

Acknowledgements

This work is supported by the National Key R&D Program of China (2018YFB1304600), the National Natural Science Foundation of China (51775541) and the CAS Interdisciplinary Innovation Team (JCTD-2018-11).

References

1. G. S. Chirikjian and J. W. Burdick, "A modal approach to hyper-redundant manipulator kinematics," *IEEE Trans. Rob. Autom.* **10**(3), 343–354 (1994).
2. G. S. Chirikjian and J. W. Burdick, "Hyper-redundant Robot Mechanisms and Their Applications," **In**: *Proc. IEEE RSJ Int. Workshop Intell. Robots Syst. (IROS)* (1991) pp. 185–190.
3. A. Wolf, H. B. Brown, R. Casciola, A. Costa, M. Schwerin, E. Shamas and H. Choset, "A Mobile Hyper Redundant Mechanism for Search and Rescue Tasks," **In**: *Proc. IEEE Int. Conf. Intell. Rob. Syst.* (2003) pp. 2889–2895.
4. J. Tang, Y. Zhang, F. Huang, J. Li, Z. Chen, W. Song, S. Zhu and J. Gu, "Design and kinematic control of the cable-driven hyper-redundant manipulator for potential underwater applications," *Appl. Sci.* **9**(6), article number 1142 (2019).
5. W. Wan, C. Sun and J. Yuan, "Adaptive caging configuration design algorithm of hyper-redundant manipulator for dysfunctional satellite pre-capture," *IEEE Access* **8**, 22546–22559 (2020).
6. X. Zhang and J. Liu, "Effective motion planning strategy for space robot capturing targets under consideration of the berth position," *Acta Astronaut.* **148**, 403–416 (2018).
7. X. Zhang, J. Liu, J. Feng, Y. Liu and Z. Ju, "Effective capture of nongraspable objects for space robots using geometric cage pairs," *IEEE/ASME Trans. Mechatron.* **25**(1), 95–107 (2020).
8. Z. Mu, T. Liu, W. Xu, Y. Lou and B. Liang, "Dynamic feedforward control of spatial cable-driven hyper-redundant manipulators for on-orbit servicing," *Robotica* **37**(1), 18–38 (2019).
9. T. Rybus, "Obstacle avoidance in space robotics: Review of major challenges and proposed solutions," *Prog. Aeosp. Sci.* **101**, 31–48 (2018).
10. M. D. Marcos, J. A. T. Machado and T.-P. Azevedo-Perdicoulis, "A fractional approach for the motion planning of redundant and hyper-redundant manipulators," *Signal Process.* **91**(3), 562–570 (2011).
11. A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. J. Robot. Res.* **4**(3), 109–117 (1985).
12. C. Qiu, Q. Cao and S. Miao, "An on-line task modification method for singularity avoidance of robot manipulators," *Robotica* **27**(4), 539–546 (2009).
13. B. Liao and W. Liu, "Pseudoinverse-type bi-criteria minimization scheme for redundancy resolution of robot manipulators," *Robotica* **33**(10), 2100–2113 (2015).
14. J. Wan, H. Wu, R. Ma and L. Zhang, "A study on avoiding joint limits for inverse kinematics of redundant manipulators using improved clamping weighted least-norm method," *J. Mech. Sci. Technol.* **32**(3), 1367–1378 (2018).
15. S. Ma and D. Nenchev, "Local torque minimization for redundant manipulators: A correct formulation," *Robotica* **14**(2), 235–239 (1996).

