CONTINUUM ROBOT MANIPULATION

by

Jinglin Li

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2015

Approved by:

____

Dr. Jing Xiao

____

Dr. Srinivas Akella

____

Dr. Min C. Shin

____

Dr. Jianping Fan

____

Dr. James M. Conrad

ABSTRACT

JINGLIN LI. Continuum robot manipulation. (Under the direction of DR. JING XIAO)

Unlike conventional robotic manipulators, continuum manipulators are inspired by invertebrate structures found in nature, such as octopus arms and elephant trunks. The shape of a continuum manipulator can continuously deform via changing the controllable degrees of freedom, such as bending, extending/contracting, and torsional turning of the arm sections. The manipulator is also passively compliant due to their infinite number of passive degrees of freedom. Hence, continuum manipulators are very flexible and particularly suitable for performing tasks in cluttered environments. Although different mechanical designs have been proposed and validated by teleoperations. There is a great need to study autonomous manipulation of a continuum manipulator.

This dissertation addresses research issues in autonomous manipulation using continuum manipulators and introduces novel algorithms, including those for collision detection, whole arm continuum grasping, task constrained manipulation, and continuum manipulation in cluttered, unknown environments. Implementation and testing results are presented to validate the effectiveness of introduced approaches.

ACKNOWLEDGMENTS

I am greatly indebted to my advisor, Prof. Jing Xiao, for her years of patient guidance, advice, constant support, and encouragement. She has given me invaluable insights and suggestions. Some of the results in this dissertation would not be possible without her guidance and feedback. Her genuine caring benefits not only my Ph.D. research and professional progress but also my overall growth.

I would also like to express my thanks to all members of my dissertation committee, Prof. Srinivas Akella, Prof.Min C. Shin, Prof. Jianping Fan and Prof. James M. Conrad. Their time, support, and help are highly appreciated.

I gratefully acknowledge my colleague, Zhou Teng, in the Robotics Lab of UNCC for providing 3D point cloud data for experiments; and collaborators Dr. Apoorva Kapadia and Dr. Ian Walker of Clemeson Universtiy for their help with the OctArm in real experiments. It has been a pleasure to work with these awesome people.

Finally, I want to express my gratitude to my family and friends here and in China, especially my parents, Dehua Li and Hongmin Guo; my uncle and aunt, Shaozhong Deng and Qunhui Guo; and my wife, Wenqian Jin, for their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

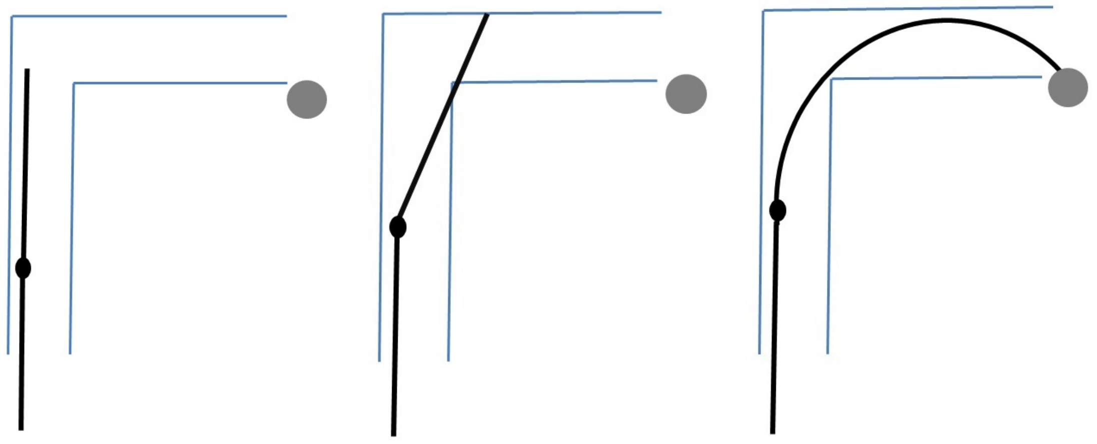
CHAPTER 1: INTRODUCTION

Robotic manipulation has been extensively studied for conventional articulated manipulators, initially due to the need of robotic applications in performing repetitive and labor-intensive tasks, for example, welding and painting in manufacturing environments. There is a greater demand for robotic applications in environments hazardous for humans, such as nuclear power plants, undersea and underground environments, and search and rescue scenes after natural disasters, such as earthquakes.

However, robotic manipulation in such hazardous environments is challenging because: 1) those environments are usually unknown or only partially known, and thus manipulation often needs to rely on sensory information, which involves uncertainty and noise; 2) those environments are usually cluttered and unstructured with objects (obstacles) of unknown types in arbitrary placements/poses. An articulated manipulator may not be flexible enough to perform certain tasks in such an environment, see Fig. 1 (a), (b) for an example.

Inspired by invertebrate structures found in nature, such as elephant trunks and octopus arms, continuum manipulators have been designed to provide more flexibility for manipulation. Those manipulators do not have rigid links, but consist of deformable segments, which give them (theoretically) infinite degrees of freedom to be compliant and robust to uncertainty while meeting the flexibility requirements of maneuvering in a cluttered space. – See Fig. 1 (c).

Although many prototypes of continuum manipulators have been designed and teleop-



(a) A manipulation task requires an arm to bend around the corner to reach the gray circular region.

(b) An articulated manipulator gets stuck at the corner of the tunnel.

(c) A continuum manipulator passes the tunnel and reaches the target region by bending its last section.

Figure 1: Illustration of a manipulation task in a narrow tunnel environment, which is better performed by a continuum manipulator.

erations have been experimented, autonomous manipulation for continuum manipulators, which is the main focus of the proposed dissertation research, has been rarely studied.

In the rest of this chapter, some background on robotic manipulation will be first introduced, including collision-free motion planning, manipulator kinematics, task constrained manipulation, and robotic grasping. Next some related work on manipulation using a continuum manipulator will be surveyed.

1.1 Collision-free Motion Planning

One of the fundamental requirements for robotic tasks is to generate collision-free motion for robots. Given the geometry of a robot and the environment (workspace), a *configuration* of a robot is defined as a complete specification of the location of every point on the robot geometry. The *configuration space* (also called C-space) contains all possible configurations of that robot, and a configuration of a robot is represented by a point in its configuration space.

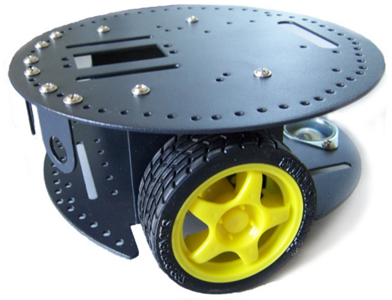
For example, a wheeled mobile robot, see Fig. 2 (a), which can translate on a 2D floor and change its orientation, has three degrees of freedom and therefore a three dimensional C-space. For a serial-chain manipulator, the position and orientation of its arm links are fully controlled by its joint motors, therefore the joint values can be used to represent its configuration. The dimensionality of the manipulator's configuration space depends on the number of joints that can be independently positioned. For example, a PUMA manipulator with six joints, see Fig. 2 (b), has a six-dimensional C-space.

Planning a collision-free path for a robot refers to finding a collision-free curve in the configuration space from an initial configuration to a goal configuration. To plan motion that is executable by a robot, certain physical constraints of the robot need to be satisfied, which limit the velocities and accelerations along the path. Therefore, we also need to *plan a trajectory* as the time profile of a path, by specifying the velocities and accelerations along the path. Once a trajectory is generated, it can be executed by the robot control system to perform the actual motion.

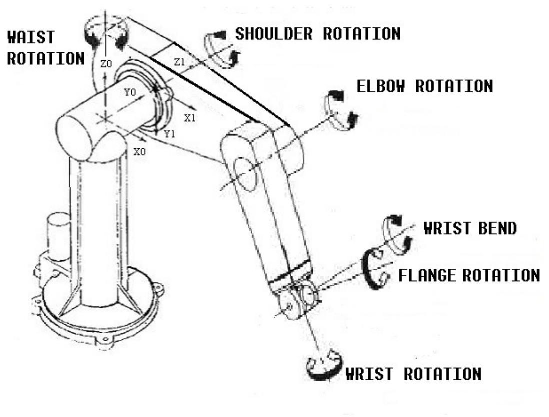
1.1.1 Motion Planning Algorithms

There are many different motion planning algorithms, which can be generally classified [45] as either *deterministic* or *non-deterministic* algorithms.

A deterministic motion planning algorithm always generates the same path, given the same input robot initial configuration and the environment, whereas a non-deterministic algorithm introduces some randomness, such that the output is not completely depending on the input. Deterministic planning algorithms include roadmap, cell decomposition and



(a) A 2 wheeled mobile robot with 3 degrees of freedom from MakerShed



(b) A PUMA manipulator consisting of 6 rotational joints

Figure 2: Examples of a mobile robot and a manipulator.

potential field methods. In general, all these methods share the same idea of building some structures deterministically that characterizes a robot's configuration space as the *free space*, defined as the set of configurations where the robot does not collide with obstacles, and the *C-space obstacles*, defined as the set of configurations where the robot collides with obstacles. Next, a collision-free path is found in the free space.

Roadmap methods try to build a roadmap that connects collision-free configurations in the C-space, so that planning a collision-free path becomes searching a path in the roadmap. Different roadmaps can be built depending on the the nature of the problems, common types are visibility graphs [20] and voronoi graphs [6]. – See Fig. 3 and Fig. 4 for illustrations. Once a roadmap is built, one can apply different graph search methods, e.g. A [31], Dijkstra's algorithm [23], Depth First Search (DFS) or Breadth First Search(BFS) methods, to find a desired path connecting an initial configuration to a goal configuration.

A cell decomposition method [18] decomposes the free space of the configuration space

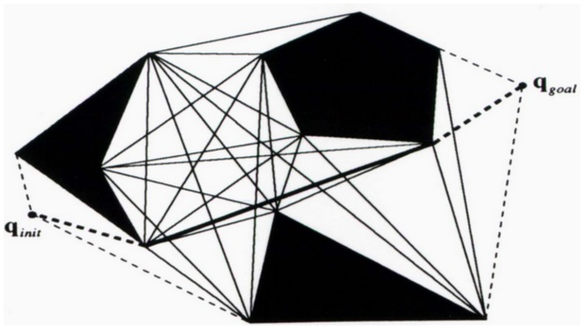


Figure 3: Finding a path in a 2D visibility graph connecting an initial configuration q_{init} to a goal configuration q_{goal} [45].

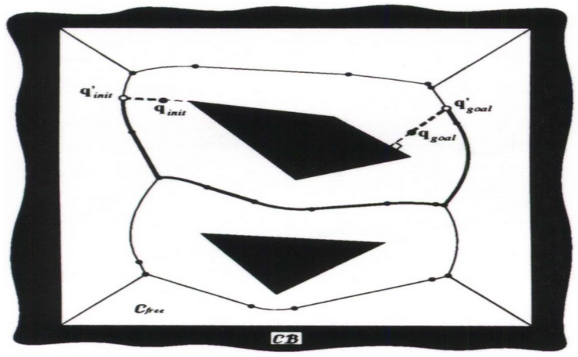


Figure 4: Finding a path in a 2D voronoi graph connecting an initial configuration q_{init} to a goal configuration q_{goal} [45].

into cells, see Fig 5. Then by connecting the free space cells, one can also build a graph connecting adjacent cells, and a collision-free path can be generated by searching the graph.

Potential field approach [38] is proposed to build an artificial potential field in the C-space that combines attraction to the goal, and repulsion from obstacles, see Fig 6. Then robot can simply follow a gradient descent motion to the goal and move away from the obstacles. However, this method can suffer from local minimal, where the robot may get stuck, and cannot reach its goal configuration. Strategies such as backtracking or random walk, can be utilized to help a robot get out of a local minimum.

Deterministic motion planning algorithms are usually used to deal with robots with low

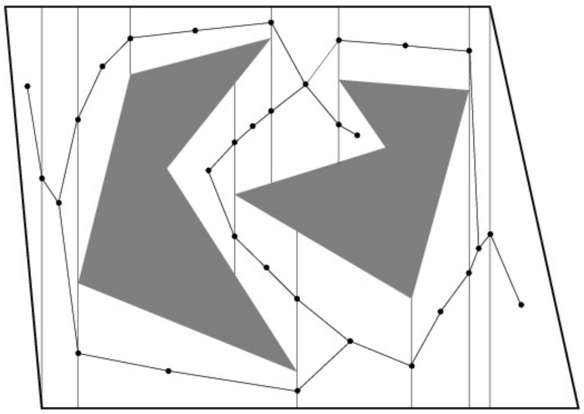
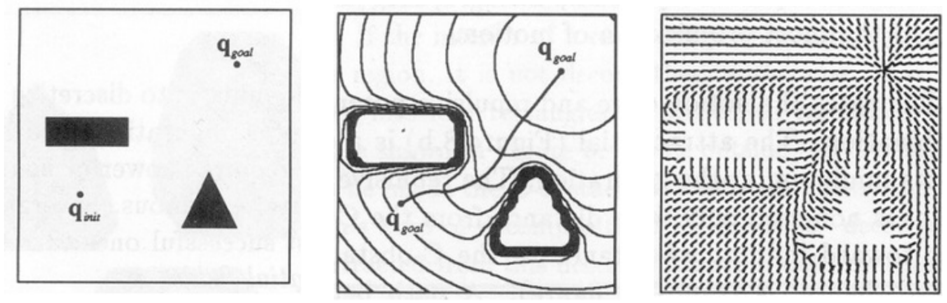


Figure 5: A 2D configuration space decomposed as cells of free space [45].



(a) A 2D configuration space with an initial configuration q_{init} and a goal configuration q_{goal}

(b) A path is found by following a gradient descent motion from q_{init} to q_{goal}

(c) An illustration of the corresponding artificial potential field

Figure 6: An example potential field built in a 2D configuration space [45].

degrees of freedom, e.g, a mobile robot with 3 degrees of freedom. However, for a high degrees of freedom robot, for example, a 6 degrees of freedom manipulator (Fig. 2 (b)), deterministically building a structure characterizing the C-space is very difficult, since it involves explicitly determining the C-obstacles in such a high dimensional C-space is too computationally expensive to be feasible.

Therefore, instead of explicitly computing the free space and the C-obstacle regions in a configuration space, a *non-deterministic* motion planning algorithm, usually generates a collision-free configuration by random sampling and collision test between the robot and obstacles in the workspace. Then it uses a local planner to build collision-free connections between the newly generated configurations to those previously found. Most sampling based algorithms are variations of two categories: Probabilistic Road Map (PRM) [37] and Rapidly exploring Random Tree (RRT)[47].

By random sampling in the C-space, PRM generates a randomized roadmap to characterize free space of the C-space, see Fig. 7. Once a roadmap is generated, then a path can be queried by searching a path connecting the initial configuration and the goal configuration. Variants of PRM are proposed to improve its performance by introducing efficient sampling techniques [4, 103] and dynamic path query schemes [13, 67, 81]. PRM assumes a known, static environment, however, for a dynamic environment involving moving obstacles, it is not efficient, since it needs to rebuild the roadmap from scratch each time when the positions or orientations of obstacles change.

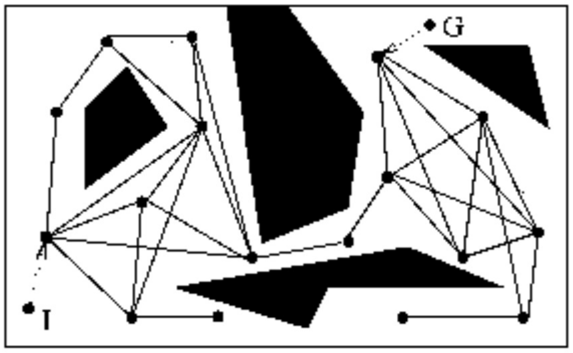


Figure 7: An illustration of a randomized roadmap built in a 2D configuration space [37].

Instead of building a roadmap, RRT expands trees gradually in C-space from both the goal configuration and the initial configuration and tries to connect both trees, see Fig. 8. Once a connection between two configuration trees is found, a path from the initial configuration to the goal configuration is obtained. Unlike PRM that has separated two phases, i.e., buiding a roadmap and then querying a path, RRT finds a path and expands the randomized tree structures at the same time, and it uses a state transition function defining the robot physical constraints under which a new configuration can be sampled, hence RRT not only generates a collision-free path, but also an executable trajectory for a robot. A PRM algorithm, on the other hand, may require connecting tens of thousands of configurations to find an executable solution.

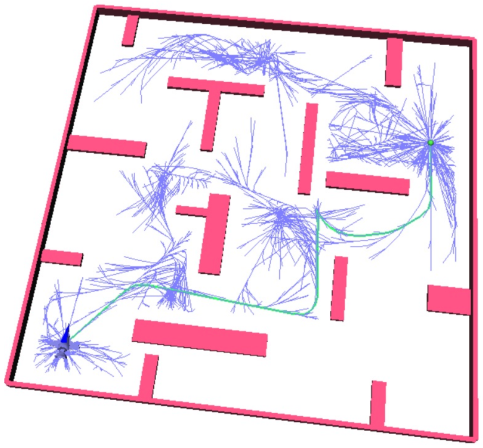


Figure 8: An illustration of randomly expanded trees, the green curve indicates a path found [47].

To deal with a dynamic environment with unforeseen obstacle motion, an algorithm, called Real-time Adaptive Motion Planning (RAMP), is proposed [96]. This algorithm is based on evolutionary computation. By mimicking the genetic modifications, i.e, exchange, mutation, cross-over, it can perform global optimization of robot motion, where a whole path or trajectory can be optimized according to a customized fitness function. Also it preserves a diverse range of paths/trajectories all the time in a “population”, to allow instant, and if necessary, drastic adjustment of robot motion to adapt to newly sensed changes in a dynamic, unforeseen environment.

1.1.2 Collision Detection

Sampling-based motion planners require checking if a sampled robot configuration is in free space or not, which means checking if the robot, placed at the considered configuration, will intersect (i.e., collide with) some obstacle in the physical (Cartesian) space, or not. Algorithms checking such intersections between the robot and obstacles are called *collision detection algorithms*. Efficient collision detection algorithms are essential to motion planning in high-dimensional configuration space. In order to characterize a large range of shapes and sizes of the objects and robots in an unified way, polygonal meshes are often used to model objects and robots in the environments. By doing that, checking the collision between an object and a robot becomes the problem of checking the intersection between polygonal meshes. A hierarchy of polygonal meshes bounding boxes/volumes are usually used to speed up the intersection checking between two meshes [94, 19, 29].

There are also algorithms focusing on collision detection among moving objects. [79] formulates the trajectories of a robot and obstacles and conducts collision checking among them at each time instant analytically. However, it is computational expensive if a trajectory is nonlinear. [78] checks the total traveled distance between two objects and if their minimum distance before moving is greater than the total traveled distance, then there is no collision. Some approaches [17, 27] approximate the sweeping volume of the robot, and another approach [7] grows the robot's volume along a path; the generated sweeping or grown volume is tested for collision against obstacles to achieve continuous collision checking. Continuous collision checking in unknown environments is addressed in [97].

For deformable object models, bounding volume hierarchies (BVHs) of simple bounding volumes, such as spheres or Axis Aligned Bounding Boxes (AABBs), are typically used; unlike rigid body models, deformable models need update (or even rebuild) its BVHs at each time step. Refitting algorithms[44, 46, 109, 94] for BVHs are introduced, and culling algorithms for BVHs are also presented in [98] and [83] using tight bounds of surface normals. With those algorithms, BVHs of objects can be refitted and queried efficiently.

More recently, some researchers have studied collision detection between objects represented as point clouds [39, 69]. Point clouds can be directly obtained from 3D object sensing, e.g., via stereo vision, laser range finders, and Microsoft Kinect. A hierarchy of spheres are introduced to approximate object surfaces [39], which also have bounding boxes to speed up collision detection. Collision detection is considered as a classification problem [69], where points of each object forme a class, and the collision probability of two objects are determined by the separability of two point classes.

1.2 Robotic Manipulation

A robotic manipulation task often requires the manipulator to plan and execute motions to meet a certain task goal under the constraints of the physical environment. During a manipulation task, the manipulator establishes contacts with the environment. Through the contact points the robots can apply forces and moments to objects. A robot manipulator usually uses its *end-effector*, where a tool is attached on, to contact with the environment. Then the tasks require the end-effector to satisfy certain *task constraints*, such as following a specified path/trajectory in work space, maintaining a fixed position and orientation, applying a certain force or moment to the objects, etc.

1.2.1 Manipulator Kinematics

For a serial-chain manipulator, the position and orientation of each arm link is represented by a coordinate *frame*, i.e., a homogeneous transformation matrix, attached at each link with respect to the same reference frame, usually the robot base. The frame attached at the manipulator's end-effector is also called the *tool frame*, where a task tool is usually attached on. A homogeneous transformation matrix ${}^{0}\mathbf{T}_i$ is often used to represent the position and orientation of ith link frame with respect to robot base, and ${}^{j}\mathbf{T}_i$ is used to represent relative position and orientation of ith link with respect to the jth link.

Forward kinematics for a serial-chain manipulator (See Fig. 9 for an example) is to find the end-effector position and orientation relative to the base coordinate system of the manipulator given the joint configurations of the arm and the geometric parameters of arm links. The transformation can be obtained by simply concatenating transformations between frames attached at adjacent links of the chain.

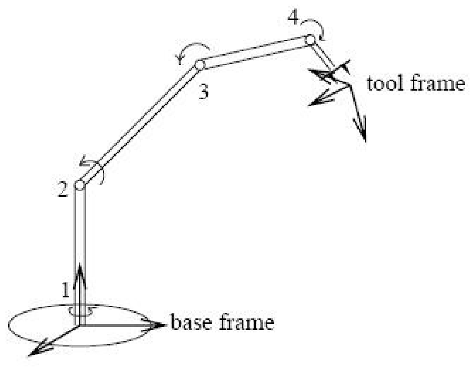


Figure 9: An example of a four-joint serial-chain manipulator.

Inverse kinematics, on the other hand, is to find the joint configurations given the position and orientation of the end-effector and the geometric parameters of the arm links.

From the kinematic analysis, a kinematic transformation between joint space and end-effector work space can be represented as:

$$\mathbf{x} = \mathbf{f}(\mathbf{q}), \tag{1}$$

where $\mathbf{x}$ is the position and orientation of the end-effector, $\mathbf{q}$ is an arm joint configuration and $\mathbf{f}$ represents the transformation function bewtween the joint configurations and the end-effector positions and orientations.

To study the impending motion of an end-effector at a specific point in time, *instantaneous kinematic* analysis follows directly from the kinematic analysis by differentiation with respect to time,

$$\mathbf{v} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}, \tag{2}$$

where $\mathbf{v}$ is a 6×1 vector representing the spatial velocity (linear and angular) of the end-effector, $\dot{\mathbf{q}}$ is a vector composed of all arm joint rates, and $\mathbf{J}(\mathbf{q})$ is a Jacobian matrix, called a *manipulator Jacobian*, parameterized by the joint configuration $\mathbf{q}$.

Each column of $\mathbf{J}(\mathbf{q})$ is associated with one joint rate $\dot{q}_i$ in $\dot{\mathbf{q}}$, indicating the contribution of $\dot{q}_i$ to the spatial velocity of the end-effector at that moment. The inverse of $\mathbf{J}(\mathbf{q})$ can be used to compute the joint rates given the end-effector's linear and angular velocities, which called *inverse instantaneous kinematics*. For a manipulator with n-DOFs, if $n > 6$, $\mathbf{J}(\mathbf{q})$ will not have an inverse and the manipulator is redundant. In that case, pseudo-inverse of $\mathbf{J}(\mathbf{q})$ can be used to compute joint rates,

$$\dot{\mathbf{q}} = \mathbf{J}^{+}(\mathbf{q})\mathbf{v}, \tag{3}$$

The pseudo-inverse $\mathbf{J}^{+}(\mathbf{q})$ can be written as:

$$\mathbf{J}^{+}(\mathbf{q}) = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{T} \tag{4}$$

where matrices $\mathbf{V}$, $\boldsymbol{\Sigma}$ and $\mathbf{U}^{T}$ can be obtained through singular-value decomposition (SVD) of $\mathbf{J}(\mathbf{q})$ [65],

$$\mathbf{J}(\mathbf{q}) = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{T} \tag{5}$$

and $\boldsymbol{\Sigma}^{+}$ is the transpose of $\boldsymbol{\Sigma}$ with all the non-zero values on its diagonal reciprocated.

1.2.2 Task Constrained Manipulation

Existing techniques for task-constrained manipulation include control schemes based on instantaneous inverse kinematics and sampling-based methods. Methods based on instantaneous inverse kinematics [107, 14, 80, 60] use (pseudo) inverse of the arm Jacobian to satisfy the task constraint. To satisfy additional constraints, such as environment constraints, they compute the related joint motion by gradient descent methods, for example, using a gradient descent method to compute the joint motion that increase the arm-to-obstacle distance to avoid obstacles. Then they project the computed joint motion onto the null space of the arm Jacobian to ensure the primary task constraint is not violated. Such methods can compute solutions efficiently but require prioritizing the constraints during a task, by which one has to compromise one constraint to satisfy the other. However in certain circumstances, it is difficult to decide which constraints is more important than the other, e.g., satisfying task constraints vs avoiding obstacles.

Instead of prioritizing different constraints, sampling-based methods [108, 82, 10] are proposed to generate arm configurations on the task constrained manifold in the configuration space and then check if the generated configurations also satisfy the environment constraints. Several techniques have been proposed: randomized gradient decent [106] is a rejection-based sampling method and thus can be computationally expensive; tangent space sampling and first-order projection techniques [82, 10] also utilize the pseudo-inverse of the arm Jacobian to efficiently generate samples on (or close to) the constrained manifold.

1.2.3 Robotic Grasping

Grasping using a robot hand/gripper has been extensively studied for decades. Various kinds of robot hands have been designed for grasping tasks, such as anthropomorphic hands Utah-MIT hand [33], DLR hand[15] and Robonaut hand[5] and non-anthropomorphic hands Barrett hand [2] and Salisbury hand [71].

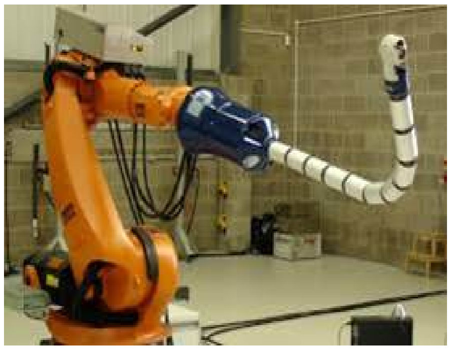
The most fundamental requirements in grasping are the abilities to hold an object in equilibrium and control the position and orientation of the grasped object relative to the palm of the hand. The *closure* [11] properties are often used to evaluate a certain grasp, where a stable grasp may require a force/form closure [26, 68, 89, 42] to be able to resist the disturbance of external forces/torques in any direction.

To plan a grasping task using a robot hand, people encountered an even larger configuration space comparing to a manipulator, since planning a grasp of a robot hand requires planning a configuration consisting of a wrist pose (6 degrees of freedom) and finger poses, which are often more than 6 degrees of freedom for fingers of dexterous hands. “Pre-grasps” are usually generated so that a stable grasp can be achieved by simply closing the gripper/fingers. [64] can generate pre-grasps by approximating complex target object shape with simple shape primitives, e.g, spheres, cylinders, cones and boxes. Learning is often used to obtain pre-grasps (e.g., [72, 76, 8, 22, 12]), and some pre-grasps are inspired by caging schemes [75]. Methods for grasping deformable objects has been proposed in [34], and grasping unknown or partially known objects has been studied in [40, 77, 70] by utilizing vision sensor to fit object geometric and textural features.

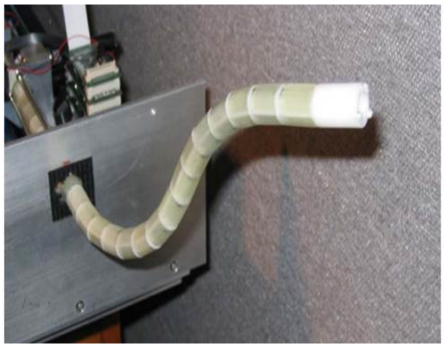
1.3 Related Work on Continuum Manipulators

Comparing to a conventional articulated manipulator, a continuum manipulator does not consist of rigid links but consists of deformable structures, so that it is compliant when in contact with the environment, which makes the manipulator adaptive to complicated environments where compliant motion is required. However, the compliance of a continuum manipulator results in a problem of controlling a robot of (theoretically infinite) high degrees of freedom with discrete set of variables. [30] studied how an octopus controls its arm movement in order to identify a general control principle of a continuum robot. Almost all continuum robots feature constant-curvature sections when intact.

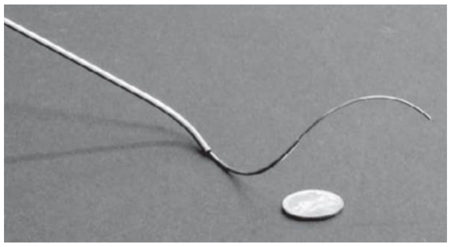
Studies on different prototypes of continuum manipulators have been conducted [32, 74, 90, 101, 100]. Building a continuum manipulator requires designing proper actuators to generate motions of a continuum manipulator, that are compliant but also capable of producing high speeds and large forces. Different designs of actuators can be used to categorize different continuum manipulators: highly articulated, hyper-redundant snake arms [1, 21], continuous pneumatic driven or tendon driven arms [99, 36, 16, 105] inspired by the biomechanical systems, and small-scale continuum manipulators with concentric tube designs [102, 24, 88]. Examples of those manipulators are shown in Fig. 10. Teleoperations of those prototypes have been experimented to validate their arm models and designs.



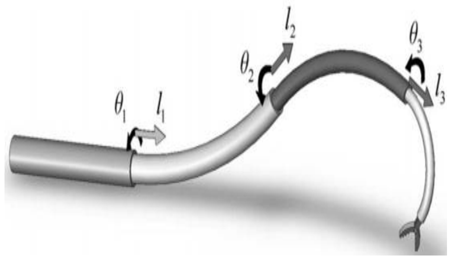
(a) OC Robotics snake arm [1]



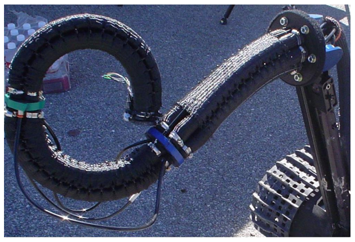
(b) Highly articulated snake arm by Degani and Choset [21]



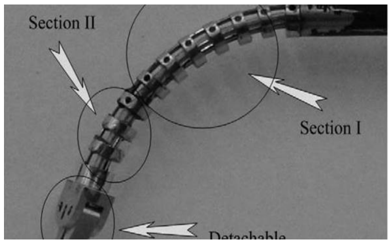
(c) A concentric tube robot of 3 tubes designed by Webster et al. [102]



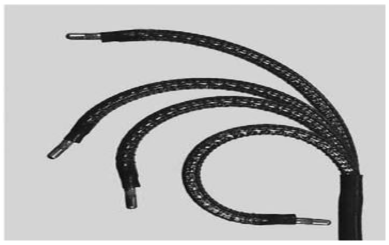
(d) The design of a concentric tube robot of 5 tubes by Dupont et al. [24]



(e) OctArm developed by Jones and Walker [99]



(f) A tendon-driven continuum arm by Xu and Simaan [105]



(g) A tendon/wire-driven continuum manipulator by Camarillo et al. [16]

Figure 10: Current prototypes of a continuum manipulator.

Existing motion planning algorithms for a continuum manipulator are mainly introduced for small-scale, surgical-use continuum manipulators [102, 24, 88]. A sampling based motion planning algorithm for a concentric tube manipulator is proposed [91, 86, 87] to guide the robot's tip to a target point in the environment; [73] proposed a method of detecting a surgical continuum manipulator by using 3D ultrasound images during a medical surgery; [92] proposed an approach to locate and register the configuration and the end-effector frame of a surgical continuum manipulator with respect to object frame by using contact sensor data and environmental stiffness data; [41] studied different structures of arm segments and found that an arm segment that is able to change its length has the best tip dexterity in surgical applications. However, all those algorithms mainly focus on the arm tip manipulation of a continuum manipulator in a surgical application, the whole arm manipulation of a continuum manipulator, as a key feature of a continuum manipulator comparing to an articulated manipulator, is not well-studied.

Only recently, 2D whole arm manipulation for a continuum manipulator are proposed [104, 62]. However, autonomous whole arm manipulation of a general n-section continuum manipulator in a 3D, especially a cluttered environment, has been rarely studied, and this is the main focus of this dissertation.

In the following chapters, the overall objectives of this dissertation will be introduced, and then novel approaches for autonomous continuum manipulation in a 3D, cluttered environment will be presented.

CHAPTER 2: RESEARCH OVERVIEW

As introduced in Chapter 1, a continuum manipulator is more flexible and compliant than a conventional articulated manipulator and thus can be more useful in performing tasks in less structured and cluttered environments. However, autonomous manipulation using continuum manipulators is not well-studied, especially for manipulation tasks in cluttered, occluded environments. In addition, most autonomous manipulation methods for conventional manipulators can't be directly applied to continuum manipulation tasks, because of the different structures of continuum manipulators, especially the concave and deformable shapes of the arm segments.

This dissertation is focused on autonomous manipulation algorithms for continuum manipulators to perform different manipulation tasks in cluttered environments. The following approaches will be introduced in particular:

- An efficient, real-time collision detection method involving multi-segment continuum manipulators and 3D polygonal mesh objects [57].
- Whole arm grasping approaches of a target 3D object for a continuum manipulator in both an open environment [52, 55] and a cluttered environment [53, 51].
- Generalized task constrained continuum manipulation approaches which consider tasks constraining the arm tip position and/or orientation in cluttered environments [56, 58].

- An approach for continuum manipulation in cluttered and unknown environments, where the information of the environment is obtained through progressive sensing [50].

Note that all approaches introduced in this dissertation only use geometric models of the manipulator and objects. In the following chapters, the manipulator model of a continuum manipulator and the object models in the environment will be introduced, and then the above approaches for autonomous continuum manipulation will be presented.

CHAPTER 3: MANIPULATOR AND OBJECT MODEL

3.1 Manipulator Model

The kinematic model of a continuum manipulator used in this dissertation is as proposed in [99]. For an n-section continuum manipulator, the i-th section is denoted as sec_i, in terms of its central circular axis seg_i with two end points: a *base* point p_{i-1} and a *tip* point p_i, and the radius of the (curved) cylinder w_i. If seg_i has a non-zero curvature, we call the circle of seg_i the ***section*** ***i's circle***, denoted by cir_i, with radius r_i, and the plane that contains seg_i the ***section plane***, denoted by P_i, as shown in Fig. 11.

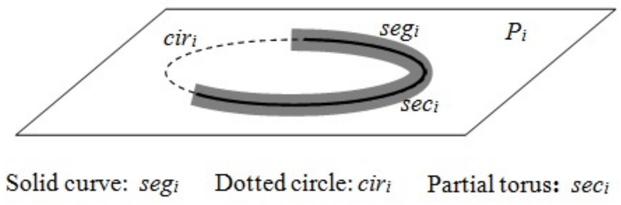


Figure 11: An arm section sec_i, its central axis seg_i on the circle cir_i and the plane P_i.

The base frame of the robot is set at p_0 with z_0 axis tangent to section 1's circle. The section i's frame is formed at p_{i-1} with the z_i axis tangent to the section circle at p_{i-1}. The base of section i is the tip of section i-1. Two adjacent sections i-1 and i are connected *tangentially* at the connection point p_{i-1}, i.e., the two sections share the same tangent at p_{i-1}.

Figure 12 illustrates the three sections of the OctArm and their respective frames. Because of the mechanical structure, section i of the OctArm can bend either along the $+z_i$ axis or the $-z_i$ axis but *not* both.

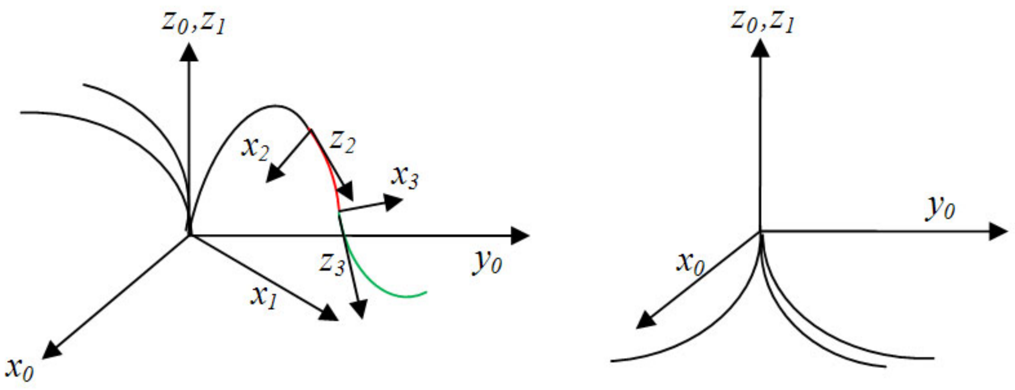


Figure 12: OctArm manipulator frames. Note that section 1 can bend along either (a) $+z_1$ axis (with different orientations) or (b) $-z_1$ axis (with different orientations), but *not* both directions of z_1.

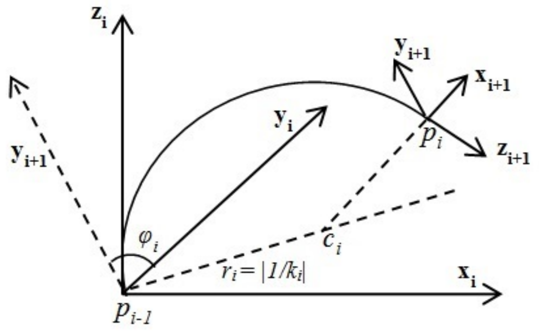


Figure 13: Section i's frame.

As shown in Fig. 13, the circle center of sec_i, c_i, always lies on the x_i axis, with $\mp 1/\kappa_i$ being the x coordinate in the i-th frame, where κ_i is the curvature. Note also that c_i lies on the positive x_i axis if $\kappa_i < 0$ and on the negative x_i axis if $\kappa_i > 0$. When $\kappa_i = 0$, section i is a straight-line segment starting from the origin p_{i-1} and along the z_i axis.

