Real-time Adaptive Motion Planning for a Continuum Manipulator

Jing Xiao and Rayomand Vatcha

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, often without a clear distinction between transport and grasping. In this paper, we address the novel problem of real-time motion planning for such a robot under uncertain conditions. We present an algorithm for on-line planning the motion of a planar continuum robot for grasping a target object amid an environment of other objects with uncertain movements. Our algorithm substantially extends the RAMP paradigm [19] for real-time adaptive motion planning to this new form of whole-arm manipulation. Simulation results are promising, demonstrating the effectiveness of our approach.**

I. Introduction

Unlike the skeletal design of traditional rigid-link robot manipulators, continuum manipulators [17] are usually defined to be those featuring continuous backbone structures, inspired by invertebrate structures found in nature, such as octopus arms [13] and elephant trunks [6]. Instead of bending only at discrete joints, continuum manipulators can deform anywhere along their structure in theory.

There is no divide between the "arm" and "hand/gripper" for a continuum manipulator to manipulate objects. While it is common to define a grasping end-effector configuration based only on the relation of the hand and the object for a conventional manipulator, for a continuum robot, there is no such "end-effector configuration" – the shape and pose of the whole arm is affected by not only how an object is grasped, but also the size and the shape of the object, and the presence of nearby obstacles. Thus, it is not suitable to pre-define "gripper" configurations without regarding the environment.

For a conventional robot manipulator, i.e., in the form of an articulated arm with a gripper, gross motion planning assumes a given goal configuration (where grasping can start) and is concerned with finding a collision free path for the manipulator or mobile manipulator connecting an initial configuration to the goal configuration. Planning is often done off-line, assuming a static environment or dynamic environment with known obstacles and their motions. To avoid the formidable challenge of constructing high-dimensional C-obstacles, sampling-based planners, notably the PRM [9] and the RRT [10] planners and variants, are widely used. Some methods take into account small-scale changes in the manipulator's environment due to new obstacles (e.g., [11] [22]). The RAMP approach [19] is effective in planning mobile manipulator motions in real-time in a dynamic environment where obstacle motions are unknown. Recently a co-evolutionary approach was introduced [1] combining gross motion planning of a conventional mobile manipulator with finding optimal goal configurations for grasping in a static environment.

There exists work on path planning for hyper redundant robots [4], characterized by vast number of joints connecting very short rigid links/structures to be flexible. Here the path planning problem is more challenging than that of conventional manipulators with far fewer degrees of freedom. There are approaches avoiding static known obstacles [3] [7] [12] and sensor-based approaches tackling unknown static obstacles for snake robots [16] [5]. However, there is little study on manipulation motion planning for active continuum manipulators, which are deformable anywhere and do not have discrete joints, although there exists considerable research on modeling and planning for manipulating passive deformable linear objects (such as wires or cables) [14] [18] [21]. It is an even greater challenge to plan motions of an active continuum robot in a dynamic environment having unknown obstacle motions, in real-time.

In this paper, we introduce an approach to motion planning of a multi-section continuum manipulator [8] in uncertain and changeable environments in real-time. Our planner extends the RAMP paradigm [19] for real-time adaptive motion planning of conventional manipulators to this new form of whole-arm manipulation in the following ways: (1) instead of assuming goal configuration(s) as given, generate and plan goal configurations of the whole arm for picking up an object *together* with the paths/trajectories leading to the goal configurations in a progressive fashion on-line and based on the specific target object, (2) introduce optimization functions for goal configurations and paths/trajectories taking into account the specific characteristics of continuum whole-arm manipulation, and (3) introduce path/trajectory modification operations exploiting the structure of a continuum manipulator. Our algorithm also employs the unique concept of *dynamic envelope* [20] for collision-checking of robot trajectories based on sensing, which does not require prediction of unknown motions of obstacles.

II. OctArm Manipulator: Configuration, Path, and Trajectory

While a continuum robot can in fact have an infinite number of degrees of freedom because it is deformable, there are only a finite number of controllable degrees of freedom when the robot is not in contact. These are the degrees of

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

R. Vatcha is a PhD student in Computer Science,University of North Carolina at Charlotte, USA. rvatcha@uncc.edu

freedom that can be directly changed by the actuators. The OctArm manipulator designed at Clemson University has three sections. According to [8], each section $i, i = 1, 2, 3$, is a part of a circle with two end points: a *base* point p_{i-1} and a *tip* point p_i. The base of the robot is set at p_0 with z_0 axis tangent to section 1's curve. The section i's frame is formed at p_{i-1} with the z axis tangent to the section curve at p_{i-1}. The base of section i is the tip of section $i-1$. Each section has three controllable variables: curvature κ_i (which can be either negative or positive), length s_i, and rotation angle ϕ from axes y_{i-1} to y_i about z_{i-1}. Note that the circle center of section i, p_{ic}, always lies on the x_i axis.

A configuration $\mathbf{C}$ of the OctArm manipulator can be expressed by the controllable variables as $[\kappa_1, s_1, \phi_1, \kappa_2, s_2, \phi_2, \kappa_3, s_3, \phi_3]^T$. Thus, we can treat this (κ, s, ϕ) space the *configuration space* of the OctArm robot. Given the position of the base point p_{i-1}, and the κ_i, s_i, and ϕ_i values, the position of the tip point p_i of the section can be computed [8].

