Article

The International Journal of Robotics Research
2020, Vol. 39(8) 957–982

DOI: 10.1177/0278364920918299
journals.sagepub.com/home/ijr

Manipulating deformable objects by interleaving prediction, planning, and control

Dale McConachie, Andrew Dobson, Mengyao Ruan and Dmitry Berenson

Abstract

We present a framework for deformable object manipulation that interleaves planning and control, enabling complex manipulation tasks without relying on high-fidelity modeling or simulation. The key question we address is when should we use planning and when should we use control to achieve the task? Planners are designed to find paths through complex configuration spaces, but for highly underactuated systems, such as deformable objects, achieving a specific configuration is very difficult even with high-fidelity models. Conversely, controllers can be designed to achieve specific configurations, but they can be trapped in undesirable local minima owing to obstacles. Our approach consists of three components: (1) a global motion planner to generate gross motion of the deformable object; (2) a local controller for refinement of the configuration of the deformable object; and (3) a novel deadlock prediction algorithm to determine when to use planning versus control. By separating planning from control we are able to use different representations of the deformable object, reducing overall complexity and enabling efficient computation of motion. We provide a detailed proof of probabilistic completeness for our planner, which is valid despite the fact that our system is underactuated and we do not have a steering function. We then demonstrate that our framework is able to successfully perform several manipulation tasks with rope and cloth in simulation, which cannot be performed using either our controller or planner alone. These experiments suggest that our planner can generate paths efficiently, taking under a second on average to find a feasible path in three out of four scenarios. We also show that our framework is effective on a 16-degree-of-freedom physical robot, where reachability and dual-arm constraints make the planning more difficult.

Keywords

Deformable objects, deformable object manipulation, motion planning, compliant motion planning, planning and control

1. Introduction

Examples of deformable object manipulation range from domestic tasks such as folding clothes to time- and safety-critical tasks such as robotic surgery. One of the challenges in planning for deformable object manipulation is the high number of degrees of freedom (DoFs) involved; even approximating the configuration of a piece of cloth in three dimensions with a 4×4 grid results in a 48-DoF configuration space. In addition, the dynamics of the deformable object are difficult to model (Essahbi et al., 2012); even with high-fidelity modeling and simulation, planning for an individual task can take hours (Bai et al., 2016). Local controllers on the other hand are able to very efficiently generate motion, however, they are only able to successfully complete a task when the initial configuration is in the "basin of attraction" of the goal (Berenson, 2013; McConachie and Berenson, 2018).

The central question we address in this work is how can we combine the strengths of global planning with the strengths of local control while mitigating the weakness of each? We propose a framework for interleaving planning and control which uses global planning to generate gross motion of the deformable object, and a local controller to refine the configuration of the deformable object within the local neighborhood. By separating planning from control we are able to use different representations of the deformable object, each suited to efficient computation for their respective roles. In order to determine when to use each component, we introduce a novel deadlock prediction algorithm that is inspired by topologically-based motion planning methods (Bhattacharya et al., 2012; Jaillet and Siméon, 2008). By answering the question "Will the local controller get stuck?", we can predict whether the local

Robotics Institute, University of Michigan, Ann Arbor, MI, USA

Corresponding author:
Dale McConachie, Robotics Institute, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109, USA.
Email: dmcconac@umich.edu