Although each section can bend passively anywhere, it has a finite number of degrees of freedom that can be directly changed by the OctArm actuators [99], which are variables: curvature κ_i, length s_i, and orientation angle ϕ_i from the plane of section i-1 to that of section i about z_i axis, i.e., the angle from y_{i-1} axis to y_i axis about z_i axis.

Through out this dissertation, the following notations will be used to describe arm sections, section frames, and section configuration variables:

n:	the number of arm sections of a continuum manipulator.
sec_i:	the arm section i, $1 \leq i \leq n$.
cir_i:	the section circle of sec_i, where the central axis of sec_i lies.
c_i:	the center of cir_i.
p_i:	the tip point of sec_i, which is also the base point ofsec_{i+1} and the origin of sec_{i+1}'s base frame.
$\mathbf{T}_i$:	the homogeneous transformation matrix of the base frame of sec_i with respect to a fixed world coordinate system; $\mathbf{T}_i$ also represents the tip frame location of sec_{i-1} (for $i > 1$).
$\mathbf{R}_i$:	the rotation matrix in $\mathbf{T}_i$ consisting of $\mathbf{x}_i$, $\mathbf{y}_i$ and $\mathbf{z}_i$ of sec_i's base frame.
$\mathbf{p}_{i-1}$:	the position vector in $\mathbf{T}_i$ for the base point p_{i-1} of sec_i.
κ_i:	the curvature of sec_i.
ϕ_i:	the rotation angle from $\mathbf{y}_i$ axis to $\mathbf{y}_{i+1}$ axis.
w_i:	width of arm section sec_i.
C :	an n-section continuum arm configuration represented as $C = \{(s_1, \kappa_1, \phi_1), ..., (s_n, \kappa_n, \phi_n)\}$.
P_i:	the section plane that contains cir_i, with plane normal parallel to $\mathbf{y}_{i+1}$.

3.2 Object Model

An object is modeled as a polygonal mesh [93]. As a common approach to speed up the proximity check, the object meshes are organized by a Bounding Volume Hierarchy (BVH) (see Fig. 14), such as an Axis-Aligned Bounding Box (AABB) tree [94] or an Oriented Bounding Box (OBB) tree [29].

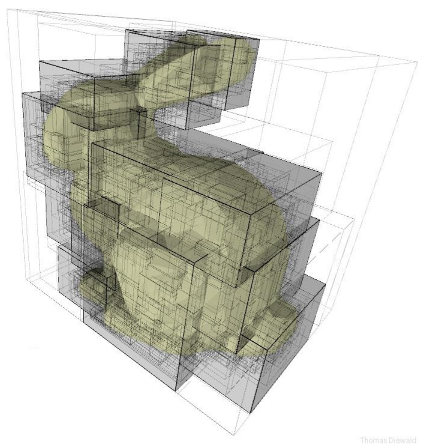
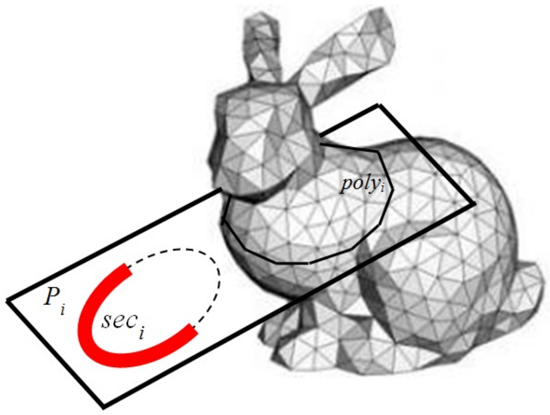
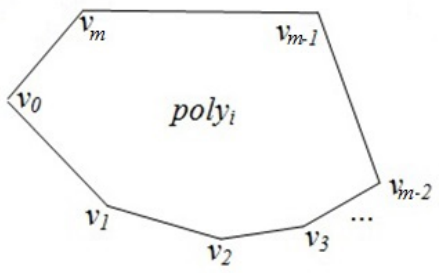


Figure 14: A hierarchy of bounding volumes for a bunny mesh.



(a) Section plane P_i intersects a bunny mesh object



(b) A cross section polygon $poly_i$ intersected by section plane P_i

Figure 15: A cross section polygon $poly_i$ of the object intersected by section plane P_i.

The cross section of the object mesh by an arm section's plane is denoted as $poly$ since it is a polygon consisting of a set of vertices $\{v_0, v_1, ...v_m\}$, see Fig. 15 for illustration. If the cross section consists of more than one connected component, $poly$ denotes the convex hull of the cross section.

3.3 Contact Determination Between Objects and the Manipulator

In order to know how close the manipulator arm is to a target object while trying to grasp it and whether a contact occurs. Starting from a configuration away from the object, once the manipulator is moved closer to the object such that the minimum distance between them is less than a small distance $d_{contact} > 0$, we call a contact has occurred between the pair of closest points on the arm and the object respectively. This threshold $d_{contact}$ actually reflects the fact that the continuum manipulator is compliant and can deform slightly under contact: we can imagine shrinking the radius of each arm section by $d_{contact}$, such that when the minimum distance between the shrunk arm and the object is less than $d_{contact}$, the arm is in fact contacting the object with some compliance.

The collision detection algorithm proposed in Chapter 4 will help us compute the minimum distance between the object and manipulator, and thus compute a list of contacts between the arm sections and the target objects.

CHAPTER 4: REAL-TIME COLLISION DETECTION INVOLVING A CONTINUUM MANIPULATOR

4.1 Overview

Collision detection between the robot and objects in the environment is essential in robot motion planning. Conventional collision detection algorithms often use a hierarchy of simple bounding volumes for the objects to speed up computation. At the lowest level of the hierarchy, collision checking is conducted between either convex parts or polygons (for generic objects approximated by polygonal meshes). Such an algorithm works well for collision detection using a conventional articulated manipulator, which consists of rigid links, and whose structures do not change or deform. However, the arm section of continuum manipulators are not rigid and may deform continuously, so that conventional collision detection algorithms using hieratical mesh models are not suitable for a continuum manipulator.

Approximating a continuum robot at a certain configuration requires a very fine polygonal mesh to be reasonably accurate, which decreases the efficiency for collision checking if a large number of polygons are in the mesh. Moreover, each time a continuum robot changes its configuration, because its whole shape deforms, the fine mesh has to be updated, which is time consuming due to mapping from the high dimensional configuration space to the Cartesian space. On the other hand, storing different meshes of different configurations beforehand for the purpose of path/trajectory planning can take too much space to be feasible.

Therefore we have proposed an efficient and novel algorithm [57] for Collision Detection between a Continuum Manipulator (CD-CoM) and environmental objects based on analytical intersection checking. The CD-CoM algorithm applies to the exact parametric model of any continuum manipulator featuring multiple sections, where each section has an uniform curvature. It also applies to any section of non-uniform curvature approximated by different uniform-curvature segments or toroidal primitives. As such, the algorithm saves significant time and space in collision checking of a path for a continuum manipulator.

4.2 Bounding Planes of Manipulator Sections

The section i, $i = 1, 2, ...$, of a continuum manipulator is bounded by two planes H_{i-1} and H_i, as shown in Fig. 16. H_i is the plane containing p_i with normal z_{i+1} and shared by section i and section $i + 1$.

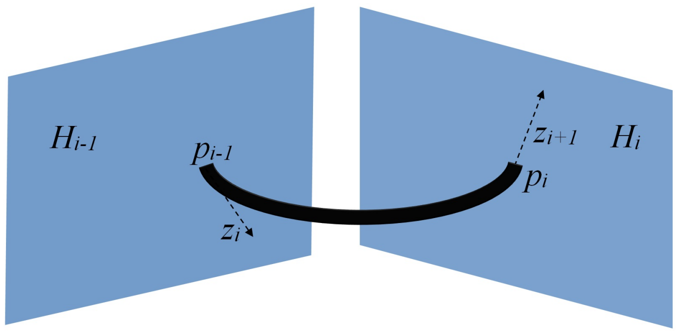


Figure 16: Two bounding planes for section i.

4.3 Manipulator Cross Sections

For each section i of the manipulator with a non-zero curvature, let plane P_i contain the section i's circle, i.e. cir_i, then the cross section of section i by P_i, denoted by cs_i, is a fan-shaped planar region with a width $2w_i$, bounded by two rays $L_{i,1}$ and $L_{i,2}$. As shown in Fig. 17, using a polar coordinate system (ρ, θ) with circle center c_i as the pole and x_i as the polar axis, the region cs_i can be described easily by bounds on ρ and θ as:

$$r_i - w_i \leq \rho \leq r_i + w_i \tag{6}$$

$$\theta_i^{min} \leq \theta \leq \theta_i^{max} \tag{7}$$

where, if $\kappa_i > 0$, then $\theta_{min} = 0$ and $\theta_{max} = s_i\kappa_i$; else, $\theta_{min} = \pi + s_i\kappa_i, \theta_{max} = \pi$.

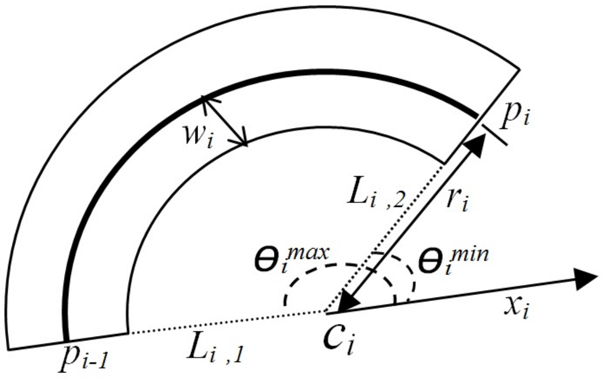


Figure 17: The cross section cs_i of section i and its polar coordinate system.

4.4 CD-CoM Algorithm

The CD-CoM algorithm (Algorithm 1) recursively traverses a hierarchy T of bounding boxes of an object mesh and checks if there is any collision between T and the exact model of a continuum manipulator featuring multiple sections of uniform curvatures at a given configuration. Its subroutine *CD-Face* (Algorithm 2) checks for collision between a section of the manipulator and a polygon f from the object, which can be a face of a bounding box or directly from the polygonal mesh of the object.

The configuration of the arm determines seg_i with end points p_{i-1} and p_i. If κ_i=0, section i is a straight cylinder with a width of $2w_i$ and a length of s_i, and the function

Algorithm 1: CD-CoM(C, T)

```
/* This algorithm checks for collisions between a continuum manipulator and an object
in polygonal mesh */
Input arm configuration $C$ = $(\kappa_1, \phi_1, s_1, ...\kappa_n, \phi_n, s_n)$, a tree of bounding boxes $T$ for
an object mesh
$Collision \leftarrow$ False
/* Check if $sec_i$ intersects the bounding box of $T$ */
for each face $f$ of the root bounding box of $T$ do
    if $\exists sec_i$, $(0<i\leq n)$, such that CD-Face($sec_i$, $f$) = True then
        $Collision \leftarrow$ True
        break
    end if
end for
if $Collision$ = False then
    /* the arm is either completely inside the bounding box or outside the bounding
    box of $T$ */
    if any point of the arm is inside the bounding box of $T$ then
        return $Collision \leftarrow$ True
    else
        return $Collision \leftarrow$ False
    end if
end if
/* Perform the same collision check with $T$'s sub-trees*/
$Collision \leftarrow$ False
if $T.LeftChild \neq NULL$ then
    $Collision \leftarrow$ CD-CoM($C, T.LeftChild$)
end if
if $Collision$ = True then
    return $Collision$
end if
if $T.RightChild \neq NULL$ then
    $Collision \leftarrow$ CD-CoM($C, T.RightChild$)
end if
/* Perform primitive collision check if $T$ is a leaf node */
if $T$ has no child then
    for each face $f$ of the mesh polygons within the leaf bounding box do
        if $\exists sec_i$, such that CD-Face($sec_i$, $f$) = True then
            return $Collision \leftarrow$ True
        end if
    end for
end if
return $Collision$.
```

Algorithm 2: CD-Face(sec_i, f)

```
/* This algorithm checks for collisions between a toroidal arm section and a polygon */
Input $sec_i$'s configuration $(\kappa_i, \phi_i, s_i)^T$, central axis $seg_i$, section width $w_i$, and a face $f$
from an object
$Collision$ ← False
if $\kappa_i = 0$ then
    $Collision$ ← SS-CollisionCheck($sec_i$, $f$)
else
    if $P_i$ intersects $f$ at line segment $l^i$ then
        $Collision$ ← CS-CollisionCheck($cs_i$, $l^i$)
    end if
    if $Collision$ = False then
        $Collision$ ← NCS-CollisionCheck($sec_i$, $f$)
    end if
end if
return $Collision$.
```

Algorithm 3: SS-CollisionCheck(sec_i, f)

```
/* This algorithm checks for collisions between a cylinder arm section and a polygon */
Compute the distance between straight-line segment $seg_i$ and the plane $Q$ that the
object face $f$ is on, denote the distance as $d(seg_i, Q)$ and closest points $p$, $q$ on $seg_i$ and
$Q$ respectively
if $d(seg_i, Q) \leq w_i$ then
    if $q$ is not on $f$ then
        compute the minimum distance between each edge $e_k$ of $f$ and $seg_i$ [25],
        denoted as $d_{min}(seg_i, e_k)$ with closest points $p$ and $q$ on $seg_i$ and $e_k$
        respectively
        if $d_{min}(seg_i, e_k) > w_i$ then
            return $Collision$ ← False
        end if
    end if
    if $p$ is not on $p_{i-1}$ or $p_i$ then
        return $Collision$ ← True
    else
        /* $f$ is closest to an end point of $seg_i$ */
        if $f$ intersects $H_i$ or $H_{i-1}$ at $l_{int}$ and the distance $d(l_{int}, p_i) \leq w_i$ then
            return $Collision$ ← True
        end if
    end if
end if
return $Collision$ ← False.
```

StraightSection-CollisionCheck (SS-CollisionCheck), presented in Algorithm 3, is called for collision checking between a cylinder (section i) and the planar face (f). This algorithm computes the minimum distance $d_{min}(seg_i, f)$ between seg_i (which becomes a line segment) and f. If $d_{min}(seg_i, f) > w_i$, it returns no collision; otherwise, depending on whether f is between the bounding planes H_{i-1} and H_i of seg_i or not and whether or not f intersects either end of section i on H_{i-1} or H_i, the algorithm can return collision or no collision. For example, if f is outside of the bounding plane H_{i-1}, i.e., not on the same side with seg_i, then even if $d_{min}(seg_i, f) < w_i$, there is still no collision between f and seg_i.

If $\kappa_i \neq 0$, seg_i is a circular curve, defining a plane P_i, and the cross section cs_i of section i by P_i is characterized by inequalities (6) and (7). To be efficient, Algorithm 2 first considers the 2D collision cases where the face f intersects P_i by calling Algorithm 4 for *cross section collision check*. If no collision is found between f and the cross section cs_i, it next checks the 3D collision cases where collisions may happen between f and the section i other than cs_i by calling Algorithm 5 for *non-cross section collision check*. Our test results show that it is substantially more efficient to first call Algorithm 4 and call Algorithm 5 only when it is necessary.

4.4.1 Cross Section Collision Check (CS-Collision Check)

For a manipulator section i with non-zero curvature, i.e., $\kappa_i \neq 0$, if a face f from an object intersects the plane P_i at a line segment l^i with vertices v_1 and v_2 [1], Algorithm 4 checks if l^i intersects the fan-shaped cross section cs_i of manipulator section i. Denote the polar coordinates of v_1 and v_2 as (ρ_1, θ_1) and (ρ_2, θ_2) respectively. Algorithm 4 classifies

[1] In the special case that the intersection is a point, $v_1 = v_2$.

Algorithm 4: CS-CollisionCheck(cs_i, l^i)

```
/* This algorithm checks for collisions between a fan-shaped planar region $cs_i$ and a
line segment $l^i$ with vertices $v_1$ and $v_2$, whose polar coordinates are $(\rho_1, \theta_1)$ and $(\rho_2, \theta_2)$
respectively. */
Case 1: if $\rho_1$ and $\rho_2$ are both below the lower bound for $\rho$, i.e., $\max(\rho_1, \rho_2) < r_i - w_i$
then
    return $Collision \leftarrow$ False

Case 2: if either $(\rho_1, \theta_1)$ or $(\rho_2, \theta_2)$ satisfies both (6) and (7) then
    return $Collision \leftarrow$ True

Case 3: if $\rho_1$ and $\rho_2$ are both above the upper bound for $\rho$, i.e., $\min(\rho_1, \rho_2) > r_i + w_i$
and $\theta_1$, $\theta_2$ both satisfy (or both do not satisfy) (7) then
compute the distance between circle center $c_i$ and $l^i$ and obtain point $q = (\rho_q, \theta_q)$ on $l^i$
if the distance is shorter than $r_i + w_i$ and $\theta_q$ satisfies (7)
    return $Collision \leftarrow$True
else
    return $Collision \leftarrow$ False

Case 4: if both $\theta_1$ and $\theta_2$ satisfy (7) then
    return $Collision \leftarrow$ True

/ * the remaining collision cases have intersections between rays $L_{i,1}$ (or $L_{i,2}$) and $l^i$ * /
Case 5: if line segment $l^i$ intersects rays $L_{i,k}$ $(k = 1, 2)$ of $cs_i$ at $p^k_{int}$ (see Fig. 17) then
if one vertex of $l^i$ satisfies (7) and is above the upper bound for $\rho$. then
    if $\rho^k_{int} \leq r_i + w_i$ then
        return $Collision \leftarrow$ True (see Fig. 18(e))
    end if
end if
if one vertex of $l^i$ satisfies (7) and is below the lower bound for $\rho$. then
    if $\rho^k_{int} \geq r_i - w_i$ then
        return $Collision \leftarrow$ True (see Fig. 18(f))
    end if
end if
if neither vertices of $l^i$ satisfies (7) then
    if one of $\rho^k_{int}$ satisfies $\rho^k_{int} \geq r_i - w_i$ then
        return $Collision \leftarrow$ True (see Fig. 18(g))
    end if
end if
return $Collision \leftarrow$ False.
```

all collision scenarios into five cases based on whether v_1 and v_2 satisfy the bounds of inequalities (6) and (7) and then check sequentially from Case 1 to Case 5 to detect all possible kinds of intersections (i.e., collisions). Fig. 18 shows examples for these cases. Appendix further shows that Algorithm 4 covers all cases of possible collisions and is complete.

4.4.2 Non-cross Section Collision Check (NCS-Collision Check)

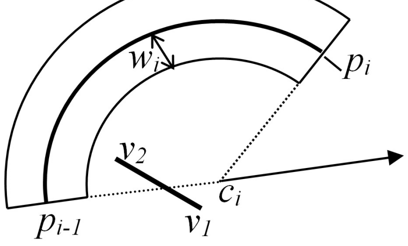
If a face f of the object does not intersect P_i or does not intersect the cross section cs_i, we need to further check if f intersects section i by Algorithm 5.

In Algorithm 5, we first check if the distance between cir_i and Q, the supporting plane of face f, is greater than the width of section i by calling Procedure 1. If so, Q has no intersection with the section i, a truncated torus, and no further collision checking is necessary. Otherwise, further collision checking is done by calling subsequently Procedure 2, for computing the distance between the end points of seg_i to f, and if necessary, also Procedure 3, for computing the distance between seg_i to an edge of f. Procedures 1, 2 and 3 are described below:

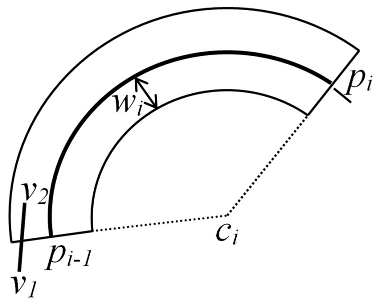
Procedure 1: Compute the minimum distance $d_{min}(cir_i, Q)$ between circle cir_i and plane Q as well as the pair of closest points p on cir_i and q on Q.

If Q intersects cir_i, then $d_{min}(cir_i, Q) = 0$, and return all pairs of intersection points p and q on cir_i and Q respectively.

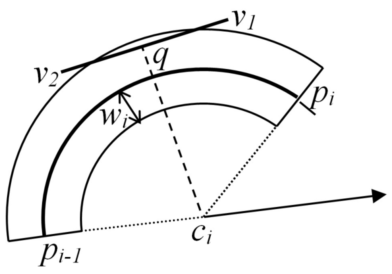
For the case Q does not intersect cir_i, if Q is parallel to plane P that cir_i is on, then the distance from any point p on cir_i to Q is the shortest distance from cir_i to Q. Project point p to Q to obtain point q. Otherwise, compute $d_{min}(cir_i, Q)$ as follows – see Fig. 19 for



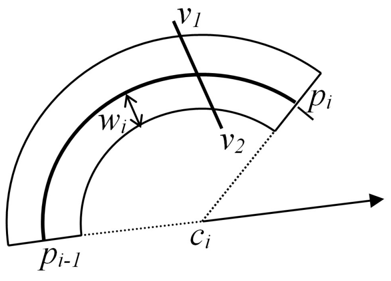
(a) Case 1



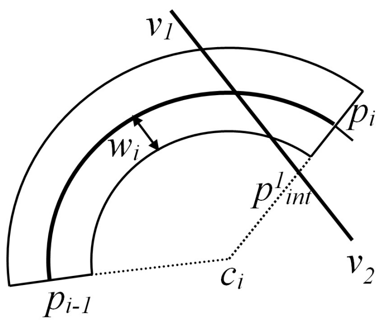
(b) Case 2



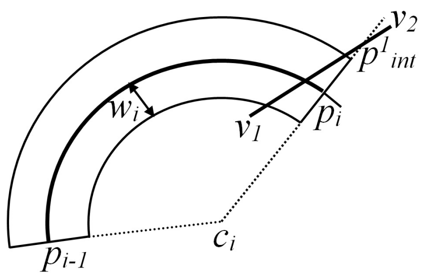
(c) Case 3



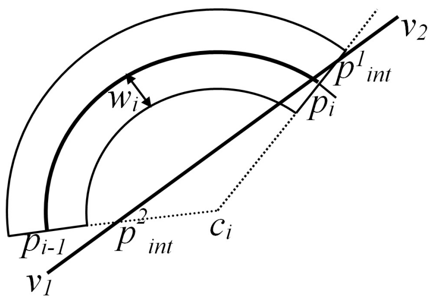
(d) Case 4



(e) Case 5



(f) Case 5



(g) Case 5

Figure 18: Some examples for cases considered in the cross section collision check.

illustration:

- Project c_i to Q and denote the point on Q as q'.

Algorithm 5: NCS-Collision check(sec_i, f)

```
/* This algorithm checks for collisions between a toroidal arm section $sec_i$ and a
polygon $f$ when $f$ does not intersect the section plane $P_i$ */
Compute the minimum distance between $cir_i$ and the plane $Q$ of face $f$ from the object,
i.e., $d_{min}(cir_i, Q)$, and corresponding points $p$, $q$ on $cir_i$ and $Q$ respectively by
Procedure 1
if $d_{min}(cir_i, Q) > w_i$ then
    return $Collision \leftarrow$ False
end if
if $q$ is on $f$ and $p$ is not on $seg_i$ then
    compute the shortest distance from $p_{i-1}$ and $p_i$ to $f$, denoted as $d_{min}(p_{i/i-1}, f)$,
    and update the pair of closest points $p$ and $q$, by Procedure 2
    if $d_{min}(p_{i/i-1}, f) > w_i$ then
        return $Collision \leftarrow$ False
    end if
end if
if $q$ is not on $f$ then
    compute the shortest distance between $seg_i$ and the edges of $f$ by Procedure 3,
    denoted as $d_{min}(seg_i, e_k)$, and update the pair of closest points $p$ and $q$
    if $d_{min}(seg_i, e_k) > w_i$ then
        return $Collision \leftarrow$ False
    end if
end if
if $p$ is not on $p_{i-1}$ or $p_i$ then
    return $Collision \leftarrow$ True
else
    /* $f$ is closest to an end point of $seg_i$ */
    if $f$ intersects $H_i$ or $H_{i-1}$ at $l_{int}$ and the distance $d(l_{int}, p_i) \leq w_i$ then
        return $Collision \leftarrow$ True
    end if
end if
return $Collision \leftarrow$ False.
```

- Project q' to P_i and denote the point on P_i as p'.
- Connect p' and c_i (which are both on P_i) and obtain a line segment that intersects cir_i at point p.
- Project p to Q, denote the projected point as q, and return the distance between p and q.

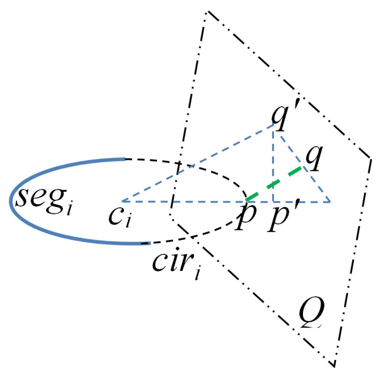


Figure 19: $d_{min}(cir_i, Q)$, indicated by the green dashed line and computed by Procedure 1.

A proof for the correctness of Procedure 1 is presented in Appendix.

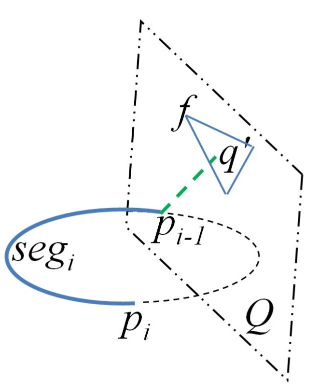
Procedure 2: Compute the minimum distance $d_{min}(p_{i/i-1}, f)$ from points p_i and p_{i-1} to face f, and obtain the pair of closest points.

- Project each point to Q and denote the closer projection (to the original point) as q' and the corresponding distance $d_{min}(p_{i/i-1}, Q)$.
- If q' is on f then return $d_{min}(p_{i/i-1}, Q)$ as $d_{min}(p_{i/i-1}, f)$, see Fig. 20(a).
- Else, find the minimum distance between each point to every edge e_k of f and return the shortest distance as $d_{min}(p_{i/i-1}, f)$, see Fig. 20(b).

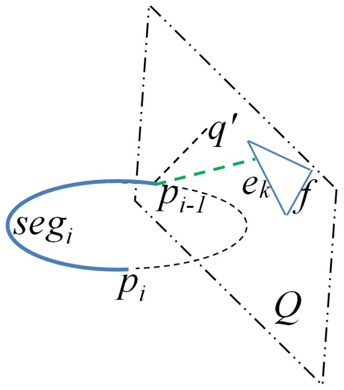
Procedure 3: Compute the minimum distance $d_{min}(seg_i, e_k)$ between seg_i and an edge e_k, and obtain the pair of closest points.

1) Compute the minimum distance between e_k and cir_i and obtain the corresponding closest points q on e_k and p on cir_i as follows.

We first establish a coordinate system: x is along the projection of e_k on P_i, z is along the normal of P_i towards e_k starting from one vertex of the projection, and y is orthogonal to x on P_i such that xyz form a right-handed system, see Fig. 21.



(a) If q' is on f, then $d_{min}(p_{i/i-1}, f)$ equals to the minimum distance from $p_{i/i-1}$ to q'.



(b) If q' is not on f, then $d_{min}(p_{i/i-1}, f)$ equals to the minimum distance from $p_{i/i-1}$ to the edges e_k of f.

Figure 20: Two cases of $d_{min}(p_{i/i-1}, f)$, indicated by the green dashed line and computed by Procedure 2.

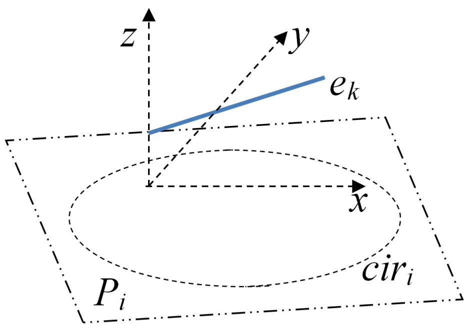


Figure 21: Coordinate system established in Procedure 3

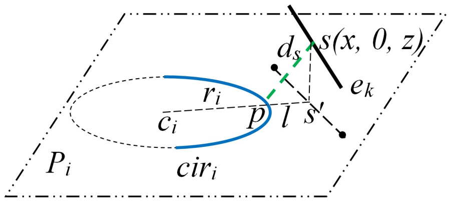
Let the center of cir_i be $(x_c, y_c, 0)$ and the radius of cir_i be r_i. The line of e_k is on the xz plane and satisfies the equation

$$z = ax + b. \tag{8}$$

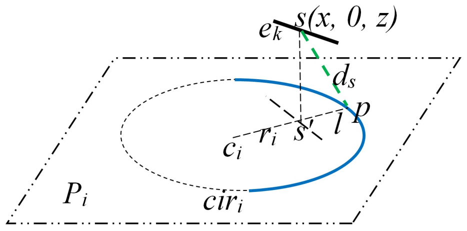
Any point s on e_k has coordinates $(x, 0, z)$, and its projection s' on P_i has coordinates $(x, 0, 0)$. The closest point to s' on cir_i is the intersection point p between the line segment from s' to the center of cir_i and cir_i, and the corresponding distance l (from s' to p) satisfies:

$$l = \begin{cases} r_i - \sqrt{(x-x_c)^2 + y_c^2} & if\ s'\ is\ inside\ or\ on\ cir_i, \\ \sqrt{(x-x_c)^2 + y_c^2} - r_i & if\ s'\ is\ out\ of\ cir_i. \end{cases} \tag{9}$$

See Fig. 22. Note that p is also the closest point on cir_i to s.



(a) The projection s' on P_i is out of cir_i.



(b) The projection s' on P_i is inside cir_i.

Figure 22: Two cases for computing the distance l.

The distance d_s from the point s on e_k to its closest point p on cir_i satisfies: $d_s = \sqrt{z^2 + l^2}$, which, with (8) and (9), results in:

$$d_s = \sqrt{(ax+b)^2 + (\sqrt{(x-x_c)^2 + y_c^2} - r_i)^2} \tag{10}$$

Let $v_{k,1}$ and $v_{k,2}$ be the vertices of e_k with coordinates $(x_1, 0, z_1)$ and $(x_2, 0, z_2)$ respectively. Let d_1 and d_2 correspond to $v_{k,1}$ and $v_{k,2}$ respectively, satisfying equation (10).

In order to find the point $s = q$ that has the minimum distance $d_s = d_q$ to cir_i, we take the derivative of d_s^2 with respect to x

$$\frac{dd_s^2}{dx} = 2a(ax+b) + \frac{2(x-x_c)(\sqrt{(x-x_c)^2 + y_c^2} - r_i)}{\sqrt{(x-x_c)^2 + y_c^2}} \tag{11}$$

and equate it to zero:

$$2a(ax+b)+\frac{2(x-x_c)(\sqrt{(x-x_c)^2+y_c^2}-r_i)}{\sqrt{(x-x_c)^2+y_c^2}}=0 \tag{12}$$

The above equation (12) can be expressed as the following quartic equation of x, which have analytical solutions [3]:

$$t_4x^4+t_3x^3+t_2x^2+t_1x+t_0=0 \tag{13}$$

where

$t_4 = a^4+2a^2+1,$

$t_3 = 2(a^2+1)(ab-x_c)-2x_c(a^2+1)^2,$

$t_2 = (a^2+1)^2(x_c^2+y_c^2)-4x_c(a^2+1)(ab-x_c)-r_i^2$

$\quad +(ab-x_c)^2,$

$t_1 = 2r_i^2x_c-2x_c(ab-x_c)^2+2(a^2+1)(ab-x_c)(x_c^2+y_c^2),$

$t_0 = (ab-x_c)^2(x_c^2+y_c^2)-r_i^2x_c^2.$

If there exist roots of (13) that correspond to points within e_k, and if the corresponding smallest d_s is smaller than $min(d_1,d_2)$, then obtain the corresponding point q on e_k and the corresponding closest point on cir_i as p. Otherwise, obtain the vertex of e_k corresponding to $min(d_1,d_2)$ as q and the corresponding closest point on cir_i as p.

2) If p is on seg_i, return the distance between p and q as $d_{min}(seg_i,e_k)$; otherwise, return the shortest distance from the two endpoints (p_i and p_{i+1}) of seg_i to e_k as $d_{min}(seg_i,e_k)$.

We introduce Procedure 1 and Procedure 3 because there is no ready algorithm for analytically computing the distance from a polygon or edge to a finite curve (seg_i) in 3D space. Our Procedure 1 and Procedure 3 introduced here are efficient and accurate.

As for Procedure 2, a related algorithm was proposed in [35] for computing the 3D distance from a point to a triangle, which uses the barycentric coordinates of a triangle to check whether the projection of a point is within the triangle. In our algorithm, however, the face f is not limited to be triangular but can be any convex polygon (e.g., it can be a face of a bounding box).

Appendix further shows that Algorithm 5 is complete.

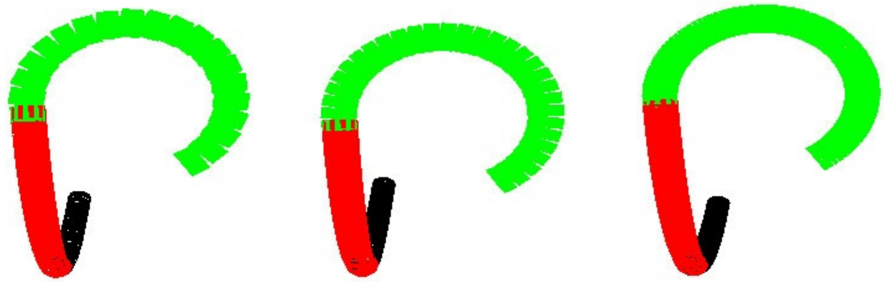

(a) 2,000 triangles per section (Mesh 1) (b) 4,000 triangles per section (c) 8,000 triangles per section (Mesh 2)

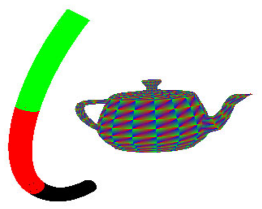
Figure 23: Three OctArm mesh models.

Table 1: Space/time required for different OctArm models at a given arm configuration

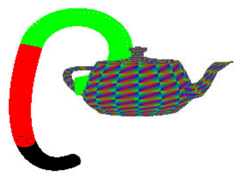
Arm Model	Exact (CD-CoM)	Mesh 1	Mesh 2
Description	9 config. parameters: κ_i, s_i, ϕ_i, $i = 1, 2, 3$	2,000 triangle/section	$8,000$ triangle/section
Space	36 byte/section	18K byte/section	72K byte/section
Compute vertices t_c	None	4.5 ms/section	18 ms/section
BVH rebuilding t_{rb}	None	9 ms/section	48 ms/section
BVH refitting t_{rf}	None	2 ms/section	12 ms/section
Updating mesh points for a new configuration?	No need	Yes	Yes

4.5 Comparative Study and Test Results

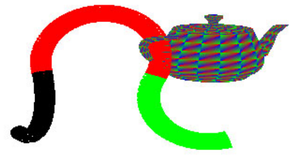
We have implemented the CD-CoM algorithm in C++ and applied it to collision detection between the three-section OctArm manipulator and polygonal mesh models of arbi-



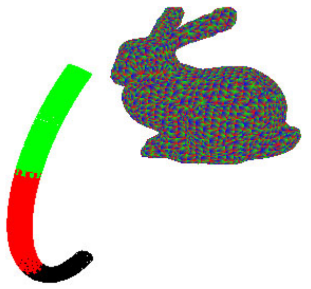
(a) Config. 1 (no collision)



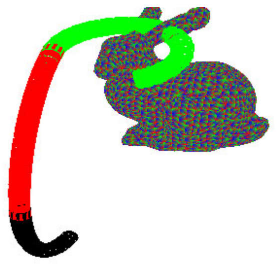
(b) Config. 2 (collision at section 3)



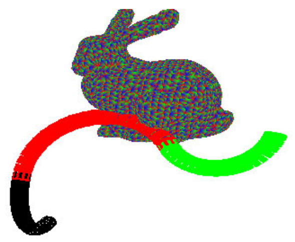
(c) Config. 3 (collision at section 2)



(d) Config. 4 (no collision)



(e) Config. 5 (collision at section 3)



(f) Config. 6 (collision at section 2)

Figure 24: Arm configurations tested for collision with a teapot and a bunny.

trary objects. With single precision float point variables in our implementation, ranging from $3.4e^{-38}$ to $3.4e^{+38}$ with 7 significant figures, we can maintain the numerical stability.

We have also compared the collision detection results using our CD-CoM algorithm with those using a mesh-based collision detection algorithm OPCODE [85], which is similar to SOLID [94, 95] or RAPID [29]. Since OPCODE uses an AABB tree on top of an object mesh to speed up collision checking, we also apply CD-CoM to the AABB tree of the same object mesh for comparison. Since our CD-CoM takes a tree of bounding boxes (in this case an AABB tree) as an input, but the OPCODE builds an AABB tree for each input object mesh, for fair comparison, we subtract the tree building time for each object from the total collision checking time that OPCODE uses. Moreover, each time the configuration of the continuum manipulator changes, the mesh of the manipulator changes too, and therefore, the AABB tree for the manipulator mesh will have to be changed. To adjust the AABB tree of the manipulator, we use the refitting method implemented in OPCODE as well. We run both the CD-CoM algorithm and the OPCODE on the same PC with a 2.4GHz core.

Fig. 23 displays three mesh models for the OctArm, from coarse to reasonably fine. As shown, Mesh 2 with 8,000 triangles per arm section is required to provide a necessarily smooth approximation. Of course, a finer mesh model will be even better for accuracy but more expensive.

Table 1 compares the space and time required by the exact arm model of the OctArm and those of the two mesh models (a coarse one and a fine one) of the OctArm for any given configuration. The space cost is computed based on the number of vertices of the corresponding mesh model. Three time costs are presented for dealing with a mesh model of the manipulator: (a) the time t_c needed for computing the positions of mesh vertices for any given arm configuration; (b) the time t_{rf} required for refitting the bounding volumes given updated positions of mesh vertices and bounding volume topology; (c) the time t_{rb} needed for rebuilding the hierarchy of bounding volumes given mesh vertices.

