2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Exact and Efficient Collision Detection for a Multi-section Continuum Manipulator

Jinglin Li and Jing Xiao

***Abstract*—Continuum manipulators, featuring "continuous backbone structures", are promising for deft manipulation of a wide range of objects under uncertain conditions in less-structured and cluttered environments. A multi-section trunk/tentacle robot is such a continuum manipulator. With a continuum robot, manipulation means a continuous whole arm motion, where the arm is often bent into a continuously deforming concave shape. To approximate such an arm with a polygonal mesh for collision detection is expensive not only because a fine mesh is required to approximate concavity but also because each time the manipulator deforms, a new mesh has to be built for the new configuration. However, most generic collision detection algorithms apply to only polygonal meshes or objects of convex primitives.**

In this paper, we propose an efficient algorithm for Collision Detection between an Exact Continuum Manipulator (CD-ECoM) and its environments, which is applicable to any continuum manipulator featuring multiple constant-curvature sections. Our test results show that using this algorithm is both accurate and more efficient in both time and space to detect collisions than approximating the continuum manipulator as polygonal meshes and applying an existing generic collision detection algorithm. The algorithm is essential for path/trajectory planning of continuum manipulators.

I. Introduction

Continuum manipulators are usually defined to be those featuring continuous back bone structures, inspired by invertebrate structures found in nature, such as octopus arms [1] and elephant trunks [2]. Almost all continuum robots feature constant-curvature sections (modulo external loading due to gravity or payload) [3] because of actuating the (theoretically infinite) degrees of freedom of the continuously bendable backbone with finite actuators. The OctArm (Fig. 1), with constant-curvature sections, is representative of the general class of continuum robots developed by researchers [3], [4]. Continuum manipulators of smaller scales are also developed for medical surgery applications [5], [6].

In path/trajectory planning for robotic manipulation, collision detection between the robot and objects in the environment is not only necessary but also the most computationally expensive component of a planning algorithm for a high-degree of freedom robot, such as a n-DOF ($n \geq 6$) robot manipulator [7]. Thus, many efficient collision detection algorithms are developed [8], [9]. Efficient algorithms often use a hierarchy of bounding volumes (e.g., [10], [11]) for the objects to speed up collision checking. At the lowest level of the hierarchy, collision checking is conducted between either convex parts or polygons (for generic objects approximated by polygonal meshes). While a conventional manipulator usually consists of rigid, convex links and can be naturally decomposed into convex parts and use an existing collision detection algorithm, a continuum manipulator often works in configurations of continuously deforming concave shapes (Fig. 1). Thus, approximating a continuum robot at a certain configuration requires a very fine polygonal mesh to be reasonably accurate, which decreases efficiency for collision checking with the large number of polygons in the mesh. Moreover, each time a continuum robot changes its configuration, because its whole shape deforms, a new mesh has to be constructed, which is time consuming, and storing different meshes of different configurations for the purpose of path/trajectory planning can take too much space to be feasible.

In this paper, we propose an efficient algorithm for Collision Detection between an Exact Continuum Manipulator (CD-ECoM) and environmental objects in polygonal meshes. The algorithm is applicable to any continuum manipulator featuring multiple constant-curvature sections. We first describe the manipulator model and object model before presenting our CD-ECoM algorithm. We also present testing results to demonstrate the effectiveness and efficiency of the algorithm.

II. Manipulator Model and Object Model

A. Exact manipulator model

As an example of a multi-section continuum manipulator, let us consider the OctArm, which consists of three sections (see Fig. 1), where each section has a constant curvature [12]. 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. The curvature of a section can change values continuously, and if the curvature becomes zero, the section becomes a (straight) cylinder (i.e., the radius goes to infinity). In general, we can represent each section, denoted as sec_i, $i = 1, 2, 3$, in terms of its central axis seg_i with two end points: a *base* point p_{i-1} and a *tip* point p_i, and the radius of the cylinder w_i. If seg_i is a curve, we call the circle that seg_i curves along the *section i's circle*, denoted by cir_i, with radius r_i.