16. J. Barraquand, B. Langlois and J.C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Trans. Syst. Man. Cybern.* **22**(2), 224–241 (1992).
17. J. Barraquand and J.C. Latombe, "Robot motion planning: A distributed representation approach," *Int. J. Robot. Res.* **10**(6), 628–649 (1991).
18. E. S. Conkur, "Path planning using potential fields for highly redundant manipulators," *Robot. Auton. Syst.* **52**(2–3), 209–228 (2005).
19. T. Lozano-Perez, "Spatial planning: a configuration space approach," *IEEE Trans. Comput.* **32**(2), 108–120 (1983).
20. L.E. Kavraki, P. Svestka, J.C. Latombe and M.H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Rob. Autom.* **12**(4), 566–580 (1996).
21. B. Dasgupta, A. Gupta and E. Singla, "A variational approach to path planning for hyper-redundant manipulators," *Robot. Auton. Syst.* **57**(2), 194–201 (2009).
22. M. D. Marcos, J. A. T. Machado and T.-P. Azevedo-Perdicoulis, "Trajectory planning of redundant manipulators using genetic algorithms," *Commun. Nonlinear Sci. Numer. Simul.* **14**(7), 2858–2869 (2009).
23. J. Zhao, L. Zhao and H. Liu, "Motion Planning of Hyper-redundant Manipulators Based on Ant Colony Optimization," **In**: *Proc. IEEE Int. Conf. Robot. Biomim., (ROBIO)* (2016) pp. 1250–1255.
24. H. Ananthanarayanan and R. Ordóñez, "A fast converging optimal technique applied to path planning of hyper-redundant manipulators," *Mech. Mach. Theory* **118**, 231–246 (2017).
25. J. J. Kuffner and S. M. Lavalle, "RRT-connect: A Efficient Approach to Single-query Path Planning," **In**: *Proc. IEEE Int. Conf. Rob. Autom.* (2000) pp. 995–1001.
26. M.V. Weghe, D. Ferguson and S.S. Srinivasa, "Randomized Path Planning for Redundant Manipulators without Inverse Kinematics," **In**: *Proc. IEEE-RAS Int. Conf. Humanoid Rob.* (2007) pp. 477–482.
27. N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner and R. Dillmann, "Humanoid Motion Planning for Dual-Arm Manipulation and Re-Grasping Tasks," **In**: *Proc. IEEE Int. Conf. Intell. Rob. Syst. (IROS)* (2009) pp. 2464–2470.
28. A. Shkolnik and R. Tedrake, "Path Planning in 1000+ Dimensions using a Task-Space Voronoi Bias," **In**: *Proc. IEEE Int. Conf. Rob. Auto.* (2009) pp. 2061–2067.
29. G. Mesesan, M.A. Roa, E. Icer and M. Althoff, "Hierarchical Path Planner using Workspace Decomposition and Parallel Task-Space RRTs," **In**: *Proc. IEEE Int. Conf. Intell. Rob. Syst. (IROS)* (2018) pp. 1–9.
30. A. A. Maciejewski and J. J. Fox, "Path planning and the topology of configuration space," *IEEE Trans. Rob. Autom.* **9**(4), 444–456 (1993).
31. J. Wang, M.Q. Meng and O. Khatib, " EB-RRT: Optimal motion planning for mobile robots," *IEEE Trans. Autom. Sci. Eng.* **17**(4), 2063–2073 (2020).
32. Y. Zhang and J. Wang, " Obstacle avoidance for kinematically redundant manipulators using a dual neural network," *IEEE Trans. Syst. Man. Cybern.* **34**(1), 752–759 (2004).
33. D. Guo and Y. Zhang, "A new inequality-based obstacle-avoidance MVN scheme and its application to redundant robot manipulators," *IEEE Trans. Syst. Man. Cybern.* **42**(6), 1326–1340 (2012).
34. D. Guo and Y. Zhang, "Acceleration-level inequality-based MAN scheme for obstacle avoidance of redundant robot manipulators," *IEEE Trans. Ind. Electron.* **61**(12), 6903–6914 (2014).