If ϕ_i=0 for all the sections, the OctArm is planar. Figure 1 shows a planar three OctArm and an example section. In this paper, we only focus on a planar OctArm.

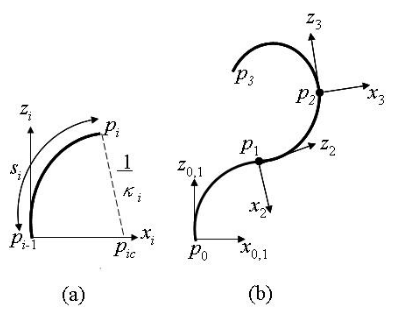


Fig. 1. A planar OctArm manipulator: (a) one section and its frame; (b) three-section arm

A *path* for an elephant trunk robot from its current configuration $\mathbf{C}_c$ to some goal configuration $\mathbf{C}_g$ can be described as a sequence of discrete configurations $\mathbf{C}_c, \mathbf{C}_1, \mathbf{C}_2, ...\mathbf{C}_m, \mathbf{C}_g$, and those intermediate configurations $\mathbf{C}_i$ $(i = 1, ..., m)$ are called *knot configurations* or *knots*. Between two knot configurations, the straight-line path segment in the (κ, s, ϕ) space is further discretized as a sequence of configurations. Thus, an entire path is discretized and represented by a sequence of configurations.

A *trajectory* adds a dimension of time to a path and is subject to the actuation and control constraints of the robot. By adding a time dimension to the (κ, s, ϕ) space of the OctArm robot, we have a *configuration-time space* of the robot, where a point $\chi = (C, t)$ is a *configuration time point* (CT-point).

III. Overview of Planning Algorithm

The RAMP paradigm [19] is motivated by the need of real-time motion planning of high-DOF robots, such as (mobile) manipulators, in dynamic environments of unknown obstacle motions. It has the following general characteristics: (1) real-time *simultaneous* planning and execution of high-DOF robot path/trajectory based on sensing; (2) *anytime* and *parallel planning* with optimization, as inspired by evolutionary computation [2], through maintaining and repeatedly updating/improving a set of trajectory candidates for a robot from its current configuration to a goal configuration; (3) great structual flexibility to allow for both on-line adaptation to different environmental scenarios and off-line extension to robots of very different nature. Specifically for (3), all major components of the RAMP algorithm can be customized, including initialization of paths/trajectories, optimization criteria and evaluation, and modification operations of paths/trajectories. The strength of RAMP lies in both its generality and its flexibility for adaptation and extension.

A. Extending RAMP to continuum robots

While the RAMP algorithm presented in [19] is directly applicable to a wide class of conventional manipulators featuring rigid links and discrete joints, planning motions of continuum manipulators with deformable sections requires further extension and customizaton of RAMP. As introduced in section I, goal configurations cannot be assumed given as in the case of conventional manipulators because they are determined by the target object and the environment and affect the whole form of the manipulator. Thus the planner has to plan the goal configurations jointly with paths/trajectories. Note that a *goal configuration* in this paper refers to a manipulator configuration from which "grasping" can start by compliantly wrapping the object[1].

In addition, a continuum robot is considerably more complex in kinematics than a conventional articulated arm/manipulator. The configuration variables for a deformable section, such as κ, s, and ϕ of the OctArm robot, are highly coupled in expressing the pose of a section in the physical space; whereas for a conventional manipulator, the pose of a link i is commonly determined by just one joint variable (relative to the link i-1). This makes it nontrivial to determine effective distance metric for path/trajectory optimization, i.e., requires new evaluation functions for optimization. It also requires that a path/trajectory be represented with more built-in flexibility in initialization.

Moreover, the planner should exploit the deformable shape and volume of a continuum manipulator in avoiding obstacles – an advantage that a conventional articulated arm does not have.

B. Real-time continuum planner

Our planner is outlined in Algorithm 1, interweaving parallel processes performed in the following three cycles as in RAMP:

- *Sensing cycle*: In each cycle new information about the environment, such as new poses of obstacles, is obtained from the sensor.
- *Planning cycle*: In each cycle, a trajectory from a set S of trajectories along with its goal configuration is

[1]We do not consider the compliant wrapping motion in this paper, which is the next research topic (see section V).

modified to adapt to changes in the environment and for optimization.

- *Adaptation cycle*: At the start of each cycle, the robot can switch to the newly found better trajectory.

As soon as the planner finds an immediate segment of a trajectory collision-free, call it L_f, the robot starts moving along L_f while continuing the planning cycles simultaneously to (1) find subsequent collision-free segments and (2) further optimize collision-free segments. Thus, at each new adaptation cycle, the robot can switch to a better trajectory[2], and in order to do that, the start CT-point of every trajectory in S is updated accordingly.

In the following sections, we describe in detail the components, i.e., **initialize**, **evaluate**, and **modify** of Algorithm 1, with extensions to RAMP as summarized in section III.A.

