2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Progressive, Continuum Grasping in Cluttered Space

Jinglin Li and Jing Xiao
Department of Computer Science
University of North Carolina at Charlotte
jli41@uncc.edu, xiao@uncc.edu

***Abstract*— Continuum manipulators, inspired by invertebrate structures in nature, such as octopus arms and elephant trunks, do not contain rigid links, can deform, and are passively compliant, which make them particularly flexible for manipulation in cluttered space. A key open issue here is how to make such a manipulator autonomously grasp an object in cluttered space, especially if the object cannot be completely seen or known before being grasped. In this paper, we address this issue by introducing an approach that enables a multi-section continuum manipulator to probe an object with its tip while gradually form a whole-arm, force-closure grasp by following closely the contour of the probed object. This real-time approach is both effective and efficient for grasping an object in a cluttered space, as evident from the test examples.**

I. Introduction

Unlike conventional articulated manipulators, continuum manipulators [1], [2] are inspired by invertebrate structures found in nature, such as octopus arms [3] and elephant trunks [4]. As such they are whole-arm manipulators, and their shapes can continuously deform to adapt to grasping a wide range of objects of different sizes and shapes via changing the controllable or active degrees of freedom. Moreover, they are also passively compliant due to their infinite number of passive degrees of freedom. Almost all continuum robots feature constant-curvature sections (modulo external loading due to gravity or payload) [2] because of actuating the (theoretically infinite) degrees of freedom of the continuously bendable backbone with finite actuators. A representative continuum manipulator is the OctArm (Fig. 1). Smaller scale continuum manipulators are also developed for medical surgery applications [5], [6].

Fig. 1. An OctArm manipulator (by the courtesy of Ian Walker)

Research on autonomous grasping for continuum manipulators has been fairly recent, and the existing methods [7]–[9] are not specifically concerned with limiting the motion of the manipulator in a tight space. [7] determines grasping poses of the OctArm manipulator based on some bounding circle of the target object. [8] introduces an approach to progressively generate a spiral, force-closure grasp for a multi-section continuum manipulator by wrapping the manipulator around the target object section by section, from a section close to the base to the tip section. However, this approach requires sufficient free space to accommodate a fairly open (i.e., unwrapped) and extended initial arm pose and the sweeping motion of each arm section. Such kind of a space may not be available in a cluttered environment.

In this paper, we tackle the problem of continuum grasping constrained by a tight space in a cluttered environment and introduce an algorithm that enables a multi-section continuum manipulator to probe an object with its tip while gradually form a force-closure grasp by molding the arm closely along the contour of the probed object. The object may not even be fully visible before the simultaneous probing and wrapping begins. 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.

The rest of the paper is organized as follows. Section II reviews the manipulator model, introduces assumptions about objects, and explains detections of contacts or collisions between the manipulator and objects. Section III describes our real-time grasping approach. Section IV presents experimental results. Section V concludes this paper and presents possible future research.

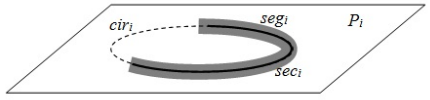
II. Manipulator and Objects

As an example of a multi-section continuum manipulator, let us consider the OctArm, which consists of three sections (see Fig. 1). Each section has a constant curvature, which can change values continously [10]. If a section has a non-zero curvature, then it is a truncated torus (when not in contact), with its central axis bent into a circular curve, and if the curvature becomes zero, the section becomes a (straight) cylinder. The length of each section can also be contracted

or extended. The orientation of each section, indicated by an angle with respect to an adjacent section, can also be changed. Curvature, length, and orientation angles are the three controllable variables for each section.

A. Manipulator model

Inspired by the OctArm, we use the following model for an n-section continuum manipulator. We represent the i-th section, 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. 2.

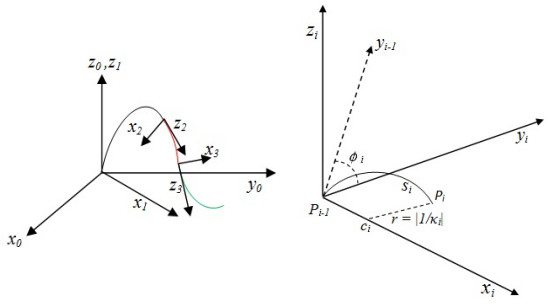


Solid curve: seg_i Dotted circle: cir_i Partial torus: sec_i

Fig. 2. 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 seg_1. The frame of section sec_i is formed at p_{i-1} with the z_i axis tangent to seg_i at p_{i-1}. The base of seg_i is the tip of seg_{i-1}. Adjacent seg_{i-1} and seg_i are connected *tangentially* at the connection point p_{i-1} as shown in Fig. 3(a), i.e., the two sections share the same tangent at p_{i-1}.