Although refitting the BVH takes substantially less time $(70\% - 80\%)$ than rebuilding the BVH, both of them cannot avoid the cost of updating the positions of mesh vertices for different arm configurations. Note that updating mesh vertices for the manipulator here is more costly that for some deformable object models, e.g., model for a waving/vibrating skirt, where positions of mesh vertices after deformation can be either computed quickly, e.g., by linear interpolation, or given in input files for each time frame (before and after deformation). Whereas, updating mesh vertices for the manipulator requires mapping from the high-dimensional configuration space of the continuum manipulator to the Cartesian space.

In general, if R is the time required for updating a single mesh model of a continuum manipulator, then to represent and store a single path of k configurations of the manipulator in a mesh model requires kR time, which could be too expensive to be feasible even for off-line motion planning. This is because motion planning usually requires examining and maintaining a vast number of configurations to search for a good path. In contrast, our CD-CoM algorithm uses the exact arm model of a continuum manipulator directly to avoid the high time and space cost of updating mesh models for different configurations and thus facilitates motion planning for a continuum manipulator.

4.5.1 Comparative Analysis of Worst-case Time Complexity

To analyse the time complexity of our algorithms, we first denote the following:

n:	the number of arm sections of the continuum manipulator.
M:	the number of bounding volumes of a polygonal mesh of an object.
m_o:	the maximum number of features (i.e., faces, edges, and vertices) of the object mesh at the lowest level of the bounding volume hierarchy.
k :	number of arm configurations to check for a path.
N:	the total number of bounding volumes is using mesh model for the manipulator.
m_a:	the maximum number of features (i.e., faces, edges, and vertices) of the mesh approximating the arm at the lowest level of the bounding volume hierarchy.

Then the worst-case time complexity of the CD-CoM Algorithm is $O[(M + m_o)n]$ for checking one arm configuration [49], and the worst-case time complexity is $O[k(M + m_o)n]$.

However, in order to use a conventional mesh-to-mesh collision detection algorithm, such as the OPCODE, the continuum manipulator will have to be approximated by a mesh.

Then the worst-case time complexity of a mesh-to-mesh collision detection algorithm is $O(MN + m_o m_a)$. Since $N >> n$ (as it requires more than one bounding volume for each arm section), and $m_a >> n$, $MN + m_o m_a >> (M + m_o)n$. Thus, the CD-CoM Algorithm has a far lower order of worst-case time complexity than OPCODE, and we can further show that $O(MN + m_o m_a) >> O[(M + m_o)n]$.

For checking collision of a path of k arm configurations, the worst-case time complexity of CD-CoM is $O[k(M + m_o)n]$; whereas, the worst-case time complexity of OPCODE is $O[k(MN + m_o m_a) + Nk]$, with the additional term Nk reflecting the time complexity of refitting the bounding volume hierarchy of the manipulator for each change of configuration. Thus, $k(MN+m_o m_a)+Nk >> k(M+m_o)n$, and moreover, $O[k(MN+m_o m_a)+Nk] >> O[k(M + m_o)n]$.

4.5.2 Test Results and Discussion

Table 2 and Table 3 present the results of collision detection between the OctArm and two different object meshes, a teapot mesh model with $1,024$ triangles and a bunny mesh model with $3,851$ triangles, at different OctArm configurations, as shown in Fig. 24. We choose the teapot and bunny because they are sophisticated, non-convex objects with many polygons in the mesh; they are as complex as many objects in practical applications. The CD-CoM algorithm takes less time than OPCODE to detect collisions, while OPCODE also takes time to re-compute the positions of mesh vertices (t_c) and to refit BVHs of mesh models (t_{rf}) for different arm configurations: for Mesh 1 of the manipulator, our algorithms takes at most 60% of the time (config. 6) that OPCODE uses for collision checking, and for the finer Mesh 2 of the manipulator, it takes at most 18% of the time (config. 6) that

OPCODE uses.

Table 2: Collision detection time between the OctArm and a teapot mesh (with 1,024 triangles)

Arm Model	Algorithm	Config. 1	Config. 2	Config. 3
Exact	CD-CoM	$<$ 1 ms	4 ms	6 ms
Mesh 1	OPCODE	7 ms	9 ms	12 ms
Mesh 2	OPCODE	36 ms	41 ms	43 ms

Table 3: Collision detection time between the OctArm and a bunny mesh (with 3,851 triangles)

Arm Model	Algorithm	Config. 4	Config. 5	Config. 6
Exact	CD-CoM	$<$ 1 ms	6 ms	10 ms
Mesh 1	OPCODE	7 ms	11 ms	16 ms
Mesh 2	OPCODE	37 ms	47 ms	53 ms

Fig. 25 compares the collision detection time for each collision check for 100 randomly sampled OctArm configurations (according to an uniform distribution) against the teapot using CD-CoM vs. OPCODE, and Table 4 compares the average and median time per collision check. The results clearly show that the CD-CoM algorithm is more efficient than OPCODE applied to even the coarse mesh model of the continuum manipulator (with only 37% on average of the time that OPCODE uses). The average time (3.2 ms) is also much shorter than the time (18 ms) reported in [44], which conducts collision detection based on spheres and updates BVHs dynamically, for meshes with comparable numbers of triangles.

We have also tested the performance of CD-CoM vs. OPCODE employed by a motion planner for grasping in a cluttered space by a continuum manipulator[54]. The planner searches a path of configurations for an n-section continuum manipulator to gradually wrap around a target object with its tip following the contour of the object without colliding with nearby obstacles. Fig. 27 shows snapshots of such a path, which consists of 160

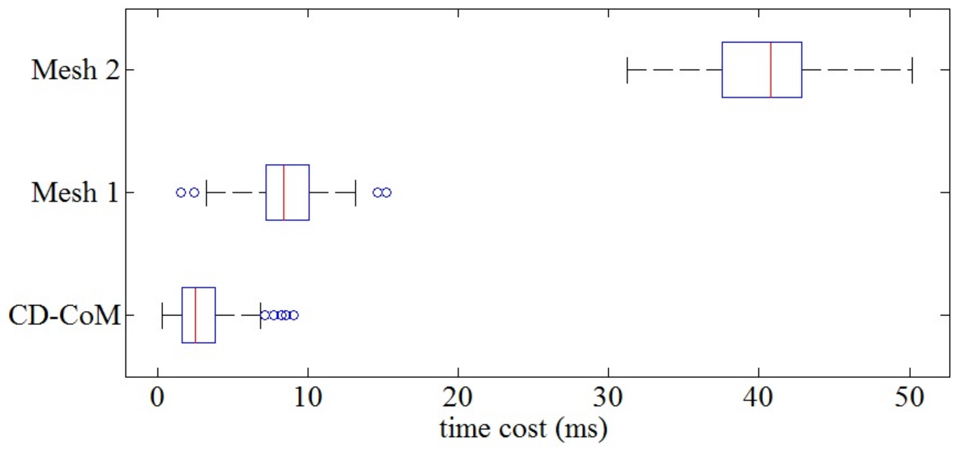


Figure 25: The box-and-whiskers plot of collision detection time between the OctArm and the teapot for 100 arm configurations.

configurations for a four-section continuum manipulator to grasp the teapot in a confined environment. To plan the path, the planner checked 1,120 configurations.

Table 4: Collision detection time between the OctArm and the teapot over 100 arm configurations

Arm Model	Algorithm	Average time	Median time
Exact	CD-CoM	3.2 ms	2.5 ms
Mesh 1	OPCODE	8.5 ms	8.38 ms
Mesh 2	OPCODE	40.8 ms	40.74 ms

Fig. 26 compares the collision detection time by CD-CoM vs. OPCODE for each collision check of the 1,120 configurations of the four-section continuum manipulator in the search of the feasible path for grasping the teapot. Table 5 compares the average and median time per collision check and the total time of collision check needed for the 1,120 configurations. The CD-CoM algorithm is clearly more efficient.

The efficiency of the CD-CoM algorithm is also demonstrated in the application of inspection with the OctArm [56].

Table 6 shows the average number of times Algorithm 4 and Algorithm 5 were called by

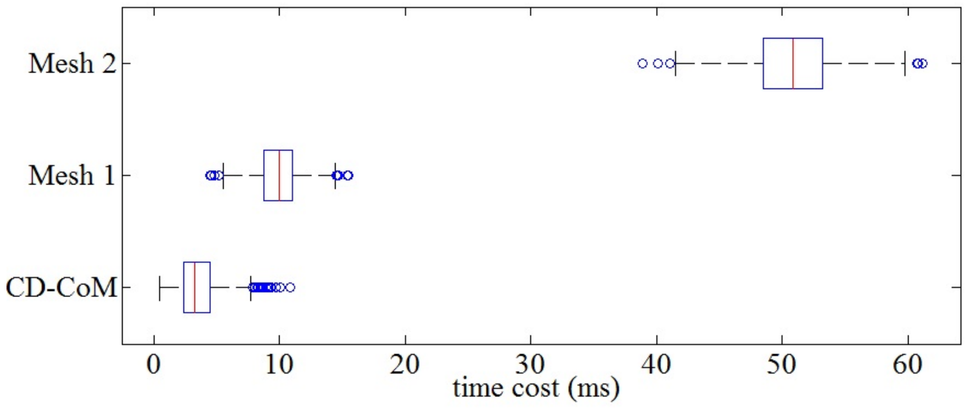


Figure 26: The box-and-whiskers plot of collision detection time for the 1,120 configurations of a four-section continuum manipulator in searching a feasible path for grasping the teapot.

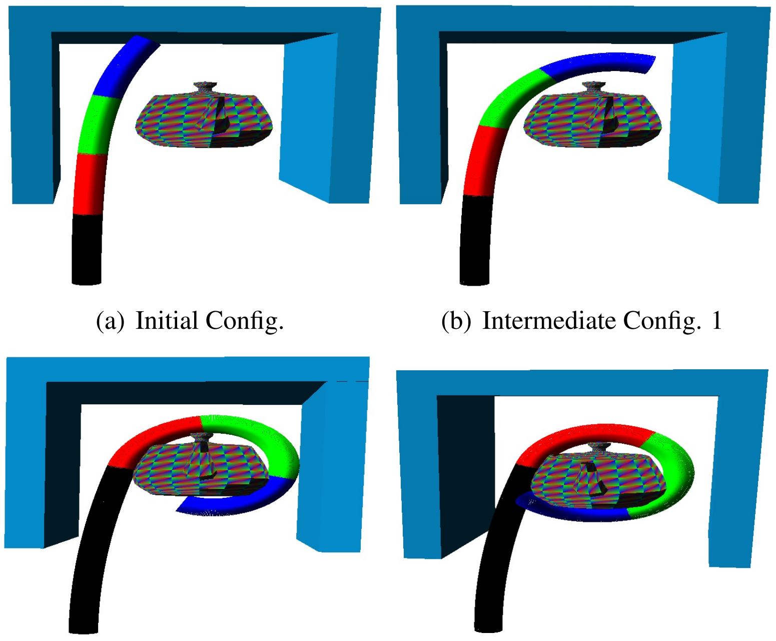


Figure 27: A few snapshots of a path of 160 configurations for grasping the teapot by a four-section continuum manipulator.

CD-CoM per configuration check over the 1,120 configurations as well as the average time for running each algorithm once. Recall that if calling Algorithm 4 can detect a collision, there is no need to call Algorithm 5, and if the face of an object does not intersect the section plane of the considered arm section, then Algorithm 4 will not be called. Clearly, Algorithm

4 is 10 times more efficient than Algorithm 5 on average, and the use of Algorithm 4 is significant in reducing the overall computation cost.

Table 5: Collision detection time over the 1,120 configurations

Arm Model	Algorithm	Average time	Median time	Total time
Exact	CD-CoM	3.5 ms	3.27 ms	3.92 s
Mesh 1	OPCODE	10.3 ms	9.98 ms	11.53 s
Mesh 2	OPCODE	50.8 ms	50.84 ms	56.89 s

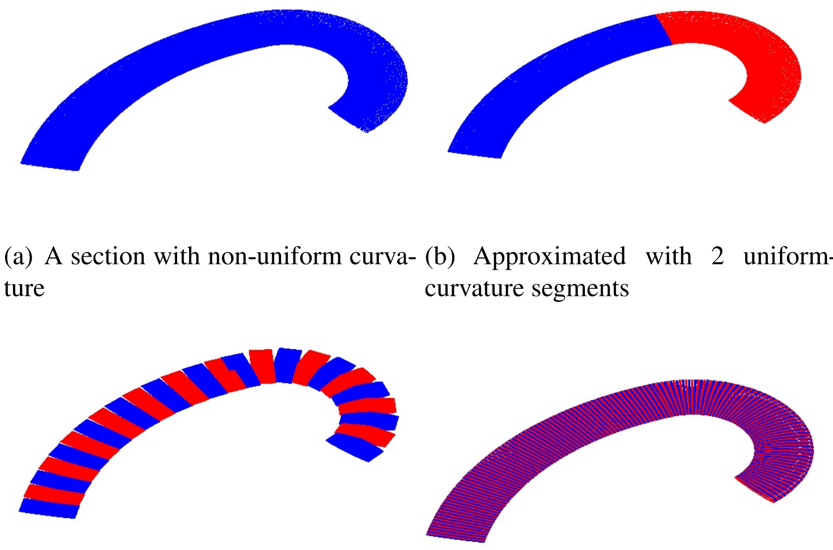
Table 6: Percentage and time cost of calling Algorithm 4 and Algorithm 5 for collision check per configuration over the 1,120 configurations

Algorithm	Algorithm 4 only	Algorithm 5 only	both
Avg. % called	44.98%	22.3%	32.72%
Avg. time/call	0.001 ms	0.01 ms	0.011 ms

Note that the total planning time for the path for grasping is 4.61s by using CD-CoM (of which 3.92s is used for collision checking, as shown in Table 5), and the average planning time per configuration is 4.11ms, which is comparable to the real-time planning time reported in [96] for a mobile manipulator with an articulated arm of six degrees of freedom (which used OPCODE for collision detection) and satisfies the requirement for real-time response [43] in the order of milliseconds. Therefore, CD-CoM is suitable for real-time motion planning for a continuum manipulator.

4.6 Remarks

In this chapter, we introduced an efficient algorithm for collision detection between a continuum manipulator and environmental objects. Such a continuum manipulator can be continuously deformed into different concave shapes by changing each section's configuration parameters (i.e., s, κ, ϕ). Unlike conventional collision detection algorithms that require mesh models, our CD-CoM algorithm uses toroidal, parametric models of a continuum manipulator instead of mesh models, and therefore, it does not need to update mesh models for different arm configurations, which requires re-computing all the positions of the vertices in a mesh and refitting the bounding volumes of the mesh. As such, our algorithm saves significant time and space. It is more efficient comparing to collision detection using even a coarse mesh model of the manipulator.



(a) A section with non-uniform curvature (b) Approximated with 2 uniform-curvature segments

(c) Approximated with 25 cylindrical segments (d) Approximated with 296 cylindrical segments

Figure 28: Comparison of different approximations of a non-uniform curvature section.

The CD-CoM algorithm is applicable to (1) exact models of continuum manipulators featuring n uniform-curvature sections ($n = 1, 2, 3, ...$) or (2) continuum manipulators with sections of non-uniform curvatures such that each non-uniform curvature section can be approximated by uniform-curvature segments. Indeed, using uniform-curvature segments, which are usually non-convex, is far more efficient than using straight cylindrical segments or finer convex primitives to approximate a non-uniform curvature section. Fig. 28 shows an example, where using only two uniform-curvature segments provides better approximation result and more efficient than using cylindrical segments. Note that we alternate blue and red colors to indicate segments used in approximation. The CD-CoM algorithm is clearly more feasible for collision detection involving non-uniform curvature sections than existing algorithms using convex primitives.

The CD-CoM algorithm is particularly suitable for continuum manipulator path planning that considers a large number of manipulator configurations in real time. CD-CoM also provides information of the minimum distance between each section of the continuum manipulator and objects when there is no collision, which could be further exploited to achieve desirable contact states for continuum manipulation and to take advantage of time and space coherence of a moving continuum manipulator and moving objects.

CHAPTER 5: DETERMINING GRASPING CONFIGURATIONS FOR A 3-SECTION CONTINUUM MANIPULATOR

5.1 Overview

Unlike a conventional articulated manipulator, where only the gripper manipulates objects, a continuum manipulator, such as a multi-section trunk/tentacle robot, is promising for deft manipulation of a wide range of objects of different shapes and sizes. Given an object, a continuum manipulator tries to grasp it by wrapping around and squeezing it. A main open problem is how to determine if the object can be grasped and if so, the whole-arm wrapping around configurations of the robot to grasp it, which we call *grasping configurations*.

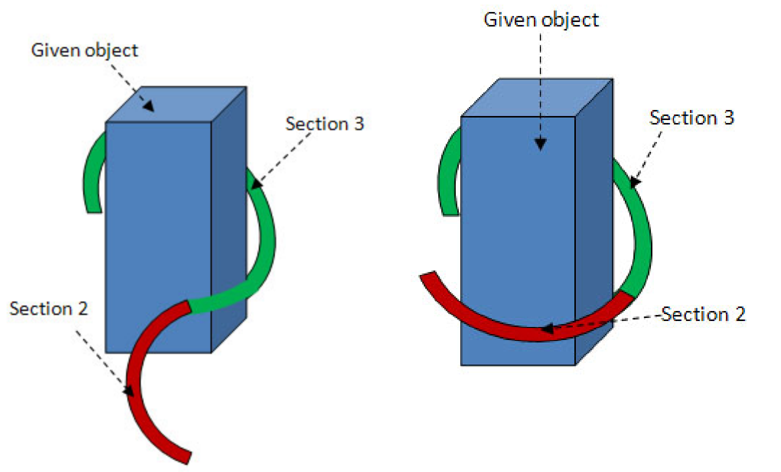
In this chapter, we propose a general and complete analysis of grasping configurations of a spatial continuum manipulator consisting of 3 sections for an given 3-D object and then formulate constraints for existence of solutions and describe how to find grasping configurations.

5.2 Object Grasping Models for a 3-section Continuum Manipulator

For a given 3D object, we define two *grasping models* for a continuum manipulator consisting of 3 sections:

- Grasping Model 1: section 3 of the robot wraps around the object, and other sections of the robot may wrap around the object but not along section 3's circle. See Figure 29.

- Grasping Model 2: both section 2 and section 3 of the robot wrap around the object along the same circle[2]. See Figure 30.



(a) Only section 3 wraps around the object. (b) Section 2 also wraps around the object along a different circle.

Figure 29: Two examples of Grasping Model 1.

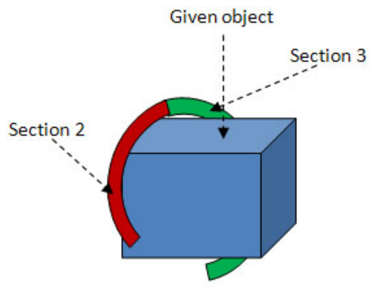


Figure 30: Grasping Model 2 where section 2 (red) and section 3 (green) are on the same circle.

To realize either grasping model, we need to first obtain bounding circles of the object, and then choose feasible bounding circles such that either the section 3 of the robot or the combined sections 2 and 3 of the robot can wrap around the object, i.e., make sure that the

[2]Similarly, the case of all sections of the robot wrapping around the object along the same circle can be considered. We omit the case because it is trivial and not very practical.

value ranges of the section curvature and length of the robot allow either grasping model to happen. Once a suitable object bounding circle for either grasping model is obtained, we can further determine (in subsequent sections of the paper) suitable configurations of the entire robot that realize the corresponding grasping model, which we call, *grasping configurations*.

Different bounding circles of an object can be obtained from different cross sections of the object. A bounding circle cir_o can be obtained automatically with the following steps:

- Select a plane through the center of mass of the object[3].
- Intersect the plane with the object to create a cross-section, which can be expressed as a polygon.
- Find the minimum bounding circle cir_o of the cross section polygon [61].

Next, we need to check if the bounding circle cir_o is of suitable size for the robot to grasp. Let r_o be the radius of cir_o. If one of the following conditions is satisfied, cir_o has a suitable size:

$$s_{3,min} \leq a\pi r_o \leq s_{3,max} \tag{14}$$

or

$$s_{3,min} + s_{2,min} \leq a\pi r_o \leq s_{3,max} + s_{2,max} \tag{15}$$

where $a \in (0, 2]$ is a coefficient determining how much the object bounding circle has to be wrapped. Its value depends on the shape, size, and material characteristics of the target object, as well as on the task of manipulation. For example, the task of pulling an object

[3]Or of the preferred grasping part, such as a handle, depending on the object.

could require a smaller a than that of picking up the object. If (14) is satisfied, cir_o is suitable for the Grasping Model 1; otherwise, if (15) is satisfied, cir_o is suitable for the Grasping Model 2.

Considering that the actual robot has a width w, for Grasping Model 1, we grow the cir_o by $w/2 - \delta$ to obtain section 3's circle that (the central axis of) section 3 needs to curve along, and for Grasping Model 2, the same grown circle will be for both the (central axes of) section 3 and section 2 to curve along. Note that δ is a small value to ensure tight wrap, taking advantage of the inherent compliance and small deformation of the OctArm. For convenience, we call such a grown circle cir_o the *object circle*.

Note that a configuration of an OctArm is in terms of only controllable variables of its sections. However, the OctArm can bend anywhere passively (i.e., with infinite passive degrees of freedom). The smooth and compliant nature of such a continuum structure allows it to gently interact with the object by adapting its shape to the object it wraps. Therefore, for either grasping model, once a corresponding configuration of the OctArm is found (to be described in the following sections of the paper), the inherent compliance of the arm will allow a tight wrap with as much continuum contact as possible (hence as much friction as possible) to make the grasp robust [63].

We describe how to find grasping configurations for each grasping model below.

5.3 Finding Configurations For Grasping Model 1

For Grasping Model 1, the object circle is the desired section 3's circle for grasping. Now, given this section 3's circle, i.e., its center position $\mathbf{c}_3$, radius r_3 (or curvature $|\kappa_3|$), and the circle plane normal, which can be expressed as $\mathbf{y}_3$ – the unit vector of the y axis of the section 3's frame, we need to solve for values of the other OctArm configuration variables: κ_1 (or r_1), ϕ_1, s_1, κ_2 (or r_2), ϕ_2, and s_2.

There are two situations of configurations for Grasping Model 1, where the section 3's circle is given and not shared by section 2:

- *general situation*: section 1 and section 2 do not share the same circle;
- *special situation*: section 1 and section 2 share the same circle.

We focus on finding grasping configurations of the general situation in this section. We will first specify constraints relating sections of the OctArm for the Grasping Model 1 and then use those constraints to solve for the unknowns to obtain grasping configurations.

The special situation can be handled in a way similar to that for determining grasping configurations for Grasping Model 2.

5.3.1 Inter-Section Constraints

For Grasping Model 1, in the general situation where no two sections of the OctArm share the same circle, the section circles have to satisfy the following:

- the section 1's circle is tangent to the section 2's circle at point p_1, which is also the end point of section 1, and
- the section 2's circle is tangent to the section 3's circle at point p_2, which is also the end point of section 2.

p_1 and p_2 are both on section 2's circle. Let l_{12} and l_{23} be the tangent lines going through p_1 and p_2 respectively, as shown in Figure 31. They must satisfy the following two constraints:

- Constraint 1: the tangent lines l_{12} and l_{23} must be coplanar.
- Constraint 2: the tangent lines l_{12} and l_{23} must be on the same circle.

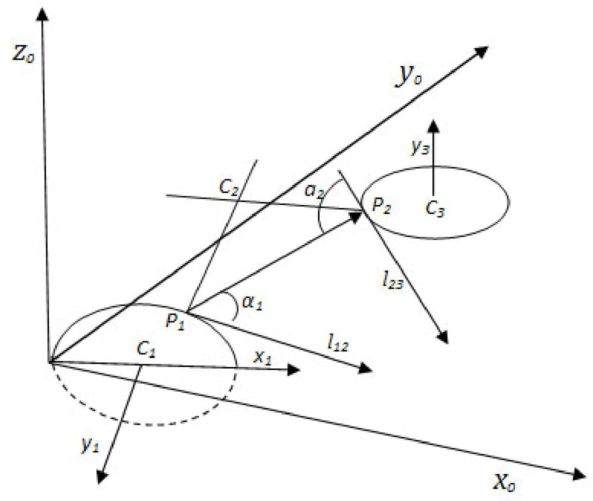


Figure 31: Two tangent lines, l_{12} and l_{23}, that pass p_1 and p_2 respectively.

Define two vectors along l_{12} and l_{23} respectively as the following:

$$\mathbf{l}_{12} = \mathbf{y}_1 \times (\mathbf{p}_1 - \mathbf{c}_1) \tag{16}$$

and

$$\mathbf{l}_{23} = \mathbf{y}_3 \times (\mathbf{p}_2 - \mathbf{c}_3) \tag{17}$$

where $\mathbf{y}_1$ and $\mathbf{y}_3$ are the unit vectors of the y axes of section 1 and section 3 respectively, which also represent plane normals of the two sections respectively; $\mathbf{c}_1$ and $\mathbf{c}_3$ are the position vectors of the centers of section 1's and section 3's circles respectively; $\mathbf{p}_1$ and $\mathbf{p}_2$ are the position vectors of p_1 and p_2 respectively, in the robot's base frame. $\mathbf{y}_1$ can be further expressed in terms of ϕ_1 as:

$$\mathbf{y}_1 = R(z_1, \phi_1)\mathbf{y}_0 \tag{18}$$

where $R(z_1, \phi_1)$ is the rotation matrix of the rotation about the z_1 axis with angle ϕ_1 and $\mathbf{y}_0$ is the (fixed) y axis of the base frame of the robot.

$\mathbf{c}_1$ can be expressed in terms of ϕ_1 and κ_1 as:

$$\mathbf{c}_1 = R(z_1, \phi_1)[-\frac{1}{\kappa_1}, 0, 0]^T \tag{19}$$

Now, for the case that l_{12} and l_{23} are *not parallel*, Constraint 1 can be expressed as:

$$(\mathbf{p}_2 - \mathbf{p}_1) \cdot (\mathbf{l}_{12} \times \mathbf{l}_{23}) = 0 \tag{20}$$

Let α_1 be the angle between $\mathbf{p}_2 - \mathbf{p}_1$ and $\mathbf{l}_{12}$ and α_2 be the angle between $\mathbf{p}_2 - \mathbf{p}_1$ and $\mathbf{l}_{23}$, as shown in Figure 32, such that[4]

$$sin(\alpha_1) = \frac{||\mathbf{l}_{12} \times (\mathbf{p}_2 - \mathbf{p}_1)||}{||\mathbf{l}_{12}||||\mathbf{p}_2 - \mathbf{p}_1||} \tag{21}$$

and

$$sin(\alpha_2) = \frac{||\mathbf{l}_{23} \times (\mathbf{p}_2 - \mathbf{p}_1)||}{||\mathbf{l}_{23}||||\mathbf{p}_2 - \mathbf{p}_1||} \tag{22}$$

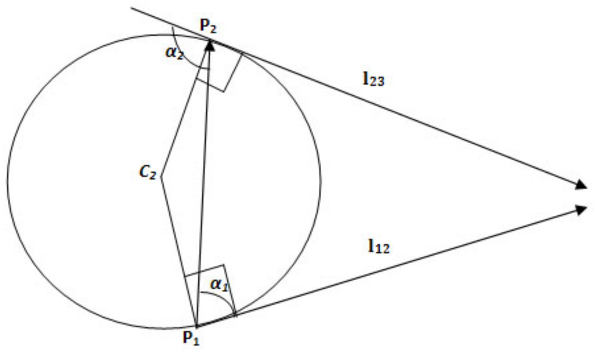


Figure 32: The tangent lines l_{12} and l_{23} and section 2's circle are on the same plane; α_1 and α_2 are complementary.

[4]by the definition of cross product.

Then, to satisfy Constraint 2, α_1 and α_2 must be either equal or complementary (see Figure 32), depending on the directions of $\mathbf{l}_{12}$ and $\mathbf{l}_{23}$, that is, the following equation must be satisfied:

$$sin(\alpha_1) = sin(\alpha_2) \tag{23}$$

For the case that l_{12} and l_{23} are *parallel*, then Constraint 1 is satisfied since

$$\mathbf{l}_{12} \times \mathbf{l}_{23} = 0 \tag{24}$$

and Constraint 2 can be expressed as:

$$(\mathbf{p}_2 - \mathbf{p}_1) \cdot \mathbf{l}_{12} = 0. \tag{25}$$

Since p_1 and p_2 are on section 1's circle and section 3's circle respectively, each can be expressed in terms of a scalar angle as derived below. Define a local coordinate system for section i's circle, as illustrated in Figure 33, such that its origin is at the circle center c_i, and two unit vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ form the orthogonal axes on the circle plane. $\mathbf{u}_i$ and $\mathbf{v}_i$ are functions of $\mathbf{y}_i$; Denote $r_i = \frac{1}{|\kappa_i|}$ as the radius of the circle for section i.

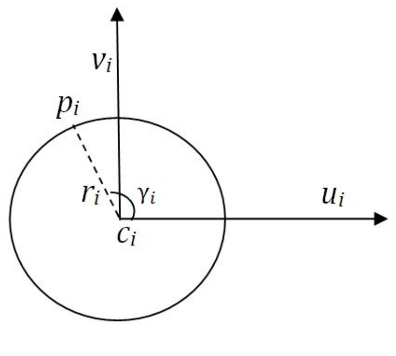


Figure 33: The section i circle's local coordinate system.

Thus, since p_1 is on the section 1's circle, its position vector (in the robot base frame) must satisfy:

$$\mathbf{p}_1 = \mathbf{c}_1 + r_1 cos(\gamma_1)\mathbf{u}_1 + r_1 sin(\gamma_1)\mathbf{v}_1 \tag{26}$$

where γ_1 is the angle from the vector $\mathbf{u}_1$ to $\mathbf{p}_1 - \mathbf{c}_1$.

Similarly, since p_2 is on the section 3's circle, its position vector must satisfy:

$$\mathbf{p}_2 = \mathbf{c}_3 + r_3 cos(\gamma_3)\mathbf{u}_3 + r_3 sin(\gamma_3)\mathbf{v}_3 \tag{27}$$

where γ_3 is the angle from the vector $\mathbf{u}_3$ to $\mathbf{p}_2 - \mathbf{c}_3$.

5.3.2 Finding Solutions

From the above, the inter-section constraint equations for the case where the two tangent lines l_{12} and l_{23} are not parallel are (20) and (23). The constraint equations for the case where l_{12} and l_{23} are parallel are equations (24) and (25).

With equations (16) to (23), we can re-write the inter-section constraint equations in terms of four variables: κ_1, ϕ_1, θ_1, and θ_3. With two equations and four variables in either case, we can solve for two variables with the other two variables assuming any values within their value ranges. If we assign values to κ_1 and ϕ_1, i.e., specify the section 1's circle, we can solve for θ_1 and θ_3. Once θ_1 and θ_3 are solved, we can further solve for section 2's circle and then the corresponding grasping configuration.

We describe the process in turn below.

5.3.2.1 Solving For θ_1 and θ_3

Now we describe how to solve for θ_1 and θ_3 for a given pair of κ_1 and ϕ_1 values within their respective ranges. Since the constraint equations (in either the non-parallel or the parallel case) written in terms of θ_1 and θ_3 are high-order, non-linear trigonometric equations of those two variables, we can only obtain numerical solutions (by Newton's method), which require good initial guesses.

In order to be efficient, we narrow down the range of search for initial guesses to be those that satisfy the length ranges of the continuum manipulator. We first discretize both θ_1 and θ_3, ranging from 0 to 2π, into small intervals. For each pair of θ_1 and θ_3 values, we then use equations (26) and (27) to compute the corresponding $\mathbf{p}_1$ and $\mathbf{p}_2$.

If $\mathbf{p}_1$ and $\mathbf{p}_2$ do not satisfy the following length constraints,

$$s_{2,max} \geq ||\mathbf{p}_2 - \mathbf{p}_1|| \tag{28}$$

and

$$s_{1,max} \geq ||\mathbf{p}_1|| \tag{29}$$

we discard the corresponding θ_1 and θ_3.

From those pairs of θ_1 and θ_3 that satisfy inequalities (28) and (29), we search for pairs that approximately satisfy the constraint equations as initial guesses of θ_1 and θ_3.

With the initial guesses, for the case where l_{12} and l_{23} are not parallel and the case where l_{12} and l_{23} are parallel, the corresponding inter-section constraints in terms of θ_1 and θ_3 are then solved numerically.

5.3.2.2 Determining Grasping Configurations

Once θ_1 and θ_3 are solved, the corresponding $\mathbf{p}_1$ and $\mathbf{p}_2$ can be solved from equations (26) and (27), and $\mathbf{l}_{12}$ and $\mathbf{l}_{23}$ can be obtained from equations (16) and (17) respectively. Next we need to solve for the plane, center, and radius of section 2's circle.

In the case where two tangent lines l_{12} and l_{23} are parallel, the center c_2 of section 2's circle is on the same line as p_1 and p_2, and thus, its position satisfies:

$$\mathbf{c}_2 = \frac{(\mathbf{p}_1 + \mathbf{p}_2)}{2} \tag{30}$$

In the case where l_{12} and l_{23} are not parallel, l_{12} and l_{23} determines the plane of the section 2. Line l_1 through p_1, perpendicular to l_{12}, and on the plane of section 2 can be determined, and similarly line l_2 through p_2, perpendicular to l_{23}, and on the plane of section 2 can be determined as shown in Figure 32. Then the intersection point of l_1 and l_2 is the center c_2 of section 2's circle.

In both cases, The radius of section 2's circle is:

$$r_2 = \|\mathbf{p}_1 - \mathbf{c}_2\| \tag{31}$$

From section 1's circle (for the specified values of κ_1 and ϕ_1) and section 2's circle, as well as p_1 and p_2, which are end-points of section 1 and section 2 respectively, the unknown configuration parameters of section 1 and section 2: s_1, κ_2, s_2, and ϕ_2 can be found easily [66]. They are valid if their values are within their respective ranges. The length s_3 of section 3 can be set to its maximum value to maximize wrapping along the section 3's circle (which is given). Now a grasping configuration of the entire OctArm is found for Grasping Model 1.

5.4 Finding Configurations for Grasping Model 2

For Grasping Model 2, the given object circle is shared by both section 2 and section 3, i.e., the section 3's circle and section 2's circle are the same, with the following known parameters: the circle center position $\mathbf{c}_3 = \mathbf{c}_2$, the radius $r_3 = r_2$, the unit normal vector of the circle plane $\mathbf{y}_2$, and $\phi_3 = 0$ or π. We only need to find the configuration parameters of section 1 that satisfy the Grasping Model 2.

Section 1's circle and section 2's circle must be tangent at point p_1, sharing the same tangent line l_{12} through p_1. Now, axis z_1 is also tangent to the section 1's circle and is the same as the z_0 axis of the robot base frame, with unit vector $\mathbf{z}_1$. Thus, l_{12} must satisfy the following two constraints:

- Constraint A: l_{12} and axis z_1 must be coplanar.
- Constraint B: l_{12} and z_1 must be on the same circle.

Define a vector along l_{12} as:

$$\mathbf{l}_{12} = \mathbf{y}_2 \times (\mathbf{p}_1 - \mathbf{c}_2) \tag{32}$$

For the case of non-parallel l_{12} and z_1, Constraint A can be expressed as:

$$\mathbf{p}_1 \cdot (\mathbf{l}_{12} \times \mathbf{z}_1) = 0 \tag{33}$$

The expression of Constraint B can be obtained similarly as:

$$\frac{||\mathbf{l}_{12} \times \mathbf{p}_1||}{||\mathbf{l}_{12}||} = \frac{||\mathbf{z}_1 \times \mathbf{p}_1||}{||\mathbf{z}_1||} \tag{34}$$

For the case of parallel l_{12} and z_1, the above two constraints become

$$(\mathbf{l}_{12} \times \mathbf{z}_1) = 0 \tag{35}$$

and

$$\mathbf{p}_1 \cdot \mathbf{z}_1 = 0 \tag{36}$$

Since $\mathbf{p}_1$ is on section 2's circle, it satisfies:

$$\mathbf{p}_1 = \mathbf{c}_2 + r_2 cos(\gamma_2)\mathbf{u}_2 + r_2 sin(\gamma_2)\mathbf{v}_2 \tag{37}$$

where $\mathbf{u}_2$ and $\mathbf{v}_2$ are functions of the given $\mathbf{y}_2$. By substituting equation (32) for $\mathbf{l}_{12}$ and then equation (37) for $\mathbf{p}_1$ in the constraint equations above, we can re-write the two constraint equations in each case in terms of a single variable γ_2. Solving the equations numerically yields at most two solutions of γ_2 (in either the non-parallel or parallel case).

Next $\mathbf{p}_1$ and $\mathbf{l}_{12}$ can be solved via equations (37) and (32), from which and $\mathbf{z}_1$, the section 1's circle and configuration variables ϕ_1, κ_1, and s_1 can be solved numerically. There are at most two grasping configurations to realize a given Grasping Model 2.

Note that with analysis and derivation similar to the above, we can find at most two grasping configurations of the special situation where section 1 and section 2 share the same circle for a given Grasping Model 1.

5.5 Implementation and Discussion

Algorithm 6 implements the method introduced above for finding grasping configurations. For Grasping Model 1, since there are fewer constraint equations than the variables in the general situation, solutions of grasping configurations depend on values of κ_1 and ϕ_1, which are input to the algorithm.

Algorithm 6 also include checking for penetrations of the robot into the target object for a found grasping configuration. Specifically, it checks if section 2 or section 1 of the robot penetrates into the object. See Figure 34 for an example. If so, the grasping configuration is not valid and should be discarded. However, if section 2 or section 1 almost touch the object, the grasping configuration is even preferred because it means a tighter wrap, as shown in Figure 29(b), recalling that the object circle is made deliberately smaller (by δ) to allow tight wrap with compliance. The collision-checking method is [57].

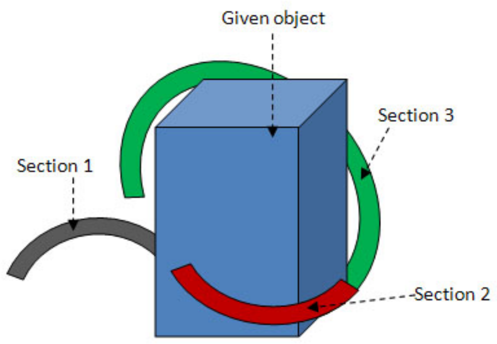


Figure 34: A grasping configuration in collision with the object.