Algorithm 1 Real-time Continuum Planner

```
Initialize a set $S$ of trajectories connecting the start configuration to one of the generated goal configurations
Evaluate all in $S$. $\Gamma_{best} \leftarrow$ best trajectory.

while a goal is not reached do
    Simultaneously sense, plan, and move:
    Sense:
    if new sensing cycle then
        Check and record collisions of trajectories in $S$.
    end if

    Plan:
    Modify a trajectory in $S$.
    if new adaptation cycle then
        Evaluate trajectories in $S$.
        Update $\Gamma_{best}$.
        $\chi_e = (\mathbf{C}_e, t_e) \leftarrow$ last CT point on the collision-free segment $L_f$ of $\Gamma_{best}$ within the adaptation cycle or before a forced stop.
        Update starting CT point of all in $S$ to $\chi_e$ (i.e., when the next adaptation cycle starts).
    end if

    Move along $\Gamma_{best}$ unless forced to stop to avoid collision.
end while
```

IV. Path/Trajectory and Goal Initialization

The initial set of paths is a mixture of randomly and deliberately generated ones, ending at a variety of goal configurations determined based on the rough information of the target object known so far. We include straight-line paths in the configuration space from the robot's current configuration to all goal configurations as deliberately generated paths. Each randomly generated path has a random number of randomly generated intermediate knot configurations. Between two randomly generated adjacent knot configurations, if the straight-line path segment is longer than a threshold D, more knot configurations are inserted along the segment to ensure that the straight-line segment between any two adjacent knots is sufficiently short. This provides the needed flexibility for a path to change shape via changing the knots later. Given a path, the corresponding trajectory is obtained as a time-minimum trajectory.

[2]Hence, an adaptation cycle is greater than a planning cycle.

Our algorithm determines goal configurations automatically for a planar OctArm robot. Multiple goal configurations facilitate placing the robot for grasping among static or moving obstacles, which will also adapt during planning. Figure 2 shows a target object with a circular bounding box of radius r. centered at p_c with position vector $\mathbf{p}_c$ in the robot's base frame, and $r_c = \|\mathbf{p}_c\|$.

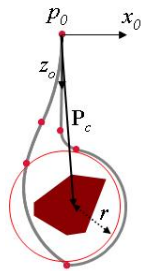


Fig. 2. A target object with a bounding circle centered at $\mathbf{P}_c$ w.r.t. the base frame and two types of goal configurations

To generate a goal configuration, our algorithm first randomly puts the robot tip in a thin belt of width Δr surrounding the bounding circle of the target object and then generates the tip positions of the other segments under the kinematic and geometric constraints of the OctArm robot. For a planar OctArm robot, the positions of the section tips p_1, p_2, and p_3 in the robot's base frame have to satisfy the constraints of the two triangles shown in Figure 3, where $r_i, i = 1, 2$ is the distance of p_{i+1} to the robot base frame, and $l_i \in [l_{i,min}, l_{i,max}]$ $(i = 1, 2, 3)$ is the length of the chord of section i, such that

$$l_{i,min} = \frac{2}{\kappa_{i,max}} sin(\frac{s_{i,min}\kappa_{i,max}}{2})$$

and

$$l_{i,max} = \frac{2}{\kappa_{i,min}} sin(\frac{s_{i,max}\kappa_{i,min}}{2}).$$

$\kappa_{i,min}$ and $\kappa_{i,max}$ are the bounds on the curvature κ_i; $s_{i,min}$ and $s_{i,max}$ are the bounds on the section arc length s_i. Note that for the target object to be within the reach of the robot arm, it is necessary that $r_c + r + \Delta r << \Sigma_{i=1}^{3} l_{i,max}$.

Algorithm 2 describes how the tip positions p_3, p_2, and p_1 are generated in turn randomly for a goal configuration that satisfies the triangle constraints in Figure 3. p_3 and p_2 are generated within the thin belt Δr surrounding the target object so that section 3 of the robot curves around the target object circle. Next, from the generated base and tip positions of p_{i-1} and p_i for section i, the curvature κ_i and length s_i can be computed [15]. If the robot at the generated configuration is not colliding with the target object, the

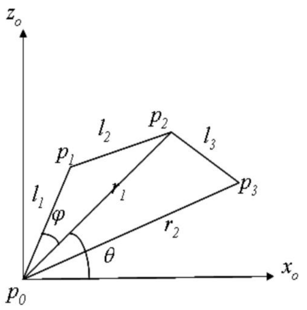


Fig. 3. Geometric constraints on the section tips p_1, p_2, and p_3 of the planar OctArm robot

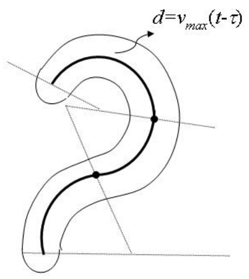


Fig. 4. A dynamic envelope surrounding the planar three-section OctArm robot

configuration is considered a valid goal configuration. Two homotopic groups of goal configurations can be generated, as shown in Figure 2.