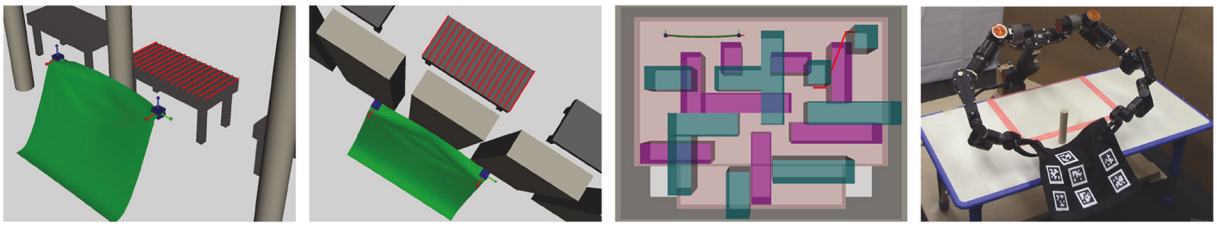

Fig. 1. Four example manipulation tasks for our framework. In the first two tasks, the objective is to cover the surface of the table (indicated by the red lines) with the cloth (shown in green). In the first task, the grippers (shown in blue) can freely move, however the cloth is obstructed by a pillar. In the second task, the grippers must pass through a narrow passage before the table can be covered. In the third task, the robot must navigate a rope (shown in green in the top left corner) through a three-dimensional maze before covering the red points in the top right corner. The maze consists of top and bottom layers (purple and green, respectively). The rope starts in the bottom layer and must move to the target on the top layer through an opening (bottom left or bottom right). For the fourth task, the physical robot must move the cloth from the far side of an obstacle to the region marked in pink near the base of the robot.
Note: Colour version of the figure is available online.

controller will be unable to achieve the task from the current configuration. If we predict that the controller will get stuck, we can then invoke the global planner, moving the deformable object into a new neighborhood from which the local controller may be able to succeed. The key to our efficient prediction is forward-propagating only the stretching constraint, assuming the object will otherwise comply to contact.

We seek to solve problems for one-dimensional and two-dimensional deformable objects (i.e., rope and cloth) where we need to arrange the object in a particular way (e.g., covering a table with a tablecloth) but where there is also complex environment geometry preventing us from directly completing the task. While we cannot claim to solve all problems in this class (in particular, in environments where the deformable object can be snagged), we can still solve practical problems where the path of the deformable object is obstructed by obstacles. In this work, we restrict our focus to controllers of the form described in Section 4.1, and tasks suited to these controllers. Examples of these types of tasks are shown in Figure 1. In our experiments we show that this iterative method of interleaving planning and control is able to successfully perform several interesting tasks where our planner or controller alone are unable to succeed.

Our contributions are: (1) a novel deadlock prediction algorithm to determine when a global planner is needed; (2) an efficient and probabilistically complete global planner for rope and cloth manipulation tasks; and (3) a framework to combine local control and global motion planning to leverage the strengths of each while mitigating their weaknesses. We present experiments in both a simulated environment and on a physical robot (Figure 1). Our results suggest that our planner can efficiently find paths, taking under a second on average to generate a feasible path in three out of four simulated scenarios. The physical experiment shows that our framework is able to effectively perform tasks in the real world, where reachability and dual-arm constraints make the planning more difficult.

A preliminary version of this work was presented in McConachie et al. (2017). This article extends this work by adding an additional experiment on a physical robotic system as well as a proof of the probabilistic completeness of our planning method. We have also improved planning times with an improved goal bias method. We also include additional related work and an expanded discussion.

2. Related work

Robotic manipulation of deformable objects has been studied in many contexts ranging from surgery to industrial manipulation (see Khalil and Payeur (2010) and Sanchez et al. (2018) for extensive surveys). In the following, we discuss the most relevant methods to the work presented here, starting with methods of simulating and planning for deformable objects. We then discuss visual servoing and learning-based methods for similar tasks. In addition to previous work in deformable object manipulation, we also discuss related work in planning/control for robot arms and ways to consider topology in planning, which we draw from for our framework. We end with a discussion of probabilistic completeness and describe why previous methods to show this property do not apply, motivating our proof method.

Much work in deformable object manipulation relies on simulating an accurate model of the object being manipulated. Motivated by applications in computer graphics and surgical training, many methods have been developed for simulating string-like objects (Bergou et al., 2008; Rungjiratananon et al., 2011) and cloth-like objects (Baraff and Witkin, 1998; Goldenthal et al., 2007). The most common simulation methods use mass–spring models (Essahbi et al., 2012; Gibson and Mirtich, 1997), which are generally not accurate for large deformations (Maris et al., 2010), and finite-element method (FEM) models (Irving et al., 2004; Kaufmann et al., 2008; Müller et al., 2002). FEM-based methods are widely used and physically well-founded, but they can be unstable when subject to contact constraints, which are especially important in this work. They also require significant tuning and are very sensitive to the discretization of the object. Furthermore, such models require

knowledge of the physical properties of the object, such as its Young's modulus and friction parameters, which we do not assume are known.