Given a Grasping Model 1, for one given pair of values κ_1=a and ϕ_1=ψ, if there are corresponding grasping configurations found by Algorithm 1, then because κ_1 and ϕ_1 can change values continuously, for each found grasping configuration $\mathbf{C}(a, \psi)$, there exists a small continuous neighborhood $B(\mathbf{C})$ of configurations corresponding to a small continuous neighborhood:

$$D(a, \psi) = \{\kappa_1, \phi_1 | \delta\kappa_1 > |\kappa_1 - a|, \delta\phi_1 > |\phi_1 - \psi|\}$$

where $\delta\kappa_1$ and $\delta\phi_1$ are small positive values. All configurations in $B(\mathbf{C})$ have very similar shapes to the found configuration $\mathbf{C}(a, \psi)$. However, for κ_1 and ϕ_1 values outside $D(a, \psi)$, their corresponding grasping configurations can be disconnected from $B(\mathbf{C})$ and belong to another continuous neighborhood of configurations, as illustrated in Figure 35. Note that for the same pair of κ_1 and ϕ_1 values, there can be multiple solutions of grasping configurations, and each belongs to a different continuous neighborhood. This is also illustrated

Algorithm 6: Finding grasping configurations

```
input object circle with known $\mathbf{c}_3$, $\mathbf{y}_3$, and $r_3$
if Grasping Model 1 then
    input $\kappa_1$ and $\phi_1$
    choose suitable initial guesses $\theta_{1guess}$, $\theta_{3guess}$;
    solve for $\theta_1$ and $\theta_3$;
else
    if Grasping Model 2 then
        $\mathbf{c}_2 = \mathbf{c}_3$, $\mathbf{y}_2 = \mathbf{y}_3$, and $r_2 = r_3$;
        choose suitable initial guesses $\theta_{2guess}$;
        solve for $\theta_2$;
    end if
end if
compute the corresponding grasping configurations $(\kappa_1, \phi_1, s_1, \kappa_2, \phi_2, s_2, \kappa_3, \phi_3, s_3)$;
for each grasping configuration do
    if the configuration passes penetration check then
        record this grasping configuration;
    end if
end for
return all recorded grasping configurations.
```

in Figure 35, where there exists points in $D(a_1, \psi_1)$, each of which corresponds to two grasping configurations in $B(\mathbf{C}_1)$ and $B(\mathbf{C}_2)$ respectively.

There are a finite number of such neighborhoods of grasping configurations for a Grasping Model 1.

By discretizing the value ranges of κ_1 and ϕ_1 with a proper resolution, we can solve for grasping configurations representing all the neighborhoods for the case of Grasping Model 1. Based on the value ranges of κ_1 and ϕ_1 (for the OctArm – see Fig. 10 (e)), we find it reasonable to set the discretization interval to be 3% of each range. For each given Grasping Model 1, by running Algorithm 1 for each pair of κ_1 and ϕ_1 from the discretization, at least one representative configuration in each neighborhood of grasping configurations can be found. Note that for a given object circle, the actual value ranges of κ_1 and ϕ_1 that can

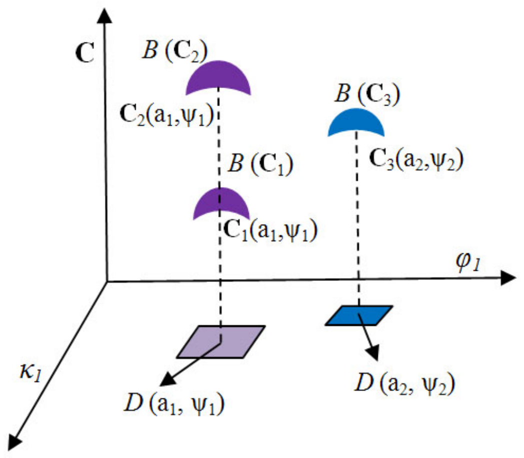


Figure 35: Grasping configurations vs. κ_1 and ϕ_1. For $(\kappa_1, \phi_1) = (a_i, \psi_i)$, $i = 1, 2$, the corresponding configurations are $\mathbf{C}_j(a_i, \psi_i)$ in neighborhood $B(\mathbf{C}_j)$, $j = 1, 2, 3$, which corresponds to neighborhood $D(a_i, \psi_i)$. $B(\mathbf{C}_1)$ and $B(\mathbf{C}_2)$ both correspond to $D(a_1, \psi_1)$.

possibly lead to grasping configurations can be smaller than the respective maximum value ranges for those parameters. For the example Grasping Model 1 object circles, the average time to run Algorithm 1 for finding grasping configurations corresponding to one pair of κ_1 and ϕ_1 is $0.05s$. The total time to find representative grasping configurations of all neighborhoods of configurations is $4s$ to $7s$ per example. This shows that our method is suitable for on-line determination of grasping configurations for the OctArm for a given object.

5.6 Test Examples

We present three examples here. We first describe types of grasping configurations and then provide representative grasping configurations found for those examples.

We can classify the OctArm grasping configurations based on the orientations ϕ_2 (i.e., the angle between planes of section 1 and section 2) and ϕ_3 into the following four types:

(1) ϕ_2 is negative and ϕ_3 is positive.

(2) ϕ_2 is positive and ϕ_3 is negative.

(3) ϕ_2 and ϕ_3 are both positive.

(4) ϕ_2 and ϕ_3 are both negative.

Table 7 shows the descriptions of three object circles, two of them are of Grasping Model 1, in terms of circles for section 3 (see Figure 36), and the other one is of Grasping Model 2, in terms of a circle that both section 3 and section 2 share.

Table 7: Example Object Circles

Grasping Model	Circle Center	Radius	Normal
1	$[45.0cm, 37.5cm, 60cm]^T$	$22.5cm$	$[-0.55cm, 0cm, 0.83cm]^T$
1	$[45.0cm, 37.5cm, 60cm]^T$	$22.5cm$	$[-0.84cm, 0.24cm, 0.48cm]^T$
2	$[6.0cm, 34.5cm, 24.75cm]^T$	$30.83cm$	$[-0.76cm, 0.013cm, 0.64cm]^T$

Table 8: Grasping Configurations in Figures 37– 39

Grasping Model 1	Fig.&Sol.	Grasping Configuration $(\kappa_i(1/cm), s_i(cm), \phi_i(\text{rad})), i = 1, 2, 3$
Type (1)	Fig. 37 Sol. 1	$(0.0228, 34.0, 0.4;\ 0.0324, 41.6, -1.85;\ 0.0443, 53.5, 0.23)$
Type (1)	Fig. 37 Sol. 2	$(0.0228, 37.1, 0.6,\ 0.0284, 38.2, -2.01,\ 0.0441, 53.5, 0.20)$
Type (2)	Fig. 37 Sol. 3	$(0.0228, 37.1, 0.6,\ 0.0143, 41.2, 1.57,\ 0.0442, 53.5, -0.37)$
Type (2)	Fig. 37 Sol. 4	$(0.0213, 39.1, 0.4,\ 0.0056, 43.7, 2.16,\ 0.0443, 53.5, -0.82)$
Type (3)	Fig. 38 Sol. 1	$(0.0228, 39.9, 0.4,\ 0.0085, 41.7, 0.23,\ 0.0444, 53.5, 0.77)$
Type (4)	Fig. 38 Sol. 2	$(0.0228, 33.2, 0.2,\ 0.0373, 43.1, -2.00,\ 0.0444, 53.5, -0.18)$
Grasping Model 2	Fig. 39	$(0.0228, 34.1, 0.4,\ 0.0324, 41.6, -1.85,\ 0.0324, 53.5, 0.0)$

For the object circle specified in row 1 of Table 7, four valid solutions are obtained by Algorithm 6 as illustrated in Figure 37, where solutions 1 and 2 belong to the type (1), and solutions 3 and 4 belong to the type (2), and corresponding configurations are shown in Table 8.

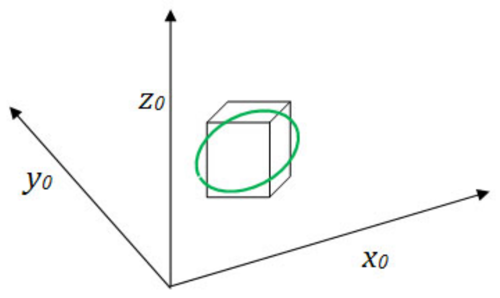
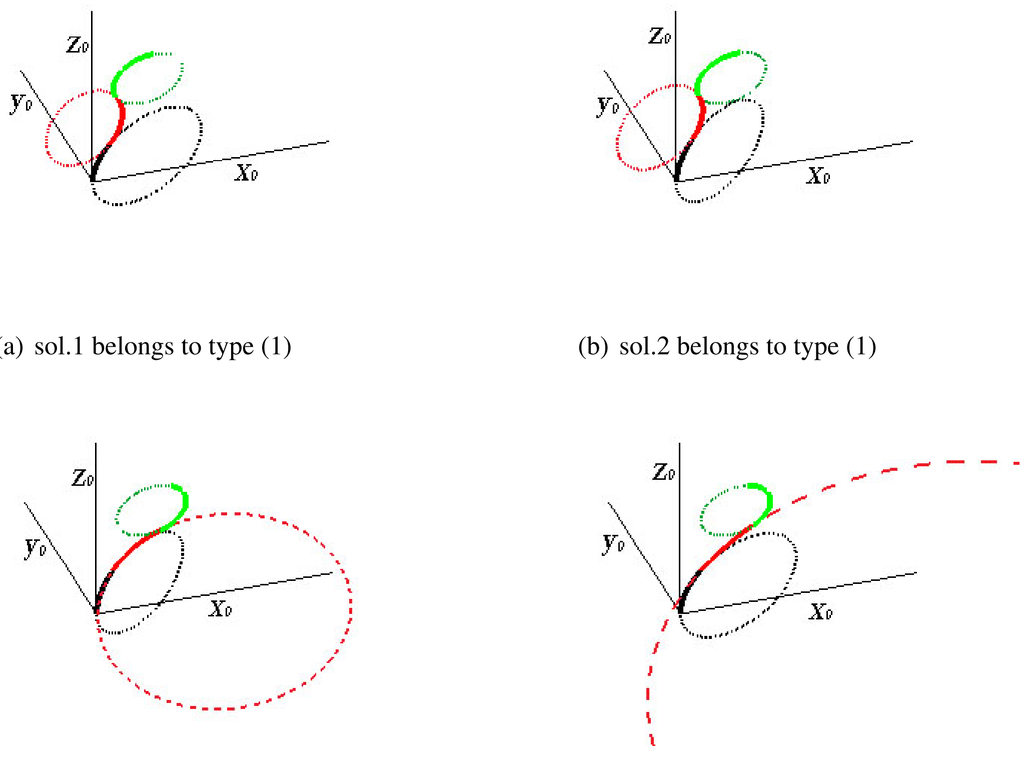


Figure 36: Grasping Model 1: a section 3's circle bounding an object.



(a) sol.1 belongs to type (1)

(b) sol.2 belongs to type (1)

(c) sol.3 belongs to type (2)

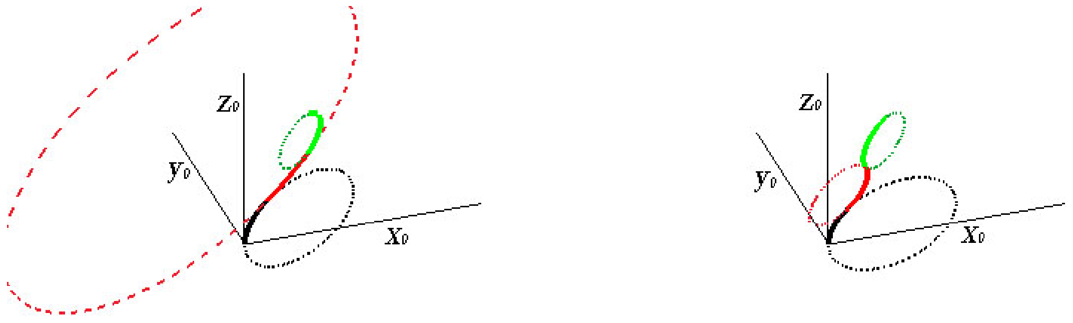
(d) sol.4 belongs to type (2)

Figure 37: Representative solutions for the object circle in row 1 of Table 7.

Note that solutions 1 and 2 belong to the same neighborhood where ϕ_1 changes continuously from 0.4 to 0.6 when $\kappa_1 = 0.0228$. Note also that solutions 2 and 3 are of different types but share the same pair of κ_1 and ϕ_1 values. This is one example to show that there can be multiple solutions for the same κ_1 and ϕ_1, which belong to separate (continuous) neighborhoods (see Figure 35 for an illustration). For this particular case, there are at most three neighborhoods of grasping configurations.

For the object circle specified in row 2 of Table 7, two solutions found by Algorithm 6 are illustrated in Figure 38, where solution 1 belongs to type (3) and solution 2 belongs to type (4). Therefore, in this case, there are only two separate neighborhoods of grasping configurations. Corresponding configurations are also shown in Table 8.

For the object circle of Grasping Model 2 in row 3 of Table 7, there is a unique solution, and the solution is shown in Figure 39, where the grasping configuration is shown in Table 8.



(a) sol.1 belongs to type (3) (b) sol.2 belongs to type (4)

Figure 38: Representative solutions for the object circle in row 2 of Table 7.

5.7 Remarks

This chapter presents a general analysis and method to determine grasping configurations for a given 3D object by a spatial continuum manipulator consisting of three constant-curvature sections and with a fixed base. The curvature, length, and the angle of each section of the manipulator are continuous variables that can be changed. Thus, a configuration

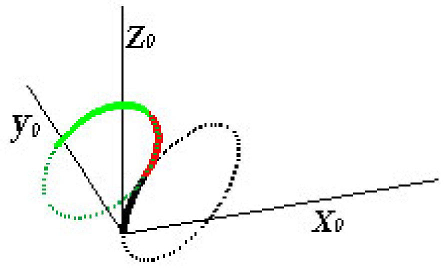


Figure 39: The unique solution for the object circle of Grasping Model 2 in row 3 of Table 7.

of the three-section manipulator is a 9-dimensional vector of those variables. Our approach first defines grasping models for a given object to facilitate grasping by the continuum manipulator. It then uses inter-section constraints in analytical equations to characterize and numerically solve for all possible grasping configurations. For a given object grasping model, the approach is implemented to determine valid grasping configurations.

CHAPTER 6: PROGRESSIVE GRASPING FOR A N-SECTION CONTINUUM MANIPULATOR

6.1 Overview

In this chapter, two real-time grasping approaches [55, 53] will be introduced to guide a general n-section continuum manipulator progressively forms a force-closure grasping configuration for a 3D target object in an open environment and a cluttered environment respectively. The grasping method introduced in Chapter 5 can computed all possible grasping configurations for a 3-section continuum manipulator. However, for a general n-section continuum manipulator, where $n \geq 3$, the computational cost of solving inter-section constraints increases significantly as n increases. Also for a cluttered environment, where the information of objects or the obstacles in the environment may not be fully observable, an on-line, real-time strategy of grasping is necessary.

In the rest of this chapter, two progressive grasping algorithms for a continuum manipulator will be introduced for open and cluttered environments respectively, both simulations and real robot experiments will be presented to validate the effectiveness and efficiency of the proposed algorithms.

6.2 Grasp Generation in an Open Environment

This section will introduce the progressive grasping approach for an open environment. Basically, the algorithm is to create a tight and force-closure grasp of a target object by an n-section continuum manipulator from an initial configuration of the manipulator arm C_0, where one arm section sec_k $(k \leq n)$ contacts the object at point p_c, and sections sec_k to sec_n share the same (rather large) section circle cir (see Fig. 44(a) for an example) bounding the object. sec_k will be the starting arm section to wrap around the target object[5]. cir can be chosen easily.

Now, for the rest of the arm sections that do not wrap around the object bounding circle, sec_{k-1} to sec_1, we can compute their configurations by inverse kinematics [66] one by one, considering that the position of the tip point p_{k-1} of sec_{k-1} is the same as the based point of sec_k, and that the section circle cir_k is tangent to cir_{k-1} at p_{k-1}, and so on. There are many solutions for the arm sections sec_{k-1} to sec_1, especially if the arm has a mobile base. Depending on the environment surrounding the target object, one can search for an optimal or near-optimal solution based on certain optimization criteria, such as avoiding obstacles and shortest section lengths, etc. For the three-section OctArm manipulator, we can compute all possible arm configurations with respect to a given object bounding circle [52], which, in this case, can be the loose circle cir.

From the arm sections around cir, sec_k to sec_n, our strategy is to make each section from sec_k to sec_n tightly wrap around the object mesh without overlap, one by one. The resulting configurations of these arm sections will form a spiral shape wrapping around the object to give a spatial grasp. From the set of contact points and normals between the arm and the object, we can check if the grasp is a force-closure grasp [26]. If it is not a force-closure grasp, our algorithm will increase the angle value $|\phi|$ for each section from sec_{k+1} to sec_n, with the effect of elongating the helix-shaped arm sections wrapping the object to find a force-closure grasp.

[5]The selection of sec_k is determined depending on the (rough) size of the object.

6.2.1 Algorithm of Generating a Grasp in an Open Environment

Algorithm 7 outlines our strategy to generate a tight grasp. The arm will wrap the object section by section from sec_k to sec_n tightly by calling Algorithm 8. If the resulting configuration of the arm forms a force-closure grasp of the object, a tight grasping configuration is achieved and output. Otherwise, the algorithm will change the angle between sec_{i+1} and sec_i, for $k \leq i \leq n$, and repeat the section by section wrapping process until either a force-closure grasp is formed or no force-closure grasp can be found after a maximum number of grasping configurations are generated and checked from the given initial configuration C_0.

Note that if n is small, i.e., the arm has only a few sections, such as the OctArm, the search for a force-closure grasp starting from C_0 can be rather systematic and exhaustive by discretizing the value of ϕ_i within its range and considering each discrete value. If n is large, randomized or heuristic techniques can be used to search for a force-closure grasp opportunistically, and thus an upper limit max on the number of configurations considered is useful. If no force-closure grasp can be found from a specific C_0, then a different initial configuration should be considered, and so on, but searching the best C_0 depends the target object posture and its surrounding environment, which is not the focus of this paper.

6.2.2 Motion of Each Arm Section in an Open Environment

Algorithm 8 decides how to curl each arm section sec_i, $k \leq i \leq n$, tightly to form a tight grasp of the object, given the configuration of the arm where sec_i contacts the object only at v_0^i. Note that for sec_k, $v_0^k = p_c$ is the only contact point between sec_k and the object before it is curved around the object by this algorithm. Let $poly_i$ be the cross-section of the object by plane P_i of sec_i. Now v_0^i is a vertex of $poly_i$.

Algorithm 7: Generate a grasp in an open environment

```
input : Arm model in the initial configuration $C_0$ around object bounding circle
        $cir$, where $sec_k$ contacts the object at $p_c$, and the object model
output: A list of contact points and normals, the corresponding grasping
        configuration $C$, and a path from $C_0$ to $C$
begin
    $C_{current} = C_0$;
    $path = \{C_0\}$;
    $conlist_k \leftarrow p_c$ with contact normal;
    $count = 0$;
    while $count < max$ do
        for $i = k$ to $n$ do
            call Algorithm 10 to make $sec_i$ wrap the object, obtain the new arm
            configuration $C_i$ and list of contact points and normals $conlist_i$;
            if "unable to wrap" then
                return "no solution"
            end
            $conlist_{i+1} \leftarrow$ the last contact point and normal in $conlist_i$;
            $path = path \cup \{C_i\}$;
            $C_{current} = C_i$;
        end
        $contactlist = \cup_k^n conlist_i$;
        if $ForceClosure(contactlist) == true$ then
            return $contactlist$ and $path$;
        end
        $C_{current} = C_0$;
        $path = \{C_0\}$;
        $conlist_k \leftarrow p_c$ with contact normal;
        for $i = k + 1$ to $n$ do
            increase $|\phi_i|$ by $\delta\phi$ within its bound to elongate the arm sections
            wrapping the object;
        end
        $count = count + 1$;
    end
    return "found no force-closure grasp";
end
```

Algorithm 8 chooses each vertex v sequentially from the next vertex v_1 in the direction of shrinking sec_i (see Fig. 40) and tries to make sec_i's tip reach the furthest vertex in that direction without penetration. In the process, the lengths of the adjacent sections to sec_i

Algorithm 8: Curl each section in an open environment

```
input  : Arm model and configuration $C_{current}$, $conlist_i$ having a single contact
         point $v_0$ between $sec_i$ and object, and object model
output : A list of contact points and normals ($cList$) between $sec_i$ and object
         mesh, and the corresponding arm configuration $C$
1  $C = C_{current}$;
2  $\{v_0, v_1, ..., v_m\} \leftarrow$ vertices of $poly_i$ in the shrinking direction of $sec_i$, as shown in
   Fig. 40;
3  for $v = v_1$ to $v_m$ do
4      if $sec_i$ can reach $v$ at a configuration $C_r$ (computed by inverse kinematics
       [66]) then
5          check penetration between the arm and the object at $C_r$;
6          if $penetration$ then
7              If expand returns an updated, penetration-free configuration $C$, then
               break;
8          else
9              call shrink to update $C$;
10             break;
11         end
12     end
13 end
14 if $sec_i$ does not collide with itself or another arm section then
15     $cList \leftarrow$ contact points and normals at configuration $C$ ;
16     return $C$ and $cList$;
17 else
18     increase $|\phi_i|$ by a small $\delta\phi$;
19     if $\phi_i$ is out of bound then
20         return ”unable to wrap”
21     end
22     go to step 1;
23 end
```

can be adjusted by calling *expand* and *shrink* procedures. If no arm section overlaps as the result, the corresponding arm configuration is returned; otherwise, the angle value $|\phi_i|$ between sec_i and sec_{i-1} is increased by a $\delta\phi$ to change the section plane P_i of sec_i, and the process of curling sec_i is repeated with the new section plane.

Procedures *expand* and *shrink* are called by Algorithm 8 to either increase or decrease the lengths of adjacent sections of sec_i for avoiding penetrations between sec_i and the object

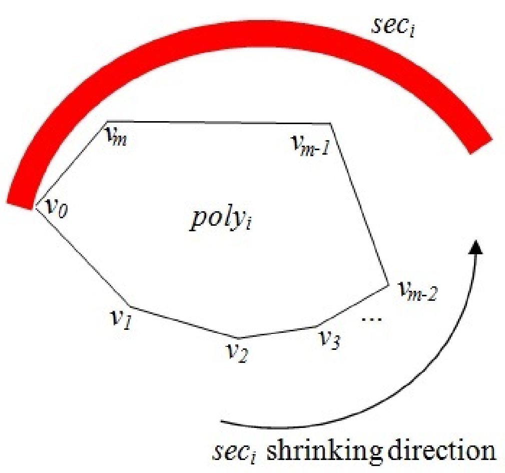


Figure 40: The vertices of $poly_i$ are ordered as $\{v_0, v_1, ..., v_m\}$ in the shrinking direction of sec_i starting from v_0.

and for further tightening the grasp respectively. Let v be the furthest vertex considered for the tip of sec_i to reach.

The procedures work as follows:

expand: if sec_i and sec_{i-1} share the same section plane, then

- **repeat:** increase the length of sec_{i-1} by a small amount and make sec_i reach v **until** $no\,penetration$ **or** sec_{i-1} *reaches its maximum length*;
- if $no\,penetration$ return the arm configuration C; otherwise return ”failed”.

shrink: if sec_i and sec_{i-1} share the same section plane, then

- **repeat:** shrink the length of sec_{i-1} and make sec_i reach v **until** a $penetration$ **or** sec_{i-1} *reaches its minimum length*;
- if $penetration$, backtrack the shrinked length of sec_{i-1} until a non-penetrated configuration is found and return this configuration; otherwise return current arm configuration;

else, return current arm configuration.

6.3 Grasp Generation in a Cluttered Environment

Next, an algorithm will be introduced to tackle the problem of continuum grasping constrained by a tight space in a cluttered environment. This algorithm enables a multi-section continuum manipulator to probe an object with its tip while gradually form a force-closure grasp by making the arm closely following along the contour of the probed object.

The process can start by having the manipulator in a contracted straight-line pose, as if a stick, and fit into the space along one side of the target object. Next, the tip of the manipulator is made to move a small step along the contour of the object, which is enabled by extending and curling every arm section, starting from the tip section. This process is repeated until the tip of the arm travelled around the object and carried the arm to form a force-closure grasp of the object, which can be a spiral grasp. During the whole process, the arm extends along the contour of the object to fit into the tight space surrounding the object and avoid colliding into other obstacles. The arm movement can be adaptive based on how far ahead along the object surface the tip can aim at.

Initially, the manipulator is put in a contracted straight-line configuration C_0, as if a stick, and fit into the (narrow) space along one side of the target object (see Fig. 41). The manipulator is put close to the nearby obstacle to maximize the clearance from the target object for maneuverability.

6.3.1 Algorithm of Generating a Grasp in a Cluttered Environment

The algorithm is outlined in Algorithm 9. It moves every section of the arm in small steps repeatedly to follow the contour of the target object while avoiding penetration into

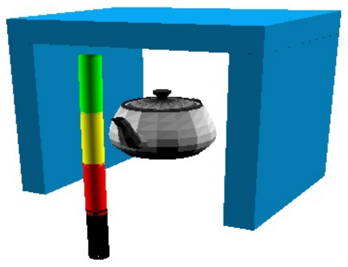


Figure 41: The initial configuration of the manipulator besides the target object (teapot).

the object or collision with surrounding obstacles, led by the tip of the arm. We call such steps *feasible moves*. The Algorithm 9 calls Algorithm 10 to generate a move for each arm section and the corresponding feasible arm configuration C_{new}, which is considered a *knot configuration* on the path towards a feasible grasping configuration.

The maximum steps each section can move can be estimated by the maximum extendable length of a section divided by the amount of extension per small step. Note that sometimes a feasible move may not be possible for sec_i so that Algorithm 10 returns "wrapping paused". However, after a feasible move for another section sec_j is generated, a feasible move for sec_i may again be possible. Moreover, different sections can have different value limits on its changeable variables. Hence, a section may not be moved exactly the *max* steps.

After all arm sections have reached their limits in curling and extending, if a force-closure grasp [26] is achieved, Algorithm 9 returns the entire path of knot configurations that leads to the force-closure grasp. Otherwise, it reports that no force-closure grasp is found. If the target object is too large or too small, a force-closure grasp may not be possible; whether the given object has a proper size to be grasped can be checked [52, 51].

6.3.2 Generation of a Knot Configuration in a Cluttered Environment

Algorithm 10 curls and extends arm section sec_i in a small step towards a targeted vertex

Algorithm 9: Generate a grasp in a cluttered environment

```
input  : arm model in the initial configuration $C_0$
output: corresponding grasping configuration $C$, a contact list $contactlist$ and a
        path from $C_0$ to $C$
1  begin
2      $C_{current} \leftarrow C_0$;
3      $path \leftarrow \{C_0\}$;
4      $count \leftarrow 0$;
5      while $count < max$ do
6          for section $i \leftarrow 1$ to $n$ do
7              call Algorithm 10 to move $sec_i$ with a small step and obtain a new arm
               configuration $C_{new}$ and the list of contact points and normals $conlist_i$;
8              $path \leftarrow path \cup \{C_{new}\}$;
9              $contactlist \leftarrow \cup_1^n conlist_i$;
10             $C_{current} \leftarrow C_{new}$;
11         end
12         if all sections have returned "$wrapping\ paused$" then
13             if $ForceClosure(contactlist) = true$ then
14                 return $path$ and $contactlist$;
15             end
16             return "no force-closure grasp is found";
17         end
18         $count \leftarrow count + 1$;
19     end
20     return "no force-closure grasp is found";
21 end
```

on the cross section polygon $poly_i$ of the object to generate a new, feasible knot configuration. It returns the corresponding feasible configuration and the list of contacts (i.e., list of pairs of contacting points between sec_i and the target object). Let v_1 be the closest vertex that sec_i's tip point p_i has not reached in the wrapping direction on the obejct's cross section $poly_i$ of sec_i, and v_f be the farthest visible vertex that has not been reached. Algorithm 10 chooses a target vertex v between v_1 and v_f (see Fig. 42) and tries to move sec_i's tip towards v without penetrating into the object or colliding with nearby obstacles.

How to choose a target vertex v among $v_1, .., v_f$ is an interesting issue. We experimented

Algorithm 10: Motion of each arm section in a cluttered environment

```
input  : arm model and configuration $C_{current}$ and object models
output: a list of contact points and normals ($cList$) between $sec_i$ and object mesh,
        and the corresponding feasible arm configuration $C_{new}$
1  let $V = \{v_1, ..., v_f\}$ be the ordered list of unreached visible vertices of $poly_i$, as
   shown in Fig. 42;
2  if $V = \varnothing$ then
3      return "$wrapping\ paused$";
4  end
5  search a vertex $v$ in $V$ based on strategies (1), (2) or (3), so that $sec_i$'s tip can
   reach $v$ in a feasible arm configuration $C_r$;
6  if no vertex in $V$ is reachable then
7      return "$wrapping\ paused$";
8  end
9  compute a new configuration $C_{new}$ after making one step linearly towards $C_r$ from
   $C_{current}$ and check if $C_{new}$ is feasible;
10 call Algorithm 11 to repair an infeasible $C_{new}$;
11 if "$repair\ fails$" then
12     return "$wrapping\ paused$";
13 end
14 $cList \leftarrow$ the list of contact points and normals at configuration $C_{new}$ (see section
   II.C);
15 return $C_{new}$ and $cList$.
```

with three strategies:

- strategy (1): searching and choosing the farthest reachable vertex along the wrapping direction;
- strategy (2): searching and choosing the nearest reachable vertex along the wrapping direction;
- strategy (3): randomly searching and choosing a reachable vertex between the nearest and the farthest vertices.

Fig. 42 shows an example of both nearest and farthest reachable vertices for sec_i.

We determine whether a vertex v on $poly_i$ is reachable by first computing the arm configuration C_r where sec_i's tip is at v (via inverse kinematics [66]) and then checking if C_r is feasible, i.e., no collision with obstacles and no penetration with the object; if so, v is reachable by sec_i; otherwise, v is not reachable. If no vertex is reachable by sec_i, then Algorithm 10 returns "$wrapping\ paused$" for sec_i.

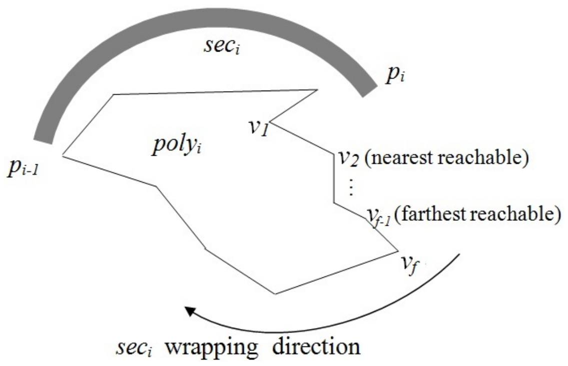

sec_i
p_i
v_1
$poly_i$
p_{i-1}
v_2 (nearest reachable)
v_{f-1} (farthest reachable)
v_f
sec_i wrapping direction

Figure 42: The unreached visible vertices of $poly_i$ as an ordered list $\{v_1, v_2, ..., v_f\}$ in the wrapping direction of sec_i, where the nearest and farthest reachable vertices are indicated.

All these strategies lead to a force-closure grasping configuration for the manipulator. However, strategy (1) generates fewer knot configurations but a longer path for the manipulator and requires less (total) planning time and fewer collision checks. Strategy (2) generates more knot configurations and requires longer (total) planning time (with more collision checks), but the path is shorter and closer to the target object contour. The performance of strategy (3) is between that of strategy (1) and strategy (2), as expected.

Once the target vertex v for the arm tip is determined, a new knot configuration C_{new} is obtained by making a small straight-line move in the configuration space from the current arm configuration $C_{current}$ to C_r.

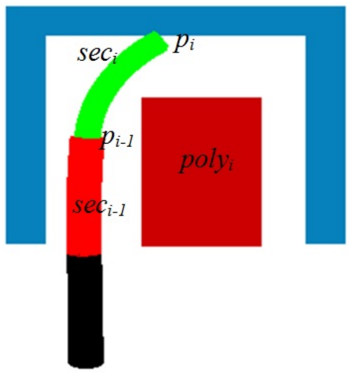
6.3.3 Repair of an Infeasible Knot Configuration in a Cluttered Environment

If the new knot configuration C_{new} generated is not feasible, then Algorithm 11 is called to repair the configuration to obtain a feasible one. Depends on whether colliding with the object or the surrounding obstacles, the corresponding arm sections will take different repair actions.

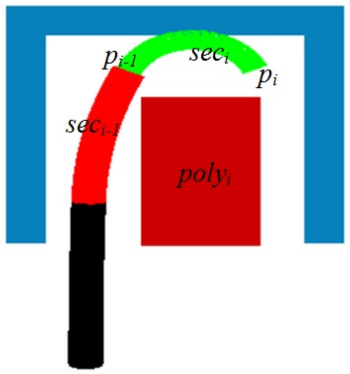
Algorithm 11: $RepairConfiguration$

```
input  : infeasible configuration $C$, the set of infeasible arm sections $S$, and
         models of colliding objects
output: a feasible arm configuration $C_{new}$ or "$repair\ fails$"
repeat
    $sec_i \leftarrow$ section of smallest index in $S$;
    $count \leftarrow 0$;
    while $sec_i$ is infeasible do
        if $count > max\_tries$ then
            return "$repair\ fails$";
        end
        if colliding with an obstacle then
            update $C$ by curling and shrinking $sec_i$ and $sec_{i-1}$ with $w_i\delta_c$ and
            $\Omega_{i-1}\delta_c$ respectively
        end
        if colliding with the target object then
            update $C$ by flattening and extending $sec_i$ and $sec_{i-1}$ with $\Omega_i\delta_c$ and
            $\Omega_{i-1}\delta_c$ respectively
        end
        if $sec_i$ collides with itself or another arm section then
            increase $|\phi_i|$ by a small $\delta\phi$;
        end
        if $C$ is out of the physical limits of the arm then
            return "$repair\ fails$";
        end
        $count \leftarrow count + 1$;
    end
    update $S$ by deleting $sec_i$ and adding new infeasible arm sections (if any) at
    configuration $C$;
until $S = \varnothing$;
$C_{new} \leftarrow C$;
return $C_{new}$.
```

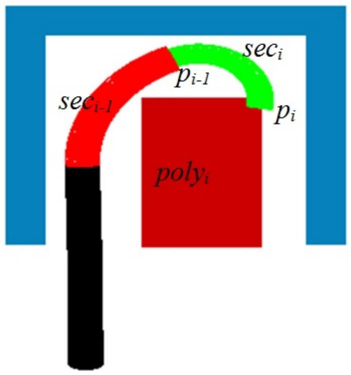
As shown in Fig. 43 (a) and (b), if an arm section sec_i collides with the surrounding obstacles, our algorithm shrinks and curls the arm section to avoid collisions; curling is to increase the curvature of sec_i and shrinking is to decrease the section length of sec_i. On the other hand, if an arm section sec_i collides with the target object, our algorithm can flatten and extend the arm section to avoid collisions as shown in Fig. 43 (c) and (d).



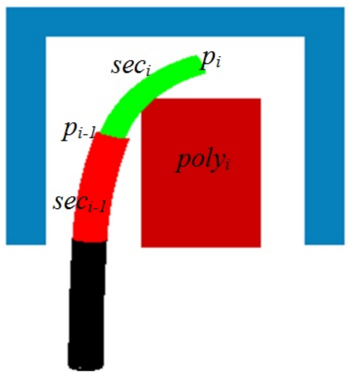
(a) sec_i (green) collides with obstacle at sec_i's tip p_i



(b) sec_i (green) collides with obstacle between p_{i-1} and p_i



(c) sec_i (green) penetrates with object at sec_i's tip p_i



(d) sec_i (green) penetrates object between p_{i-1} and p_i

Figure 43: Illustration of four collision cases between an arm section and the object or the obstacles.

Depending on where a collision happens at sec_i, our algorithm also considers sec_{i-1}. As shown in 43 (a) and (c), if a collision happens close to the sec_i's tip, adjusting sec_i alone would be sufficient to repair the collision. However, if the collision occurs close to sec_i's base, repairing should also involve adjusting the neighboring arm section sec_{i-1}, by either curling, flattening, shrinking or extending.

We assign two normalized weights Ω_i and Ω_{i-1} for the amounts of adjustments of sec_i and sec_{i-1} respectively:

$$\Omega_i = l_i/s_i, \quad \Omega_{i-1} = 1.0 - \Omega_i,$$

where l_i is length from the section base p_{i-1} to the first point on sec_i's central curve that is either in collision or closest to where collision occurs, and s_i is the current section length of sec_i from its base to tip. The longer l_i is, the closer the collision is to the section tip, and vise versa.

Denote δ_c as the amount of configuration adjustment of each section at each step, then if sec_i is collided, the weighted configuration adjustments of sec_i and sec_{i-1} for each step are $\Omega_i\delta_c$ and $\Omega_{i-1}\delta_c$ respectively.

If sec_i collides with itself or another arm section, a loop is formed by the arm section(s). Our algorithm increases the orientation angle ϕ_i of sec_i to break the loop and form a spiral wrap around the object.

Note that Algorithm 11 repairs from the infeasible section of the smallest index, i.e., the infeasible section closest to the base of the manipulator. Let sec_i be such a section. After sec_i is repaired, the other infeasible sections after sec_i (closer to sec_n) may also be repaired. If not, or if a section newly becomes infeasible due to the repair of sec_i, Algorithm 11 continues to repair infeasible sections from the section of the smallest index, and so on. On the other hand, if sec_i cannot be repaired after several tries or has reached its limits, the algorithm returns "$repairfails$", since repairing another infeasible section after sec_i will not change the pose of sec_i and thus will not make sec_i feasible.

For an n-section continuum manipulator, the complexity (in the worse case) of repairing an arm configuration in Algorithm 11 is $O(nmax)$, where max represents the maximum number of attempts allowed for repairing one section. However, our experimental results show that the average number of tries for each arm section is less than 4 to find a feasible configuration and the average time of finding it is less than 30 ms for all strategies, see Table 10. This shows that the heuristics used in Algorithm 11 are very effective. Note that if a feasible solution exists, our algorithm can usually find it by gradually following the object contour. Note also that a feasible solution exists for progressive wrapping if the object size is proper and there is sufficient space around it.