Each section i has three degrees of freedom, or controllable variables: curvature κ_i, length s_i, and orientation angle ϕ_i from y_{i-1} axis to y_i axis about z_i axis. Fig. 3(b) shows one example seg_i, its frame, and controllable variables.



(a) Frames of different sections (b) Section i and its controllable variables

Fig. 3. Section frames

The configuration of the entire arm is determined by the control variables of each section. We denote an n-section continuum arm configuration as $C = \{(s_1, \kappa_1, \phi_1), ..., (s_n, \kappa_n, \phi_n)\}$.

B. Objects in the environments

An object in the environment is modeled as a polygonal mesh. To wrap around an object, each section sec_i of the manipulator can be viewed as following the cross section polygon $poly_i$ of the object mesh by the plane P_i of sec_i, as shown in Fig. 4

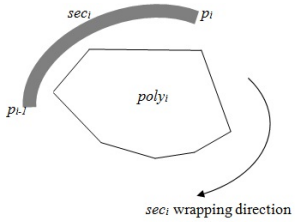
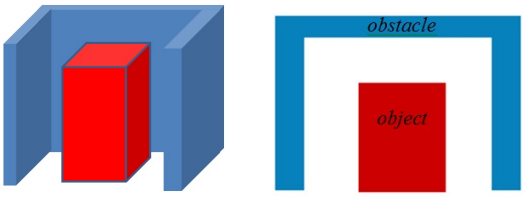


Fig. 4. The manipulator section sec_i and $poly_i$ of the target object

If a target object for grasping is in a cluttered space, as shown in Fig. 5, not all parts of the object may be visible by the manipulator before grasping. This is modeled as not all vertices of $poly_i$ is visible for arm section sec_i.



(a) A "U" shape obstacle (blue) surrounding a target object (red) (b) Top view of an obstacle (blue), a target object (red)

Fig. 5. Obstacle (blue) and the target object (red)

C. Determination of feasible configurations and contacts

For continuum manipulation in a cluttered environment, we need to make sure that the manipulator arm will make necessary contacts with the target object for grasping while avoiding collisions with other objects in the environment. We call a manipulator configuration that neither penetrates into the target object nor collides with obstacles a *feasible configuration*. A feasible configuration of the manipulator arm may involve multiple contacts with the target object. We use the Algorithm 1 in [8] to check whether a given arm configuration is feasible or not, which also returns the list of pairs of contact points between the manipulator and the target object for a feasible configuration. This algorithm builds upon an efficient method that we introduced in [11] to detect collisions between a continuum manipulator and objects. It further determines whether a contact happens by checking the minimum distance between the continuum manipulator and the target object; a contact happens if the minimum distance is less than a small threshold $d_{contact}$.

III. Grasp Generation

Now we are ready to introduce our on-line algorithm to generate progressively a whole-arm, force-closure grasp of

an object in a cluttered space with a continuum manipulator. 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. 6). The manipulator is put close to the nearby obstacle to maximize the clearance from the target object for maneuverability.

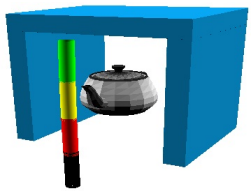


Fig. 6. The initial configuration of the manipulator besides the target object (teapot)

A. Main algorithm

The algorithm is outlined in **Algorithm 1**. It moves every section of the arm in small steps repeatedly to follow the contour of the target object while avoiding penetration into the object or collision with surrounding obstacles, led by the tip of the arm. We call such steps *feasible moves*. The **Algorithm 1** calls **Algorithm 2** 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.

Each section can move at most max steps, where max is computed 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 2** 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 max steps.

After all arm sections have reached their limits in curling and extending, if a force-closure grasp [12] is achieved, **Algorithm 1** returns the entire path of knot configurations that leads to the force-closure grasp. Otherwise, it reports that no force-closure grasp is found.

B. Generation of a knot configuration

Algorithm 2 curls and extends arm section sec_i in a small step towards a targeted vertex 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 $poly_i$, and v_m be the farthest visible vertex that has not been reached. **Algorithm 2** chooses a target vertex v between v_1 and v_m (see Fig. 7) and tries to move sec_i's tip towards v without penetrating into the object or colliding with nearby obstacles.