Motion planning for manipulation of deformable objects is an active area of research (Jiménez, 2012). Saha et al. (2008) presented a probabilistic roadmap (PRM) (Kavraki et al., 1996) that plans for knot-tying tasks with rope. Rodriguez and Amato (2006) study motion planning in fully deformable simulation environments. Their method, based on Rapidly-exploring Random Trees (RRTs) (LaValle, 2006), applies forces directly to an object to move it through narrow spaces while using the simulator to compute the resulting deformations. Frank et al. (2011) presented a method that pre-computes deformation simulations in a given environment to enable fast multi-query planning. Other sampling-based approaches have also been proposed (Anshelevich et al., 2000; Burchan Bayazit et al., 2002; Gayle et al., 2005; Lamiraux and Kavraki, 2001; Moll and Kavraki, 2006; Roussel et al., 2015). However, all of these methods either disallow contact with the environment or rely on potentially time-consuming physical simulation of the deformable object, which is often very sensitive to physical and computational parameters that may be difficult to determine. In contrast, our method uses simplified models for control and motion planning with far lower computational cost. In addition, the use of a local controller is not considered in the previous methods, instead relying on a global planner (and, thus, implicitly on the accuracy of the simulator) to generate a path that completes the entire task.

Model-based visual servoing approaches bypass planning entirely, and instead use a local controller to determine how to move the robot end-effector for a given task (Hirai and Wada, 2000; Smolen and Patriciu, 2009; Wada et al., 2001). Our recent work (Berenson, 2013; McConachie and Berenson, 2018) as well as that of Navarro-Alarcon and Liu (2018); Navarro-Alarcon et al. (2014, 2016) bypasses the need for an explicit deformable object model, instead using approximations of the Jacobian to drive the deformable object to the attractor of the starting state. More recent work by Hu et al. (2018) has enabled the use of Gaussian process regression while controlling a deformable object. Rather than using only a planner or only a controller, our framework uses both components, each when appropriate.

Approaches based on learning from demonstration avoid planning and deformable object modeling challenges entirely by using offline demonstrations to teach the robot specific manipulation tasks (Huang et al., 2015; Schulman et al., 2016); however, when a new task is attempted, a new training set needs to be generated. In our application we are interested in a way to manipulate a deformable object without having a high-fidelity model or training set available a priori. For instance, imagine a robot encountering a new piece of clothing for a new task. While it may have models for previously seen clothes or training sets for previous tasks, there is no guarantee that those models or training sets are appropriate for the new task.

Park et al. (2014) considered interleaving planning and control for arm reaching tasks in rigid unknown environments. In their method, they assumed an initially unknown environment in which they plan a path to a specific end-effector position. This path is then followed by a local controller until the task is complete, or the local controller gets stuck. If the local controller gets stuck, then a new path is planned and the cycle repeats. In contrast, our controller is performing the task directly rather than following a planned reference trajectory, incorporating deadlock prediction into the execution loop, while our global planner is planning for both the robot motion as well as the deformable object stretching constraint.

Our planning method has some similarity to topological (Bhattacharya et al., 2012; Jaillet and Siméon, 2008) and tethered robot (Brass et al., 2015; Kim and Likhachev, 2015) planning techniques; these methods use the topological structure of the space to define homotopy classes, either as a direct planning goal, or as a way to help inform planning in the case of tethered robots. Planning for some deformable objects, in particular rope or string, can be viewed as an extension of the tethered robot case where the base of the tether can move. This extension, however, requires a very different approach to homotopy than is commonly used, particularly when working in three-dimensional space instead of a planar environment. In our work, we use *visibility deformations* from Jaillet and Siméon (2008) as a way to encode homotopy-like classes of configurations.

Previous approaches to proving probabilistic completeness for efficient planning of underactuated systems rely on the existence of a steering function to move the system from one region of the state space to another, or choosing controls at random (Karaman and Frazzoli, 2013; Kunz and Stilman, 2015; LaValle and Kuffner, 2001; Li et al., 2016). For deformable objects, a computationally efficient steering function is not available, and using random controls can lead to prohibitively long planning times. Roussel et al. (2015) bypassed this challenge by analyzing completeness in the submanifold of quasi-static contact-free configurations of extensible elastic rods. In contrast, we show that our method is probabilistically complete even when contact between the deformable object and obstacles is considered along the path. Note that it is especially important to allow contact at the goal configuration of the object to achieve coverage tasks. Li et al. (2016) presented an efficient asymptotically optimal planner that does not need a steering function, however, they do rely on the existence of a contact-free trajectory where every point in the trajectory is in the interior of the valid configuration space. Our proof of probabilistic completeness is based on Li et al. (2016), but we allow for the deformable object to be in contact with obstacles along a given trajectory.