Algorithm 2 Goal Generation

```
Input information of target object: $\mathbf{p}_c, r_c, r, \Delta r$.
Pick point $p$ along $\mathbf{p}_c$ with $\|\mathbf{p}-\mathbf{p}_c\| \in [r, r+\Delta r]$.
Obtain $p_3$ by rotating $p$ about $p_c$ a random angle.
Randomly pick $\alpha \in [\frac{s_{3,min}}{r+\Delta r}, \frac{s_{3,max}}{r+\Delta r}]$.
Obtain $p_2$ by rotating $p_3$ about $p_c$ either $\alpha$ or $-\alpha$.
if $r_1 > l_{1,max} + l_{2,max}$ then
    return "Object too far".
end if
$x_2 \leftarrow$ x coordinate of $p_2$.
repeat
    if $x_2 > 0$ then
        randomly pick $\varphi$ in $[0, \min(\frac{\pi}{4}, \frac{\pi}{2} - \theta)]$,
    else
        randomly pick $\varphi$ in $[-\min(\frac{\pi}{4}, \theta - \frac{\pi}{2}), 0]$.
    end if
    Randomly pick $l_1 \in [l_{1,min}, l_{1,max}]$.
    $l_2 = \sqrt{r_1^2 + l_1^2 - 2r_1 l_1 \cos\varphi}$.
until $l_2 \in [l_{2,min}, l_{2,max}]$ or a time limit is reached.
if $l_2$ is outside $[l_{2,min}, l_{2,max}]$ then
    return "Failure".
end if
Generate $p_1$ with coordinates $(l_1\cos(\theta+\varphi), l_1\sin(\theta+\varphi))$.
Return $p_1$, $p_2$, and $p_3$.
```

To generate a random knot configuration, we use forward kinematics: for $i = 1$ to 3, randomly generate κ_i and s_i within their bounds respectively, and then compute the corresponding tip point p_i [8].

The objective of initialization is to create a diverse set of trajectories with diverse goal configurations, which form a foundation for further improvements during subsequent planning cycles.

V. Trajectory and Goal Evaluation

We evaluate the quality of a path/trajectory, which takes into account the quality of the goal configuration that the path ends, by different cost evaluation functions based on different circumstances. The objective is to make the robot select a trajectory that can lead it to a goal most efficiently and also avoid obstacles. In any case, a trajectory is better if it has a lower cost. We consider collision-free trajectories always better than trajectories that are not collision-free.

As shown in Algorithm 1, evaluation of trajectories in S happens after initialization and before an adaptation cycle starts for the planner to select the best trajectory for execution. After a trajectory is modified to be a new one, it is also evaluated to compare to the trajectories in S to replace the worst trajectory in S (if the modified one is better than the worst) depending on need (see Section VI).

We first explain collision checking and then explain evaluation functions for trajectory selection.

A. Collision detection

Collision detection is performed by employing the concept of dynamic envelope [20] based on sensing, which does not require prediction of unknown motions of obstacles. This approach checks for the intersection between the dynamic envelope that surrounds the robot at configuration-time point $(\mathbf{C}, t)$ and obstacles to see if the robot will be surely collision-free or not at $(\mathbf{C}, t)$ based on sensing at current time $\tau < t$. The size of the envelope is determined by $d = v_{max}(t-\tau)$, where v_{max} is an upper-bound on obstacle speeds. Figure 4 shows a dynamic envelope of the planar three-section OctArm robot. The shape of the dynamic envelope is the result of connecting the fan-shape envelope for each section and a circle at the tip point of the robot. We developed an efficient method to check intersections between such a dynamic envelope for the OctArm robot and polygonal obstacles. A collision-free trajectory segment found by the above collision-detection method is guaranteed collision-free [20] regardless how obstacles move in the future. Hence, a collision-free trajectory connecting the robot's current configuration to a goal configuration, once found, will remain collision-free so that the robot can execute it without the worry of collision again.

In a static environment, i.e., $v_{max} = 0$, the dynamic envelope is reduced to the robot itself, and collision checking at current time τ is between the robot at $(\mathbf{C}, t)$ and the (static) obstacles.

B. Evaluation functions

To evaluate a trajectory, we consider the path length, the goal quality, and the quality of the immediate collision-free trajectory segment L_f (of a trajectory that is not collision-free).

Length: the sum of the distances between two consecutive configurations along the discretized trajectory. We use two distance measures to compute the distance between two consecutive configurations $\mathbf{C}_j$ and $\mathbf{C}_{j+1}$ on a trajectory, $d_{ks\phi}(\mathbf{C}_j, \mathbf{C}_{j+1})$ and $d_w(\mathbf{C}_j, \mathbf{C}_{j+1})$:

$$d_{\kappa s\phi}(\mathbf{C}_j, \mathbf{C}_{j+1}) = \|\mathbf{C}_{j+1} - \mathbf{C}_j\|$$
$$d_w(\mathbf{C}_j, \mathbf{C}_{j+1}) = \Sigma_{i=1}^{3} u_i(\|\mathbf{p}_{ic}^{j+1} - \mathbf{p}_{ic}^{j}\|)$$

where $\mathbf{p}_{ic}^{j}$ and $\mathbf{p}_{ic}^{j+1}$ are the center positions of the circle of section i at configurations $\mathbf{C}_j$ and $\mathbf{C}_{j+1}$ respectively, and u_i is the weight for section i. Larger weights are placed on sections closer to the robot base. d_w captures well the difference between the two poses of the robot in the workspace.