The base frame of the robot is set at p_0 with z_0 axis tangent to seg_1. The frame of section i, $i = 2, 3$, is formed at p_{i-1} with the z_i axis tangent to seg_i at p_{i-1}. The base of section i is the tip of section i-1. Two adjacent seg_{i-1} and seg_i are connected *tangentially* at the connection point p_{i-1} as shown in Fig.

J. Li is a PhD student in Computer Science, University of North Carolina at Charlotte, USA. jli41@uncc.edu

J. Xiao is with the faculty of Computer Science, University of North Carolina at Charlotte, USA. xiao@uncc.edu

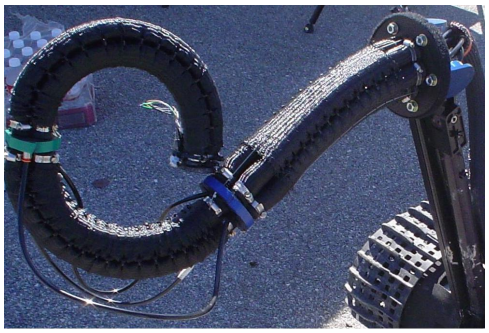
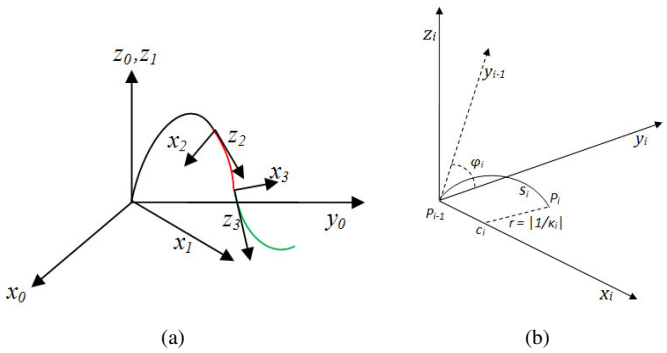


Fig. 1. An OctArm manipulator (by the courtesy of Ian Walker)

2.(a), i.e., the two sections share the same tangent at p_{i-1}. For clarity, we consistently use black, red, and green colors to draw section 1, section 2, and section 3 of the OctArm in this paper.

Each section i, $i = 1, 2, 3$, has three degrees of freedom that can be directly changed by the OctArm actuators [12], which are 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. 2.(b) shows one example seg_i, its frame, and controllable variables.



(a) (b)

Fig. 2. Section frames

Note that the center of cir_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 $\kappa_i = 0$, seg_i is a straight-line segment along the z axis, and c_i can be considered at either $+$ or $-$ infinity along the x axis.

The configuration of the entire arm is determined by the control variables of each section. This model can be extended to describe an n-section ($n > 3$) continuum manipulator.

B. Separating planes of manipulator sections

There is a plane separating two sections of a continuum manipulator. We call the plane between section $i-1$ and section i as H_{i-1}, which contains p_{i-1}, and its normal is along z_i. Clearly for each manipulator section i, it has two planes H_{i-1} and H_i separating it from its neighboring sections, as shown in Fig. 3.

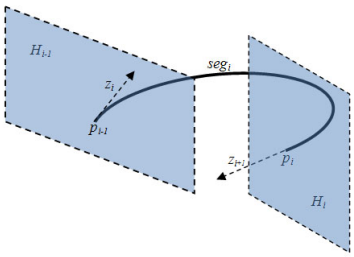


Fig. 3. Two separating planes for section i

C. Manipulator cross sections

For each section i of OctArm with a non-zero curvature, we define a plane P_i as a plane that contains the section i's circle, i.e. cir_i, then the cross section of OctArm 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. 4, 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:

$$r_i - w_i \leq \rho \leq r_i + w_i \tag{1}$$

$$\theta_i^{min} \leq \theta \leq \theta_i^{max} \tag{2}$$

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

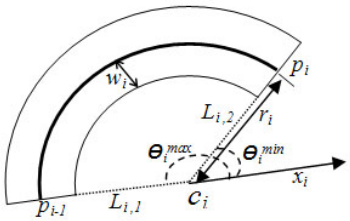


Fig. 4. The cross section (cs_i) of section i and its polar coordinate system

D. Object model

An object is modeled as a polygonal mesh consisting of J faces, denoted as $\{f_1, ..., f_J\}$, where each face consists of K edges, denoted as $\{e_1, ..., e_K\}$, and each edge has two vertices. Each face f_j is on the plane Q_j. We use the polygon file format[1] to represent the mesh. This format uses a vertex table to store the x, y and z values of all vertices and an index list to indicate which face and edge that each vertex belongs to.