3. Problem statement

Define the robot configuration space to be $\mathbb{C}_r$. We assume that the robot configuration can be measured exactly. Denote an individual robot configuration as $q_r \in \mathbb{C}_r$. This set can be partitioned into a valid and invalid set. The valid set is referred to as $\mathbb{C}_r^{valid}$, and is the set of configurations where the robot is not in collision with the static geometry of the world. The invalid set is referred to as $\mathbb{C}_r^{inv} = \mathbb{C}_r \setminus \mathbb{C}_r^{valid}$.

We assume that our model of the robot is purely kinematic, with no higher-order dynamics. We assume that the robot has two end-effectors that are rigidly attached to the object. The configuration of a deformable object is a set $\mathcal{P} \subset \mathbb{R}^3$ of $P = |\mathcal{P}|$ points. We assume that we have a method of sensing $\mathcal{P}$. The rest of the environment is denoted by $\mathcal{O}$ and is assumed to be both static and known exactly. We assume that the robot moves slowly enough that we can treat the combined robot and deformable object as quasi-static. Let the function $f(q_r, \mathcal{P}, \dot{q}_r)$ map the system configuration $(q_r, \mathcal{P})$ and robot movement $\dot{q}_r$ to the corresponding deformable object movement $\dot{\mathcal{P}}$. We assume that the deformable object will be damaged if it is stretched beyond a factor λ_s from the relaxed state. Let $D \in \mathbb{R}^{P^2}$ be the symmetric matrix of pairwise distances between all points of $\mathcal{P}$ in its relaxed state. We assume that there are no other deformable object properties (such as bending energy) that are relevant to the task.

We define a task based on a set of T target points $\mathcal{T} \subset \mathbb{R}^3$, a function $\rho : \mathcal{P} \times \mathcal{T} \rightarrow \mathbb{R}^{\geq 0}$, which measures the alignment error between $\mathcal{P}$ and $\mathcal{T}$, and a termination function $\Omega(\mathcal{P})$, which indicates whether the task is finished. Let a robot controller be a function $C(q_r, \mathcal{P}, \mathcal{T})$,[1] which maps the system state $(q_r, \mathcal{P})$ and alignment targets $\mathcal{T}$ to a desired robot motion $\dot{q}_r^{cmd}$. In this work, we restrict our discussion to tasks and controllers of the form introduced in our previous work (Berenson, 2013; McConachie and Berenson, 2018); these controllers are local, i.e., at each time t they choose an incremental movement $\dot{q}_r^{cmd}$ which reduces the alignment error as much as possible at time $t+1$.

The problem we address in this work is how to find a sequence of N_e robot commands $\{\dot{q}_{r,1}^{cmd}, \ldots, \dot{q}_{r,N_e}^{cmd}\} = \dot{Q}_r^{cmd}$ such that each motion is feasible, i.e., it should not bring the grippers into collision with obstacles, should not cause the object to stretch excessively, and should not exceed the robot's maximum velocity $\dot{q}_r^{\max}$. Let these feasibility constraints be represented by $A(\dot{q}_r) = 0$. Then the problem we seek to solve is

$$\begin{aligned} \text{find} \quad & \dot{Q}_r^{cmd} \\ \text{s.t.} \quad & \Omega(\mathcal{P}_{N_e}) = \texttt{true} \\ & A(\dot{q}_{r,t}^{cmd}) = 0, \quad t = 1, \ldots, N_e \end{aligned} \tag{1}$$

where $\mathcal{P}_{N_e}$ is the configuration of the deformable object after executing $\dot{Q}_r^{cmd}$.

Solving this problem directly is impractical in the general case for two major reasons. First, modeling a deformable object accurately is very difficult in the general case, especially if it contacts other objects or itself. Second, even given a perfect model, computing precise motion of the deformable object requires physical simulation, which can be very time consuming inside a planner/controller where many potential movements need to be evaluated. We seek a method that does not rely on high-fidelity modeling and simulation; instead, we present a framework combining both global planning and local control to leverage the strengths of each in order to efficiently perform the task.