35. A. Hassan, M. EI-Habrouk and S. Deghedie, "Inverse kinematics of redundant manipulators formulated as quadratic programming optimization problem solved using recurrent neural networks: a review," *Robotica* **38**(8), 1495–1512 (2020).
36. Z. Zhang, L. Zheng, J. Yu, Y. Li and Z. Yu, "Three recurrent neural networks and three numerical methods for solving a repetitive motion planning scheme of redundant robot manipulators," *IEEE/ASME Trans. Mech.* **22**(3), 1423–1434 (2017).
37. M.S. Menon, V.C. Ravi and A. Ghosal, "Trajectory planning and obstacle avoidance for hyper-redundant serial robots," *J. Mech. Robot.* **9**(4), 041010 (2017).
38. K. P. Ashwin, A. N. Chaudhury and A. Ghosal, "Efficient representation of ducts and cluttered spaces for realistic motion planning of hyper-redundant robots through confined paths," *Comput.-Aided Des.* **119**, article number 102777 (2020).
39. G. S. Chirikjian and J. W. Burdick, "An Obstacle Avoidance Algorithm for Hyper-redundant Manipulators," **In**: *Proc. IEEE Int. Conf. on Robotics and Automation* (1990) pp. 625–631.
40. H. Choset and W. Henning, "A follow-the-leader approach to serpentine robot motion planning," *J. Aerosp. Eng.* **12**(2), 65–73 (1999).
41. F. Fahimi, H. Ashrafiuon and C. Nataraj, "Obstacle avoidance for spatial hyper-redundant manipulators using harmonic potential functions and the mode shape technique," *J. Rob. Syst.* **20**(1), 23–33 (2003).
42. Z. Mu, T. Liu, W. Xu, Y. Lou and B. Liang, "A hybrid obstacle-avoidance method of spatial hyper-redundant manipulators for servicing in confined space," *Robotica* **37**(6), 998–1019 (2019).
43. S. Ma, M. Watanabe and H. Kondo, "Dynamic Control of Curve-Constrained Hyper-redundant Manipulators," **In**: *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom. (CIRA)* (2001) pp. 83–88.
44. S. Sreenivasan, P. Goel and A. Ghosal, "A real-time algorithm for simulation of flexible objects and hyper-redundant manipulators," *Mech. Mach. Theory* **45**(3), 454–466 (2010).
45. M.S. Menon, G.K. Ananthasuresh and A. Ghosal, "Natural motion of one-dimensional flexible objects using minimization approaches," *Mech. Mach. Theory* **67**, 64–76 (2013).
46. M.S. Menon, B. Gurumoorthy and A. Ghosal, "Efficient simulation and rendering of realistic motion of one-dimensional flexible objects," *Comput. -Aided Des.* **75**, 13–26 (2016).
47. X. Zhang, J. Liu, Z. Ju and C. Yang, "Head-raising of snake robots based on a predefined spiral curve method," *Appl. Sci.* **8**(11), 1-20 (2018).
48. F. Fahimi, H. Asharafiuon and C. Nataraj, "An improved inverse kinematic and velocity solution for spatial hyper-redundant robots," *IEEE Trans. Rob. Autom.* **18**(1), 103–107 (2002).
49. W. Xu,Z. Mu, T. Liu and B. Liang, "A modified modal method for solving the mission-oriented inverse kinematics of hyper-redundant space manipulators for on-orbit servicing," *Acta Astronaut.* **139**, 54–66 (2017).
50. H. Ananthanarayanan and R. Ordóñez, "Real-time inverse kinematics of (2n+1) DOF hyper-redundant manipulator arm via a combined numerical and analytical approach," *Mech. Mach. Theory* **91**, 209–226 (2015).
51. E. K. Xidias, "Time-optimal trajectory planning for hyper-redundant manipulators in 3D workspaces," *Robot. Comput.-Integr. Manuf.* **50**, 286–298 (2018).

52. A. H. Barr, "Superquadrics and angle-preserving transformations," *IEEE Comput. Graph. Appl.* **1**(1), 11–23 (1981).
53. G. Bradshaw and C. O'Sullivan, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Trans. Graph.* **23**(1), 1–26 (2004).
54. S. Stolpner, P. Kry and K. Siddiqi, "Medial spheres for shape approximation," *IEEE Trans. Pattern Anal. Mach. Intell.* **34**(6), 1234–1240 (2012).

Appendix: Symbols Utilized in this Study

Symbols utilized in this study are summarized in Table III-VI, and acronyms utilized in this study are concluded in Table VII.

Table III . Symbols utilized in this study (part 1).