[1]also known as Stanford Triangle Format [13]

Algorithm 1 CD-ECoM

Input Multi-section manipulator configuration $(\kappa_i, \phi_i, s_i, i = 1, ..., n)$, section width w_i, and faces of object mesh f_j, $j = 1, 2, ..., J$;
$Collision$ = False;
for each face f_j and each arm section i **do**
  if $\kappa_i == 0$ **then**
    $Collision$ = SS-CollisionCheck(seg_i, f_j)
  else
    if P_i intersects f_j at line segment l_j^i **then**
      $Collision$ = CS-CollisionCheck(cs_i, l_j^i).
    end if
    if $Collision$ == False **then**
      $Collision$ = NCS-CollisionCheck(seg_i, l_j^i).
    end if
  end if
  if $Collision$ == True **then**
    Return $Collision$.
  end if
end for
Return $Collision$.

Algorithm 2 SS-CollisionCheck(seg_i, f_j)

Compute the distance between seg_i and Q_j, denote the distance as $d(seg_i, Q_j)$ and closest points found on seg_i and Q_j as p, q respectively;
if $d(seg_i, Q_j) \leq w_i$ **then**
  if q is not in f_j **then**
    Compute the minimum distance between each edge e_k^j of f_j and seg_i, denoted as $d_{min}(seg_i, e_k^j)$ with closest points p and q on seg_i and e_k^j respectively.
    if $d_{min}(seg_i, e_k^j) > w_i$ **then**
      Return $Collision$ = False.
    end if
  end if
  if p is not on p_{i-1} or p_i **then**
    Return $Collision$ = True.
  else
    if q is on H_{i-1} (or H_i) **then**
      Return $Collision$ = True.
    end if
  end if
end if
Return $Collision$ = False.

III. CD-ECoM Algorithm

The CD-ECoM algorithm (**Algorithm 1**) checks if there is any collision between the exact model of a continuum manipulator featuring multiple constant-curvature sections at a given configuration and an object in polygonal mesh. The configuration of the arm determines seg_i with end points p_{i-1} and p_i. If $\kappa_i = 0$, where section i is a straight cylinder with a width of $2w_i$ and a length of s_i, then the function *straightSection-CollisionCheck* (SS-CollisionCheck), presented in **Algorithm 2**, is called to do the collision checking between a cylinder (section i) and a planar face (f_j).

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 (1) and (2). To be efficient, the **Algorithm 1** repeatedly uses a divide and conquer strategy by decomposing the collision detection problem into many simpler cases in different levels. It first considers the cases where the object mesh intersects P_i by calling **Algorithm 3** for *cross section collision check*. If no collision is found, it next checks the cases where the object mesh does not intersect P_i by calling **Algorithm 4** for *non-cross section collision check*. Both **Algorithm 3** and **Algorithm 4** further divide the problem into simpler cases to solve them efficiently. These two algorithms are further explained in the following subsections.

A. Cross section collision check (CS-collision check)

For a manipulator section i with non-zero curvature, i.e., $\kappa_i \neq 0$, if face f_j of an object intersects the plane P_i at a line segment l_j^i with vertices v_1 and v_2[2], **Algorithm 3** checks if l_j^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 3** partitions all scenarios into five cases based on whether v_1 and v_2 satisfy the bounds of inequalities (1) and (2) to detect intersections (i.e., collisions). Fig. 5 shows examples for these cases.