6.4 Simulation Tests and Real Robot Experiments

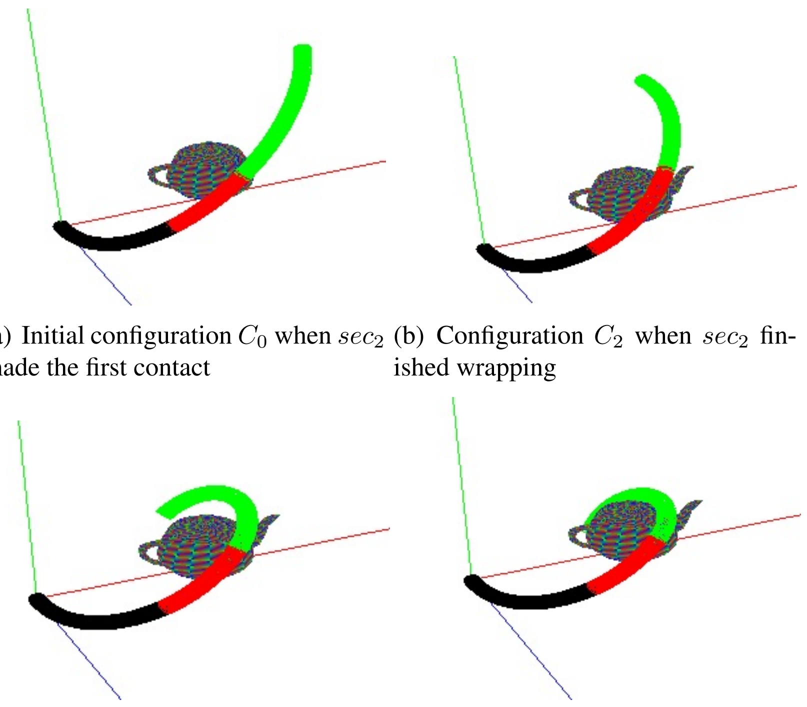
All the algorithms in both open and cluttered environment are implemented on a 2.40GHz Intel(R) Xeon(R) CPU with 4.00 GB RAM and tested in several grasping examples with a continuum manipulator. Both simulation tests and real robot experiments are conducted.

6.4.1 Simulations of Progressive Grasping in an Open Environment

Algorithm 7 was tested with different grasping examples in an open environment with three-section and four-section continuum manipulators. Fig. 44 shows a path of three-section arm configurations leading to a force-closure grasp found by our method. Fig. 45 shows a path of four-section arm configurations leading to a force-closure spiral grasp found by our method.

Note that from the configuration C_3 in Fig. 45(c), directly curling sec_4 will result a collision between sec_4 and sec_2; therefore, Algorithm 8 increased the angle ϕ_4 between

sec_4 and sec_3 and then curled sec_4 to create the stable, spiral grasp shown in Fig. 45(d).



(a) Initial configuration C_0 when sec_2 made the first contact

(b) Configuration C_2 when sec_2 finished wrapping

(c) Configuration after adjusting sec_2's length to make sec_3 wrap tighter

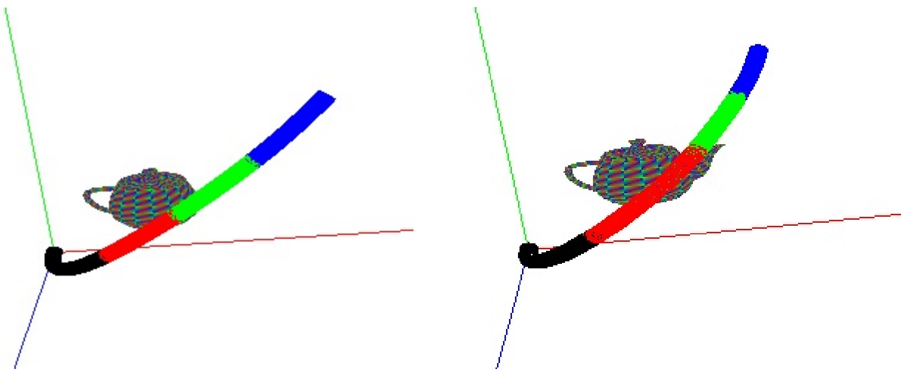
(d) Configuration C_3 when all sections finished wrapping

Figure 44: A path of configurations leading to a stable grasp by a three-section manipulator.

Table 18 shows the time costs of finding the paths leading to the force-closure grasps of Fig. 44 and Fig. 45 respectively. In each case, configurations C_i, $i > 0$, are the arm configurations after sec_i of the arm is finished curling/wrapping around the object, and these configurations are in the $path$ returned by Algorithm 7. As shown in the table, the time for finding a path to form a force-closure grasp is 1–2 seconds, which is fast enough for real-time arm operations.

6.4.2 Simulations of Progressive Grasping in a Cluttered Environment

Algorithm 9 was also implemented and tested on a four-section continuum manipulator. We first tested and compared the three alternative strategies for choosing target vertices


(a) Initial configuration C_0 when sec_2 made the first contact (b) Configuration C_2 when sec_2 finished wrapping

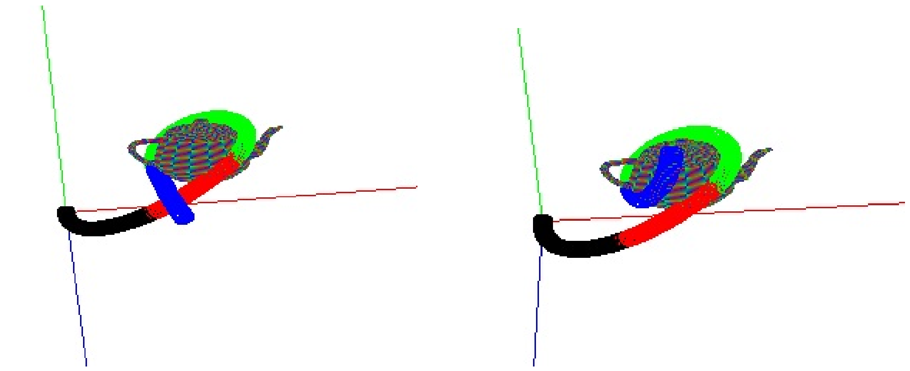

(c) Configuration C_3 when sec_3 finished wrapping (d) Configuration C_4 when all sections finished wrapping

Figure 45: A path of configurations leading to a stable grasp by a four-section manipulator.

used in Algorithm 10, and then we also tested our algorithms in different environments with different target objects and obstacles.

6.4.2.1 Comparing three alternative strategies

We tested the three alternative strategies for choosing target vertices in Algorithm 10 for a task of grasping a teapot in a tight space, as shown in Fig. 41, and compared the results, as shown in Table 10. The following parameters of performance were considered:

- # knot configurations : number of feasible knot configurations generated for a path;
- Planning time per knot: average time required for generating one knot configuration in a path;

Table 9: Time cost (ms) for finding the paths leading to the force-closure grasps in Fig. 44 and Fig. 45 respectively

computing time	config. C_0	config. C_2	config. C_3	config. C_4	force closure check
Fig. 44:	50	430	390	—	233
Fig. 45:	42	320	540	610	306

- Total planning time: the total time needed for generating all knot configurations in a path;
- Avg. # collision detection: the average number of collision checks performed in generating a feasible knot configuration;
- Avg. # tries per section: the average number of attempts made for repairing an infeasible arm section;
- Total path length: the sum of distances (in work space) between adjacent knot configurations in a path.

Table 10: Comparison of simulation results from using the three alternative strategies in Algorithm 10

Strategies	# knot config.	Plan. time / knot (ms)	Total plan. time (s)	Avg.# col. checks	Avg. # tries / section	Path length (cm)
Strategy (1)	43	24.31	1.04	13	3.2	302.41
Strategy (2)	195	10.58	2.06	5	1.4	269.62
Strategy (3)	141	13.61	1.91	7	1.6	278.49

How to compute the distance between two configurations is an important issue in obtaining the length of a path. We have found that using the straight-line distance between two configurations in the configuration space of the manipulator is not a good metric. This is because a shorter straight-line distance in the configuration space does not correspond to closer distance between volumes of arm points corresponding to the two poses in the physical (i.e., Cartesian) space. Thus, for two arm configurations C_j and C_{j+1} on a path, we compute the distance between them using the following metric:

$$d(C_j, C_{j+1}) = \Sigma_{i=1}^{n} |p_i^{j+1} - p_i^j|$$

where p_i is the tip point of section i for an n-section continuum manipulator.

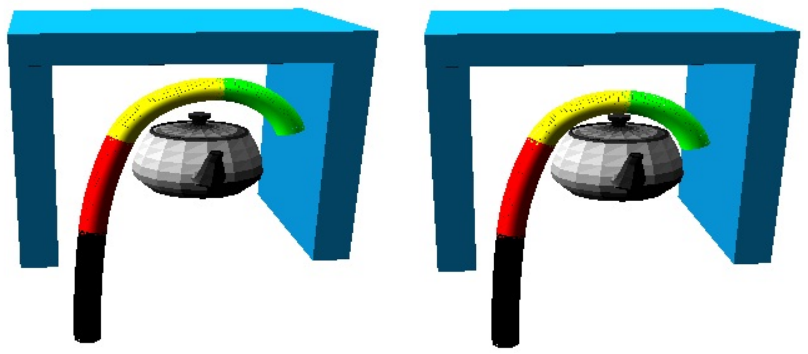
Now the length of a path with N knot configurations can be computed as:

$$l(path) = \Sigma_{j=1}^{N} d(C_j, C_{j+1})$$

The above measure for path length provides a more intuitive characterization consistent with the arm pose changes in the Cartesian space.

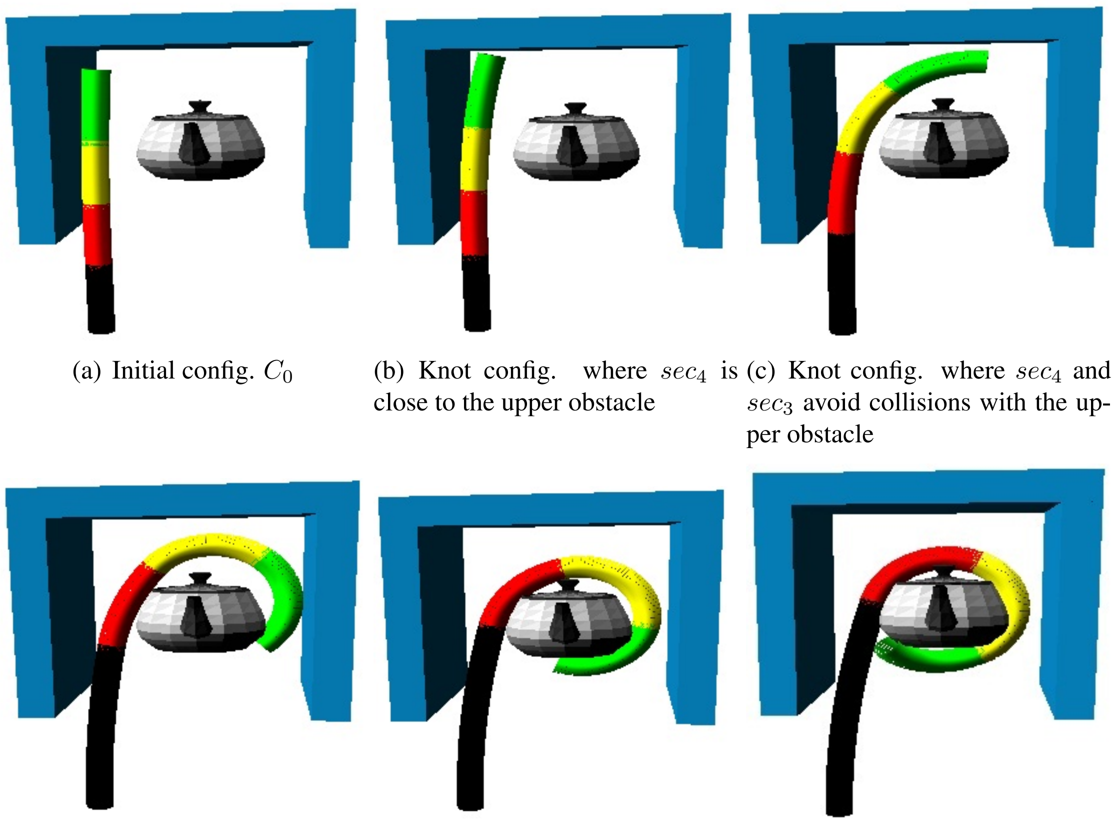
As shown in Table 10, strategy (1) generates the longest path, while strategy (2) generates the shortest one. The reason is that strategy (2) generates paths that are closer to the contour of the object, see Fig. 46. The length of the path generated by strategy (3) is between those of strategies (1) and (2). Also note that strategy (1) generates the fewest number of knot configurations for a path and requires the least amount of total time to generate a path. However, it takes the most amount of time for generating one knot configuration on average. This is because strategy (1) takes more time to search for the furthest reachable vertex of the object by the tip of the arm and the corresponding feasible configuration, which also involves more collision checks.

Fig. 47 shows a path generated by our algorithm using strategy (3), leading the arm to a stable, force-closure grasp while avoiding collision with the ”U” shaped structure and penetration into the object.



(a) Configuration by strategy (1) (b) Configuration by strategy (2)

Figure 46: Two example configurations generated by strategies (1) and (2) respectively, where the configuration generated by strategy (2) is closer to the target object.



(a) Initial config. C_0 (b) Knot config. where sec_4 is close to the upper obstacle (c) Knot config. where sec_4 and sec_3 avoid collisions with the upper obstacle

(d) Knot config. where sec_4 contacts the teapot (e) Knot config. where sec_4 compliantly wraps around the teapot (f) Final force-closure grasping config. where sec_4 is twisted to avoid collision with sec_1

Figure 47: Snapshots of knot configurations leading to a force-closure grasp by a four-section continuum manipulator while avoiding obstacles.

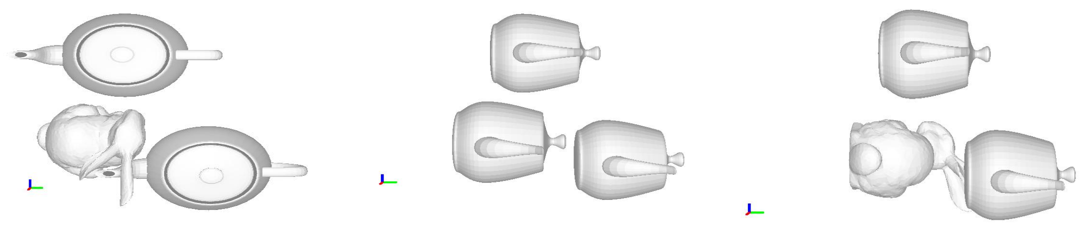
6.4.2.2 Grasping in Different Environment Settings

We also tested our algorithms for grasping tasks in different cluttered environments, as shown in Fig. 48, to demonstrate the effectiveness of our algorithms. Note that the environments in Fig. 48(a) and Fig. 48(c) show different arrangements of the target object (bunny) for grasping and different arrangements of the obstacles (two teapots). We use those different arrangements to show that our algorithms work robustly in dealing with different shapes/ways of grasping the target object and different ways of arm interaction with obstacles.

Snapshots of the three different grasping tasks of Fig. 48 are shown in Fig. 49, 50 and 51, respectively. The corresponding performance parameter values for each task are shown in Table 11. Note that to avoid any bias on the strategy for choosing target vertices in Algorithm 10, we used strategy (3), which randomly chooses target vertices, for all those different grasping tasks. Also, for different environments, we used different arm initial configurations to begin the grasp.

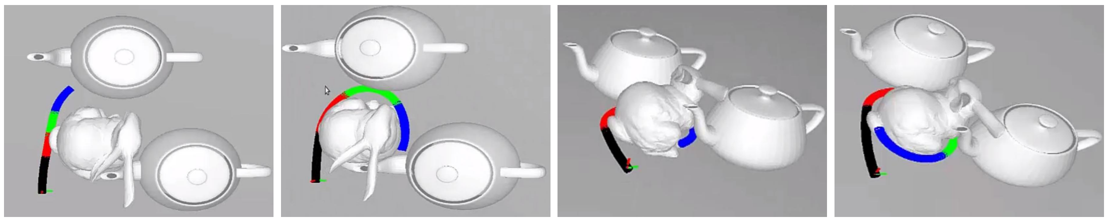
Note that the final grasping configurations are different in different environment settings, since the grasping configuration of a target object depends on the path leading to it, which is affected by the initial configuration of the arm, the strategy of choosing target vertices, physical limits of the arm (i.e., length, curvature, orientation), geometry of the target object surface, and also surrounding obstacles.

As our algorithm assumes known polygonal meshes of the target object and surrounding obstacles, it can compute each grasping path off-line. However, the simulation results show that our algorithm can compute an entire grasping path in just a few seconds, and thus, it can also be applied online.



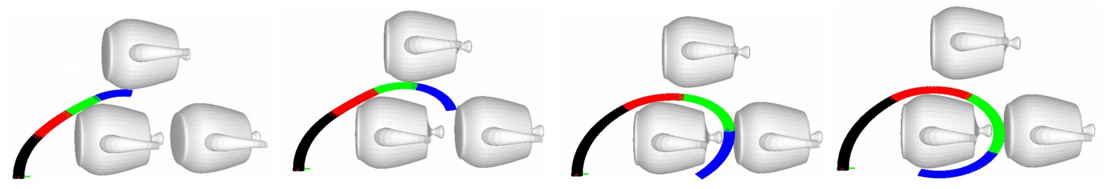
(a) Environment (1): the bunny is the target object for grasping
(b) Environment (2): the lower-left teapot is the target object for grasping
(c) Environment (3): the bunny is the target object for grasping

Figure 48: Three different cluttered environments for testing, with the target object surrounded by other objects as obstacles in each case



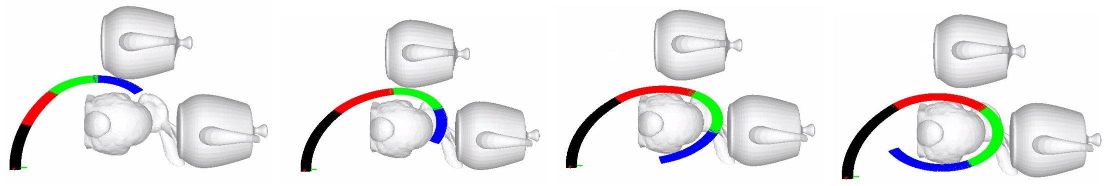
(a) Initial config.
(b) A knot config.
(c) A knot config.
(d) Grasping config.

Figure 49: Snapshots of knot configurations of a 4-section arm grasping a bunny in Environment (1) shown in Fig. 48(a)



(a) Initial config.
(b) A knot config.
(c) A knot config.
(d) Grasping config.

Figure 50: Snapshots of knot configurations of a 4-section arm grasping a teapot in Environment (2) shown in Fig. 48(b)



(a) Initial config.
(b) A knot config.
(c) A knot config.
(d) Grasping config.

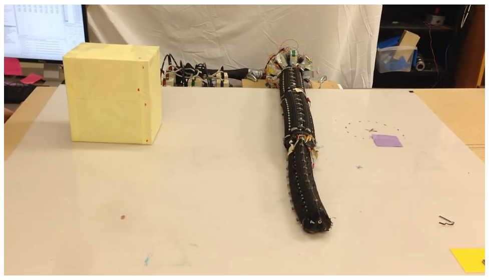
Figure 51: Snapshots of knot configurations of a 4-section arm grasping a bunny in Environment (3) shown in Fig. 48(c)

Table 11: Performance of grasping tasks in different environments in Fig. 48

Env.	# knot config.	Plan. time per config	Total plan. time	Avg.# col. checks	Avg. # tries per section	Path length
Env. (1)	109	27.42(ms)	2.98(s)	19	4.70	284.41(cm)
Env. (2)	118	28.34(ms)	2.99(s)	21	5.28	263.11(cm)
Env. (3)	96	26.12(ms)	2.49(s)	18	4.75	273.12(cm)

6.4.3 Real Robot Experiments for Progressive Grasping

We have also tested our algorithms using the 3-section continuum manipulator OctArm. We have conducted experiments of grasping in both an open environment and a cluttered environment. Fig. 52 (a) shows a yellow target box for grasping in an open space, and Fig. 52 (b) shows the environment with two obstacles placed around the target object.


(a) An environment without any obstacle; the manipulator tries to grasp the yellow target box.

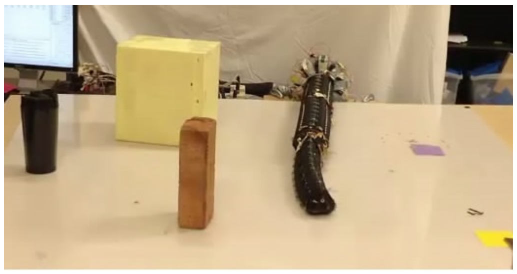

(b) A more cluttered environment with two obstacles added: a black coffee mug and a red brick.

Figure 52: Two environments for real experiments.

The object and obstacles are detected by an overhead Microsoft Kinect sensor - see Fig. 53 for the sensor setup, and their poses and dimensions are estimated. Fig. 54 shows an image captured by the Kinect camera and the segmented target object, the robot, and the obstacles. Table 12 presents the estimated center positions and heights of the object/obstacle on the table top.

For the open environment shown in Fig. 52(a), the Algorithm 7 can make the arm grasp

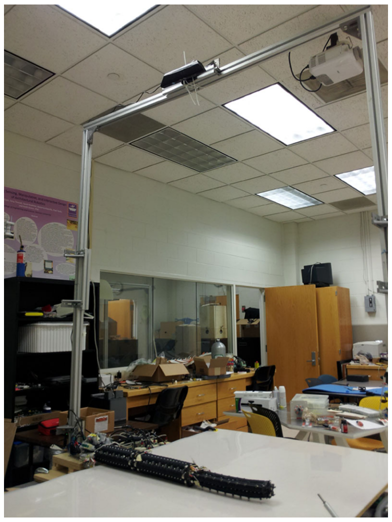
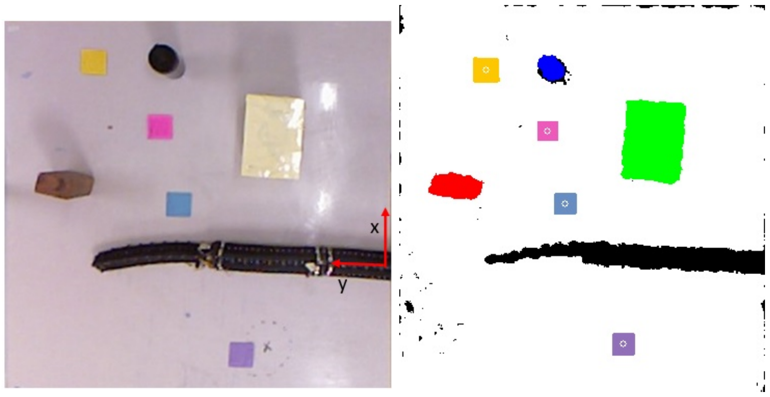


Figure 53: Experimental set up with a table top OctArm, overseen by an overhead Microsoft Kinect.



(a) An image captured by the overhead sensor.

(b) An image of segmented object, obstacles, and robot.

Figure 54: An image captured by the overhead sensor (left) and detected object and obstacles (right). Note that the color stickers on the table are used to calibrate the camera and to facilitate pose estimation of the object and obstacles.

Table 12: Estimated object/obstacles positions in Fig. 54

Object/Obstacles	Est. center position (x, y) (cm)	Height (cm)
Target box	$(37.5, 35.4)$	23.3
Brick	$(20.9, 96.7)$	19.2
Coffee mug	$(90.2, 60.5)$	18.1

the object successfully, see Fig. 55 for some snapshots and Table 13 for the time cost. But it causes the manipulator to hit obstacles in the more cluttered environment of Fig. 52 (b)

Table 13: Time cost of Algorithm 7 for the grasping task of Fig. 52 (a)

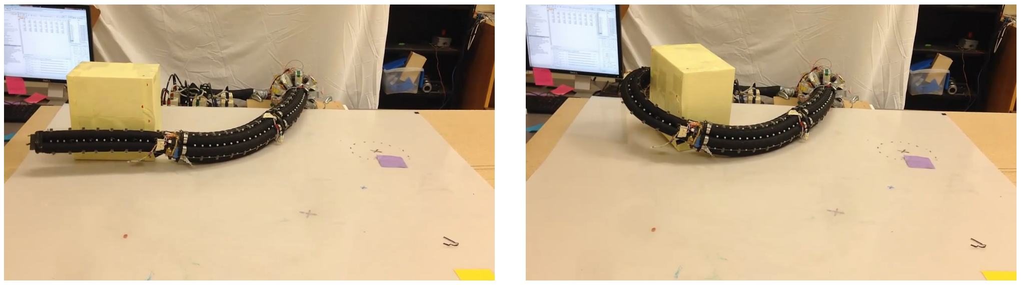
Obj. detect. time	# config. planned	Total plan. time	Avg. plan. time / config.	Avg.# colli. checks	Execut. time
$0.18(s)$	94	$1.02(s)$	$10.8(ms)$	5	$15(s)$

Table 14: Time cost of Algorithm 9 for the grasping task of Fig. 52 (b)

Obj. detect. time	# config. planned	Total plan. time	Avg. plan. time / config.	Avg.# colli. checks	Execut. time
$0.18(s)$	157	$2.1(s)$	$13.37(ms)$	7	$25(s)$

(see Fig. 56).

Next, Algorithm 9 of this paper is applied to the environment of Fig. 52 (b), with strategy (3) used (whose performance is between strategy (1) and strategy (2)); the manipulator grasps the object successfully while avoiding the surrounding obstacles by adjusting the length and curvature of every arm section and then reaches a force closure grasping configuration. Fig. 57 shows a sequence of snapshots, and Table 14 shows the time cost for this experiment. Note that the average planning time for finding a feasible arm configuration is only about 13 ms.



(a) The manipulator made the first contact with the object.

(b) The manipulator successfully grasped the target box.

Figure 55: The manipulator tries to grasp the target box in an open environment.

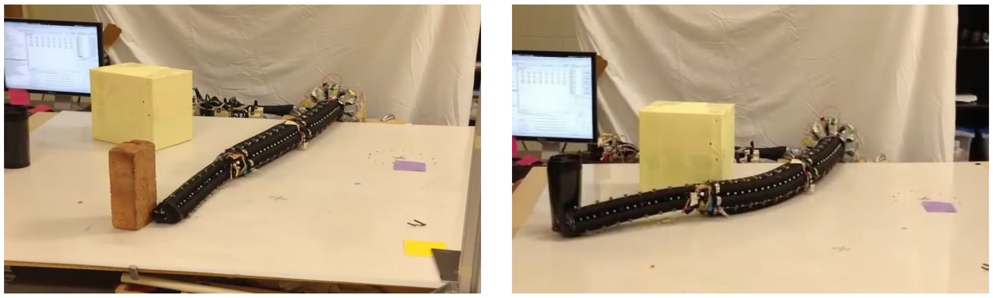

(a) The manipulator collides with the brick. (b) The manipulator collides with the coffee mug.

Figure 56: The method in [55] can not make the manipulator avoid the obstacles.

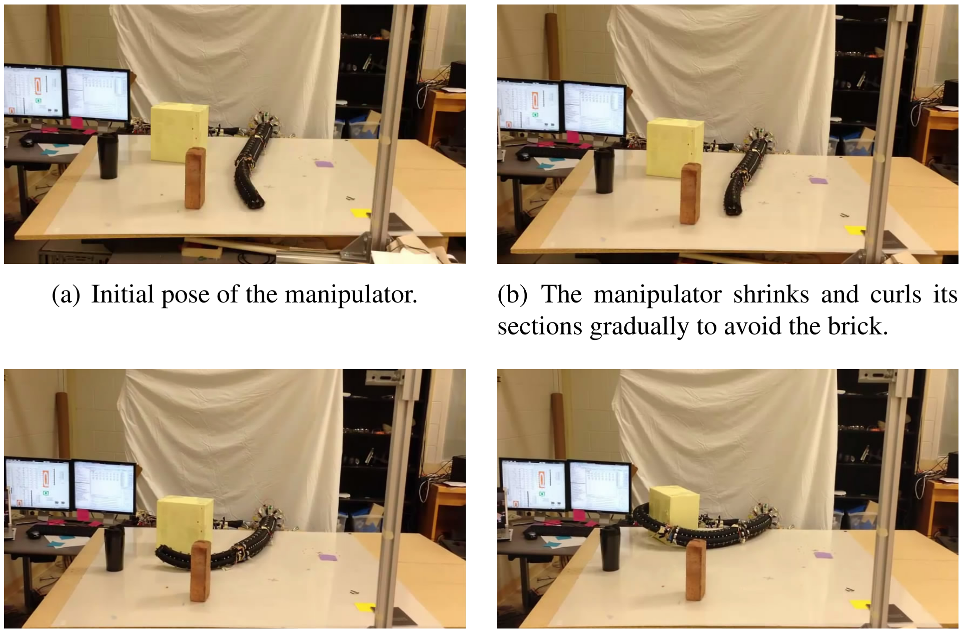

(a) Initial pose of the manipulator. (b) The manipulator shrinks and curls its sections gradually to avoid the brick. (c) The manipulator passes the brick and continues to grasp the object without hitting the coffee mug. (d) The manipulator successfully grasps the target object.

Figure 57: The method introduced in this paper successfully generates a feasible path of the continuum manipulator leading to a stable grasping configuration while avoiding obstacles in a cluttered environment in real time.

6.5 Remarks

This chapter introduced two progressive grasping approaches for a general n-section continuum manipulator to achieve force-closure grasps of a target object in both open and cluttered environments. The motion of each arm section is achieved by changing the curvature, length, and orientation angle of each arm section around the object. The result is a helix shaped arm configuration around the target object tightly. Both algorithms have time complexities linear to the number of arm sections, and the effectiveness and efficiency of them are validated in both simulations and real experiments. They can also be extended to enable grasping an object not fully visible initially by gradually extending the manipulator to explore the surface of the object, if the manipulator is equipped with sensors at its tip.

CHAPTER 7: TASK CONSTRAINED CONTINUUM MANIPULATION IN CLUTTERED ENVIRONMENTS

7.1 Overview

A continuum manipulator has flexible arm segments which are suitable for carrying out tasks in cluttered environments. For example, it can bend and curl its arm to reach the target behind an obstacle without collision, Fig. 58 shows one example of a cluttered pipe environment: certain pipes behind can be reached by a continuum manipulator for inspection but are difficult to reach by an articulated manipulator. Thus, it would be useful for a continuum manipulator to be able to carry a certain device at its tip and perform task-related motion, which usually requires the tip of the manipulator to follow a certain path or a trajectory in the work space.

In this chapter, we will first introduce the method [56, 59] for task constrained manipulation in a cluttered environment that only constrain the arm tip's position, and then introduce a more generalized task constrained manipulation approach [58] that also considers the arm tip's orientation.

7.2 Task Constrained Manipulation Constraining Arm Tip Position

In this section task constraints in the Cartesian space are defined in terms of restrictions on the position of the tip of a continuum manipulator required by a task. Specifically, we focus on the case that the tip of the manipulator has to follow a curve in the Cartesian space, i.e., the task constraint is defined by a *task curve*. A task curve could be viewed as either

Figure 58: A cluttered pipe environment (courtesy of EPRI).

describing the goal of a task or ending at the goal configuration of a task. Here we assume the task curve is either given (for simple tasks) or obtained through motion planning of the tip point [10, 37].

A main challenge of the task constrained manipulation using a continuum manipulator is the high redundancy of the continuum manipulator. Even though the configuration space of a continuum arm is constrained by the task curve that the tip of the arm has to follow, it is still huge because of the high dimensionality, e.g., 9-D for the OctArm, which consists of three arm sections. Direct search of such a space, even by random sampling, would be very inefficient. Therefore, our strategy is to constrain the arm shape to search from a subspace and gradually relax the subspace as needed to find feasible arm configurations.

7.2.1 Constrained Arm Shape

We denote a *super-section* $Ssec_{1,i}$ formed by sections from sec_1 to sec_i when the central axes of these connected sections share the same curvature and on the same plane (i.e., $\phi_j = 0, 1 < j \leq i$), and therefore on the same circle, see Fig. 59 for illustration. The super-section $Ssec_{1,i}$ now behaves like a large, single section with three degrees of freedom: curvature κ_1, rotation angle ϕ_1, and the sum $s_{1,i}$ of section lengths from sec_1 to sec_i. In the rest of the paper, we use *arm segment* i to denote either the arm section sec_i or the super-section $Ssec_{1,i}$.

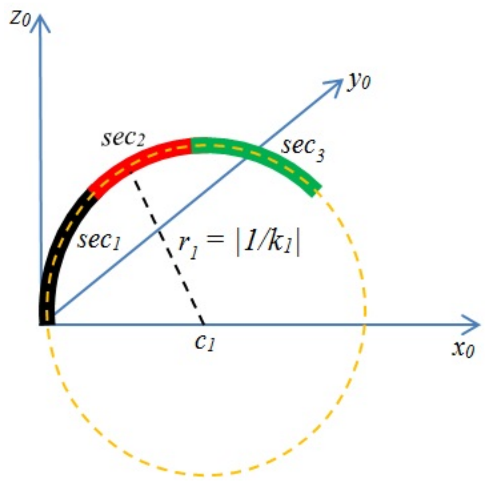


Figure 59: A super-section $Ssec_{1,3}$ consisting of sec_1 (black), sec_2 (red) and sec_3 (green) with their central axes on the x_0-z_0 plane

In general, the *arm configuration* of an n-section continuum manipulator is determined by the control variables of each section, denoted as $C = \{(s_1, \kappa_1, \phi_1), ..., (s_n, \kappa_n, \phi_n)\}$. We further denote the *partial arm configuration* for sections from sec_k to sec_i ($1 \leq k \leq i$) as $C_{k,i}$. Note that $C_{i,i}$ indicates the configuration of a single section sec_i; whereas, $C_{1,n}$ also indicates an (entire) arm configuration.

An arm (or partial arm) configuration is called a *feasible* configuration if it does not make the manipulator collide with obstacles and also satisfies the task constraint for the task.

7.2.2 Planning for Task-constrained Manipulation

Given a task curve, which can be described in terms of a discrete sequence of points $G = \{g_1, g_2, ..., g_m\}$ in the Cartesian space that the tip of a continuum manipulator must follow, the problem of planning for task-constrained manipulation is to find a feasible path of arm configurations for the tip to follow the task curve while the arm avoiding obstacles efficiently. This problem can be divided into two intertwined subproblems: search of arm shapes in the configuration space of the manipulator and search of base positions for all arm sections in the Cartesian space, so that a corresponding arm configuration can be computed by [66] given the base positions for arm sections. In the following, we address these subproblems and present our approach for planning.

7.2.2.1 Search of Arm Shapes

Beginning from the most constrained arm shape, i.e., all arm sections form one super-section, our strategy searches from a subspace and gradually relax the subspace as needed to find feasible arm configurations. This core idea is presented in a recursive function Algorithm 13, which is called by the main algorithm Algorithm 12 for our approach.

Algorithm 12 starts by considering the shape of the arm in the simplest form as one super-section $Ssec_{1,n}$, whose tip needs to reach the first point g_1 on the task curve, and calls Algorithm 13 to get a feasible arm configuration. Algorithm 13 first computes the corresponding configuration of the super-section by inverse kinematics [66]. If no feasible arm configuration exists (i.e., either no inverse solution for the tip at the target point or the solution is not free of obstacles), our algorithm relaxes the shape of the arm by breaking it into a shorter super-section $Ssec_{1,n-1}$ and the last arm section sec_n, whose tip must reach the point g_1 on the task curve. This requires deciding the target position q for the base of sec_n, i.e., also the tip of $Ssec_{1,n-1}$, to find a feasible configuration for sec_n, and the corresponding partial arm configuration $C_{1,n-1}$ when the tip of $Ssec_{1,n-1}$ reaches q.

If there is no feasible configuration for super section $Ssec_{1,n-1}$, Algorithm 13 further breaks down $Ssec_{1,n-1}$ into a shorter super-section $Ssec_{1,n-2}$ and arm section sec_{n-1}, and so on. So in general, the considered arm shape can be in terms of a super section $Ssec_{1,j}$ and arm sections sec_{j+1}, ..., sec_n, $1 \leq j < n$. In the most relaxed case, where $j = 1$, the arm shape consists of no super section and just n arm sections.

For the arm tip to reach the next point g_t, $t = 2, ..., m$, on the task curve, Algorithm 12 calls Algorithm 13 to decide the next arm configuration C_t based on the current feasible arm configuration C_{t-1} where the arm tip is at g_{t-1}. Algorithm 13 first tries to decide the configuration of the last arm segment, which can be either section sec_n or super-section $Ssec_{1,n}$, to reach the new tip position g_t without changing the segment's base position via inverse kinematics [66]. If the solution for the segment exists and is feasible, then the resulting arm configuration C_t is feasible, because the rest of the arm below the last segment is intact as part of the feasible configuration C_{t-1}. Otherwise, the base position of the last segment is changed in order to find a feasible configuration.

Our strategy is very efficient for several reasons. For collision checking between the arm and the obstacles (which may also include the target object), we use the collision detection algorithm [57] that treats a super-section just as a regular arm section as a primitive unit for efficient check. The fewer sections (including a super-section) in an arm, the less time for collision checking of an arm configuration. Also, the fewer sections, the fewer target points connecting sections need to be searched. Moreover, a super-section can be considered just as a regular section for inverse kinematics [66]. Thus, the fewer sections in an arm, the less time for solving inverse kinematics. Finally, for the arm tip to move from point g_{t-1} to point g_t on the task curve, the new arm configuration C_t can often re-use the previous arm configuration C_{t-1} for some arm sections without repeatedly solving for inverse kinematics and checking for collisions.

```
Algorithm 12: $GenPath$
input  : Task curve $G = \{g_1, g_2, ..., g_m\}$, arm model and an initial config. $C_0$
output: A $path$ of arm configuration for the tip to follow $G$ or report failure
begin
    $path \leftarrow$ null;
    $C_{current} \leftarrow C_0$, $flag \leftarrow$ "super-section";
    for $t = 1$ to $m$ do
        $p_n \leftarrow g_t$;
        if $NewArmConfig(C_{current}, p_n, n, flag)$ returns $C_{1,n}$ then
            $C_{current} \leftarrow C_{1,n}$;
            $path \leftarrow path \cup \{C_{current}\}$;
        else
            return "No path is found";
        end
    end
    return $path$;
end
```

7.2.2.2 Search of Base Position for An Arm Segment

As described earlier, if a considered arm shape is in terms of a super-section $Ssec_{1,j}$ and arm sections sec_{j+1}, ..., sec_n, $1 \leq j < n$, the base positions of sections sec_{j+1}, ..., sec_n need to be determined given that the tip of sec_n must reach a target point g_t on the task curve, so that a feasible arm configuration can be computed via inverse kinematics [66]. If the base of the manipulator is not fixed, then the base position for $Ssec_{1,j}$ also needs to be determined.