Symbol	**Definition**
RRT	Rapidly exploring random tree
RRTSC	The obstacle avoidance algorithm, proposed in this study, using combination of RRT algorithm and shape control method
En_limit	Environment boundary which is utilized to limit the sampling space of the RRT algorithm and represented by $En\_limit = (X_l,\ X_r,\ Y_l,\ Y_r,\ Z_l,\ Z_r)$
X_l	Left limit coordinate value of En_limit on x axis, the same as Y_l and Z_l
X_r	Right limit coordinate value of En_limit on x axis, the same as Y_r and Z_r
En_inner	Inner part of 3D task space which is represented by $En\_inner = \{P \mid P \in En\_limit\}$
En_outer	Outer part of 3D task space which is denoted by $En\_outer = \{P \mid P \notin En\_limit\}$
RRT_path	Collision free path obtained via the RRT algorithm
CSI_path	Cubic spline interpolation curve obtained after smoothing RRT_path
r	Obstacle envelope sphere's radius
$d_{\max}$	Maximum distance from the centroid of an arbitrary geometry entity to its envelope boundary
OB_i	Obstacle envelope sphere's center data
$Tree$	Representation of a rapidly exploring random tree when writing pseudo code
$Start\_node$	Start path point of RRT_path
End_node	End path point of RRT_path
One_RRT_path	A line segment constructed by one leaf node and its parent node
$Step_{\text{tree}}$	Length of One_RRT_path, or sampling step length of the RRT algorithm

Table IV . Symbols utilized in this study (part 2).

Symbol	**Definition**
$Col\_det\_RRT()$	A function utilized to detect collision between One_RRT_path and obstacles
$Add\_node()$	A function utilized to add New_node to $Tree$
$flag\_tree$	A flag variable to determine whether $Tree$ has reached to End_node
$Extend\_tree()$	A function to continually add New_node to $Tree$
$Find\_path()$	A function used to obtain RRT_path form $Tree$
$flag\_RRT$	A flag variable to judge whether One_RRT_path collides with obstacles
δ	A coefficient vector used to get discretization point of One_RRT_path
$Rand\_node$	Random sampling points in En_inner
$Nearst\_node$	A node in $Tree$ closest to $Rand\_node$
New_node	A new node which can be added to $Tree$
n_{ob}	Number of envelope spheres
$OB1$	Obstacle representation in Figs. 5c-5e, the same as $OB2$
θ_1	Joint angle of a manipulator in Fig. 5c, the same as θ_2
$P(s,t)$	An arbitrary point on a backbone curve
$P_{\text{int}}(t)$	Initial point of a backbone curve in base frame, defined as $[x_{\text{int}},\ y_{\text{int}},\ z_{\text{int}}]^{\text{T}}$
x_{int}	x value of P_{int}, the same as y_{int} and z_{int}
s	Independent variable representing arc length of a backbone curve
t	Independent variable representing time
$l(t)$	Total length of a backbone curve at time t
$F(\sigma,t)$	Unit vector tangent to a backbone curve at $s=\sigma$
$F'(\sigma,t)$	Projection vector of $F(\sigma,t)$ on $x-y$ plane of base frame
$K(\sigma,t)$	Angle between $F'(\sigma,t)$ and y axis of base frame at $s=\sigma$
$T(\sigma,t)$	Angle between $F'(\sigma,t)$ and $F(\sigma,t)$
$b_1(t)$	A coefficient to specify orientation at the start point of a backbone curve, the same as $b_3(t)$

Table V . Symbols utilized in this study (part 3).