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

B. Non-cross section collision check (NCS-collision check)

If a face f_j of the object does not intersect P_i or the cross section cs_i, we need to further check if f_j intersects section i by **Algorithm 4**.

In **Algorithm 4**, we first check if the distance between cir_i and Q_j, the supporting plane of face j, is greater than the width of section i by calling **Procedure 1**. If so, Q_j has no intersection with the section i, a truncated torus, and no further collision checking is necessary. If Q_j intersects the section i, then further collision checking is done by calling subsequently **Procedure 2**, and if necessary, also **Procedure 3**. **Procedures 1,2 and 3** are described below:

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

- Project c_i to Q_j and denote the point on Q_j as q'.
- 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_j, denote the projected point as q, and return distance between p and q.

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

- Project each point to Q_j and denote the closer projection

Algorithm 3 CS-CollisionCheck(cs_i,l_j^i)

Case 1: **if** ρ_1 and ρ_2 are both below the lower bound for ρ, i.e., $\max(\rho_1, \rho_2) < r_i - w_i$ **then**
Return $Collision$ = False.

Case 2: **if** either (ρ_1, θ_1) or (ρ_2, θ_2) satisfies both (1) and (2) **then**
Return $Collision$ = True.

Case 3: **if** ρ_1 and ρ_2 are both above the upper bound for ρ, i.e., $\min(\rho_1, \rho_2) > r_i + w_i$ **then**
Compute the distance between circle center c_i and l_j^i to obtain point $q = (\rho_q, \theta_q)$ on l_j^i.
if q is within cs_i, i.e., ρ_q and θ_q satisfy inequalities (1) and (2) respectively **then**
Return $Collision$ = True.
else
Return $Collision$ = False.
end if

Case 4: **if** both θ_1 and θ_2 satisfy (2) **then**
Return $Collision$ = True.

Case 5: **if** line segment l_j^i intersects either ray $L_{i,k}$ ($k = 1, 2$) of cs_i (see Fig. 4) **then**
if l_j^i is colinear to either ray **then**
Return $Collision$ = True.
end if
if At least one of ρ_{int}^k satisfies (1) **then**
Return $Collision$ = True. (Fig. 5(e))
end if
if l_j^i intersects both rays at one point above the upper bound for ρ and one point below the lower bound for ρ **then**
Return $Collision$ = True. (Fig. 5(f))
end if
if l_j^i intersects only one ray at a point above the upper bound for ρ **and** the vertex of l_j^i with the smaller ρ value satisfies (2) **then**
Return $Collision$ = True. (Fig. 5(g))
end if
if l_j^i intersects only one ray at a point below the lower bound for ρ **and** the vertex of l_j^i with the greater ρ value satisfies (2) **then**
Return $Collision$ = True. (Fig. 5(h))
end if
Return $Collision$ = False.

(to the original point) as q' and the corresponding distance $d_{min}(p_{i/i-1}, Q_j)$.

- If q' is on f_j then return $d_{min}(p_{i/i-1}, Q_j)$ as $d_{min}(p_{i/i-1}, f_j)$.
- Else, find the minimum distance between each point to every edge of f_j and return the shortest distance as $d_{min}(p_{i/i-1}, f_j)$.