Algorithm 13 generates a new base position q_{new} for an arm segment according to its current base position q and its required tip position p_i by searching in a neighborhood Q of q in the Cartesian space. The scope of the neighborhood can be determined by the range of length of the segment such that from any point in Q, p can be reached. Q can also be determined based on task-specific information. Depending on the shape and size of Q,

```
Algorithm 13: $NewArmConfig(C, p_i, i, flag)$
input  : Arm configuration $C$, new tip position $p_i$ for segment $i$, $flag$ to indicate if
         segment $i$ is a “super-section”
output: a feasible new partial arm configuration $C_{1,i}$ or “no feasible solution”
begin
    if segment $i$ can reach $p_i$ from its current base in a feasible configuration $C_{i,i}$
    then
        return the new $C_{1,i}$ from $sec_1$ to $sec_i$, where configurations of sections
        with indices lower than segment $i$ remain the same as in $C$;
    end
    if $flag$ = “super-section” then
        break segment $i$ into $Ssec_{1,i-1}$ and $sec_i$;
        $flag \leftarrow$ “section”;
        return $NewArmConfig(C, p_i, i, flag)$;
    end
    if $i = 1$ with fixed base then
        return “no feasible solution”;
    end
    $\#tries \leftarrow 0$;
    while $\#tries < max$ do
        generate $q_{new}$ as a new base point for $sec_i$;
        if segment $i$-1 is a super-section then
            $flag \leftarrow$ “super-section”;
        else
            $flag \leftarrow$ “section”;
        end
        if $NewArmConfig(C, q_{new}, i-1, flag)$ returns $C_{1,i-1}$ then
            if $sec_i$ can reach $p_i$ from $q_{new}$ in a feasible configuration $C_{i,i}$ then
                return new arm configuration $C_{1,i} \leftarrow C_{1,i-1} \cup C_{i,i}$ ;
            end
        end
        $\#tries = \#tries + 1$;
    end
    return “no feasible solution”
end
```

either random sampling or systematic sampling of Q can be used to find q_{new}.

The found q_{new} not only ensures that sec_i's configuration $C_{i,i}$ is *feasible*, but also that a corresponding partial arm configuration $C_{1,i-1}$ with its tip at q_{new} is *feasible*. This means that, in the worst case, base positions of sec_1 to sec_{i-1} will also be changed by the recursive

Algorithm 13, and the number of base positions searched per segment is limited by the threshold max. Even though the worst-case time complexity for base point search is max^i, the average time in practice is much, much less because (a) a tip position of a segment (with 3-DOFs) can usually be reached from many base positions in a sizable continuous region (or conversely, from a base position, many tip positions can be reached, forming the "dexterous workspace" of the arm segment), and (b) either the current q is already feasible or a feasible q_{new} can often be obtained by a small change of the current q. The experimental results in the following section for an example of inspection task confirms the efficiency of base point search.

7.2.3 Task of Inspection

In this section, we describe how to apply our strategy for task-constrained manipulation to the task of autonomously inspecting the surface of an object, such as a pipe, for possible defects in a cluttered environment (e.g., see Fig. 58).

7.2.3.1 Task Description

Assuming that an inspection device (e.g., a camera) is mounted on the tip of a continuum manipulator. The tip of the manipulator will have to cover the entire surface of an object following certain specific path, which defines task constraints. Without losing generality, we consider the tip scan the object surface line by line (or curve by curve if the surface is not flat), and each line/curve scan defines a task curve. For example, Fig. 60 shows a cylindrical object and N task curves for inspecting it; each task curve is a circle surrounding the object, and all task curves are parallel.

In general, we can describe a task curve with respect to the object frame o-xyz. Let z be

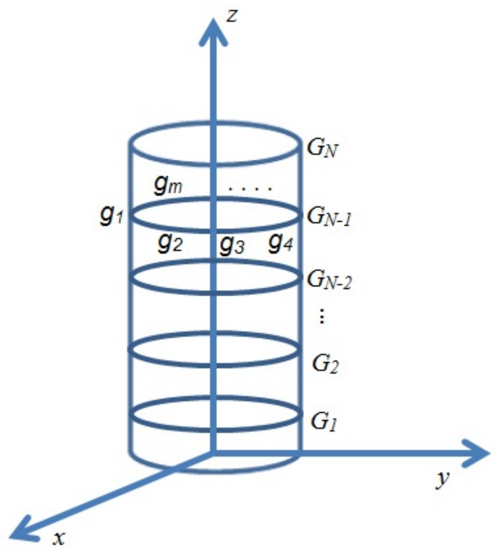


Figure 60: A cylindrical object and the task curves for inspection

the axis that task curves stack along (see, for example, Fig. 60). Each point on a task curve can be further described by cylindrical coordinates (d, θ, z). By using such a cylindrical coordinate system in the Cartesian space, the possible neighborhood Q for searching the base point of a segment can usually be considered as 2-D (instead of 3-D) with a more or less fixed d value (i.e., fixed distance to the z axis).

Consider sec_i. If its current base point q does not result in a feasible configuration $C_{i,i}$, in searching a better q_{new}, the θ and z values of q are adjusted in the direction of approaching its target tip point; if $C_{1,i-1}$ is not feasible, the θ and z values of q are adjusted in the direction of approaching the robot base point.

We define the direction of a scan (to complete a task curve) based on if the continuum manipulator extends or contracts itself. If the manipulator arm extends itself in a scan, the scan is called in *forward direction* and is a *forward scan*; Otherwise, if the manipulator arm contracts in a scan, the scan is called in *backward direction* and a *backward scan*.

7.2.3.2 Inspection algorithm

Algorithm 14 outlines the main algorithm for the inspection task. The manipulator alternates a *forward scan* with a *backward* scan to visit all *task curves* around the object. In this way, after the manipulator finishes a forward scan, with the arm largely extended, a small update of its tip along the z axis of the object frame is often sufficient to set it on course for the next scan in reverse direction (i.e., a backward scan).

In the first scan, our strategy is applied to lead the arm follow its first *task curve*, and all base points of arm sections are recorded along the way (for each arm shape consists of one or more arm sections in addition to the super-section). In a subsequent scan, the recorded base points of arm sections from the previous scan are re-used with small adjustments, as the algorithm calls Algorithm 13, taking advantage of the fact that the current task curve is more or less a simple shift of the previous task curve along the z axis.

Clearly, the time taken for generating a path of arm configurations for a subsequent scan is usually shorter than that for the first scan.

7.2.4 Implementation and Results

We have implemented all the algorithms on a 2.40GHz Intel(R) Xeon(R) CPU with 4.00 GB RAM and tested them on a three-section continuum manipulator with a fixed base for the task of inspecting a tilted cylindrical object in a cluttered space as shown in Fig. 61. Ten scans are defined by 10 task curves $\{G_1, G_2, ...G_{10}\}$, and each *task curve* is discretized into 500 target points for the tip of the arm to visit. We set $max = 6$ for $\#tries$ in Algorithm 13. Our algorithms return a feasible path of arm configurations for every task curve. Snapshots of one forward and backward scan, as the results of applying our inspection algorithm, are

```
Algorithm 14: $Inspection$
input  : A set of task curves $\{G_1, G_2, ...G_N\}$ for inspection, the initial arm
         configuration
output: The feasible path of arm configurations for inspection or report failure
1  begin
2      $Path \leftarrow null$;
3      $scan \leftarrow 1$;        // "1" indicates forward scan
4      call Algorithm 12 with task curve $G_1$;
5      if no feasible path of arm configurations is returned then
6          return "No solution";
7      end
8      record the feasible path $path_1$ and section base positions for every
       configuration on $path_1$;
9      $Path \leftarrow Path \cup path_1$;
10     for $i = 2$ to $N$ do
11         $scan \leftarrow 1 - scan$;     // change scan direction
12         $path_i \leftarrow path_{i-1}$;     // copy previous path
13         foreach point $g_t$ on $G_i$ along scan direction, $1 \leq t \leq m$ do
14             $p_n \leftarrow g_t$;
15             call Algorithm 13 to update configuration $C_t$ on $path_i$ with the new tip
               position $p_n$;
16             if no feasible configuration is returned then
17                 return "No solution";
18             end
19         end
20         record the updated $path_i$;
21         $Path \leftarrow Path \cup path_i$ ;
22     end
23     return $Path$
24 end
```

shown in Fig. 62.

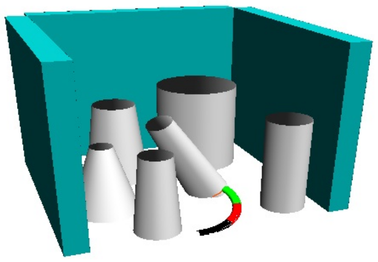
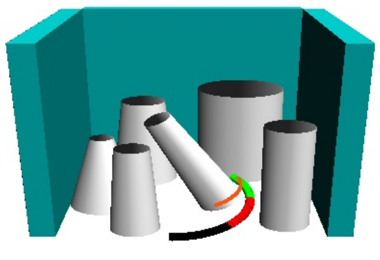
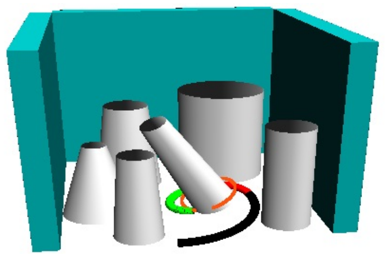
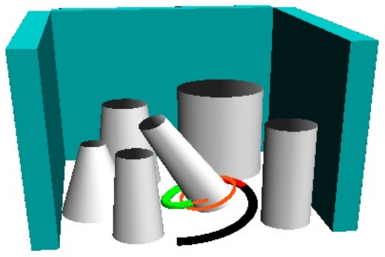


Figure 61: A target object (the middle oblique object) and obstacles in a cluttered environment


(a) Forward scan for G_1 with the arm shape of $Ssec_{1,3}$


(b) Forward scan for G_1 with the arm shape of sec_1, sec_2 and sec_3


(c) Backward scan for G_2 with the arm shape of sec_1, sec_2 and sec_3

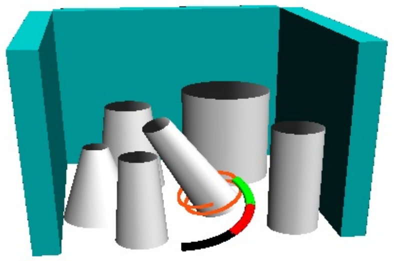

(d) Backward scan for G_2 with the arm shape of $Ssec_{1,3}$

Figure 62: Snapshots of a forward scan of *task curve* G_1 and a subsequent backward scan of *task curve* G_2, with the results shown as the red curves

In Table 15, we present the related costs of the two scans, with the forward scan as the first scan and the backward scan obtained by updating the results of the forward scan. As expected, the backward scan takes much less time than the forward scan (about half of the time of the forward scan) to find feasible configuration. The table also shows different time costs to find a feasible arm configuration under different arm shapes. For the most constrained and simplest shape $Ssec_{1,3}$, as expected, our algorithm used the least amount of time on average (2 ms in both forward and backward scans) to find a feasible configuration. Whereas, it took the most amount of time to find a feasible configuration on average for the most relaxed shape with three single sections sec_1, sec_2 and sec_3 (and no supersection) (16.0 ms and 8.0 ms in the forward and backward scans respectively). Note that the average $\#tries$ to search a suitable base position for each arm segment (i.e., a section or a super-section) is quite low. In the case the arm forms a single super-section, no search of base point is needed (due to the fixed base of the manipulator).

Table 15: Time cost of the scans illustrated in Fig. 62

scan	arm shape	# config.	avg. # tries	# collision checks	plan. time	avg. plan. time
G_1	$Ssec_{1,3}$	173	0	173	0.34 (s)	2.0 (ms)
	$Ssec_{1,2}$, sec_3	95	1.55	385	0.77 (s)	8.1 (ms)
	sec_1, sec_2, sec_3	232	2.43	1866	3.73 (s)	16.0 (ms)
G_2	$Ssec_{1,3}$	173	0	173	0.34 (s)	2.0 (ms)
	$Ssec_{1,2}$, sec_3	95	0.75	194	0.38 (s)	4.1 (ms)
	sec_1, sec_2, sec_3	232	1.07	930	1.86 (s)	8.0 (ms)

7.3 Manipulation With Arm Tip Position and/or Orientation Constrained

In this section, a more generalized task constraint are considered, which constrains on certain components of the position and/or orientation of the tip frame of a continuum manipulator during the motion of the (mobile) manipulator. One can view the task constraints as corresponding to a certain task manifold $\mathcal{M}$ in the 6-dimensional (6-D) space (position and orientation) of the tip frame.

As the manipulator moves, such constraints manifest to a continuous path for the tip frame, which can be discretized as a sequence of tip frame configurations in terms of homogeneous transformation matrices of the tip frame with respect to the world frame: $G = \{\mathbf{T}_{t,1}, \mathbf{T}_{t,2}, ...\mathbf{T}_{t,M}\}$. Depending on specific tasks, G can be either given directly by a task, such as an inspection task, or the result of path planning [48, 10, 37, 45] of the tip frame as a point in the manifold $\mathcal{M}$ in the 6-D configuration space of the tip frame. In either case, here we assume that G is known and focus on planning the corresponding motion of the (mobile) continuum manipulator.

While performing a task, the motion of the continuum manipulator is also constrained by the environment, i.e., the manipulator should be collision-free with respect to the obstacles in the environment. We call an arm configuration a *feasible* configuration, if the tip frame satisfies a configuration in G and the mobile continuum manipulator is also collision-free with respect to the obstacles in the environment. Obstacles are represented as polygonal meshes, which can be based on point clouds from sensing or known object models. Collision detection between a continuum manipulator arm and such obstacles can be efficiently done by the method introduced in [57].

7.3.1 Planning for Constrained Continuum Manipulation

The goal is to plan motion for an n-section (mobile) continuum manipulator to satisfy both task constraints and environment constraints. However, finding a *feasible* configuration satisfying both task and environment constraints for an n-section (mobile) continuum manipulator requires us to search in a $3n$ dimensional configuration space. One can sample directly in such a high dimensional configuration space, and check if a sample satisfy both task constraints and environment constraints, however such approach is inefficient for a high dimensional search space; even with more sophisticated sampling techniques [82, 10], only an approximated solution can be obtained from linear projection functions, and (further) gradient decent modification is required. Another way, as of the approach in Chapter 7.2, is searching in the workspace instead of in the arm's configuration space, and using workspace heuristics to help find feasible arm configurations, which also relies on inverse kinematics to build the connection between workspace and the arm's configuration space. However, existing inverse kinematic methods for continuum manipulators either do not constrain the arm tip's orientation [66] or do not address the redundancy of arm solutions [28] to find feasible arm solutions that satisfy both task constraints and environment constraints.

Our approach builds a mapping from certain key points on the manipulator in the workspace to the arm's configuration space analytically, then by (heuristically) searching the key points under task and environment constraints in the workspace, corresponding arm configurations are computed efficiently that satisfy both task and environment constraints.

Assume that initially the manipulator is at a collision-free configuration such that its tip position is close to the position of $\mathbf{T}_{t,1}$ of path G. Our approach is to obtain the next collision-free configuration of the entire mobile continuum manipulator with its tip frame at configuration $\mathbf{T}_{t,j}$, $1 \leq j \leq M$, taking advantage of spatial coherence with the previous feasible configuration of the manipulator. Algorithm 15 outlines the approach, which calls the strategy Algorithm 16 to obtain a feasible manipulator configuration.

```
Algorithm 15: $GeneralizedTaskConstrainedManipulation$
  input : Manipulator model, a path of tip frames $\{\mathbf{T}_{t,1}, \mathbf{T}_{t,2}, , ..., \mathbf{T}_{t,M}\}$, the initial
          arm configuration $C_0$ and base frame configuration $\mathbf{T}_{1,0}$
  output: A feasible path of arm and base configurations, or report failure
1 $path \leftarrow$ null;
2 for $j \leftarrow 1$ to $M$ do
3     if $GenConfig(C_{j-1}, \mathbf{T}_{1,j-1}, n, \mathbf{T}_{t,j})$ returns a feasible arm configuration $C_j$
      and base configuration $\mathbf{T}_{1,j}$ with the arm tip frame at $\mathbf{T}_{t,j}$ in $G$ then
4         $path \leftarrow path \cup \{(C_j, \mathbf{T}_{1,j})\}$;
5     else
6         return "no solution";
7     end
8 end
9 return $path$;
```

Algorithm 16 finds a feasible manipulator configuration by computing a feasible config-

Algorithm 16: $GenConfig(C_{j-1}, \mathbf{T}_{1,j-1}, i, \mathbf{T}_{i+1,j})$

```
input  : Manipulator model, previous manipulator config. $C_{j-1}$ and $\mathbf{T}_{1,j-1}$,
         current section index $i$, and $sec_i$ tip frame configuration $\mathbf{T}_{i+1,j}$
output: A feasible partial arm configuration $A_i$ from $sec_1$ to $sec_i$ and base
         configuration for step $j$, or returns $A_i$ is not collision-free; for $i = n$,
         $A_n = C_j$.
1  $\#tries \leftarrow 0$;
2  Partially specify $sec_i$'s base (either position or orientation);
3  Compute $sec_i$'s base location $\mathbf{T}_{i,j}$ and $(s_i, \kappa_i, \phi_i)$ by constrained inverse
   kinematics;
4  if $(s_i, \kappa_i, \phi_i)$ is collision-free and $i > 1$ then
5      $A_{i-1} \leftarrow GenConfig(C_{j-1}, \mathbf{T}_{1,j-1}, i-1, \mathbf{T}_{i,j})$;
6  end
7  if $(s_i, \kappa_i, \phi_i)$ is not collision-free or $A_{i-1}$ ($i > 1$) is not collision-free then
8      repeat
9          $\#tries \leftarrow \#tries + 1$;
10         if $\#tries > max$ then
11             return "$A_i$ is not collision-free";
12         end
13         modify $sec_i$ and its base to update $\mathbf{T}_{i,j}$ and $(s_i, \kappa_i, \phi_i)$;
14         if $i > 1$ and $(s_i, \kappa_i, \phi_i)$ is collision-free, update $A_{i-1}$ with
           $A_{i-1} \leftarrow GenConfig(C_{j-1}, \mathbf{T}_{1,j-1}, i-1, \mathbf{T}_{i,j})$;
15     until $(s_i, \kappa_i, \phi_i)$ is collision-free and $A_{i-1}$ ($i > 1$) is collision-free;
16 end
17 if $i = 1$ then
18     return $A_1 \leftarrow \{(s_1, \kappa_1, \phi_1)\}$ and $\mathbf{T}_{1,j}$;
19 else
20     return $A_i \leftarrow \{(s_i, \kappa_i, \phi_i)\} \cup A_{i-1}$ and $\mathbf{T}_{1,j}$;
21 end
```

uration for each arm section in the order from sec_n to sec_1, so that the tip frame location for each sec_i is input from the computation for sec_{i+1}, with the tip frame location for sec_n being a given $\mathbf{T}_{t,j}$ on the path G. It further uses recursion to make sure that the determination of the base frame location for sec_i ($n \geq i > 1$), which is the tip frame for sec_{i-1}, will not only result in a feasible configuration of sec_i but also lead to a feasible configuration of sec_{i-1}, and so on, in order to find a feasible configuration of the entire arm.

There are two key procedures in Algorithm 16:

1. given the tip frame configuration of sec_i, partially specifying its base frame, taking into account environment constraints, and

2. computing the configuration of sec_i, satisfying both the tip and the base constraints, by applying task constrained inverse kinematics.

We describe the two procedures in the following subsections.

7.3.1.1 Base Search For an Arm Section

We make an initial guess of its base frame location at step j along path G, $\mathbf{T}_{i,j}$, in a neighborhood of its previous location $\mathbf{T}_{i,j-1}$ and taking into account the obstacles in the workspace. Here we do not fully specify $\mathbf{T}_{i,j}$ in the initial guess so that the unspecified degrees of freedom can be used to find a section configuration (s_i, κ_i, ϕ_i) of sec_i that satisfies the given tip frame location $\mathbf{T}_{i+1,j}$ of sec_i. Note that a single section has 9 degrees of freedom: 6 for its base and 3 for the arm section itself. If a base frame is fully specified along with a given tip frame[6], there may not be a solution for the arm section configuration. Thus, we only specify (i.e., make a guess of) 3 degrees of freedom: either the new base position $\mathbf{p}_{i-1,j}$, or the new base orientation in terms of the rotation matrix $\mathbf{R}_{i,j}$, but not both. We can subsequently use the specified degrees of freedom of the section base, together with those from the section tip to determine both the section configuration and the remaining degrees of freedom of the section base.

To preserve the arm shape from C_{j-1} to C_j as much as possible, we make the initial guesses of section bases from sec_n to sec_1. The tip change of sec_n from $\mathbf{T}_{t,j-1}$ to $\mathbf{T}_{t,j}$ determines the range of change for the base of sec_n, which is the tip of sec_{n-1} and determines

[6]i.e., with more than 9 total degrees of freedom

the range of change for the base of sec_{n-1}, and so on.

The initial guess of $\mathbf{p}_{i-1,j}$ is chosen randomly in a small neighborhood of $\mathbf{p}_{i-1,j-1}$ as:

$$\mathbf{p}_{i-1,j} = \mathbf{p}_{i-1,j-1} + \Delta\mathbf{p}_{i-1}, \tag{38}$$

where $\Delta\mathbf{p}_{i-1}$ is a vector of a small magnitude obtained by the tip position change of sec_i (i.e., within the neighborhood) from arm shape in C_{j-1}.

Similarly, the initial guess of $\mathbf{R}_{i,j}$ can be expressed as:

$$\mathbf{R}_{i,j} = \mathbf{R}_{i,j-1}\Delta\mathbf{R}_i, \tag{39}$$

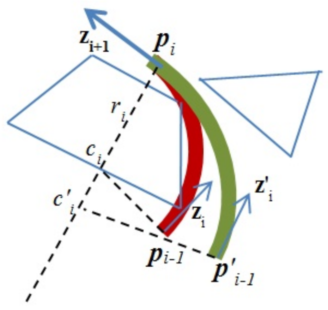
where $\Delta\mathbf{R}_i$ is a 3×3 rotation matrix obtained by the small orientation change of sec_i's tip frame from the previous orientation at step $j - 1$.

If the initial guess of sec_i's base does not lead to a feasible manipulator configuration, as shown in Algorithm 16, which means that the arm configuration is not collision-free (as the configuration satisfies task constraints), our strategy is to modify the configuration of sec_i heuristically to obtain a feasible configuration, which also leads to a modified base location of sec_i:

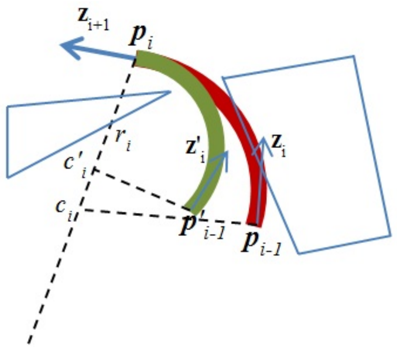
- If sec_i collides with an obstacle at its concave side, see Fig. 63(a), increase the section circle's radius r_i, i.e., decrease curvature κ_i, in small steps, while maintaining the tip constraints, to compute a new base position $\mathbf{p}'_{i-1}$ or a new axis $\mathbf{z}'_i$, which can (hopefully) provide a collision-free configuration.

- If sec_i collides with an obstacle at its convex side, see Fig. 63(b), decrease in small steps r_i, i.e., increase curvature κ_i, to obtain a new base location.

- If sec_i collides with obstacles at both its concave and convex sides, shrink s_i in order to get rid of collision at one side and adjust κ_i to get rid of collision at the other side as described above. However, if sec_i cannot be made collision-free, the recursive algorithm (Algorithm 16) will change the tip of sec_i (i.e., change the base of sec_{i+1}), subsequently change the feasible configuration of sec_{i+1} before running again the above procedure to find a feasible configuration for sec_i.



(a) sec_i has collision at its concave side, a new section configuration is generated by increasing r_i



(b) sec_i has collision at its convex side, a new section configuration is generated by decreasing r_i

Figure 63: Examples of obtaining a collision-free section configuration (green), which satisfies the same tip constraints as the collided configuration (red) obtained from the initial guess of the section base location.

7.3.1.2 Constrained Inverse Kinematics For an Arm Section

Computing a section configuration that satisfies constraints on tip and base frames is essentially an inverse kinematics problem. However, existing methods on inverse kinematics for continuum manipulators either do not constrain the arm tip's orientation [66] or do not address the redundancy of arm solutions [28] to find feasible solutions that satisfy both task constraints and environment constraints.

Next we focus on how to solve for the configuration of a single section sec_i $(i = 1, ..., n)$

under different tip and base constraints.

To be general, we consider a *tip constraint* for sec_i as a fixed tip position $\mathbf{p}_i$ and one or two axes of the tip frame, given either the section base position or orientation. We can categorize different kinds of constraints in the following cases:

- Constraint case (1): constrains $\mathbf{p}_i$ and *one* tip frame axis with given section base position $\mathbf{p}_{i-1}$.
- Constraint case (2): constrains $\mathbf{p}_i$ and *one* tip frame axis with given section base orientation $\mathbf{R}_i$.
- Constraint case (3): constrains $\mathbf{p}_i$ and *two* tip frame axes with given section base position $\mathbf{p}_{i-1}$.
- Constraint case (4): constrains $\mathbf{p}_i$ and *two* tip frame axes with given section base orientation $\mathbf{R}_i$.

For different constraint cases, we now determine the remaining variables of sec_i's tip and base frames, i.e., vectors of either position or frame axes, and then compute the corresponding section configuration.

To determine the remaining frame variables, we first compute the section plane P_i to obtain the following constraining conditions on the frame variables:

- $\mathbf{z}_{i+1}$, $\mathbf{z}_i$, $\mathbf{x}_{i+1}$, and points p_i and p_{i-1} are on the same section plane P_i;
- p_{i-1} and p_i are both on the section circle cir_i;
- $\mathbf{z}_i$ and $\mathbf{z}_{i+1}$ are tangent to cir_i at p_{i-1} and p_i respectively.

Compute the section plane P_i for different constraint cases

Constraint case (1): if $\mathbf{x}_{i+1}$ or $\mathbf{z}_{i+1}$ is fixed as the tip orientation constraint, then P_i is decided by $\mathbf{p}_i$, $\mathbf{p}_{i-1}$ and any point on the ray starting from p_i and in the direction of $\mathbf{x}_{i+1}$ or $\mathbf{z}_{i+1}$; if $\mathbf{y}_{i+1}$ is given as the tip constraint, then P_i passes p_i and with normal parallel to $\mathbf{y}_{i+1}$.

Constraint case (2): if $\mathbf{x}_{i+1}$ or $\mathbf{z}_{i+1}$ is fixed as the tip orientation constraint, then P_i passes p_i and its normal is decided by the cross product between $\mathbf{x}_{i+1}$ (or $\mathbf{z}_{i+1}$) and $\mathbf{z}_i$; if $\mathbf{y}_{i+1}$ is given, then P_i passes p_i with its normal parallel to $\mathbf{y}_{i+1}$.

Constraint cases (3) and (4): P_i passes p_i with its normal parallel to $\mathbf{y}_{i+1}$.

Compute the frame variables on P_i for different constraint cases

For Constraint cases (1) and (3), as illustrated in Fig. 64:

- Draw the perpendicular bisector l of the line segment connecting p_i and p_{i-1}.
- If Constraint case (1), where the section tip frame is not fully constrained, compute tip frame axes $\mathbf{x}_{i+1}$, $\mathbf{y}_{i+1}$, and $\mathbf{z}_{i+1}$ as follows:
  - If only $\mathbf{z}_{i+1}$ is fixed, then section circle cir_i's center c_i can be computed as the intersection between l and the line passing p_i and perpendicular to $\mathbf{z}_{i+1}$; $\mathbf{x}_{i+1}$ is parallel to the line decided by p_i and c_i, and $\mathbf{y}_{i+1} = \mathbf{z}_{i+1} \times \mathbf{x}_{i+1}$.
  - If only $\mathbf{x}_{i+1}$ is fixed, then compute c_i as the intersection between l and the line passing p_i and along $\mathbf{x}_{i+1}$; $\mathbf{z}_{i+1}$ is along the line passing p_i and perpendicular to $\mathbf{x}_{i+1}$, and $\mathbf{y}_{i+1} = \mathbf{z}_{i+1} \times \mathbf{x}_{i+1}$.

- If only $\mathbf{y}_{i+1}$ is fixed, then c_i can be any point (other than p_i) on l[7], $\mathbf{x}_{i+1}$ is along the line passing p_i and c_i, and $\mathbf{z}_{i+1}$ is on the line passing p_i and perpendicular to $\mathbf{x}_{i+1}$.

- Now with $\mathbf{z}_{i+1}$ and c_i determined, $\mathbf{z}_i$ can be determined tangent to cir_i at p_{i-1}. Denote unit vector $\mathbf{u}_i$ from p_i to c_i and unit vector $\mathbf{v}_i$ from p_{i-1} to c_i, and let vector $\mathbf{w} = \mathbf{z}_{i+1} \times \mathbf{u}_i$, which is normal to the section plane P_i, then $\mathbf{z}_i = \mathbf{v}_i \times \mathbf{w}$.

- $\mathbf{x}_i$ and $\mathbf{y}_i$, if $i > 1$, will be determined by the above procedure as tip frame axes for sec_{i-1}. $\mathbf{x}_1$ and $\mathbf{y}_1$ can be any two orthogonal vectors that form a right-handed coordinate system with $\mathbf{z}_1$ at p_0, and one can use a fixed $\mathbf{x}_1$ and $\mathbf{y}_1$ for each step from $j = 1$ to $j = M$ for convenience and continuity.

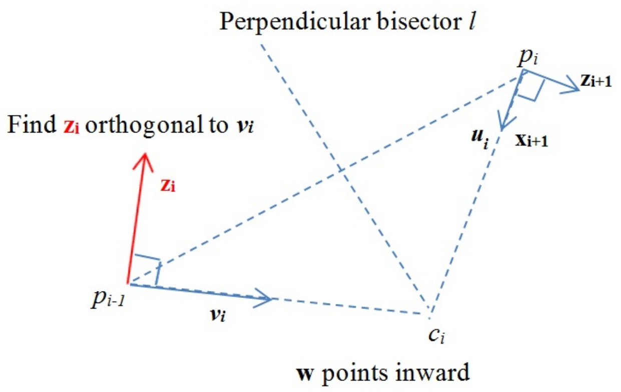


Figure 64: Determining frame variables on P_i for Constraint cases (1) and (3).

For Constraint cases (2) and (4), as illustrated in Fig. 65:

- If Constraint case (2), where the tip frame is not fully constrained, compute $\mathbf{x}_{i+1}$, $\mathbf{y}_{i+1}$ and $\mathbf{z}_{i+1}$ as follows:

[7] In this case, there will be infinite number of solutions.

  - If $\mathbf{z}_{i+1}$ is given, $\mathbf{x}_{i+1}$ is parallel to the line l' on P_i passing p_i and perpendicular to $\mathbf{z}_{i+1}$, $\mathbf{y}_{i+1} = \mathbf{z}_{i+1} \times \mathbf{x}_{i+1}$.
  - If $\mathbf{x}_{i+1}$ is given, $\mathbf{z}_{i+1}$ should be on P_i, orthogonal to $\mathbf{x}_{i+1}$ and passing p_i, $\mathbf{y}_{i+1} = \mathbf{z}_{i+1} \times \mathbf{x}_{i+1}$.
  - If $\mathbf{y}_{i+1}$ is given, $\mathbf{x}_{i+1}$ and $\mathbf{z}_{i+1}$ can be any pair of orthogonal unit vectors on P_i passing p_i.

- Denote l as a line perpendicular to $\mathbf{z}_i$ on P_i and c_i as the intersection between l and l'.
- p_{i-1} is on l, satisfying $|\overline{p_{i-1}c_i}| = |\overline{p_ic_i}|$ and $\mathbf{z}_i = \mathbf{v}_i \times \mathbf{w}$ as in Constraint cases (1) and (3).

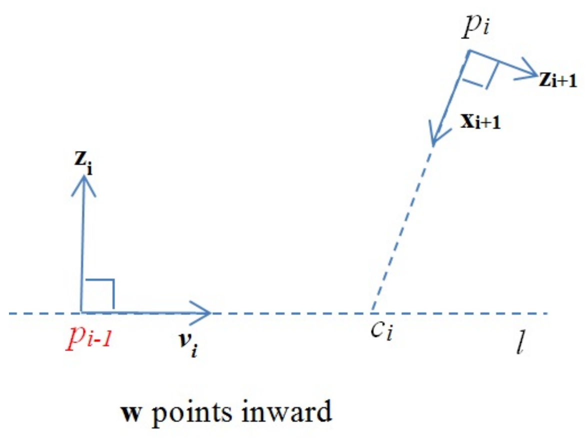


Figure 65: Determining frame variables on P_i for Constraint cases (2) and (4).

Computing the section configuration

After the sec_i's base frame and tip frame are determined, the corresponding values of section configuration variables κ_i, s_i and ϕ_i can be determined as follows,

- c_i is computed as the intersection between the line along $\mathbf{x}_{i+1}$ and the one on P_i and perpendicular to $\mathbf{z}_i$.

- The radius r_i of cir_i can be computed as $r_i = |\overline{c_i p_i}|$. $\kappa_i = 1/r_i$, if c_i on the negative side of $\mathbf{x}_{i+1}$, and $\kappa_i = -1/r_i$, if c_i on the positive side of $\mathbf{x}_{i+1}$, see Fig. 66 for illustration.

- Denote θ_i as the angle between the two vectors from c_i to the two end points p_i and p_{i-1} respectively, then s_i can be computed as $s_i = \theta_i r_i$, see Fig. 66 for illustration.

- ϕ_i is computed as the angle from $\mathbf{y}_i$ to $\mathbf{y}_{i+1}$ about the $\mathbf{z}_i$ axis.

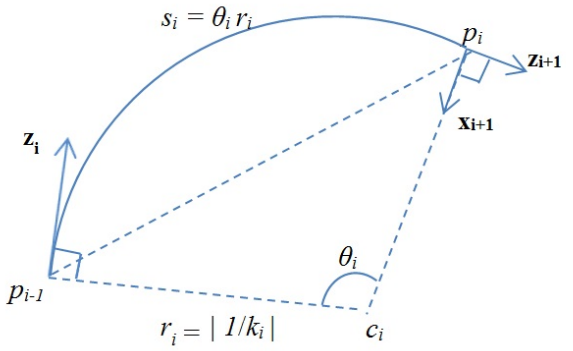


Figure 66: Computing s_i and κ_i on section plane P_i.

7.3.1.3 Multi-section Search Space and Complexity Discussion

For an n-section continuum manipulator with a fully constrained tip (i.e., both position and orientation are constrained) by a task and a mobile base on a 2 dimensional floor, by applying the above task constrained inverse kinematics from sec_n to sec_1, one can *uniquely* map a set of section bases $(p_{n-1}, p_{n-2}, ..., p_0)$ in workspace to an n-section arm configuration. The following characterizes the search spaces for different section bases:

- $p_{n-1} \in \mathbb{R}^2$, where $\mathbb{R}^2$ represents 2 dimensional workspace, since the base p_{n-1} of sec_n should be on the section plane P_n pre-defined by the task constraints.
- $p_{i-1} \in \mathbb{R}^3$, $1 < i < n$, since p_{i-1} can be searched in the 3 dimensional neighborhood of its previous location in the workspace.
- $p_0 \in \mathbb{R}^2$, since the base of the manipulator moves on the 2 dimensional floor.

Algorithm 16 searches the section bases from p_{n-1} to p_0 in a depth-first search fashion with complexity of $O(b^n)$, where b is the branching factor (i.e., number of candidates searched for a single arm section), and n is the number of arm sections. In the worst case, the time complexity is $O(max^n)$, but because of the workspace heuristic used in the search, our method significantly reduced the branching factor b. In the simulation examples presented later, b is close to 1, so that Algorithm 16 has an almost linear time complexity to the number of arm sections n.

As our algorithm combines heuristic and randomized search[8], it can cover all possible space for search under the task constraints by increasing b to approach probabilistic completeness.

7.3.2 Task of Inspection

In this section, we apply our approach of task constrained continuum manipulation to the inspection of a structure in a cluttered environment and present experimental results.

We assume that an inspection device (e.g., a camera or a scanner) is rigidly attached to the tip of the manipulator and facing the object of inspection in certain fixed angle, which

[8]Each randomized search is in a small scale determined by heuristics based on the workspace and the manipulator kinematics.

requires that the tip of the manipulator maintain a fixed orientation with respect to the surface of the object as it moves to inspect the entire object. The inspection consists of multiple scans to cover the surface of the object, where the path of each scan is described as a sequence of task frames $G = \{\mathbf{T}_{t,1}, \mathbf{T}_{t,2}, ...\mathbf{T}_{t,M}\}$.

7.3.3 Implementation and Experiments

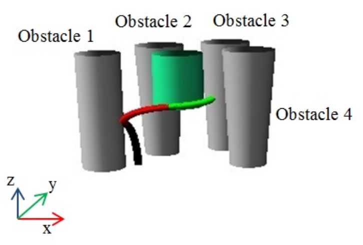
We have implemented all the algorithms on a PC with 2.40GHz Intel(R) Xeon(R) CPU and tested them in both simulation and real-world experiments of inspection with a three-section continuum manipulator, where each section has a width of 4 cm. Specifically, the following describes three simulations for inspection in cluttered space:

- Simulation 1: the tip of the manipulator is kept tangent to the surface of a cylindrical object for inspection, with 4 horizontal line scans, and each line scan is discretized into a path of 200 task frames for the tip of arm to follow. The base of the arm is mobile with its orientation fixed. Fig. 67 (a) shows the task environment.