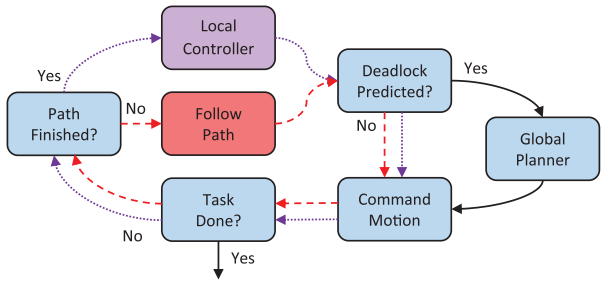

Fig. 2. Block diagram showing the major components of our framework. On each cycle we use either the local controller (dotted purple arrows) or a planned path (dashed red arrows) to predict whether the system will be deadlocked in the future, planning a new path is needed to avoid deadlock.
Note: Colour version of the figure is available online.

4. Interleaving planning and control

Global planners are effective at finding paths through complex configuration spaces, but for highly underactuated systems such as deformable objects, achieving a specific configuration is very difficult even with high-fidelity models; this means that we cannot rely on them to complete a task independent of a local controller. In order for the local controller to complete the task, the system must be in the correct basin of attraction. From this point of view it is not the planner's responsibility to complete a task but rather to move the system into the right basin for the local controller to finish the task. By explicitly separating planning from control we can use different representations of the deformable object for each component; this allows us to use a highly simplified model of the deformable object for global planning to generate gross motion of the deformable object, while using an independent local approximation for the controller. The key question then is when should we use global planning versus local control?

Our framework can be broken down into three major components: (1) a global motion planner to generate gross motion of the deformable object; (2) a local controller for refinement of the configuration of the deformable object; and (3) a novel deadlock prediction algorithm to determine when to use planning versus control. Figure 2 shows how

1. A specific controller may have additional parameters (such as gains in a PID controller), but we do not include such parameters here to keep $C(\ldots)$ in a more general form.

these components are connected, switching between a local controller loop and planned path execution loop as needed. In the following sections, we describe each component in turn, starting with the local controller.

4.1. Local control

The role of the local controller is not to perform the whole task, but rather to refine the configuration of the deformable object locally. For our local controller we use a controller of the form introduced in Berenson (2013) and McConachie and Berenson (2018). These controllers locally minimize error ρ while avoiding robot collision and excessive stretching of the deformable object.

An outline of how these controllers function is shown in Algorithm 1; first, for every target point $\mathcal{T}_i \in \mathcal{T}$ we define a workspace navigation function pointing towards $\mathcal{T}_i$ using Dijkstra's algorithm. This gives us the shortest collision-free path between any point in the workspace and the target point, as well as the distance traveled along that path. These navigation functions are used to define the best direction to move the deformable object $\dot{\mathcal{P}}_e$ and the relative importance of each part of the motion W_e in order to locally reduce error as much as possible at each timestep (lines 1 and 2). These error reduction terms are then combined using relative importance weight λ_w with stretching avoidance terms $\dot{\mathcal{P}}_s, W_s$ to define the desired manipulation direction and importance weights $\dot{\mathcal{P}}_d, W_d$ at each timestep (lines 3 and 4). If these terms conflict, then stretching correction takes precedence. We then find the best robot motion to achieve the desired deformable object motion, while preventing collision between the robot and obstacles (line 5).

Given the current system state $(q_r, \mathcal{P})$ FindBestRobotMotion$(q_r, \mathcal{P}, \dot{\mathcal{P}}_d, W_d)$ is solving the following problem:

$$\begin{aligned} \underset{\dot{q}_r}{\text{argmin}} \quad & \|f(q_r, \mathcal{P}, \dot{q}_r) - \dot{\mathcal{P}}_d\|_{W_d} \\ \text{subject to} \quad & \|\dot{q}_r\| \leq \dot{q}_r^{\max} \\ & (q_r + \dot{q}_r) \in \mathbb{C}_r^{valid} \end{aligned} \tag{2}$$

How (2) is solved depends on the particular robot; details for each function in Algorithm 1 are given in Appendix B.

An important limitation of this approach is that the individual navigation functions are defined and applied independently of each other; this means that the navigation functions that are combined to define the direction to move the deformable object can cause the controller to move the end-effectors on opposite sides of an obstacle, leading to poor local minima, i.e., becoming stuck. Figure 3 shows our motivating example of this type of situation. Other examples of this kind of situation are shown in Section 7. In addition, while this local controller prevents collision between the robot and obstacles, it does not explicitly have any ability to *go around* obstacles.