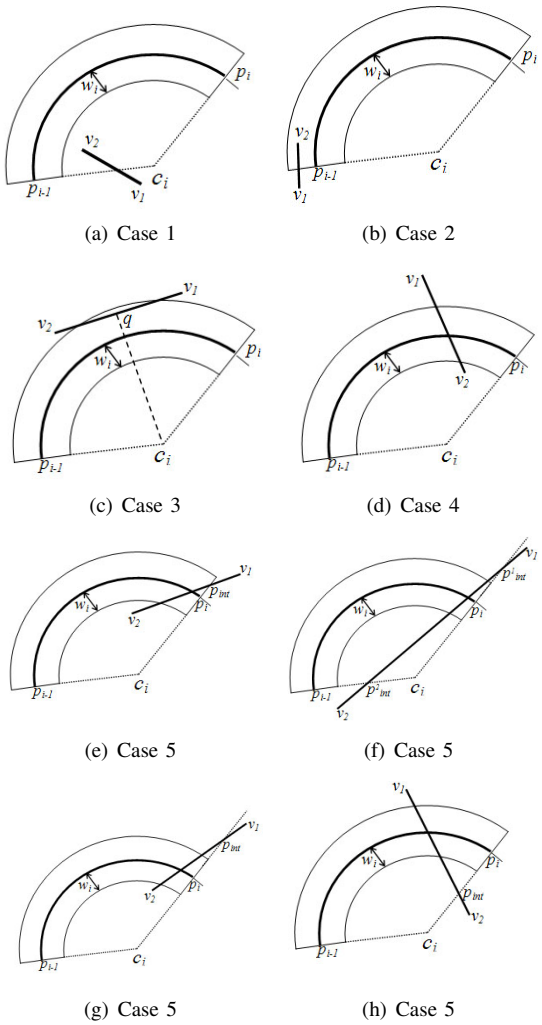


Fig. 5. Some examples for cases considered in the cross section collision check

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:

- Project e_k to P_i and denote it as e_k^i.
- If e_k^i intersects cir_i at point p, then
  - If p is on seg_i, return the distance from p to e_k as $d_{min}(seg_i, e_k)$, as shown in Fig. 6(a).
  - If p is not on seg_i, then compute the shortest distance from p_i and p_{i-1} to e_k, and set it as $d_{min}(seg_i, e_k)$, as shown in Fig. 6(b).
- Else, project c_i to the line containing e_k^i at point p', and let p be the intersection point between cir_i and the line passing c_i and p'.
  - If p is on seg_i and p' is outside cir_i, then the distance from p to e_k is $d_{min}(seg_i, e_k)$. See Fig. 6(c).
  - If p is on seg_i and p' is inside cir_i, then one of the vertices v of e_k is the closest point on e_k to seg_i. See Fig. 6(d). The following computes the shortest distance from v to seg_i:

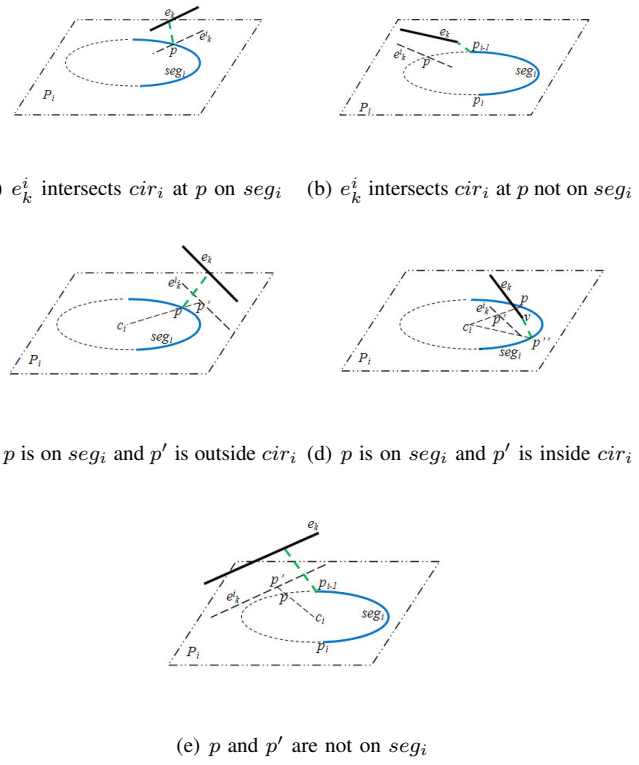
Algorithm 4 NCS-Collision check(seg_i, f_j)