Algorithm 1: *GenGrasp*

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 ← C_0;
3      path ← {C_0};
4      count ← 0;
5      while count < max do
6          for section i ← 1 to n do
7              call Algorithm 2 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 ← path ∪ {C_new};
9              contactlist ← ∪_1^n conlist_k;
10             C_current ← 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 ← count + 1;
19     end
20     return "no force-closure grasp is found";
21 end
```

How to choose a target vertex v among $v_1, .., v_m$ is an interesting issue. We experimented with three strategies: (1) searching and choosing the farthest visible and reachable vertex along the wrapping direction, (2) searching and choosing the nearest visible and reachable vertex, and (3) randomly searching and choosing a reachable vertex between the nearest and the farthest visible vertices. Fig. 7 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 [13]) 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 2** returns "$wrapping\ paused$" for sec_i.

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

Algorithm 2: $SecWrap$

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}

let $V = \{v_1, ..., v_m\}$ be the ordered list of unreached and visible vertices of $poly_i$, as shown in Fig. 7;
if $V = \varnothing$ **then**
  return "$wrapping\ paused$";
end
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;
if *no vertex in* V *is reachable* **then**
  return "$wrapping\ paused$";
end
compute a new configuration C_{new} after making one step linearly towards C_r from $C_{current}$ and check if C_{new} is feasible;
call **Algorithm 3** to repair an infeasible C_{new};
if "$repair\ fails$" **then**
  return "$wrapping\ paused$";
end
$cList \leftarrow$ the list of contact points and normals at configuration C_{new} (see section II.C);
return C_{new} and $cList$.

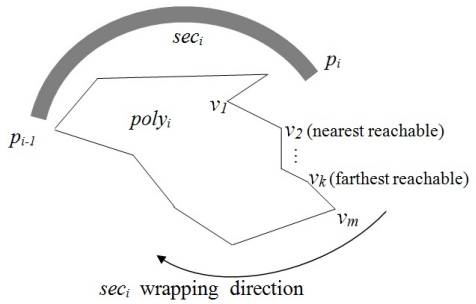


Fig. 7. The unreached vertices of $poly_i$ as an ordered list $\{v_1, v_2, ..., v_m\}$ in the wrapping direction of sec_i, where the nearest and farthest reachable vertices in the visible range are indicated

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. In section IV, we will further present and discuss the experimental results comparing the three strategies.

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.

C. Repair of an infeasible knot configuration

If the new knot configuration C_{new} generated is not feasible, then **Algorithm 3** is called to repair the configuration to obtain a feasible one. **Algorithm 3** categorizes infeasible cases in the following four types and takes different repair actions on an infeasible section sec_i accordingly.

- **Case 1**: arm section sec_i collides with obstacles at tip point p_i, as shown in Fig. 8 (a).
- **Case 2**: arm section sec_i collides with obstacles at a point between base point p_{i-1} and tip p_i, as shown in Fig. 8 (b).
- **Case 3**: arm section sec_i penetrates the object at tip point p_i, as shown in Fig. 8 (c).
- **Case 4**: arm section sec_i penetrates the object at a point between base point p_{i-1} and tip p_i, as shown in Fig. 8 (d).