Algorithm 1 LocalController$(q_r, \mathcal{P}, \mathcal{T}, D, \lambda_s, \lambda_w)$

1: $\mathcal{T}_c \leftarrow$ CalculateCorrespondences$(\mathcal{P}_t, \mathcal{T})$
2: $\dot{\mathcal{P}}_e, W_e \leftarrow$ FollowNavigationFunction$(\mathcal{P}_n, \mathcal{T}_c)$
3: $\dot{\mathcal{P}}_s, W_s \leftarrow$ StretchingCorrection$(D, \lambda_s, \mathcal{P})$
4: $\dot{\mathcal{P}}_d, W_d \leftarrow$ CombineTerms$(\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s, \lambda_w)$
5: $\dot{q}_r^{cmd} \leftarrow$ FindBestRobotMotion$(q_r, \mathcal{P}, \dot{\mathcal{P}}_d, W_d)$

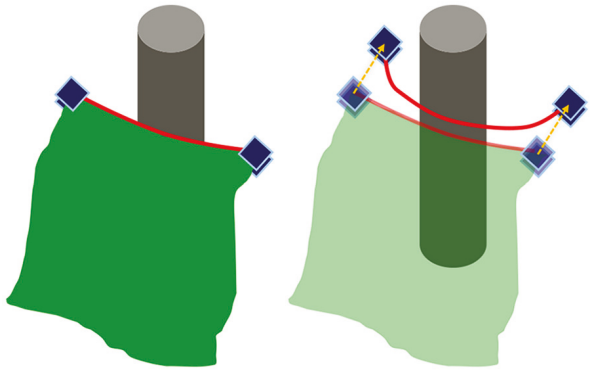

Fig. 3. Motivating example for deadlock prediction. The local controller moves the grippers on opposite sides of an obstacle, while the geodesic between the grippers (red line) cannot move past the pole, eventually leading to overstretch or tearing of the deformable object if the robot does not stop moving towards the goal.
Note: Colour version of the figure is available online.

In order to address these limitations we introduce a novel deadlock prediction algorithm to detect when the system $(q_{r,t}, \mathcal{P}_t)$ is in a state that will lead to deadlock (i.e., becoming stuck) if we continue to use the local controller.

4.2. Predicting deadlock

Predicting deadlock is important for two reasons: first, we do not want to waste time executing motions that will not achieve the task; second, we want to avoid the computational expense of planning our way out of a cul-de-sac after reaching a stuck state. By predicting deadlock before it happens we address both of these concerns. The key idea is to detect situations similar to Figure 3 where the local controller will wrap the deformable object around an obstacle without completing the task. We also need to detect situations where no progress can be made owing to an obstacle being directly in the path of the desired motion of the robot.

Let $E(q_r, \mathcal{P}, \dot{q}_r^{cmd}) = \dot{q}_r^{act}$ be the true motion of the robot when $\dot{q}_r^{cmd}$ is executed for unit time; in this section, we predict the future state of the system, thus it is not sufficient to consider only $\dot{q}_r^{cmd}$, we must also consider $\dot{q}_r^{act}$. Modeling inaccuracies as well as the deformable object being in contact can lead to meaningful differences between $\dot{q}_r^{cmd}$ and $\dot{q}_r^{act}$. Specifically, when a deformable object is in contact with the environment, tracking $\dot{q}_r^{cmd}$ perfectly may lead to a constraint violation (i.e., overstretch or tearing of the deformable object).

We consider a controller to be deadlocked if the commanded motion produces (nearly) no actual motion, and the task termination condition is not met:

$$\begin{aligned} \|\dot{q}_{r,t}^{act}\| &\approx 0 \\ \Omega(\mathcal{P}_t) &= \texttt{false} \end{aligned} \tag{3}$$

In general, we cannot predict whether the system will get stuck in the limit; to do so would require a very accurate simulation of the deformable object. Instead we predict whether the system will get stuck within a prediction horizon N_p timesteps. We divide our deadlock prediction algorithm into three parts and discuss each in turn: (1) estimating gross motion; (2) predicting overstretch; and (3) progress detection.