Compute the minimum distance between cir_i and Q_j, i.e., $d_{min}(cir_i, Q_j)$, and corresponding points p, q on cir_i and Q_j respectively by **Procedure** 1.
if $d_{min}(cir_i, Q_j) > w_i$ **then**
  Return $Collision$ = False.
end if
if q is on f_j **and** p is not on seg_i **then**
  Compute the shortest distance from p_{i-1} and p_i to f_j, denoted as $d_{min}(p_{i/i-1}, f_j)$, and update the pair of closest points p and q, by **Procedure** 2;
  if $d_{min}(p_{i/i-1}, f_j) > w_i$ **then**
    Return $Collision$ = False.
  end if
end if
if q is not on f_j **then**
  Compute the shortest distance between seg_i and the edges of f_j by **Procedure** 3, denoted as $d_{min}(seg_i, e_k^j)$, and update the pair of closest points p and q;
  if $d_{min}(seg_i, e_k^j) > w_i$ **then**
    Return $Collision$=False.
  end if
end if
if p is not p_i or p_{i-1} **then**
  Return $Collision$ =True.
else
  if q is on H_i or H_{i-1} **then**
    Return $Collision$ =True.
  end if
end if
Return $Collision$=False.

- Find the intersection point, denoted as p'', between the ray pointing from c_i to the projection of v on P_i and cir_i.
- Compute the distance from p'' to v as $d_{min}(seg_i, e_k)$.

– If p is not on seg_i, then compute the shortest distance from p_i and p_{i-1} to e_k as $d_{min}(seg_i, e_k)$, as shown in Figures 6(e).

We introduce **Procedure 1** and **Procedure 3** because there is no ready algorithm in computational geometry for analytically computing the distance from a flat face or edge to a curve (cir_i or seg_i) in 3D space. One could discretize cir_i (or seg_i) and f_j (or its edge e_j) into two point clouds and then compute the closest pair of points between them [14], but this would be an approximation and is computationally expensive. Another possibility is to formulate the problem of finding the minimum distance involving a curve as an optimization problem, which, however, may only be solved numerically because the distance function involving a curve is of high-order and non-linear. Again, such a method can be computationally expensive. In contrast, the **Procedure 1** and **Procedure 3** introduced here are efficient and accurate.



(a) e_k^i intersects cir_i at p on seg_i (b) e_k^i intersects cir_i at p not on seg_i

(c) p is on seg_i and p' is outside cir_i (d) p is on seg_i and p' is inside cir_i

(e) p and p' are not on seg_i

Fig. 6. Illustrations for **Procedure** 3, the blue partial circle is the OctArm section seg_i and the green dotted line segment indicates the distance obtained for each case.

As for **Procedure 2**, a related algorithm was proposed in [15] 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_j is not limited to be triangular but can be any convex polygon.

IV. Test Results and Discussion

We have implemented the CD-ECoM algorithm, applied it to collision detection between the OctArm manipulator and polygonal mesh models of arbitrary objects. We have also compared the collision detection results using our CD-ECoM algorithm with those using a mesh-based collision detection algorithm OPCODE [16], which is similar to SOLID [10], [17] or RAPID [11] but often faster. We have run both the CD-ECoM algorithm and the OPCODE on the same computer with a 2.4GHz core.

Fig. 7 displays three mesh models for the OctArm, from coarse to reasonably fine. The mesh model 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 I compares the space and time required for acquiring the exact arm model of the OctArm and two mesh models (a coarse one and a fine one) of the OctArm for any given

TABLE I
SPACE/TIME REQUIRED FOR DIFFERENT OCTARM MODELS AT A GIVEN ARM CONFIGURATION

Arm Model	Exact	Mesh 1	Mesh 2
Description	9 configuration parameters: κ_i, s_i, ϕ_i, $i = 1, 2, 3$	2,000 triangles per section 3 vertices per triangle.	8,000 triangles per section. 3 vertices per triangle.
Space (byte)	36	18K	72K
Building time (ms)	None	50	193
Rebuild model for a new configuration?	No need	Yes	Yes

TABLE II
COLLISION DETECTION TIME (MS) BETWEEN THE OCTARM AND A TEA POT MESH (WITH 1,024 TRIANGLES)

Arm Model	Algorithm	Config. 1	Config. 2	Config. 3
Exact	CD-ECoM	42	8	12
Mesh 1	OPCODE	17	10	15
Mesh 2	OPCODE	47	21	38