If a collision occurs near the tip p_i of sec_i, repairing simply involves either curling or flattening sec_i. Curling is achieved by increasing the curvature of sec_i, and flattening means decreasing the curvature of sec_i. However, if a collision occurs near the base p_{i-1} of sec_i, repairing also involves adjusting the neighboring arm section sec_{i-1}, by either curling, flattening, or extending the length of sec_{i-1}. If sec_i collides with itself or another arm section, a (physical) loop is formed by the arm section(s). Thus, the algorithm increases the orientation angle ϕ_i of sec_i to break the loop and form a spiral wrap around the object.

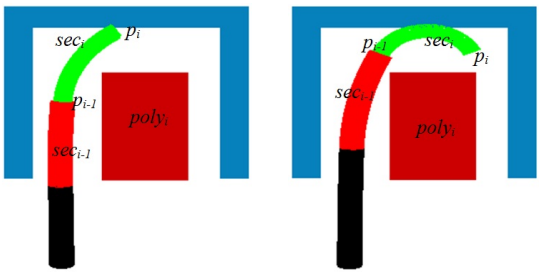
Note that repairing starts from the infeasible section of smallest index, i.e., the infeasible section closest to the base of the manipulator. Let sec_i be such a section. After sec_i is repaired, the other infeasible sections sec_{i+k}, where $0 < k \leq n\text{-}i$ is an integer, may also be repaired. If not, or if a section newly becomes infeasible due to the repair of sec_i, **Algorithm 3** 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 "$repair fails$", since repairing other infeasible sections sec_{i+k} will not change the pose of sec_i and thus will not make sec_i feasible.

IV. Implementation and Tests

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 four-section continuum manipulator for the task of grasping a teapot in a tight space as shown in Fig. 6. We tested the three alternative strategies for choosing target vertices used in **Algorithm 2**, as described in Section III.B, and compared the results, as shown in Table I.

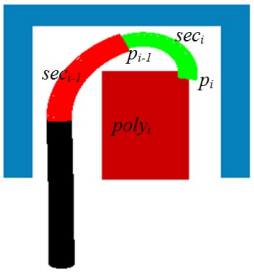
We use the following parameters of performance:

- # 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;
- Total planning time: the total time needed for generating all knot configurations in a path;

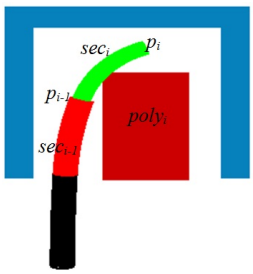


(a) **Case 1:** sec_i (green) collides with obstacle at sec_i's tip p_i

(b) **Case 2:** sec_i (green) collides with obstacle between p_{i-1} and p_i



(c) **Case 3:** sec_i (green) penetrates with object at sec_i's tip p_i



(d) **Case 4:** sec_i (green) penetrates object between p_{i-1} and p_i

Fig. 8. Illustration of four infeasible cases between an arm section and the object or the obstacles

- Avg. # collision detection: the average number of collision detections performed in generating a feasible knot configuration;
- Total path length: the sum of distances between adjacent knot configurations in a path.

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 I, 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. 9. The length of the path generated by strategy (3) is between those of strategies (1) and (2). Aslo 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 (among the visible vertices) and the corresponding feasible configuration, which also involves more collision checks. The attached video also compares the motions generated by these three different strategies respectively.

Fig. 10 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.

Algorithm 3: *RepairConfiguration*

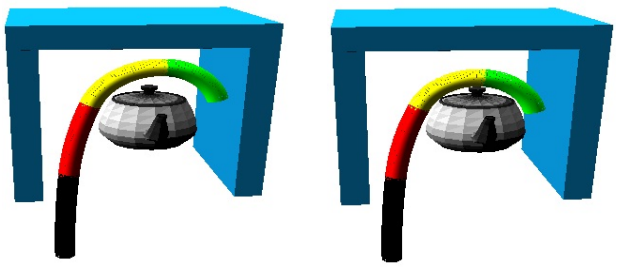
input : infeasible configuration C, the set of infeasible arm sections S, and models of colliding objects
output: a feasible arm configuration C_{new} or "*repair fails*"