4.2.1. Estimating gross motion. The idea central to our prediction (Algorithm 2) is that while we may not be able to determine precisely how a given controller will steer the system, we can capture the gross motion of the system and estimate whether the controller will be deadlocked. We split the prediction into two parts. First, we assume that controller C is able to manipulate the deformable object with a reasonable degree of accuracy within a local neighborhood of the current state. This allows us to approximate the motion of the deformable object by following the task-defined navigation functions for each $\mathcal{P}_i \in \mathcal{P}$. Examples of this approximation are shown in Figure 4.

Next we use a simplified version of *LocalController*( ), which omits the stretching avoidance terms (Algorithm 1, lines 3 and 4) to predict the commands sent to the robot. These terms are omitted as they can be sensitive to the exact configuration of the deformable object, which is not considered in this approximation. If we are executing a path, then we can use the planned path directly to predict overstretch.

Algorithm 2 PredictDeadlock($\rho, q_{r,t}, \mathcal{P}_t, B_t, \mathcal{T}, L_{\max}, N_p$, Path)

```
1: ConfigHistory ← [ConfigHistory, q_t]
2: ErrorHistory ← [ErrorHistory, ρ(P_t)]
3: BandPredictions ← []
4: T_c ← CalculateCorrespondences(P_t, T)
5: for n = t, ..., t + N_p − 1 do
6:     if Path ≠ ∅ then
7:         Ṗ_e, W_e ← FollowNavigationFunction(P_n, T_c)
8:         P_{n+1} ← P_n + Ṗ_e
9:         q̇^cmd_{r,n} ← FindBestRobotMotion(q_{r,n}, P_n, Ṗ_e, W_e)
10:        q_{r,n+1} ← q_{r,n} + q̇^cmd_{r,n}
11:    else
12:        q_{r,n+1} ← q_{r,n} + FollowPath(Path)
13:    end if
14:    B_{n+1} ← ForwardPropagateBand(B_n, q_{r,n+1})
15:    BandPredictions ← [BandPredictions, B_{n+1}]
16: end for
17: if PredictOverstretch( BandPredictions, L_max) or
18: NoProgress(ConfigHistory, ErrorHistory) then
19:    return true
20: else
21:    return false
22: end if
```

Algorithm 3 ForwardPropagateBand(B, q_r)

```
1: (p_0, p_1) ← ForwardKinematics(q_r)
2: B ← [p_0, B, p_1]
3: B ← InterpolateBandPoints(B)
4: B ← RemoveExtraBandPoints(B)
5: B ← PullTight(B)
6: return B
```

4.2.2. Predicting overstretch. Next we introduce the notion of a *virtual elastic band* between the robot's end-effectors. This elastic band represents the shortest path through the deformable object between the end-effectors. The band approximates the constraint imposed by the deformable object on the motion of the robot; if the end-effectors move too far apart, then the elastic band will be too long, and thus the deformable object is stretched beyond a task-specified maximum stretching factor λ_s. Similarly, if the elastic band gets caught on an obstacle and becomes too long, then the deformable object is also overstretched. By considering only the geodesic between the end-effectors, we are assuming that the rest of the deformable object will comply to the environment, and does not need to be considered when predicting overstretch. The elastic band representation allows us to use a fast prediction method, but does not account for the part of the material that is slack. We discuss this trade-off further in Section 9. This virtual elastic band is based on Quinlan's path deformation algorithm (Quinnlan, 1994) and is used both in deadlock prediction as well as global planning (Sections 4.3 and 5).

Denote the configuration of an elastic band at time t as a sequence of $N_{b,t}$ points $B_t \subset \mathbb{R}^3$. The number of points used to represent an elastic band can change over time, but for any given environment and deformable object there is an upper limit $N_b^{\max}$ on the number of points used. Define Path(B) to be the straight line interpolation of all points in B. Define the length of a band to be the length of this straight line interpolation. At each timestep the elastic band is initialized with the shortest path between the end-effectors through the deformable object, and then "pulled" tight using the internal contraction force described in Quinlan (1994: section 5), and a hard constraint for collision avoidance. The endpoints of the band track the predicted translation of the end-effectors (Algorithm 3). This band represents the constraint that must be satisfied for the object not to tear. By considering only this constraint on the object in prediction, we are implicitly relying on the object to comply to contact as it is moved by the robot. We discuss the limitations of this assumption in the discussion (Section 9).