Figure 5 shows two examples, where $d_{\kappa s\phi}(\mathbf{C}_j, \mathbf{C}_{j+1})$ is the same in both cases, but the distance $d_w(\mathbf{C}_j, \mathbf{C}_{j+1})$ in case (b) is much greater than that in case (a). We use d_w to compute the length of a path when the robot is far from the goal configuration. Only when the robot is very close to the goal configuration so that a straight-line path (in the (κ, s, ϕ) space) is likely collision-free, we switch to use $d_{\kappa s\phi}$ to compute length, which favors the straight-line path towards the goal.

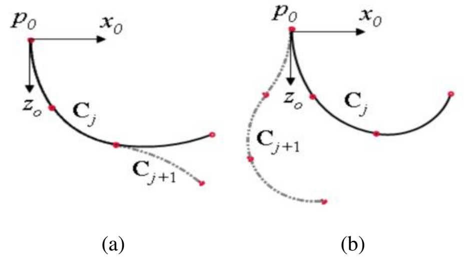


Fig. 5. Two examples to show the difference between $d_{\kappa s\phi}(\mathbf{C}_j, \mathbf{C}_{j+1})$ and $d_w(\mathbf{C}_j, \mathbf{C}_{j+1})$: $d_{\kappa s\phi}$ has the same value for both (a) and (b), but d_w in case (b) is much greater than that in case (a).

GoalCost: measures the quality of a goal configuration $\mathbf{C}_g$ by the difference between it and a configuration where the robot fully surrounds the target object. Let $\mathbf{p}_{3c}$ be the center position of the circle of section 3 of the robot, and let $\rho_3 = 1/\kappa_3$ be the radius of section 3. $GoalCost$ can be computed as:

$$GoalCost = b_1(|r - \rho_3| + \|\mathbf{p}_c - \mathbf{p}_{3c}\|) + b_2(2\pi r - s_3)$$

where b_1 and b_2 are weights.

SegCost: measures the quality of the immediate collision-free trajectory segment L_f of a trajectory that is not collision-free. Let $(\mathbf{C}_f, t_f)$ be the last CT point on L_f. **SegCost** is a weighted combination of (a) a cost negatively proportional to the number of collision-free configurations in L_f, (b) the d_w distance between $\mathbf{C}_f$ and the goal configuration of the path, and (c) a cost negatively proportional to the time available at $\mathbf{C}_f$ once the robot reaches it before a collision may happen. w_3, w_4, and w_5 are the respective weights.

Algorithm 3 decides the proper evaluation function to choose the best trajectory to execute in a new adaptation cycle. If there exist collision-free trajectories or trajectories with sufficiently long L_f, the evaluation function combines path **Length** and **GoalCost** for optimization; otherwise, **SegCost** is used to favor a trajectory with better obstacle avoidance.

Algorithm 3 Selection of the best trajectory

```
Input robot's current (collision-free) configuration $\mathbf{C}_c$
Input the current time $\tau$.
$S_f = \{\Gamma | \Gamma$ is completely collision-free and $\Gamma \in S\}$
**if** $S_f$ is not empty **then**
    Return the best $\Gamma \in S_f$ based on:
        $eval_1(\Gamma) = w_1 Length + w_2 GoalCost$    (1)
**else**
    **if** CT-point $(\mathbf{C}_c, \tau + \delta t)$ is collision free **then**
        **return** the best $\Gamma \in S$ based on Eq. (1)
    **else**
        **return** the best $\Gamma \in S$ based on:
            $eval_2(\Gamma) = SegCost$    (2)
    **end if**
**end if**
```

VI. Trajectory and Goal Modifications

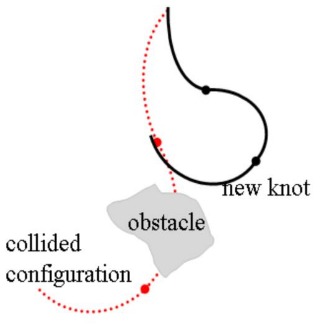
Initial trajectories in S are evolved into better ones during cycles of modifications in Algorithm 1. In each planning cycle, a trajectory is randomly selected to undergo one of the stochastic modification operations below:

- *Modify*: a randomly selected knot configuration is changed to a randomly generated configuration.
- *Insert*: a new knot configuration is randomly generated and inserted to the path.
- *Delete*: a randomly selected knot configuration is deleted.
- *Repair*: a new knot configuration is created out of a collided configuration to reduce collision.
- *Curl-up*:a new knot configuration is created to make the robot curl-up to reduce collision.
- *Shrink*: a new knot configuration is created out of a collided configuration by shrinking the robot size to reduce collision.