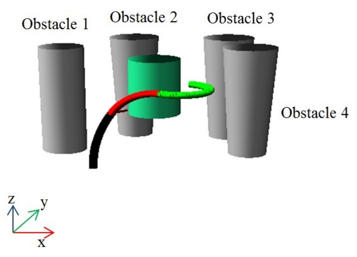
- Simulation 2: the tip of the manipulator is kept orthogonal to the surface of a cylindrical object for inspection, but the tip of the sec_1 is kept tangent to the object face. This example simulates that, in addition to the scanner at the tip of sec_3, another scanner is attached at the tip of sec_1, facing the target object along the $\mathbf{y}$ axis of sec_1's tip. The two scanners are used to scan the object surface simultaneously in a total of 4 horizontal line scans (with each device covering 2 line scans), and each line scan is discretized into a path of 200 task frames. The base of the arm is mobile with its orientation fixed as of Simulation 1, the task environment is similar to Simulation 1 as shown in Fig. 67 (a).

- Simulation 3: the tip of the manipulator is kept a $45°$ angle from the surface of the polygonal object for inspection with 60 vertical raster scans, and each vertical scan consists of a path of 30 task frames. The base has a fixed position. Fig. 67 (b) shows the task environment.

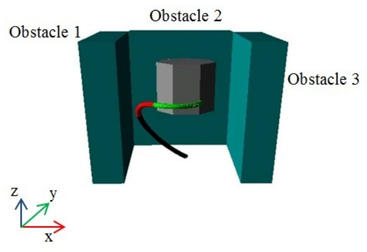
Table 16 displays the object positions and the closest distance from the object surface to each obstacle. Table 17 displays the Constraint cases of each arm section in the three simulations.



(a) a cylindrical object for inspection in Simulation 1



(b) a cylindrical object for inspection in Simulation 2



(c) a polyhedral prism for inspection in Simulation 3

Figure 67: Environments for the three simulations.

Table 16: Object positions (x, y, z) and the closest distance from the object surface to each obstacle for each simulation, see Fig. 67 for example illustrations

Simulation	Object Position	Dist. to Obst. 1	Dist. to Obst. 2	Dist. to Obst. 3	Dist. to Obst. 4
1	(50, 35, 50) (cm)	13.07 cm	8.5 cm	17.49 cm	8.5 cm
2	(50, 35, 46) (cm)	20.16 cm	20.8 cm	28.8 cm	16.8 cm
3	(70, 60, 46) (cm)	19 cm	10.4 cm	13 cm	N/A

Table 17: Constraint case of each arm section

Simulation	sec_3	sec_2	sec_1
1	Constr. case (3)	Constr. case (1)	Constr. case (2)
2	Constr. case (3)	Constr. case (1)	Constr. case (4)
3	Constr. case (3)	Constr. case (1)	Constr. case (1)

A real experiment involving the OctArm robot is also conducted with a fixed arm base (suspended on the ceiling) to preform vertical raster scans with the tip position constrained, as shown in Fig. 68.

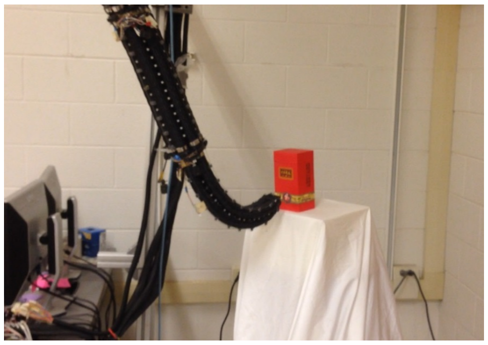
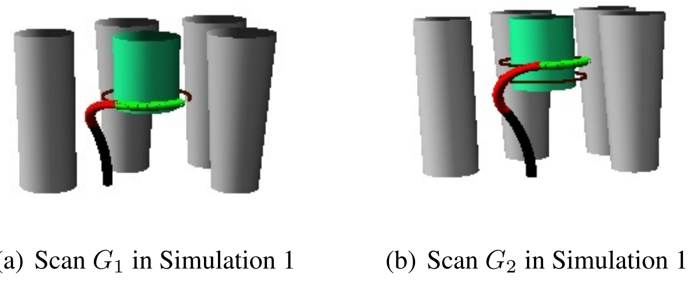


Figure 68: The real experiment: using the OctArm to perform vertical raster scans of one surface of a red box.

7.3.3.1 Results

Our algorithm returns a feasible path of arm configurations for each scan in every simulation/experiment. We set the maximum number of tries max in Algorithm 16 for generating a feasible configuration as $max = 10$. Snapshots of different scans in different simulations are shown in Fig. 69, Fig. 70 and Fig. 71.



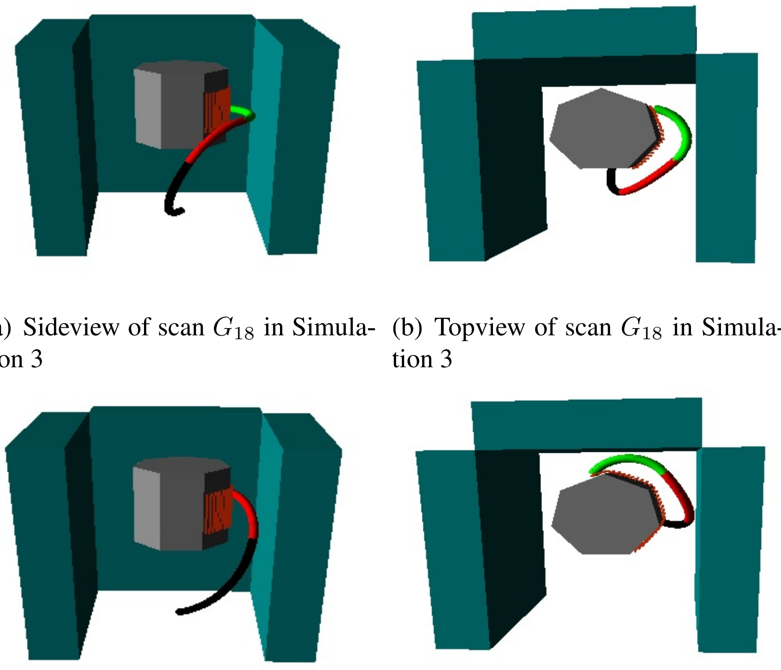
(a) Scan G_1 in Simulation 1 (b) Scan G_2 in Simulation 1

Figure 69: Snapshots of Simulation 1, where the tip of the arm is kept tangent to the surface of the object and the scans are carried out horizontally.

Tables 18 and 19 present the related costs of the scans in Simulation 1 and Simulation

(a) Scan G_1 adnG_3 in Simulation 2 with scanners at sec_1 and sec_3's tips respectively (b) Scan G_2 and G_4 in Simulation 2 with scanners at sec_1's and sec_3's tips respectively

Figure 70: Snapshots of Simulation 2, where the tip of the arm and the tip of sec_1 is kept orthogonal and tangent to the surface of the object respectively; scans are carried out horizontally.


(a) Sideview of scan G_{18} in Simulation 3 (b) Topview of scan G_{18} in Simulation 3

(c) Sideview of scan G_{30} in Simulation 3 (d) Topview of scan G_{30} in Simulation 3

Figure 71: Snapshots of Simulation 3, where the tip of the arm is kept 45^o facing the object surface and the scans are carried out vertically.

2 respectively. Note that the time cost for each scan from G_1 to G_4 increases gradually, because the surrounding obstacles create a narrower space at the top, and thus more computation time was required to adjust the section bases for the scans of the upper part of the

Table 18: Time cost of each scan (with 200 configurations) for Simulation 1, see Fig. 69

scan	total # tries	avg. # tries per config.	# collision checks	total plan. time	avg. plan. time/config.
G_1	18	0.09	218	1.22 (s)	6.1 (ms)
G_2	30	0.15	230	1.42 (s)	7.1 (ms)
G_3	86	0.43	286	1.83 (s)	9.0 (ms)
G_4	102	0.51	302	2.09 (s)	10.45 (ms)

Table 19: Time cost of each scan (with 200 configurations) for Simulation 2, see Fig. 70

scan	total # tries	avg. # tries per config.	# collision checks	total plan. time	avg. plan. time/config.
G_1 & G_3	38	0.19	235	1.39 (s)	6.95 (ms)
G_2 & G_4	55	0.275	253	1.65 (s)	8.25 (ms)

Table 20: Time cost of Simulation 3 with 60 vertical scans and 30 configurations for each scan, see Fig. 71

total # config.	total # tries	avg. # tries per config.	# collision checks	total plan. time	avg. plan. time/config.
1800	240	0.13	2040	11.24 (s)	6.24 (ms)

Table 21: Time costs of two vertical scans (each with 10 configurations) of the real experiment, see Fig. 68

scans	total # tries	avg. # tries per config.	# collision checks	total plan. time	execut. time
up	1	0.1	11	66 (ms)	8.50 (s)
down	1	0.1	11	62 (ms)	8.50 (s)

object. Table 20 presents the related cost of 60 scans in Simulation 3. Table 21 presents the cost of 2 vertical scans in the real experiment.

Also note that the average $\#tries$, which reflects the number of additional searches of each section base for a feasible section, is quite low (less than 0.6) in Algorithm 16 for all simulations and the real experiment, which indicates that our strategy of making initial guesses for the section bases is efficient for computing *feasible* configurations.

7.4 Remarks

This chapter introduced approaches to constrained continuum manipulation in cluttered environments for an n-section continuum manipulator. Our algorithms plan a path for the continuum manipulator that satisfies task constraints, which constrain the position (and/or orientation) of the tip frame of the continuum manipulator during motion, as well as environment constraints, which require the motion of the manipulator to be collision-free. Although the worst case complexity of our algorithms is exponential to the number of arm sections, the average time complexity is still linear to the number of arm sections due to the workspace heuristic used in the search. We have applied our approaches to the task of inspection and conducted a number of simulation and real experiments. The results show that our approaches can generate feasible paths correctly and efficiently in a cluttered environment. The fast computation for each feasible manipulator configuration (in a few milliseconds) along each scan indicates that our approach can be used to conduct real-time simultaneous planning and execution of task-constrained and collision-free paths by a continuum manipulator.

CHAPTER 8: CONTINUUM MANIPULATION IN CLUTTERED, OCCLUDED ENVIRONMENTS

8.1 Overview

For a manipulation task in a largely unknown environment, where a target object is only partially visible inside a tight structure, on-line planning of manipulation motion is necessary while the unknown structure is sensed at the same time as the manipulator tries to access the target object. More importantly, it is necessary to detect, as early as possible, whether the target object can be accessed without damaging the robot or disturbing the structure (e.g., in a search and rescue scenario after an earthquake), i.e., whether a solution exists for the continuum manipulator to fetch the target object. However, approaches proposed in the previous chapters either assume the environment and the target object are known or fully visible before manipulation or assume the target object is accessible in a cluttered environment, so that a path for the manipulator to perform a task always exists. Clearly such assumptions are not realistic in an unknown environment.

This chapter introduces an on-line, deterministic method to determine, as early as possible, if a solution exists for a multi-section continuum manipulator to access a target object that is nested in an unknown cluttered space for manipulation, such as grasping, and to find a solution if one exists for the robot to execute simultaneously as it explores the unknown environment via sensing. In such an environment, the target object is assumed known, however, the obstacles surrounding the target object is unknown (but static) to the manipulator.

8.2 Target Object and Task Environment Model

We consider a target object nested in an unknown, cluttered but static space. We further assume that a distance sensor, e.g., a RGB-D sensor, IR distance sensor, or LIDAR sensor, is attached to the tip of the continuum manipulator, which we call the tip sensor or camera, so that the manipulator can sense the space between the target object and the surrounding structure or obstacles as its tip moves. Initially, the manipulator tip is outside the structure looking into the space between the object and the structure or obstacles. The target object is assumed known, either represented directly by a polygonal mesh or as the result of recognition and re-construction from point clouds after pose estimation [84] from which a polygonal mesh can also be built.

However, the obstacles surrounding the target object are not initially known; they are observed as growing point clouds by the tip sensor of the manipulator as the manipulator moves.

8.2.1 Belts for Grasping on Target Object and Gaps

We assume some prior knowledge about surface regions of the object that are suitable for a continuum manipulator to wrap around. Specifically, the target object can have one or more closed “belts” on its surface based on human user input. Each “belt” can be viewed as rolling a rectangle along the surface of the target object such that two opposite edges of the rectangle meet. Given the target object mesh, each “belt” is pre-constructed by the following steps:

- Use a plane to cut the object in a certain way (based on human input) to obtain the boundary polygon $poly_1$ of the cross section.

- $i \leftarrow 1$.
- Repeat
  - move the plane in the direction a normal to $poly_i$ some distance δ to produce another boundary polygon $poly_{i+1}$,
  - $i \leftarrow i + 1$

  until $(i-1)\delta = w_{belt}$, where w_{belt} is a pre-determined belt width.
- Obtain the convex hull CH of the point clouds from $poly_1$, ..., $poly_i$.

We call CH a *belt for grasping* on the target object, which is also a polygonal mesh around the object. Depending on the shape of the object, there can be one or more belts for grasping, based on human input. Fig. 72 shows some examples. Each possible grasping solution of the target object, in terms of the pose of the continuum manipulator, relates to a belt for grasping on the object, in that the manipulator wraps around the belt at that pose.

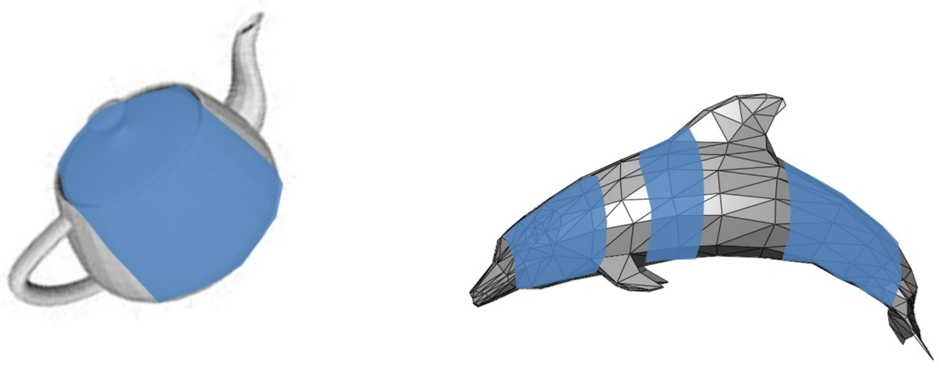

(a) A teapot (b) A dolphin model

Figure 72: Examples of different target objects, where blue regions indicate belts for grasping.

Once a belt for grasping is selected, the tip sensor of the manipulator starts observing the belt, first from the initial configuration of the manipulator outside of the surrounding structure/obstacles of the target object. Assume that the sensor can see as wide as the width of the belt. Thus, the *reachable region* R_1 on the belt is bounded by B_1 and B_2, where B_1 is the piece-wise linear curve segment consisting of a set of nearest visible and reachable vertices, and B_2 is the piece-wise linear curve segment consisting of the furthest visible and reachable vertices, by the tip of the manipulator. By observing the space between R_1 and the surrounding visible obstacles via the tip sensor, we aim to decide if the manipulator can get through, and if so, the continuum manipulator will move through the space along R_1, and the tip sensor will have a new view of the surrounding space of another reachable region R_2 of the belt to decide again if the manipulator can get through that space, and so on. If the manipulator can pass all observed spaces, it can successfully wrap around the object; on the other hand, if the manipulator observes a space, knowing that it cannot pass through, it will withdraw without further entering that space.

In general, let R_j be the jth surface region of the belt that the tip sensor observes, $j = 1, 2, ...$, we define gap_j as the space between R_j and the obstacles in the Voronoi region of R_j. Fig. 73 shows an example gap. Clearly, a gap can vary a lot in shape and size depending on the shape and size of R_j and obstacles in the Voronoi region of R_j.

8.2.2 Task Environment Model.

We describe how to characterize the space of a gap as related to whether the continuum manipulator can get through the gap. If there is a sequence of gaps for the manipulator to get through, the portions of the manipulator inside gap_j must form an arc with a non-zero

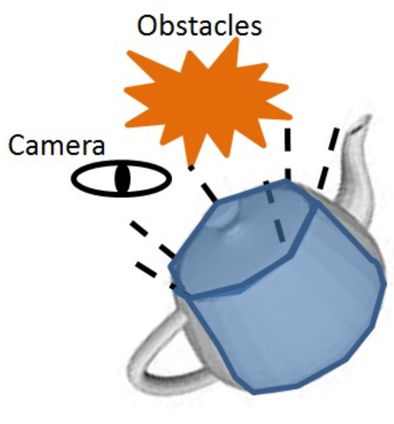


Figure 73: Example of a gap.

curvature, i.e., cannot be a straight-line segment, because a straight-line segment cannot bend over a sharp corner smoothly. Note that the arc can be formed by either a portion of an arm section, an entire arm section, or multiple arm sections of the manipulator.

Such an arc is a function of the following parameters:

$$arc = f(u, v, \theta) \tag{40}$$

where u is the starting point of the arc on the boundary curve segment B_j of R_j (which is piece-wise linear) at the beginning of gap_j, v is the ending point of arc_j on B_{j+1} of R_j at the end of the gap, and θ is the angle between the z axis at u and the edge or one of the edges on B_j that contains u.

Denote the height vector of arc as $\mathbf{h}$ (which is the maximum distance vector from the cord to the arc). The shape of arc is described by h, the arc angle α, and the length $l = |\overline{uv}|$, as depicted in Fig. 74, which satisfy the following equation (see Appendix):

$$tan(\alpha) = \frac{4lh}{l^2 - 4h^2}. \tag{41}$$

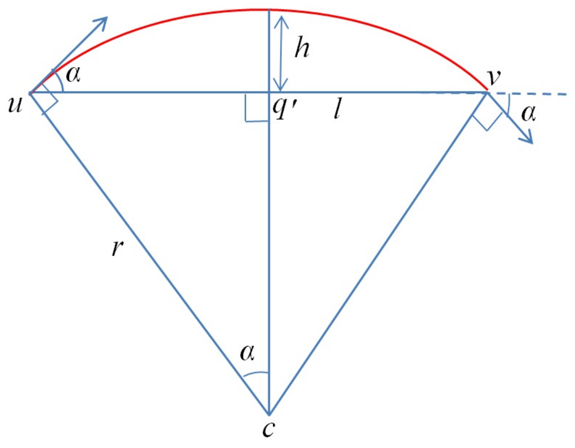


Figure 74: Circular arc and its parameters.

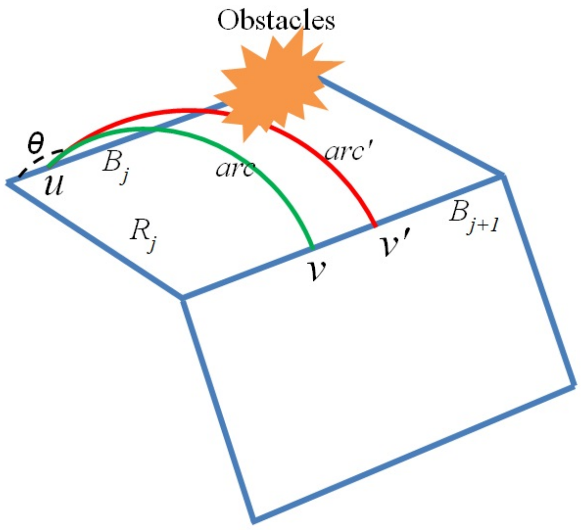


Figure 75: Different arcs passing a gap_j.

We are interested in the maximum value of h without causing a collision of the manipulator following arc with obstacles in gap_j. As shown in Fig. 75, depending on different points u and v and the angle θ, the maximum value of h can be different for not causing collisions, which characterizes the wideness (or narrowness) of the gap on the plane of arc. Let O_j denote the set of surface points of obstacles in gap_j. We can define the gap width function $d(u, v, \theta)$ as:

$$d(u, v, \theta) = max(h|arc \cap O_j = \emptyset), \tag{42}$$

Hence, d is the maximum height h for arc to pass gap_j without collision.

By systematically changing u, v, and θ, we can compute $d(u, v, \theta)$ for all possible arcs passing gap_j. u and v can be changed by discretizing the corresponding edges of R_j. For each pair of u and v, θ can be changed by rotating the z axis at u about the line segment $\overline{uv}$ in small angular increment in both directions, so long as the corresponding arc does not collide with the reachable region R_j.

Now, for each set of u, v, and θ, we find the maximum height $d(u, v, \theta)$ that will not make arc collide with O_j through a binary search:

- Consider the plane of arc. Find the point in O_j on the plane of arc that is furthest from R_j, and denote the distance as d_{far}, and find the point in O_j on the plane of arc closest to R_j, and denote the distance as d_{near}. A binary search tree is utilized to organize the points on the plane of arc [9]. Thus, querying each point in a point cloud of k points yields an average computational complexity of $O(log\ k)$.

- Conduct a binary search to find the maximum h in $[d_{near}, d_{far}]$ that does not cause arc colliding with O_j, and save the value as $d(u, v, \theta)$. Fig. 76 illustrates the simple binary search.

The above algorithm uses the collision detection algorithm in [57] to check for collisions between points and arcs.

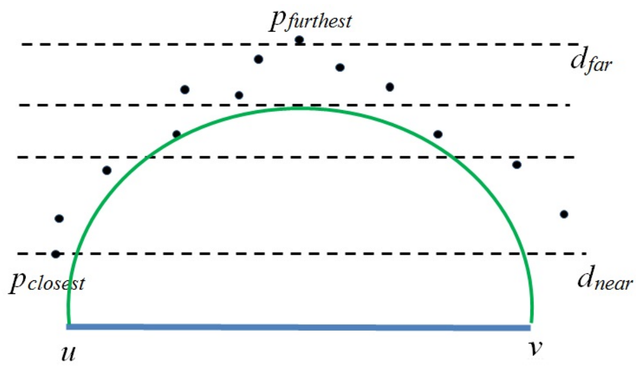


Figure 76: An arc and the binary search of the maximum height that does not cause arc to collide with obstacle points.

8.3 Single Gap Constraints

In order for the continuum manipulator to pass through gap_j without collision, its section(s) should be able to follow *at least* one arc, with end points u, v, and orientation θ, whose height h satisfies:

$$0 \leq h \leq d(u, v, \theta) - w, \tag{43}$$

where w is the width of the manipulator arm (which may vary from section to section). Inequality (43) is called a *single gap constraint*. If the manipulator arm cannot form any arc that satisfies the above height constraint in gap_j, it cannot pass gap_j without collision, i.e., no solution exists.

For gap_1, the manipulator only needs to satisfy the above constraint (43) to pass through the gap following certain arc_1. However, for gap_j, $1 < j \leq N$, an arc_j that can make the manipulator pass through gap_j also needs to satisfy constraints from gap_{j-1}, as described in the next section.

8.4 Serial Gap Constraints

Once the manipulator passes through gap_1, i.e., its tip sensor is at the boundary of gap_1 looking into a new gap_2, the constraints on the arc parameters in gap_2 must be considered. In order to pass through gap_2, not only that the manipulator must be able to follow an arc in gap_2, say arc_2, that satisfies the corresponding single gap constraint (43), but arc_2 and the arc arc_1 that the manipulator followed in gap_1 must share the same tangent vector $\mathbf{a}_{1,2}$ at their junction between gap_1 and gap_2 to ensure tangent continuity of the manipulator arm shape, as shown in Fig. 77. It is not difficult to see that if α_1 increases, α_2 has to decrease to satisfy the tangent continuity constraint.

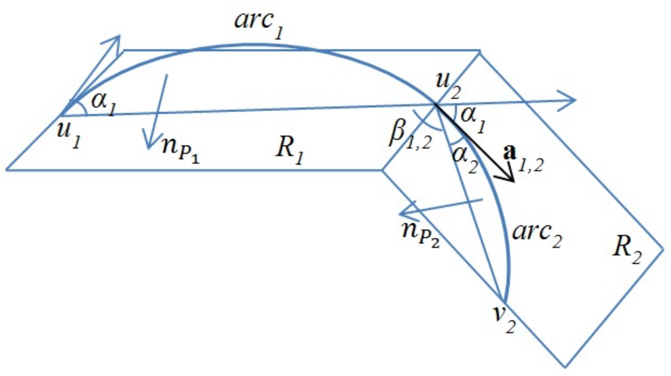


Figure 77: arc_1 and arc_2 are connected by a common tangent vector $\mathbf{a}_{1,2}$.

In general, for any $j > 1$, given the common tangent $\mathbf{a}_{j-1,j}$ of arc_{j-1} and arc_j in gap_{j-1} and gap_j respectively, we can derive the following additional constraint on α_j in terms of α_{j-1} and parameters relating gap_{j-1} and gap_j (see Appendix for the derivation):

$$(\mathbf{n}_{A_{j-1}} \cdot \mathbf{n}_{A_j}) sin\alpha_{j-1} sin\alpha_j + cos\alpha_{j-1} cos\alpha_j = -cos\beta_{j-1,j} \tag{44}$$

where $\mathbf{n}_{A_{j-1}}$ and $\mathbf{n}_{A_j}$ are the normals of the planes A_{j-1} of arc_{j-1} and A_j of arc_j respectively, and $\beta_{j-1,j}$ is the angle between the chord $\overline{u_{j-1}u_j}$ of arc_{j-1} and the chord $\overline{u_j v_j}$ of

arc_j, as shown in Fig. 77.

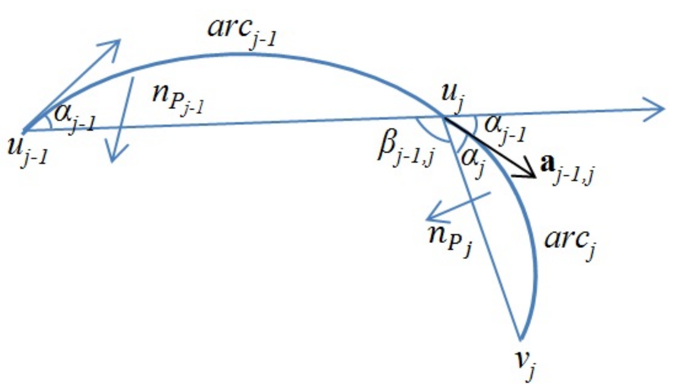


Figure 78: $\mathbf{a}_{j-1,j}$ and how it relates arc_{j-1} parameters to arc_j parameters.

The above Equation (44), together with Equation (41), can be further converted to a constraint relating h_j to h_{j-1}. The effect is to narrow the range of values for h_j, because the upper bound for h_{j-1} (to satisfy its single gap constraint inequality (43)) introduces a non-zero lower bound $d_{arc_{j,j-1}} > 0$ for h_j, as shown in Fig. 78. Therefore, the following *combined constraint* on h_j should hold:

$$d_{arc_{j,j-1}} \leq h_j \leq d_j(u_j, v_j, \theta_j) - w. \tag{45}$$

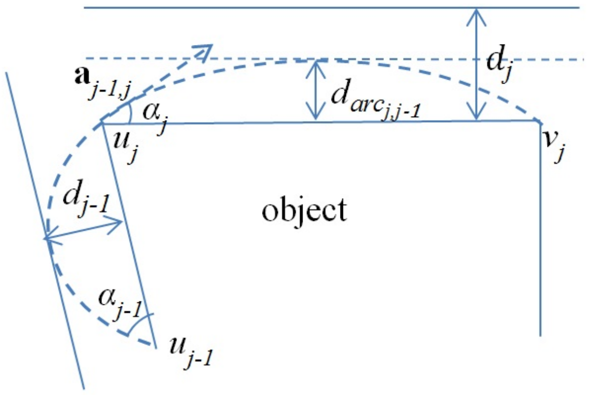


Figure 79: A lower bound for h_j resulted from the upper bound for h_{j-1} because of tangent continuity and the single gap constraint on arc_{j-1}.

8.5 Existence and Generation of Solution

The continuum manipulator tries to go through gap_1, ..., gap_N, one by one, by first sensing the encountered gap and then checking if the related constraints can be satisfied. For gap_1, if the manipulator arm width w is greater than the maximum h under the single gap constraint (43), then the manipulator cannot get into the gap – no solution exists, which is an obvious case. If the manipulator can get through gap_1 with its tip reaching the edge of gap_2, then the combined constraint (45) must be satisfied (for $j = 2$) before the manipulator can get through gap_2, and if it cannot be satisfied, then there is no solution forward, and the manipulator has to withdraw from gap_1. In general, the manipulator can encounter gap_j, $j > 1$, only if it can get through gap_1, ..., gap_{j-1}. If there is no solution to go through gap_j, the manipulator has to withdraw from gap_{j-1} to gap_1.

8.5.1 Strategy for Checking Gap Constraints and Existence of Arm Solution

Gaps are observed and checked in turn as they are encountered in real time for feasible arm configurations. Our strategy, shown in Algorithm 17, ensures that all feasible arcs in a gap_j, $j > 1$, connect to (stored) feasible arcs from gap_{j-1} back to gap_1 and detects the cases where the manipulator cannot pass through all the gaps without colliding obstacles.

Let K be the average number of arcs for each gap, and there are a maximum of N gaps that the arm can encounter. Our strategy trades space for time efficiency. Although its worst-case space complexity is $O(K^N)$, its worst-case time complexity is $O(K + (N - 1)K^2)$, and since K is more or less a constant (depending on the resolution of u, v and θ), this strategy has a worst-case time complexity linear to N.

Algorithm 17: $CheckingGapConstraints$

```
1  At $gap_1$, for every arc (obtained from varying $u$, $v$, and $\theta$) the corresponding gap
   width $d$ is obtained and stored;
2  next, those arcs that the manipulator can follow to satisfy the single gap constraint
   are stored as *feasible* ones in set $S_1$;
3  **if** $S_1 = \emptyset$ **then**
4      **return** “No arm solution exists” and exit.
5  **end**
6  $j \leftarrow 1$;
7  **while** *a new gap is encountered* **do**
8      $j \leftarrow j + 1$;
9      after the manipulator follows a feasible arc in $S_{j-1}$ through $gap_{j-1}$ and
       encounters $gap_j$, for each of the arcs in $gap_j$ that is connected to a feasible arc
       in $gap_{j-1}$ with tangential continuity at the connection point, the corresponding
       gap width $d$ is obtained and stored;
10     next, those arcs in $gap_j$ that the manipulator can follow to satisfy the serial gap
       constraint are stored as feasible arcs *with their connected feasible arcs in
       $gap_{j-1}$ indicated* in set $S_j$;
11     **if** $S_j = \emptyset$ **then**
12         **return** “No arm solution exists” and exit;
13     **end**
14 **end**
15 **return** information of feasible arc sequences from $gap_1$ to $gap_j$.
```

8.5.2 Generation of a Feasible Arm Configuration

Among the feasible arcs found for gap_j, $j \geq 1$, the one whose starting position is closest to the tip of the arm in its current configuration is chosen as arc_j for the arm tip to follow, and the resulting new arm configuration [58] is characterized by the following: the curvature of the manipulator tip section sec_n is the same as the curvature of arc_j, and the orientation of sec_n is aligned with arc_j, which also means that the z-axis of the tip frame is always along the tangent of arc_j, for each tip position on arc_j; next the base position of sec_n can be decided on arc_j based on the length range of sec_n and the length of arc_j, and the z-axis of the base frame of sec_n can be also along the tangent of arc_j.

After the base position and orientation of sec_n are determined, since they are the tip position and orientation of sec_{n-1}, if sec_n cannot cover the entire arc_j, the above partial inverse procedure is applied to find the pose of the next arm section sec_{n-1}, and so on, until arc_j is covered by some sections of the arm.

Note that when the manipulator encounters gap_j, if $j > 1$, the manipulator sections are already in gap_{j-1} through gap_1 along a particular sequence of arcs. In order for the tip section of the arm to move forward to follow the closest feasible arc_j in gap_j, if the current arc_{j-1} in gap_{j-1} does not satisfy the serial link constraint with arc_j, arc_{j-1} will be updated to the closest (feasible) arc in S_{j-1} connected to arc_j (along with its sequence of arcs from gap_{j-2} to gap_1, if $j > 2$), and the arm sections will be updated accordingly to follow the new sequence of arcs.

8.6 Implementation and Examples

We have implemented and tested in simulation our approach for on-line progressively determining whether a target object in an unknown cluttered space can be fetched and if so, generating the motion for a continuum manipulator to fetch the object. The implementation uses a computer with a 2.40GHz CPU. A three-section continuum manipulator with a section width w of 2.54cm and an RGB-D camera attached at its tip is used to fetch an object in a cluttered environment with unknown obstacles.

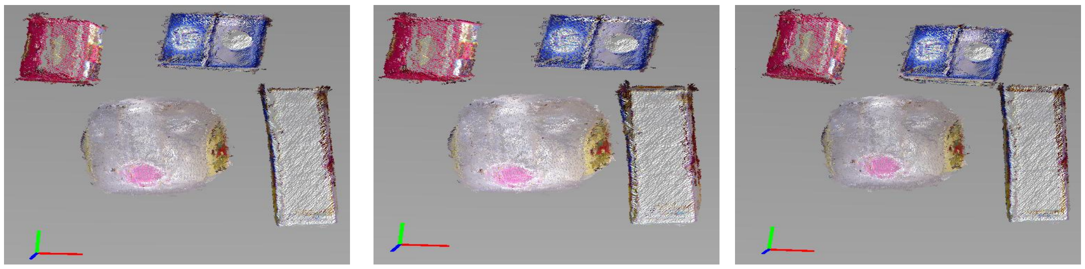
We conducted experiments with two different target objects in environments shown in Fig. 80 with three different arrangements of surrounding obstacles on the object's left, back and right sides:

- Example one: the target object is a point cloud of a milk container, re-constructed from RGB-D sensing [84].

- Example two: the target object is a polygonal mesh of a bunny, which is of complex shape.

In Fig. 80, the top views of the environments are shown. We assume that the environment has a low ceiling (that is not shown in Fig. 80 for the reader's viewing convenience) so that the manipulator cannot fetch the target object from above. Also note that, only the target object and its pose are known to the manipulator; the obstacles are unknown to the manipulator, which has to rely on its tip sensor to gradually sense the obstacles in each gap.

A pre-determined wrapping belt with a width of 15.1cm on each target object is shown in Fig. 81.



(a) Environment (1): obstacles are placed at the left, back and right sides of the target obejct.

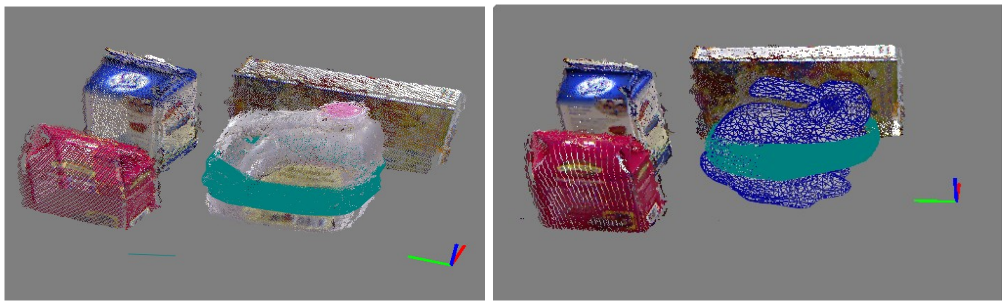
(b) Environment (2): obstacle on the right side is closer to the target object.

(c) Environment (3): obstacle at the back is closer to the target object.

Figure 80: Top view of environments with three different arrangements of obstacles surrounding the milk container (a target object).

In Environment (1) shown in Fig. 80 (a), all gap constraints are satisfiable for both examples, as shown in Fig. 82 and Fig, 84 respectively. Note that three gaps are observed in Example one and four gaps are observed in Example two.

In Environment (2) shown in Fig. 80 (b), obstacles on the right are closer to the target



(a) Wrapping belt on the milk container (b) Wrapping belt on the bunny

Figure 81: A pre-determined wrapping belt on each target object.

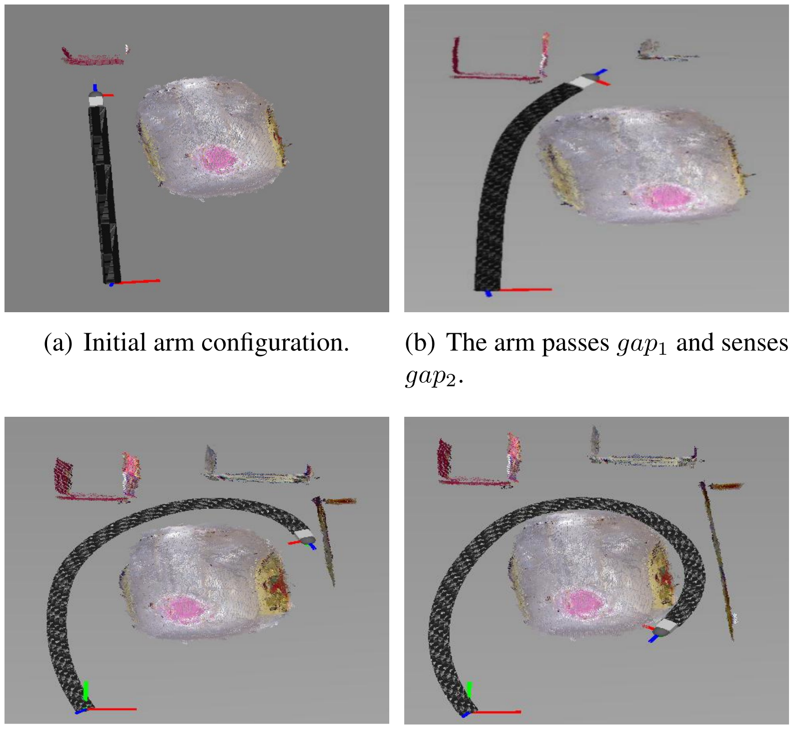
object than those in Environments (1) and (3). In Example one with the milk container, the arm is stuck in front of gap_3, since gap_3 is too narrow so that the single gap constraints cannot be satisfied, as shown in Fig. 83 (a). In Example two with the bunny, the arm passes gap_3 but is stuck in front of gap_4, since gap_4 is too narrow to pass, as shown in Fig. 85 (a).