TABLE III
COLLISION DETECTION TIME (MS) BETWEEN THE OCTARM AND A BUNNY MESH (WITH 3,851 TRIANGLES)

Arm Model	Algorithm	Config. 4	Config. 5	Config. 6
Exact	CD-ECoM	107	6	15
Mesh 1	OPCODE	58	60	69
Mesh 2	OPCODE	98	105	127

TABLE IV
COLLISION DETECTION TIME (MS) AVERAGED OVER 100 ARM CONFIGURATIONS

Arm Model	Algorithm	Average time (ms)
Exact	CD-ECoM	7.69
Mesh 1	OPCODE	8.62
Mesh 2	OPCODE	30.63

configuration. The time and space for building a bounding volume hierarchy for a mesh model is not included in the table. It is important to note that when the OctArm changes its configuration, the exact model changes accordingly without the need for rebuilding, but that is not the case for the mesh models. Like many other continuum manipulators, a new configuration of the OctArm involves deformation of the arm (caused by changed curvature and length of any arm section); therefore, the mesh model for the previous configuration cannot be simply updated by coordinate transformation and used again. Instead, a new mesh model for each new configuration is needed, costing the same time and storage space again.

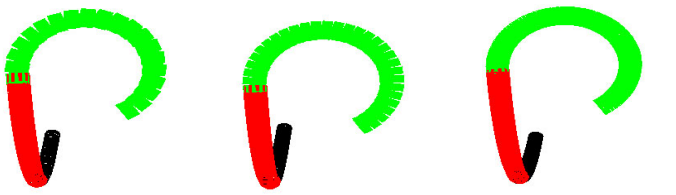
In general, if T and S are the time and space respectively required for building a single mesh model of a continuum manipulator, then to represent and store a single path of m configurations of the manipulator in a mesh model requires mT time and mS space, which could be too expensive to be feasible 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-ECoM algorithm uses the exact arm model of a continuum manipulator directly to avoid the high time and space cost of building and re-building mesh models for different configurations and thus facilitates motion planning for a continuum manipulator.

Table II and Table III present the results of collision detection between the OctArm and two different object meshes, a tea pot mesh model with $1,024$ triangles and a bunny mesh model with $3,851$ triangles, at different OctArm configurations, as shown in Fig. 8.

In configuration 1 and configuration 4, the arm is not in collision, and this is the most expensive case for collision detection with our CD-ECoM algorithm because no bounding volume of the object is used. On the other hand, OPCODE uses bounding volume hierarchies to speed up computation. As the result, the CD-ECoM algorithm takes more time than OPCODE for the coarse mesh 1 model of the arm. However, it is easy to use a bounding volume hierarchy for the object in our CD-ECoM algorithm to speed up the algorithm further. On the other hand, for the finer mesh 2 model of the arm, even without using a bounding volume hierarchy for the object, our CD-ECoM algorithm takes comparable time for collision detection.

For the rest of the configurations of the OctArm, our CD-ECoM algorithm takes less time than OPCODE to detect collisions. For the finer mesh 2, it takes at most only about 40% (configuration 2) and 11% (configuration 6) of the time that OPCODE uses for the tea pot mesh and the bunny mesh respectively.

Table IV compares the average time per collision check for 100 random OctArm configurations against the tea pot using CD-ECoM vs. OPCODE. The results show that the CD-ECoM algorithm is generally more efficient than OPCODE applied to even a coarse mesh model of the continuum manipulator.

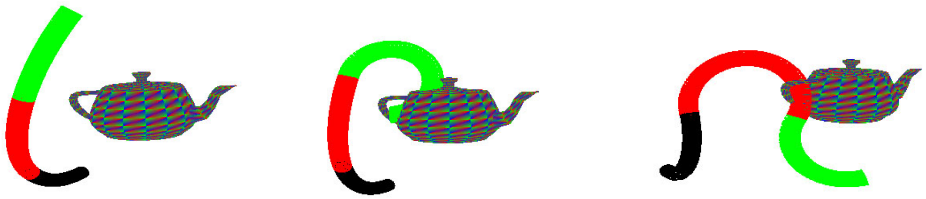


(a) 2,000 triangles per section (b) 4,000 triangles per section (c) 8,000 triangles per section