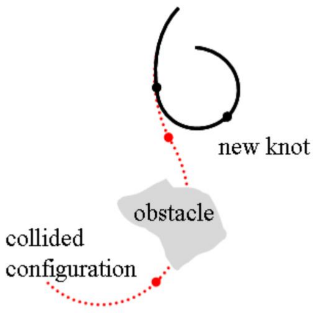
In addition, the goal configuration of the trajectory, if in collision with an obstacle, is also modified. A modified trajectory is used to either replace the worst one in S, if it is better than the worst, based on Eq. (1) in Algorithm 3, or replace a random non-best trajectory, with a probability. However, when the robot is forced to stop its motion, the planner will use the modified trajectory to replace a random non-best trajectory in S to increase the chance of finding a trajectory with a collision-free segment for the robot to move along again.

While the first three operations are generic RAMP operations, the last three, *Repair*, *Curl-up*, and *Shrink*, are designed to exploit the structure of a continuum robot, as detailed below. Given a trajectory, *Repair*, *Curl-up*, and *Shrink* all identify the earliest configuration $\mathbf{C}_c$ on the trajectory that is not collision-free and create a less colliding configuration $\mathbf{C}_{new}$ based on it. Each of these operators then uses $\mathbf{C}_{new}$ to replace $\mathbf{C}_c$ as a new knot configuration in the path.

To create $\mathbf{C}_{new}$, *Repair* does the following. Starting from $\mathbf{C}_c$, it first shortens the lengths of the sections from the base to the colliding section k closest to the base and changes the curvatures of these sections in the direction to "pull them back" (see Figure 6(a)), all in random amounts; next it maximally shortens the lengths of the sections from section k to the tip section and changes the curvatures of these sections in the direction to "push them forward" in random amounts (see Figure 6(a)). The resulting $\mathbf{C}_{new}$ reduces the original collision at $\mathbf{C_c}$. Repeated repairing could change a colliding trajectory to a collision-free one.



(a) repair



(b) curl-up

Fig. 6. The effects of *Repair* and *Curl-up* operations

Curl-up takes advantage of the deformable body of the continuum robot to reduce collisions. From the colliding section k closest to the base at the colliding configuration $\mathbf{C}_c$, it changes the curvatures of the sections from section k to the tip to "curl up" those sections in one direction and also maximally shortens the lengths of these sections. The resulting configuration $\mathbf{C}_{new}$ reduces the original collision at $\mathbf{C_c}$ and could eliminate it (see Figure 6(b)). If there are more colliding configurations ahead of $\mathbf{C}_c$ on the considered path/trajectory, repeated use of *Curl-up* could change the path/trajectory to a collision-free one.

Shrink tries to contract the robot body as much as possible. To do so, it starts from the colliding configuration $\mathbf{C_c}$, change the curvatures of all sections to the average curvature (so that they all have the same curvature), and then shrink all sections to their respective minimum lengths. The resulting configuration is $\mathbf{C}_{new}$, which is then used to replace $\mathbf{C_c}$ as a new knot configuration.

Our planner selects the modification operators by probability. It selects all operators randomly, except for *Modify*, which is given a higher probability of selection to increase diversity of paths. If *Repair* is selected to modify a path, it is applied two times consecutively so that two new knot configurations are created out of the first collided configuration of the original path and that of the once repaired path respectively.

Our planner also adapts the goal configurations generated during initialization to be better ones and free of obstacles in an uncertain environment. In each planning cycle, the goal configuration $\mathbf{C}_g$ of a trajectory Γ, if in collision, is modified through mutations in one of the stochastic steps of Algorithm 2: mutate p_3, mutate α, re-select φ, and re-select l_1. Note that "mutate" means small change of values, but "re-select" can result in large change of values. With such modifications, the basic shape of $\mathbf{C}_g$ is just mutated and not changed too much. The resulting new configuration, if better than $\mathbf{C}_g$ (i.e., collision-free and has lower $GoalCost$), is used to replace $\mathbf{C}_g$ for not only trajectory Γ but also all the trajectories that end at $\mathbf{C}_g$ in the set S.

While the set of goal configurations can be improved from its initial set, the above modifications still maintain the basic diversity of the initial set because a goal configuration is only replaced by an improved version through small changes in p_3 and α. The two homotopic groups of goal configurations, as depicted in Figure 2, will be maintained. Our planner further takes advantage of such diversity to avoid goal-related local optima: it allows the goal configuration of a path to be changed to one from a different homotopic group with some probability.

VII. Simulation

We have tested our algorithm on a planar OctArm robot in simulation environments. Each environment consists of randomly generated polygonal obstacles of arbitrary shapes and sizes. It also includes a randomly generated target object for the robot, also in arbitrary shape and size (in the wide range that the robot can "grasp" it). The obstacles can be either static or move randomly. The simulated robot has the same value ranges on (κ, s) for each section as the actual robot at Clemson University. It can have either a static base or a base that translates horizontally. The simulation was conducted on a Dell Precision T5400.

A. Assumptions

In the simulation environment, we assume that the poses of both the target object and the obstacles are sensed in sensing cycles, with a frequency of 20 Hz. Even though the obstacles may move randomly, they will not run over a stopped robot. This essentially assumes that the obstacles, which can be either moved by people or are other robots, are not malicious. Our robot is capable of avoiding others during its motion, but when it is static, others are not assumed to harm it.