Symbol	**Definition**
$b_2(t)$	A coefficient to specify orientation at the end point of a backbone curve, the same as $b_4(t)$
$a_1(t)$	A coefficient which can be used to adjust shape of a backbone curve, the same as $a_2(t)$ and $a_3(t)$
$c_i(t)$	Modal participation factors, the same as $d_i(t)$
$f_i(s)$	Mode functions, the same as $g_i(s)$
n_1	The number of mode functions for $K(s,t)$
n_2	The number of mode functions for $T(s,t)$
l_{link}	Length of links for a hyper-redundant manipulator
n_{link}	Number of links for a hyper-redundant manipulator
eps	A coefficient used to adjust matching accuracy of l_{link}
P_i	Connecting points of link model for a hyper-redundant manipulator
P'	Equally spaced points on a backbone curve
P_{target}	Target point for a backbone curve or a hyper-redundant manipulator
$P_i\|_{i=n_{\text{link}}}$	Connecting point P_i at $i = n_{\text{link}}$
CLS_path	Additional connecting line segment between a cubic spline interpolation curve and a backbone curve
SSC_curve	Static shape control curve combing CLS_path and CSI_path
$DSCB\_curve$	Dynamic shape control backbone curve
$P^j_{\text{hrm}(m)}$	The connecting point, labeled m, of the link model for a hyper-redundant manipulator, at phase j and the phase refers to the position of $P_{\text{hrm}(m)}\|_{m=n_{link}}$ on SSC_curve or $DSCB\_curve$
$P^j_{\text{hrm\_ssc}(i)}$	$P^j_{\text{hrm}(m)}$ on static shape control curve SSC_curve
$P^j_{\text{hrm\_dscb}(k)}$	$P^j_{\text{hrm}(m)}$ on dynamic shape control curve $DSCB\_curve$
$n_{\text{link\_ssc}}$	Number of links constrained via static shape control curve SSC_curve

Table VI . Symbols utilized in this study (part 4).

Symbol	Definition
$n_{\text{link\_dscb}}$	Number of links constrained via dynamic shape control backbone curve $DSCB\_curve$
Con_p	Point connecting SSC_curve and $DSCB\_curve$
$Cal\_SSC\_p()$	A function to calculate $P^j_{\text{hrm\_ssc}(i)}$ via SSC_curve, based on Eq. (18)
n_{phase}	Total number of phases during a shape control cycle, and the phase, in this study, means a specific position of the end effector on CSI_path
n_{config}	Total number of configurations of the manipulator during a shape control cycle
$Ind_{\text{p\_dis\_csi}}$	Full index number of discrete points of cubic spline interpolation path
$n_{\text{p\_dis\_csi}}$	The total number of discrete points of cubic spline interpolation path
$\Delta Ind_{\text{p\_dis\_csi}}$	Index increment of $Ind_{\text{p\_dis\_csi}}$
$Ind_{\text{p\_dis\_csi\_ee}}$	Partial index number of discrete points of cubic spline interpolation path for updating the position of the end effector
$Build\_DSCB\_curve()$	A function, programmed via Eqs. (2) and (5), to build the $DSCB\_curve$
$Cal\_DSCB\_p()$	A function, programmed via Eqs. (11)-(16), to calculate $P^j_{\text{hrm\_descb}(k)}$
$Combine\_p()$	A function utilized to combine $P^j_{\text{hrm\_ssc}(i)}$ and $P^j_{\text{hrm\_descb}(k)}$ to obtain $P^j_{\text{hrm}(m)}$
$Generate\_RRT()$	A function that integrates a RRT algorithm
$Shape\_control()$	A function that integrates two shape control methods
CF_path	A collision-free path from a start point to a goal point
$DSCBFS\_curve$	Dynamic shape control backbone curve in free space
$DSCBOS\_curve$	Dynamic shape control backbone curve in obstacle space
$DSCBETS\_curve$	Dynamic shape control backbone curve in entire task space
δs	A small curve increment added to an existing curve

Table VII . Acronyms utilized in this study.

Acronym	**Full spelling**
P	Position or point
n	Number
l	Length
r	Radius
d	Distance
CLS	Connecting line segment
CSI	Cubic spline interpolation
SSC	Static shape control
DSCB	Dynamic shape control
hrm	Hyper-redundant manipulator
hrm_ssc	Hyper-redundant manipulator constrained by the static shape control (curve)
hrm_dscb	Hyper-redundant manipulator constrained by the dynamic shape control backbone (curve)
RRT	Rapid exploring random tree
RRTSC	Rapid exploring random tree and shape control
CF	Collision-free
DSCBFS	Dynamic shape control backbone (curve) in free space
DSCBOS	Dynamic shape control backbone (curve) in obstacle space
DSCBETS	Dynamic shape control backbone (curve) in entire task space
OB	Obstacle
En	Environment
int	Initial
Con	Connecting
Cal	Calculate
Col	Collision
det	Detection
config	Configuration
Ind	Index
dis	Discrete or discretization
ee	End effector
p_dis_csi	Discrete points of cubic spline interpolation (path)