In Environment (3) shown in Fig. 80 (c), obstacles at the back of the target object are closer to the object than those of Environments (1) and (2). In Example one, the arm passes gap_2 but is stuck in front of gap_3, since $gaps_2$ (at the back of the object) is too narrow so that the serial gap constraints cannot be satisfied, as shown in Fig. 83 (b). In Example two, the arm passes gap_3 but is stuck in front of gap_4, since gap_3 is too narrow so that the serial gap constraints between gap_3 and gap_4 cannot be satisfied, as shown in Fig. 85 (b).

Table 22 and Table 23 show the following data from running our algorithm for Example one and Example two respectively:

- t, time cost of checking constraints for each gap;
- # arcs, the number of arcs checked for passing each gap_j with the resolution of u and v along the belt edges being 1cm and that of θ being 10^o.

- $|S|$, the number of feasible arcs found that satisfy related gap constraints in each gap.



(a) Initial arm configuration.

(b) The arm passes gap_1 and senses gap_2.

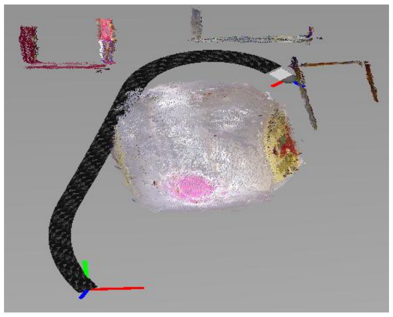
(c) The arm passes gap_2 and senses gap_3.

(d) The arm passes gap_3 and forms a grasp to fetch the object.

Figure 82: Snapshots of Example one in Environment (1), where only the visible surface patches of obstacles to the tip camera are shown.

Clearly it takes longer time for checking constraints for gap_2 and gap_3 than for gap_1 due to the computation cost for handling more than one gap. However, because of the serial gap constraints, the more gaps the arm has passed, the fewer *feasible* arcs are in S_j for a newly encountered gap_j; as a result, the time cost of checking gap_j, $j > 1$, decreases as j increases. Thus, the algorithm delivers real-time performance in milliseconds in checking the existence of solution and generating a solution (if one exists) progressively, guided by sensing in an unknown environment.

Note that gap_1 has $\#$ arcs significantly smaller than that of gap_2 and gap_3, because the


(a) In Environment (2), the arm is stuck in front of gap_3 since gap_3 too narrow.

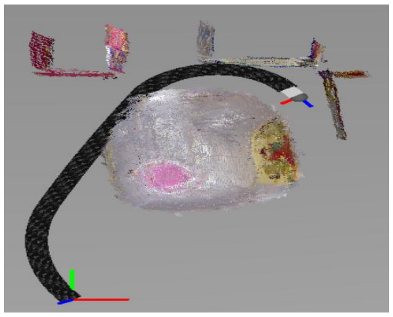
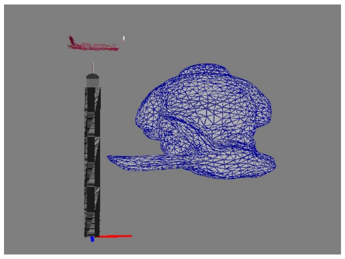
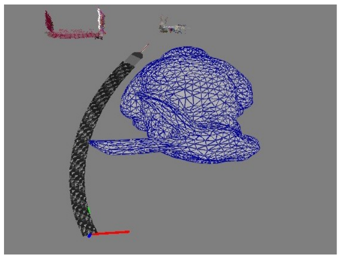
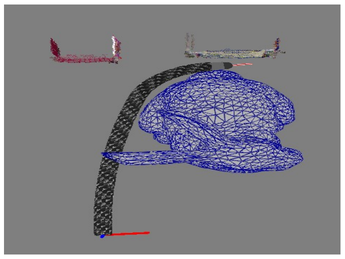
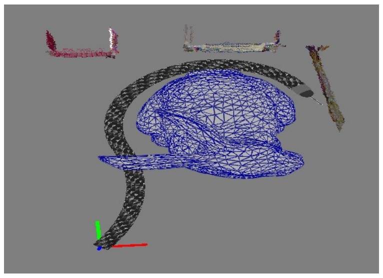

(b) In Environment (3), the arm is stuck in front of gap_3 since gap_2 is too narrow.

Figure 83: No grasping solution for Example one in Environments (2) and (3) because the arm cannot pass gap_3.


(a) Initial arm configuration.


(b) The arm passes gap_1 and senses gap_2.


(c) The arm passes gap_2 and senses gap_3.


(d) The arm passes gap_3 and senses gap_4.

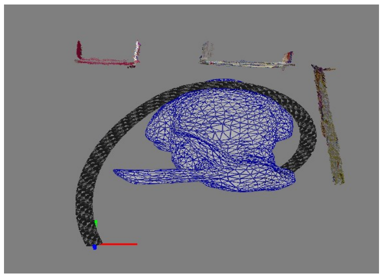
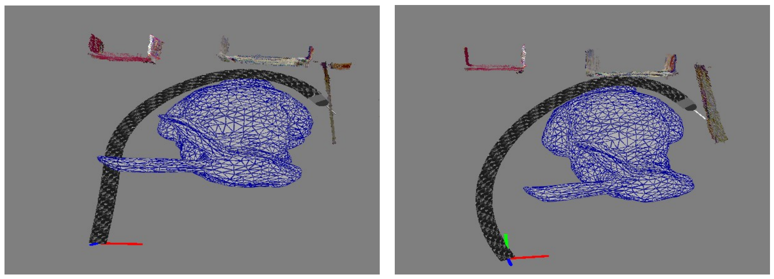

(e) The arm passes gap_4 and forms a grasp to fetch the object.

Figure 84: Snapshots of Example two in Environment (1), where only the visible surface patches of obstacles to the tip camera are shown.

arm has a fixed base, the starting points of the arcs in gap_1 are determined and fewer. Whereas, the starting point u of an arc to be checked for gap_j, $j > 1$, can be the ending



(a) In Environment (2), the arm is stuck in front of gap_4 since gap_4 is too narrow.

(b) In Environment (3), the arm is stuck in front of gap_4 since gap_3 is too narrow.

Figure 85: No grasping solution for Example two in Environments (2) and (3).

Table 22: Time cost for checking each gap, the number of arcs checked, and the number of (feasible) arcs in S for each gap in Example one

Milk container	Environment (1)			Environment (2)			Environment (3)		
	t (ms)	# arcs	$\|S\|$	t (ms)	# arcs	$\|S\|$	t (ms)	# arcs	$\|S\|$
gap_1	0.8	272	225	0.8	272	225	0.8	272	225
gap_2	8.6	2925	1800	8.6	2925	1800	8.6	2925	900
gap_3	5.6	1920	240	5.6	1920	0	4.6	1565	0

Table 23: Time cost for checking each gap, the number of arcs checked, and the number of (feasible) arcs in S for each gap in Example two

Bunny mesh	Environment (1)			Environment (2)			Environment (3)		
	t (ms)	# arcs	$\|S\|$	t (ms)	# arcs	$\|S\|$	t (ms)	# arcs	$\|S\|$
gap_1	0.8	272	225	0.8	272	225	0.8	272	225
gap_2	7.7	2640	1080	7.7	2640	1080	7.7	2640	420
gap_3	5.6	1942	576	5.6	1942	576	3.9	1354	168
gap_4	4.7	1632	96	4.7	1632	0	2.6	920	0

point v of any feasible arc in S_{j-1}.

Since the obstacles are presented as surface point clouds from real-world RGB-D sensing, and the simulated image from the tip sensor of the manipulator is the point clouds that the tip sensor can see, we can argue that the computation time costs displayed in Table 22 and 23 are similar to the computation time costs processing RGB-D sensing of a real tip sensor in the real-world environment of the same setting. Hence, our algorithm can be similarly efficient with real RGB-D sensing.

8.7 Remarks

This chapter addressed the problem of on-line determining if a continuum manipulator could fetch a partially visible object nested in an unknown cluttered space. We model the space surrounding a belt for grasping on the target object as a chain of gaps, which are observed progressively one after another by the tip sensor of the manipulator. By forming constraints relating the manipulator to the gaps, our method can quickly determine wether the manipulator can successfully get through a gap as soon as the gap is sensed, so that the manipulator and the surrounding environment will not get damaged. If a solution exists, our approach can find it in a few milliseconds so that the robot can execute it while exploring the rest of the unknown environment via sensing. The algorithm of checking gap constraints has a time complexity linear to the number of gaps observed. The efficiency and effectiveness of this approach has been demonstrated by experiments of fetching an object in simulated environments with obstacles represented as point clouds from real-world RGB-D sensing. Although the target object is assumed known, this approach can be extended to deal with partially known or unknown target object by also sensing and building the belt for grasping progressively.

CHAPTER 9: CONCLUSIONS

This dissertation focuses on autonomous manipulation using a multi-section continuum manipulator in cluttered environments. Novel approaches are introduced to address related issues effectively and efficiently.

Approaches introduced in this dissertation are general because they are applicable to a general n-section continuum manipulator, and the arm model used in this dissertation (as described in Chapter 3) can represent arm shapes of most prototypes of continuum manipulators [101], from small scale continuum manipulators [102, 24, 21, 105] for medical use to larger scale pneumatic/tendon driven continuum manipulators [99, 36, 16] even though they may have different mechanical designs. To apply the manipulation algorithms of this dissertation onto a specific prototype of a continuum manipulator, one just needs to assign values of the parameters to reflect the physical constraints of the manipulator, i.e., length, orientation, curvature limits. Some manipulators may have fewer variables than the general model represents; in such cases, some variables in the general model can be assigned constants to reflect those manipulators.

First, an efficient, real-time collision detection algorithm (CD-CoM) between a continuum manipulator and 3D mesh objects is introduced. This collision detection algorithm uses a parametric model for a continuum manipulator. Comparing to conventional mesh-to-mesh based collision detection algorithms, it is more efficient and saves a significant amount of time for building/refitting an arm model when the arm shape changes. It applies to any continuum manipulator whose shape can be modeled as consisting of toroidal and cylindrical primitives. The CD-CoM algorithm is particularly suitable for continuum manipulator path planning that considers a large number of manipulator configurations in real time. The CD-CoM algorithm also provides information of the minimum distance between each section of the continuum manipulator and objects if there is no collision, which could be further exploited to achieve desirable contact states for continuum manipulation and to take advantage of time and space coherence of a moving continuum manipulator and moving objects. Note that, all the manipulation algorithms proposed in this dissertation use CD-CoM for checking collisions/contacts between a continuum manipulator and objects/obstacles in the environment.

Second, several whole arm grasping algorithms are proposed for a continuum manipulator grasping 3D objects in both open and cluttered environments. Given a 3D target object, one of our approaches (introduced in Chapter 5) first defines grasping models for a given object to facilitate grasping by the continuum manipulator. It then uses inter-section constraints in analytical equations to characterize and numerically solves for all possible grasping configurations. For a given object grasping model, the approach is implemented to determine all possible valid grasping configurations, but with a time complexity exponential to the number of arm sections. Two progressive grasping approaches are then introduced (in Chapter 6) to achieve a feasible grasping configuration of the target object in real time for either an open environment or a cluttered environment. Progressive grasping algorithms are validated in both simulations and real experiments with a time complexity linear to the number of arm sections. Note that the progressive grasping algorithms can also be extended to enable grasping an object not fully visible initially by gradually extending the manipulator to explore the surface of the object, if the manipulator is equipped with sensors at its tip.

Third, task constrained continuum manipulation methods in cluttered environments are introduced. The introduced algorithms plan a path for a continuum manipulator satisfying task constraints, which constrain the position and/or orientation of the tip frame of the continuum manipulator during motion, as well as environment constraints, which require the motion of the manipulator to be collision-free. In addition, a general method of constrained inverse kinematics for a continuum manipulator is also introduced, which considers different combinations of position and orientation constraints of an arm section's tip and base. Example inspection tasks are performed with a number of simulations and real experiments. The results show that our approaches can generate feasible paths for the manipulator correctly and efficiently in a cluttered environment, and even though the worst-case time complexity of our algorithms is exponential to the number of arm sections, the average time complexity is still linear to the number of arm sections due to the effective workspace heuristic used in the search. The fast computation for each feasible manipulator configuration (in a few milliseconds) in a cluttered environment indicates that our approach can be used to conduct real-time simultaneous planning and execution of task-constrained and collision-free paths by a continuum manipulator.

Fourth, an approach for continuum manipulation in cluttered and unknown environments is proposed, where information about the environment is obtained through progressive sensing. By forming constraints relating the manipulator to the gaps between object and surrounding obstacles, this method can quickly determine whether the manipulator can successfully get through a gap as soon as the gap is sensed, so that the manipulator and the surrounding environment will not get damaged. If a solution exists, the approach can find it in a few milliseconds so that the robot executes it while exploring the rest of the unknown environment via sensing. The algorithm of checking gap constraints has a time complexity linear to the number of gaps observed. The efficiency and effectiveness of this approach has been demonstrated by experiments of fetching an object in simulated environments with obstacles represented as point clouds from real-world RGB-D sensing. Although so far the target object is assumed known, this approach can be extended to deal with partially known or unknown target object by also sensing and building the belt for grasping progressively.

However, the proposed approaches have some limitations. For example, the physical properties of a continuum manipulator and objects are not considered, only geometric models are used. The manipulator is assumed without external force disturbance, such as payloads, except contact force, and the deformation of arm sections while in contact with the environment is not modeled. Future research could include studying the physical properties of a continuum manipulator by considering internal/external forces applied to the manipulator through force sensing, and also the deformations of the manipulator due to interaction with objects in the environments, so that more accurate estimations of the arm shapes and object poses can be obtained. As a continuum manipulator is highly compliant, it can be suitable to interact with people for different applications, such as collaborating with human workers to transfer objects or assisting a human patient. Human robot physical interaction is another interesting future research topic.

REFERENCES

[1] OC Robotics, available at: www dot ocrobotics dot com.

[2] Barrett technology inc. available at: www dot barrett dot com.

[3] M. Abramowitz and I. A. Stegun. Solutions of quartic equations. In *Hand-book of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, pages 17–18. Dover, 1972.

[4] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based prm for 3d workspaces. *In Proc. Int. Workshop on Algorithmic Foundations of Robotics*, pages 155 – 168, 1998.

[5] R. Ambrose, H. Aldridge, R. Askew, R. Burridge, W. Bluethman, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark. ROBONAUT:NASA space humanoid. *IEEE Intelligent Systems Journal*, 2000.

[6] F. Aurenhammer. Voronoi diagrams a survey of a fundamental geometric data structure. *ACM Computing Surveys*, pages 345 – 405, 1991.

[7] B. Baginski. Efficient dynamic collision detection using expanded geometry models. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1714–1719, 1997.

[8] T. Baier-Löwenstein and J. Zhang. Learning to grasp everyday objects using reinforcement-learning with automatic value cut-off. *IEEE Transactions on Robotics and Automation*, pages 669 – 679, 2003.

[9] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509, 1975.

[10] D. Berenson, S. Srinivasa, and J. Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 2011.

[11] A. Bicchi. On the closure properties of robotic grasping. *Int. J. Robot. Res.*, 14:319 – 334, 1995.

[12] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data driven grasp synthesis - a survey. *IEEE Transactions on Robotics*, 30(2):289 – 309, 2014.

[13] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. *Proc. IEEE International Conference on Robotics and Automation*, pages 521 – 528, 2000.

[14] O. Brock, O. Khatib, and S. Viji. Task-consistent obstacle avoidance and motion behavior for mobile manipulation. In *Proc. IEEE International Conference on Robotics and Automation*, volume 1, pages 388–393, 2002.

[15] J. Butterfa, M. Grebenstein, H. Liu, and G. Hirzinger. DLR-Hand II: Next generation of a dextrous robot hand. *Proceedings of IEEE International Conference on Robotics and Automation*, 2001.

[16] D. B. Camarillo, C. F. Milne, C. R. Carlson, and M. R. Zinn. Mechanics modeling of tendon-driven continuum manipulators. *IEEE Transactions on Robotics*, 24(6):1262 – 1273, 2008.

[17] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, 6:291–302, 1990.

[18] B. Chazelle. Approximation and decomposition of shapes. *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 145 – 185, 1987.

[19] B. Chazelle. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14:105–116, 1995.

[20] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Chapter 15: Visibility graphs. *Computational Geometry, Springer-Verlag*, pages 307 – 317, 2000.

[21] A. Degani, H. Choset, A. Wolf, and M. A. Zenati. Highly articulated robotic probe for minimally invasive surgery. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 4167 – 4172, 2006.

[22] R. Detry, E. Baseski, N. Krüger, M. Popovic, Y. Touati, O. Kroemer, J. Peters, and J. Piater. Learning object-specific grasp affordance densities. *In IEEE Int. Conf. on Development and Learning*, pages 1 – 7, 2009.

[23] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269 – 271, 1959.

[24] P. E. Dupont, J. Lock, B. Itknowitz, and E. Butler. Design and control of concentric tube robots. *IEEE Trans. Robotics*, 26:209–225, 2010.

[25] D. Eberly. Distance between two line segments in 3d, 1999. www dot geometrictools dot com/.

[26] C. Ferrari and J. Canny. Planning optimal grasps. *Proc. IEEE International Conference on Robotics and Automation*, pages 2290–2295, 1992.

[27] A. Foisy and V. Hayward. A safe swept volume method for collision detection. *The Sixth Int. Symp. of Robotics Research*, pages 61–68, 1993.

[28] I. S. Godage, E. Guglielmino, and D. T. Branson. Novel modal approach for kinematics of multisection continuum arms. *Proc. IEEE/RSJ Internaltion Conference Intelligent Robots and Systems*, 2011.

[29] S. Gottschalk, M. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Computer Graphics, Proceedings of SIGGRAPH'96*, pages 171–180, 1996.

[30] Y. Gutfreund, T. Flash, Y. Yarom, G. Fiorito, I. Segev, and B. Hochner. Organization of octopus arm movements: a model system for studying the control of flexible arms. *J. of Neurosci.*, 16:7297 – 7307, 1996.

[31] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100 – 107, 1968.

[32] S. Hirose. Biologically inspired robots. *Oxford University Press*, 1993.

[33] S. C. Jacobsen, E. K. Iversen, D. Knutti, R. Johnson, and K. Biggers. Design of the Utah/M.I.T. dextrous hand. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1520–1532, 1986.

[34] Y. Jia, F. Guo, and H. Lin. Grasping deformable planar objects: Squeeze, stick/slip analysis, and energy-based optimalities. *International Journal of Robotics Research*, 33:866–897, 2014.

[35] M. W. Jones. 3d distance from a point to a triangle. Technical Report CSR-5-95, Department of Computer Science, University of Wales Swansea, 1995.

[36] R. Kang, D. Branson, T. Zheng, E. Guglielmino, and D. Caldwell. Design, modeling and control of a pneumatically actuated manipulator inspired by biological continuum structures. *Bioinspiration&Biomimetics*, 2013.

[37] L. E. Kavraki, P. Svestka, J. C.Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[38] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90 – 98, 1986.

[39] J. Klein and G. Zachmann. Point cloud collision detection. *Computer Graphics Forum*, 23 (3):567 – 576, 2004.

[40] G. Kootstra, M. Popovic, J. Jorgensen, K. Kuklinski, K. Miatliuk, D. Kragic, and N. Kruger. Enabling grasping of unknown objects through a synergistic use of edge and surface information. *International Journal of Robotics Research*, 31:1190 – 1213, 2012.

[41] K.Xu and X.Zheng. Configuration comparison for surgical robotic systems using a single access port and continuum mechanisms. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3367 – 3374, 2012.

[42] K. Lakshminarayana. Mechanics of form closure. *Amer. Soc. Mech. Eng. Tech. Rep.*, 1978.

[43] P. Laplante and S. Ovaska. *Real-Time Systems Design and Analysis: Tools for the Practitioner*. IEEE Press, Wiley, 4th edition, 2011.

[44] T. Larsson and T. Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Workshop On Virtual Reality Interaction and Physical Simulation*, 2005.

[45] J. C. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[46] C. Lauterbach, S. Yoon, and D. Manocha. Rt-deform: Interactive ray tracing of dynamic scenes using bvhs. In *In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.

[47] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Technical report, Iowa State University*, 1998.

[48] S. M. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006.

[49] H. Lewis and C. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 1981.

[50] J. Li, Z. Teng, and J. Xiao. Can a continuum manipulator fetch an object in an unknown cluttered space? *submitted to IEEE Robotics and Automation Letters*, 2015.

[51] J. Li, Z. Teng, J. Xiao, A. Kapadia, A. Bartow, and I. Walker. Autonomous continuum grasping. *Proc. IEEE/RSJ Internaltion Conference on Intelligent Robots and Systems*, 2013.

[52] J. Li and J. Xiao. Determining grasping configurations for a spatial continuum manipulator. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4207–4214, 2011.

[53] J. Li and J. Xiao. Progressive, continuum grasping in cluttered space. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[54] J. Li and J. Xiao. Progressive, continuum grasping in cluttered space. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4563–4568, 2013.

[55] J. Li and J. Xiao. Progressive generation of force-closure grasps for an n-section continuum manipulator. *Proc. of IEEE International Conference on Robotics and Automation*, pages 4016–4022, 2013.

[56] J. Li and J. Xiao. Task-constrained continuum manipulation in cluttered space. *Proc. of IEEE International Conference on Robotics and Automation*, pages 2183–2188, 2014.

[57] J. Li and J. Xiao. An efficient algorithm for real time collision detection involving a continuum manipulator with multiple uniform-curvature sections. *Robotica*, FirstView Article:1–21, 2014, DOI: 10.1017/S0263574714002458.

[58] J. Li and J. Xiao. A general formulation and approach to constrained, continuum manipulation. *Advanced Robotics*, Special Issue: Continuum robots and manipulation, 29(13):889–899, 2015, DOI: 10.1080/01691864.2015.1052846.

[59] J. Li, J. Xiao, R. Grizzi, and J. Lindberg. Inspection in cluttered space with a continuum manipulator. *Video submission to International Conference on Applied Robotics for the Power Industry (CARPI)*, Foz do Iguassu, Brazil, Oct. 2014.

[60] E. Lutscher and G. Cheng. Constrained manipulation in unstructured environment utilizing hierarchical task specication for indirect force controlled robots. *Proc. of IEEE International Conference on Robotics and Automation*, pages 3471–3476, 2014.

[61] N. Magiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Comp.*, 12:759–776, 1983.

[62] A. D. Marchese, R. K. Katzschmann, and D. Rus. Whole arm planning for a soft and highly compliant 2d robotic manipulator. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 554 – 560, 2014.

[63] W. McMahan, V. Chitrakaran, M. Csencsits, D. Dawson, I. Walker, B. Jones, M. Pritts, D. Dienno, M. Grissom, and C. Rahn. Field trials and testing of the octarm continuum manipulator. *Proc. IEEE International Conference on Robotics and Automation*, pages 2336–2341, 2006.

[64] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. *Proc. IEEE International Conference on Robotics and Automation*, pages 1824 – 1829, 2003.

[65] E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26 (9):394 – 395, 1920.

[66] S. Neppalli, M. A. Csencsits, B. A. Jones, and I. Walker. A geometrical approach to inverse kinematics for continuum manipulators. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[67] C. L. Nielsen and L. E. Kavraki. A two level fuzzy PRM for manipulation planning. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[68] T. Omata and K. Nagata. Rigid body analysis of the indeterminate grasp force in power grasps. *IEEE Trans. Robot. Autom.*, 16:46 – 54, 2000.

[69] J. Pan, S. Chitta, and D. Manocha. Probabilistic collision detection between noisy point clouds using robust classification. *International Symposium on Robotics Research (ISRR)*, 2011.

[70] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 31:538 – 553, 2012.

[71] C. Pellerin. The salisbury hand. *Industrial Robot: An International Journal*, 1991.

[72] R. Pelossof, A. Miller, P. Allen, and T. Jebara. An svm learning approach to robotic grasping. *Proc. IEEE International Conference on Robotics and Automation*, pages 3512 – 3518, 2004.

[73] H. Ren and P. E. Dupont. Tubular enhanced geodesic active contours for continuum robot detection using 3d ultrasound. *Proceedings of IEEE International Conference on Robotics and Automation*, 2012.

[74] G. Robinson and J. B. C. Davies. Continuum robots a state of the art. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 2849 – 2854, 1999.

[75] A. Rodriguez, M. T. Mason, and S. Ferry. From caging to grasping. *The International Journal of Robotics Research (IJRR)*, 31(7):886–900, June 2012.

[76] E. L. Sauser, B. D. Argall, G. Metta, and A. G. Billard. Iterative learning of grasp adaptation through human corrections. *Robotics and Autonomous Systems*, 60(1), 2011.

[77] A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 27:157 – 173, 2008.

[78] F. Schwarzer, M. Saha, and J. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Transactions on Robotics*, 21:338–353, 2005.

[79] A. Schweikard. Polynomial time collision detection for manipulator paths specified by joint motions. *IEEE Transactions on Robotics and Automation*, pages 865–870, 1991.

[80] S. Sentis and O. Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2:505–518, 2005.

[81] G. Song, S. L. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. *Proc. IEEE International Conference on Robotics and Automation*, pages 1500 – 1505, 2001.

[82] M. Stilman. Global manipulation planing in robot joint space with task constraints. *IEEE Transactions on Robotics*, 26:576–584, 2010.

[83] M. Tang, S. Curtis, S. Yoon, and D. Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 25–36, New York, NY, USA, 2008. ACM.

[84] Z. Teng and J. Xiao. Surface-based general 3D object detection and pose estimation. *Proc. IEEE International Conference on Robotics and Automation, Hong Kong, China*, pages 5473–5479, 2014.

[85] P. Terdiman. OPCODE: Optimized collision detection. Available: www dot codercorner dot com/OPCODE dot htm, 2003.

[86] L. G. Torres and R. Alterovitz. Motion planning for concentric tube robots using mechanics-based models. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5153–5159, 2011.

[87] L. G. Torres, C. Baykal, and R. Alterovitz. Interactive-rate motion planning for concentric tube robots. *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1915–1921, 2014.

[88] L. G. Torres, R. J. Webster, and R. Alterovitz. Task-oriented design of concentric tube robots using mechanics-based models. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4449–4455, 2012.

[89] J. C. Trinkle. The mechanics and planning of enveloping grasps. *PH.D. Thesis*, University of Pennsylvania, Philadelphia 1987.

[90] D. Trivedi, C. D. Rahn, W. M. Kier, and I. D. Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Applied Bionics and Biomechanics*, 5(3):99–117, 2008.

[91] K. Trovato and A. Popovic. Collision-free 6d non-holonomic planning for nested cannulas. *Proc. SPIE Medical Imaging*, Volume 7261, 2009.

[92] S. Tully, A. Bajo, G. Kantor, H. Choset, and N. Simaan. Constrained filtering with contact detection data for the localization and registration of continuum robots in flexible environments. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3388 – 3394, 2012.

[93] G. Turk and M. Levoy. Zippered polygon meshes from range images. *SIGGRAPH*, pages 311–318, 1994.

[94] G. van den Bergen. Efcient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–13, 1998.

[95] G. van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[96] J. Vannoy and J. Xiao. Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes. *IEEE Transactions on Robotics*, 24:1199–1212, 2008.

[97] R. Vatcha and J. Xiao. Perceiving guaranteed continuously collision-free robot trajectories in an unknown and unpredictable environment. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1433–1438, 2009.

[98] P. Volino and N. M. Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum (EuroGraphics Proc.)*, 13:155–166, 1994.

[99] B. Walker and I. Walker. Kinematics for multi-section continuum robots. *IEEE Transactions on Robotics*, 22:45–53, 2006.

[100] I. D. Walker. Continuous backbone continuum robot manipulators. *ISRN Robotics*, 2013.

[101] R. J. Webster and B. A. Jones. Design and kinematic modeling of constant curvature continuum robots: a review. *International Journal of Robotics Research*, 2010.

[102] R. J. Webster, J. M. Romano, and N. J. Cowan. Mechanics of precurved-tube continuum robots. *IEEE Trans. Robot.*, 25(1), 2009.

[103] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. *Proc. IEEE International Conference on Robotics and Automation*, pages 1024 – 1031, 1999.

[104] J. Xiao and R. Vatcha. Real-time adaptive motion planning for a continuum manipulator. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5919–5926, 2010.

[105] K. Xu and N. Simaan. Actuation compensation for flexible surgical snake-like robots with redundant remote actuation. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 4148 – 4154, 2006.

[106] J. H. Yakey, S. M. LaValle, and L. E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17:951–958, 2001.

[107] Y. Yamamoto and X. Yun. Coordinated obstacle avoidance of a mobile manipulator. In *Proc. IEEE International Conference on Robotics and Automation*, volume 3, pages 2255–2260, 1995.

[108] Z. Yao and K. Gupta. Path planning with general end-effector constraints. *Robotics and Autonomous Systems*, pages 316–327, 2007.

[109] G. Zachmann and R. Weller. Kinetic bounding volume hierarchies for deforming objects. *ACM Int'l Conf. on Virtual Reality Continuum and its Applications*, 2006.

[110] D. Zwillinger. *CRC Standard Mathematical Tables and Formulae, 31st edition*. Chapman and Hall/CRC, 2002.

APPENDIX A: PROOFS OF ALGORITHMS IN CHAPTER 4

Completeness of Algorithm 4

In Algorithm 4, all possible cases are enumerated based on the positions of the two vertices v_1 and v_2 of l^i, described in polar coordinates (ρ, θ). Each vertex may either be within the angular bound (see inequality (7)) or be out of it, indicated by "in" or "out" respectively. Each vertex may be above the upper bound of the radius of the fan-shaped cs_i, within the radius bounds, or below the lower bound of the radius (see inequality (6)), indicated by "a", "in" and "b" respectively. Therefore, each vertex can have $2 * 3 = 6$ different arrangements and l^i, which contains two vertices, can have a total of $6^2 = 36$ possible arrangements. The following table shows that all of them have been considered in Algorithm 4.

Table 24: All cases considered in Algorithm 4

θ_1	θ_2	ρ_1	ρ_2	Cases in Alg. 4	# configs
-	-	b	b	Case 1	4
in	-	in	-	Case 2	6
out	in	-	in	Case 2	3
in	in	a/b	in	Case 2	2
in	in	a	a	Case 3	1
out	out	a	a	Case 3	1
in	in	a	b	Case 4	1
in	in	b	a	Case 4	1
in	out	a	-	Case 5(e)	3
in	out	b	a/in	Case 5(f)	2
out	in	a/in	b	Case 5(f)	2
out	in	-	a	Case 5(e)	3
out	out	a	b/in	Case 5(g)	2
out	out	b	a/in	Case 5(g)	2
out	out	in	-	Case 5(g)	3
					Sum: 36

Completeness of Algorithm 5

Algorithm 5 first computes the minimum distance between cir_i and the supporting plane Q of face f by calling *Procedure 1*, and also obtains corresponding closest points p, q on cir_i and Q respectively. If the minimum distance $d_{min}(cir_i, Q) > w_i$, Algorithm 5 returns "$Collision$ $\leftarrow$ False"; otherwise (when $d_{min}(cir_i, Q) \leq w_i$), based on whether q is on f and whether p is on seg_i Algorithm 5 partitions all situations into the following 3 cases:

(1) *q is on f but p is not on seg_i*:

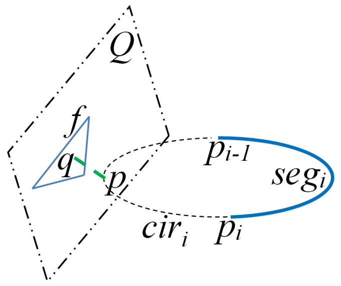
In this case the minimum distance between seg_i and f is the shortest distance from the two endpoints p_i or p_{i-1} to f, as shown in Fig. 86(a), and computed by *Procedure 2*. If $d_{min}(p_{i/i-1}, f) \leq w_i$ and f is within (or intersects) the section bounding volume defined by H_i and H_{i-1}, Algorithm 5 returns "$Collision$ $\leftarrow$ True", otherwise returns "$Collision$ $\leftarrow$ False".

(2) *q is not on f*:

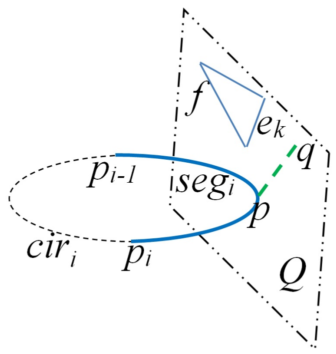
In this case the closest point q on f to seg_i is on one edge e_k of f, as shown in Fig. 86(b), which is computed by *Procedure 3*. If $d_{min}(e_k, f) \leq w_i$ and f is within (or intersects) the section bounding volume defined by H_i and H_{i-1}, Algorithm 5 returns "$Collision$ $\leftarrow$ True", otherwise returns "$Collision$ $\leftarrow$ False".

(3) *q is on f and p is on seg_i*:

In this case $d_{min}(cir_i, Q)$ is the minimum distance between seg_i and f. If f is out of section bounding volume, the algorithm returns "$Collision$ $\leftarrow$ False"; otherwise it returns "$Collision$ $\leftarrow$ True".



(a) q is on f but p is not on seg_i



(b) q is not on f

Figure 86: Examples of cases (1) and (2) above.

Proof for Correctness of *Procedure 1*:

If Q and P are not parallel and intersect each other at a line l_{int}, we denote p_j as any point (other than p) on cir_i and q_j as its projection on Q and prove that the distance from p to Q (i.e., $|\overline{pq}|$) is always shorter than that from p_j to Q (i.e., $|\overline{p_j q_j}|$) – See Fig. 87 for an illustration.

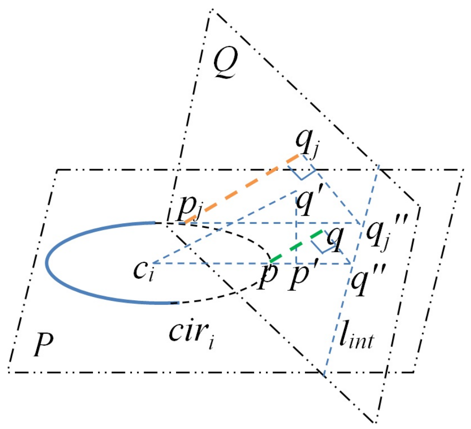


Figure 87: $|\overline{pq}|$ is the shortest distance from cir_i to Q, and the orange dashed line segment indicates the distance from any other point p_j on cir_i to Q.

Since $\overline{c_i q'}$ is perpendicular to Q and $\overline{q'p'}$ is perpendicular to P, the triangle $\triangle c_i q' p'$ defines a plane A perpendicular to both Q and P and also to the line l_{int} and intersects l_{int} at point q''. Since p is on $\overline{c_i p'}$ and $\overline{pq}$ is perpendicular to Q, p and q are also on plane A, and so is the triangle $\triangle pqq''$. Now, by connecting $\overline{p_jq_j}$ and l_{int} with their common normal, we get point q_j'' on l_{int}, and the triangle $\triangle p_jq_jq_j''$ also defines a plane perpendicular to l_{int}. Thus, two right-angled triangles $\triangle c_iq'p'$ and $\triangle p_jq_jq_j''$ are parallel, and since $\angle qq''p = \angle q_jq_j''p_j$ is the angle between Q and P, $\triangle p_jq_jq_j'' \sim \triangle pqq''$. Since $|\overline{pq''}|$ is the shortest distance to l_{int} from cir_i, $|\overline{pq''}| < |\overline{p_jq_j''}|$, and therefore $|\overline{pq}| < |\overline{p_jq_j}|$.

APPENDIX B: DERIVATIONS FOR EQUATIONS IN CHAPTER 8

Derivation of Equation (41)

As shown in Fig. 74, we denote q' as the middle point of line segment $\overline{uv}$, c and r as the center and radius of arc, then we will have $\angle ucq' = \alpha$.

In the right angle triangle $\triangle q'uc$, we have,

$$tan(\alpha) = \frac{l}{2(r-h)} \tag{46}$$

We also have,

$$r^2 = (\frac{l}{2})^2 + (r-h)^2$$

which can be simplified as,

$$r = \frac{4h^2 + l^2}{8h} \tag{47}$$

From (47) and (46), we can derive Equation (41).

Also if we denote $\mathbf{I}$ as the unit vector pointing from q' to c, we can further derive:

$$\mathbf{c} = \mathbf{q}' + (r-h)\mathbf{I}, \tag{48}$$

where $\mathbf{c}$ and $\mathbf{q}'$ represent the position vector of c and q' respectively.

Derivation of Equation (44)

: Basically, as shown in Fig. 88, a trihedral angle is formed at u_j by three planes A_{j-1}, A_j and the one containing u_{j-1}, u_j and v_j. If we denote the dihedral angle between A_{j-1}

and A_j as $\gamma_{j-1,j}$, then according to the law of cosines for a trihedral angle [110], we have:

$$cos\gamma_{j-1,j}sin\alpha_{j-1}sin\alpha_j + cos\alpha_{j-1}cos\alpha_j = cos(\pi - \beta_{j-1,j}) \tag{49}$$

where $cos\gamma_{j-1,j}$ can be expressed by the unit normals of planes A_{j-1} and A_j as:

$$cos\gamma_{j-1,j} = \mathbf{n}_{A_{j-1}} \cdot \mathbf{n}_{A_j} \tag{50}$$

By substituting (50) into (49), we have Equation (44).

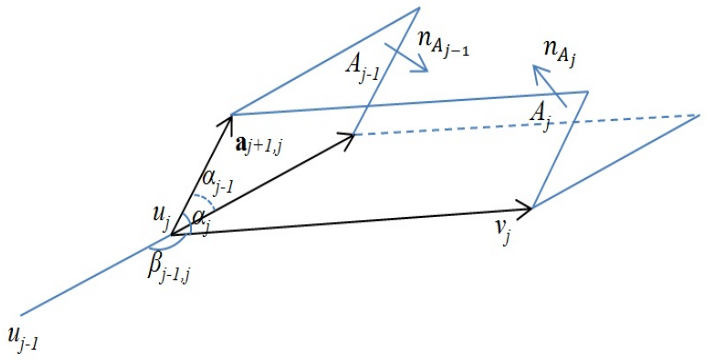


Figure 88: A trihedral angle formed by three planes A_{j-1}, A_j, and the one containing u_{j-1}, u_j and v_j.