```
repeat
    sec_i ← section of smallest index in S;
    count ← 0;
    while sec_i is infeasible do
        if count = #tries + 1 then
            return "repair fails";
        end
        Case 1: update C by curling sec_i with a small step;
        Case 2: update C by curling sec_{i-1}, i > 1, and sec_i with small steps;
        Case 3: update C by flattening sec_i with a small step;
        Case 4: update C by flattening and extending sec_{i-1}, i > 1;
        if sec_i collides with itself or another arm section then
            increase |φ_i| by a small δφ;
        end
        if C is out of the physical limits of the arm then
            return "repair fails";
        end
        count ← count + 1;
    end
    update S by deleting sec_i and adding new infeasible arm sections (if any) at configuration C;
until S = ∅;
C_new ← C;
return C_new.
```

TABLE I

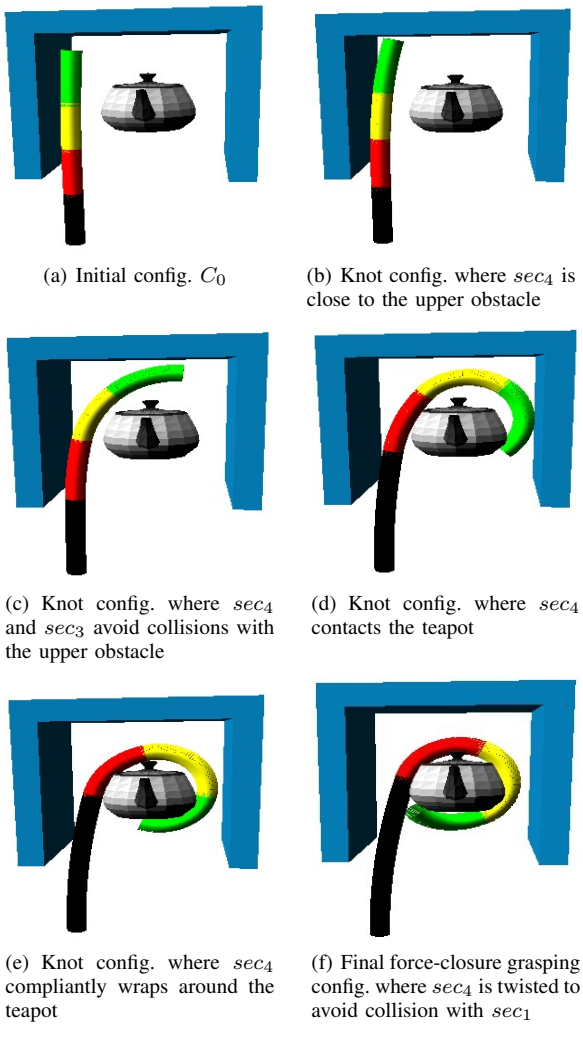
COMPARISON OF RESULTS FROM USING THE THREE ALTERNATIVE STRATEGIES IN **Algorithm 2**

Strategies to choose a vertex	# knot config.	Planning time per knot	Total planning time	Avg.# collision checks	Path length
Strategy (1) (farthest)	50	26(ms)	$1.3(s)$	14	$311.41(cm)$
Strategy (2) (nearest)	210	12.38 (ms)	$2.6(s)$	6	$279.62(cm)$
Strategy (3) (random)	160	13.75 (ms)	$2.2(s)$	7	$298.49(cm)$



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

Fig. 9. 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

Fig. 10. Snapshots of knot configurations leading to a force-closure grasp by a four-section continuum manipulator while avoiding obstacles

V. CONCLUSIONS AND FUTURE WORK

We have introduced a real-time approach for continuum, whole-arm grasping of objects in cluttered environments. Our approach allows an n-section continuum manipulator to progressively generate a force-closure grasp by following closely the contour of the target object in a narrow space. It enables 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, e.g., a camera at its tip. Our approach can also be extended to inspection applications in cluttered environments. For the next step, we plan to incorporate external sensors and apply our method to grasping in cluttered real-world environments with a real continuum manipulator.

ACKNOWLEDGMENT

This work is supported by the US National Science Foundation grant IIS-0904093.

REFERENCES

[1] G. Robinson and J. B. C. Davies, "Continuum robots - a state of the art," *Proc. CDEN Design Conf.*, pp. 2849–2854, 1999.
[2] 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*, vol. 5(3), pp. 99–117, 2008.
[3] W. McMahan, B. A. Jones, I. D. Walker, V. Chitrakaran, A. Seshadri, and D. Dawson, "Robotic manipulators inspired by cephalopod limbs," *Proc. CDEN Design Conf.*, pp. 1–10, 2004.
[4] R. Cieslak and A. Moreck, "Elephant trunk type elastic manipulator a tool for bulk and liquid type materials transportation," *Robotica*, vol. 17, pp. 11–16, 1999.
[5] R. J. Webster, J. M. Romano, and N. J. Cowan, "Mechanics of precurved-tube continuum robots," *IEEE Trans. Robot.*, vol. 25(1), 2009.
[6] J. Furusho, T. Katsuragi, T. Kikuchi, T. Suzuki, H. Tanaka, Y. Chiba, and H. Horio, "Curved multi-tube systems for fetal blood sampling and treatments of organs like brain and breast," *J. Comput. Assist. Radiol. Surg*, vol. 1, pp. 223–226, 2006.
[7] J. Li and J. Xiao, "Determining "grasping configurations for a spatial continuum manipulator," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2011.
[8] J. Li and J. Xiao, "Progressive generation of force-closure grasps for an n-section continuum manipulator," *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013.
[9] J. Xiao and R. Vatcha, "Real-time adaptive motion planning for a continuum manipulator," *IROS*, 2010.
[10] B. A. Jones and I. D. Walker, "Kinematics for multisection continuum robots," *IEEE Trans. Robot.*, vol. 22, pp. 43–55, 2006.
[11] J. Li and J. Xiao, "Exact and efficient collision detection for a multi-section continuum manipulator," *ICRA*, 2012.
[12] C. Ferrari and J. Canny, "Planning optimal grasps," *ICRA*, 1992.
[13] S. Neppalli, M. A. Csencsits, B. A. Jones, and I. Walker, "A geometrical approach to inverse kinematics for continuum manipulators," *IROS*, 2008.