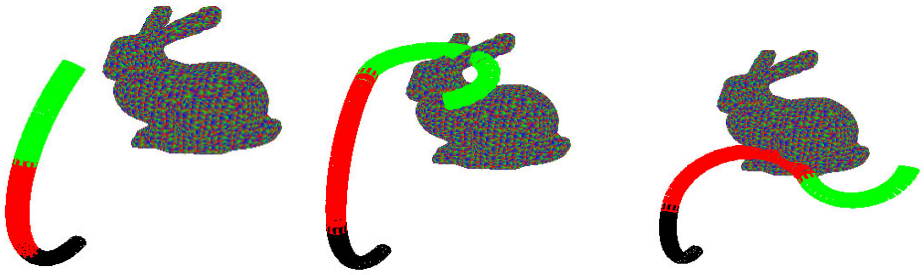
Fig. 7. Three OctArm mesh models

V. CONCLUSIONS AND FUTURE WORK

This paper proposes an efficient algorithm for collision detection between an exact continuum manipulator and environmental objects in polygonal meshes. The CD-ECoM algorithm



(a) Configuration 1 (no collision) (b) Configuration 2 (collision with section 3) (c) Configuration 3 (collision with section 2)



(d) Configuration 4 (no collision) (e) Configuration 5 (collision with section 3) (f) Configuration 6 (collision with section 2)

Fig. 8. Arm configurations tested for collision with a tea pot and a bunny

is applicable to general continuum manipulators featuring n constant-curvature sections (where $n = 1, 2, 3, ...$) that can be continuously deformed into different concave shapes by changing each section's configuration parameters (i.e., κ, θ, s). Unlike conventional collision detection algorithms that require mesh models, our CD-ECoM algorithm uses the exact model of a continuum manipulator and therefore, it does not need to rebuild mesh models for different arm configurations and saves significant time and space. It is also generally more efficient in collision detection comparing to collision detection using a sufficiently fine mesh model of the manipulator. Our algorithm is especially suitable for continuum manipulator path planning that considers a large number of manipulator configurations.

Our next steps include incorporating hierarchical bounding boxes into our CD-ECoM algorithm, which can reduce the collision detection time for collision-free arm configurations, and integrating this algorithm of collision detection into motion planning for continuum manipulation.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] G. Robinson and J. B. C. Davies, "Continuum robots - a state of the art," *Proc. CDEN Design Conf.*, pp. 2849–2854, 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. C. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[8] P. Jimnez, F. Thomas, and C. Torras, "3d collision detection: A survey," *Computers and Graphics*, vol. 25, pp. 269–285, 2000.

[9] M. C. Lin and S. Gottschalk, "Collision detection between geometric-models: A survey," *Proc. of IMA Conf. on Mathematics of Surfaces*, pp. 37–56, 1998.

[10] G. van den Bergen, "Efficient collision detection of complex deformable models using aabb trees," *Journal of Graphics Tools*, vol. 2(4), pp. 1–13, 1997.

[11] S. Gottschalk, M. C. Lin, and D. Manocha, "Obb-tree: A hierarchical structure for rapid interference detection," *Proc. of ACM SIGGRAPH*, 1996.

[12] B. A. Jones and I. D. Walker, "Kinematics for multisection continuum robots," *IEEE Trans. Robot.*, vol. 22, pp. 43–55, 2006.

[13] G. Turk and M. Levoy, "Zippered polygon meshes from range images," *SIGGRAPH*, pp. 311–318, 1994.

[14] "CGAL, Computational Geometry Algorithms Library." http://www.cgal.org.

[15] 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.

[16] P. Terdiman, "Opcode: Optimized collision detection." Available: www.codercorner.com/OPCODE.htm, 2003.

[17] G. van den Bergen, "A fast and robust gjk implementation for collision detection of convex objects," *Journal of Graphics Tools*, vol. 4(2), pp. 7–25, 1999.