We assume the robot can move with a constant speed with instant acceleration/deceleration to simplify trajectory generation. We also ignore the width of the OctArm robot for simplicity.

B. Implementation and results

In our implementation, the key parameters of the planner have values as shown in Table I. The first row gives the number of trajectories in S, number of goal configurations generated, and the maximum $d_{\kappa s \phi}$ distance D between two adjacent knot configurations. The rest of the rows provide weight values for the evaluation functions. A new adaptation cycle starts either after every 3 planning cycles or when the robot finishes executing the collision-free trajectory segment L_f.

We applied our real-time planner to different simulation environments of randomly generated obstacles. In the graphical display, 1 unit=15 cm in real world. The upper bound on the obstacle speed is $v_{max} = 1$ unit/s. The robot speed is greater than 1 unit/s.

The attached video shows three examples of running our planner: one is a fixed-base robot in a static environment, and the other two are a robot with a horizontally moving base in two dynamic environments with unknown obstacle motions: exp. 1 (with three dynamic obstacles) and exp. 2 (with three static and six dynamic obstacles), which are also shown in Figure 7. In each case the target object is indicated by the bounding circle, and Figure 7 also shows the goal configurations automatically generated. In all cases, the robot is initially in the vertical configuration stretched (i.e., with zero curvature for all sections).

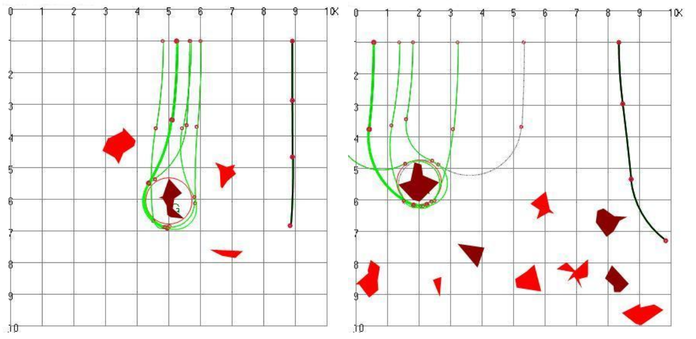

(a) Exp. 1 (with 3 dynamic obstacles) (b) Exp. 2 (with 3 static and 6 dynamic obstacles)

Fig. 7. Snapshots from two experiments, where the robot is in dark green, possible goal configurations are in light green, static obstacles are in burgundy, and dynamic obstacles are in red.

Table II shows the results averaged over 20 runs for exp. 1 and exp. 2 respectively. The results show that there are just a few forced stops in both cases (when the robot did not find a collision-free segment to follow), which is also confirmed by the high percentage of replacing the worst in S with a modified trajectory in each planning cycle. Recall that during forced stops, the planner uses a modified trajectory to replace a randomly chosen non-best one in S (see section VI). The percentage of adaptation based on $eval_1$ of Algorithm 3 is quite high for the relatively simpler exp. 1, but it is significantly lower for the more challenging exp. 2, showing that the robot spent more time avoiding obstacles by adaptation based on $eval_2$. Note that the $\%$adaptation by $eval_2$ (not shown in Table II) is $1-\%$adaptation by $eval_1$ (shown in Table II), which is about 51% for exp. 2. For both experiments, however, the robot always successfully reached a goal configuration.

Table III shows the performance of the stochastic modification operators for exp. 1 and exp. 2, averaged over 20 runs. For every time an operator is applied to a path, if the modification results in a better path, the operator is considered successful. The $\%$Success indicates the ratio between number of successful times and total number of times an operator is used. The result shows that all operators were quite useful, but the operators specifically designed for the continuum robot: *repair*, *curl-up*, and *shrink* were more effective than the generic operators *insert* and *modify*. In fact, when we ran exp. 2 without using *repair*, *curl-up*, and *shrink*, the robot could not successfully reach a goal configuration, showing that these operators were critical. The *delete* operator is the only operator that reduces the number of knot configurations. Its high $\%$success for both experiments shows that it was highly effective in shortening lengths of trajectories.

TABLE I
PARAMETER VALUES OF THE PLANNER

$\|S\|$	5	# **goals**	5	D	5
w_{1-5}	6	2	1	2	2
u_{1-3}	0.6	0.25	0.15		
b_{1-2}	3	1			

TABLE II
AVERAGE RESULTS OF 20 RUNS OF EACH EXPERIMENT

exp.	total time(s)	#stops	# plan cycles	% replace worst	% adaptation based on $eval_1$
1	28.74	3.05	186.15	59.66%	76.78%
2	55.41	2.35	369.35	77.87%	49.12%

TABLE III
%SUCCESS OF MODIFICATION OPERATORS AVERAGED OVER 20 RUNS OF EACH EXPERIMENT

exp.	repair	curl-up	shrink	insert	modify	delete
1	71%	79.45%	65.27%	59.51%	61.44%	76.67%
2	76.18%	79.19%	77.78%	70.40%	73.60%	77.83%

VIII. CONCLUSIONS & FUTURE WORK

We have introduced a real-time adaptive motion planner for a planar continuum manipulator to grasp an object amid an environment of arbitrary obstacles with uncertain movements. Our planner plans goal configurations of the whole arm together with the paths/trajectories leading to goal configurations in real-time to tackle changes in the environment, simultaneously as the manipulator moves. It extends the general RAMP paradigm [19] by exploiting the specific characteristics of the continuum manipulator to be effective. It also introduces an efficient real-time collision-checking algorithm based on the unique concept of dynamic envelopes [20] so that a found collision-free trajectory is guaranteed collision-free no matter how obstacles move.

As the next step, we will extend the planner in two directions: (1) extending it to handle compliant wrapping and picking up a target object starting from the goal configurations considered in this paper, and (2) extending it to a spatial continuum manipulator with non-zero ϕ's. The flexible nature of our planner facilitates both extensions. We will also test our planner on the real OctArm at Clemson University with real sensors, collaborating with Professor Ian Walker's group.

Acknowledgment

This work is supported by the US National Science Foundation grant IIS-0904093. The authors also thank Ian Walker and Bryan Jones for helpful discussions regarding the Clemson OctArm manipulator.

References

[1] D. Berenson, J. Kuffner, and H. Choset, "An Optimization Approach to Planning for Mobile Manipulation," *Proc. of IEEE Int. Conf. Robotics and Automation*(ICRA), pp. 1187-1192, May 2008.
[2] P.P. Bonissone, R. Subbu, N. Eklund, and T.R. Kiehl, "Evolutionary algorithms + domain knowledge = real-world evolutionary computation," *IEEE Trans. Evolutionary Computation*, 10(3):256-280, April 2006.
[3] G.S. Chirikjian and J.W. Burdick, "An Obstacle Avoidance Algorithm for Hyper-Redundant Manipulators," *ICRA*, May 1990.
[4] G.S. Chirikjian and J.W. Burdick, "A Hyper-Redundant Manipulator," *IEEE Robot. and Automat. Magazine*, pp. 22-29, Dec. 1994.
[5] H. Choset and W. Henning, "A follow-the-leader approach to serpentine robot motion planning," *ASCE J. Areospace Engin.*, 12(2):65-73, April 1999.
[6] R. Cieslak and A. Morecki, "Elephant trunk type elastic manipulator a tool for bulk and liquid type materials transportation," *Robotica*, 17:11-16, 1999.
[7] B. Dasgupta, A. Gupta, and E. Singla, "A variational approach to path planning for hyper-redundant manipulators," *J. Robotics and Autonomous Sys.*, 57(2):194-201, Feb. 2009.
[8] B.A. Jones and I.D. Walker, "Kinematics for multisection continuum robots," *IEEE Trans. Robot.*, 22(1):43-55, Feb. 2006.
[9] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, 12(4):566-580, 1996.
[10] S.M. LaValle and J.J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*(IJRR), 20(5):378-400, May 2001.
[11] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *IJRR*, 21:999–1030, 2002.
[12] S. Ma and M. Konno, "An obstacle avoidance scheme for hyper-redundant manipulators-global motion planning in posture space," *ICRA*, pp. 161-166, April 1997.
[13] W. McMahan, B.A. Jones, I.D. Walker, V. Chitrakaran, A. Seshadri, and D. Dawson, "Robotic manipulators inspired by cephalopod limbs," *Proc. of the CDEN Design Conf.*, Montreal, Canada, pp. 1-10, July 2004.
[14] M. Moll and L.E. Kavraki, "Path Planning for Deformable Linear Objects," *IEEE Trans. on Robotics*, 22(4):625636, August 2006.
[15] S. Neppalli, M.A. Csencsits, B.A. Jones, and I.D. Walker, "Closed-form Inverse Kinematics for Continuum Manipulators," *Advanced Robotics*, 23:2077-2091, 2009.
[16] D. Reznik and V. Lumelsky, "Motion planning with uncertainty for highly redundant kinematic structures I. "Free Snake" Motion," *ICRA*, pp. 1747-1752, July 1992.
[17] G. Robinson and J.B.C. Davies, "Continuum Robots - A State of the Art," *ICRA*, pp. 2849-2854, May 1999.
[18] M. Saha and P. Isto, "Motion Planning for Robotic Manipulation of Deformable Linear Objects," *ICRA*, pp. 2478-2484, May 2006.
[19] J. Vannoy and J. Xiao, "Real-time Adaptive Motion Planning (RAMP) of Mobile Manipulators in Dynamic Environments with Unforeseen Changes," *IEEE Trans. Robotics*, 24(5):1199-1212, Oct. 2008.
[20] R. Vatcha and J. Xiao, "Perceived CT-space for motion planning in unknown and unpredictable environments," *8th Int. Workshop Alg. Foundations of Robotics*, Guanajuato, Mexico, Dec. 2008.
[21] H. Wakamatsu, E. Arai, S. Hirai, "Knotting/unknotting Manipulation of Deformable Linear Objects," *IJRR*, 25(4):371-395, April 2006.
[22] Y. Yang and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments," *Robotics Sci. & Sys. II*, The MIT Press, pp. 279-286, 2006.