Let L_{t+n} be the length of the path defined by the virtual elastic band B_{t+n} at timestep n in the future, and $L_{\max}$ be the

longest allowable band length. To use this length sequence to predict if the controller will overstretch the deformable object, we perform three filtering steps: an annealing low-pass filter, a filter to eliminate cases where the band is in freespace, and the detector itself which predicts overstretch. We use a low-pass annealing filter with annealing constant $\alpha \in [0, 1)$ to mitigate the effect of numerical and approximation errors which could otherwise lead to unnecessary planning:

$$\begin{aligned} \tilde{L}_{t+1} &= L_{t+1} \\ \tilde{L}_{t+n} &= \alpha \tilde{L}_{t+n-1} + (1-\alpha) L_{t+n}, \quad n = 2, \ldots, N_p \end{aligned} \tag{4}$$

Second, we discard from consideration any bands which are not in contact with an obstacle; we can eliminate these cases because our local controller includes an overstretch avoidance term that will prevent overstretch in this case in general. Finally, we compare the filtered length of any remaining band predictions to $L_{\max}$; if after filtering there is an estimated band length $\tilde{L}$ that is larger than $L_{\max}$, then we predict that the local controller will be stuck. An example of this type of detection is shown in Figure 5, where the local controller will wrap the cloth around the pole, eventually becoming deadlocked in the process.

4.2.3. Progress detection. Lastly, we track the progress of the robot and task error to estimate whether the controller C is making progress towards the task goal. This is designed to detect cases when the robot is trapped against an obstacle. Naïvely we could look for instances when $\dot{q}_r^{act} = 0$, however owing to sensor noise, actuation error, and using discrete math in a computer, we need to use a threshold instead. At the same time we want to avoid false positives, where the robot is moving slowly but task error is decreasing. To address these concerns, we record the configuration of the robot (stored in ConfigHistory) and the task error (stored in ErrorHistory) every time we check for deadlock, and introduce three parameters to control what it means to be making progress: history window N_h, error improvement threshold β_e, and configuration distance threshold β_m. If over the last N_h timesteps, the improvement in error is less than β_e, and the robot has moved less than β_m, then we predict that the controller will not be able to reach the goal from the current state and trigger global planning.

4.3. Setting the global planning goal

In order to enable efficient planning, we need to approximate the configuration of the deformable object in a way that captures the gross motion of the deformable object without being prohibitively expensive to use. We use the same approach from Section 4.2.2, but the interpretation in this use is slightly different; the virtual elastic band is a proxy for the leading edge of the deformable object. To define the leading edge, we again use the geodesic between the grippers. In this way, we can plan to move the deformable object to a different part of the workspace without needing to simulate the entire deformable object, instead the deformable object conforms to the environment naturally.

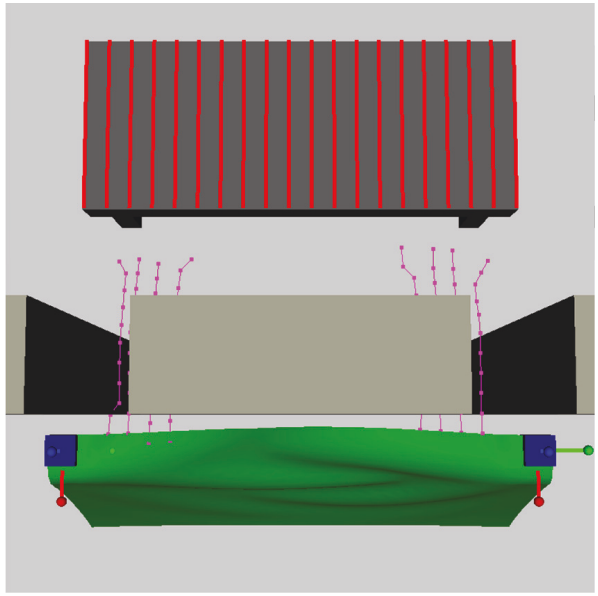

Fig. 4. Example of estimating the gross motion of the deformable object for a prediction horizon $N_p = 10$. The magenta lines start from the points of the deformable object that are closest to the target points (according to the navigation function). These lines show the paths those points would follow to reach the target when following the navigation function.

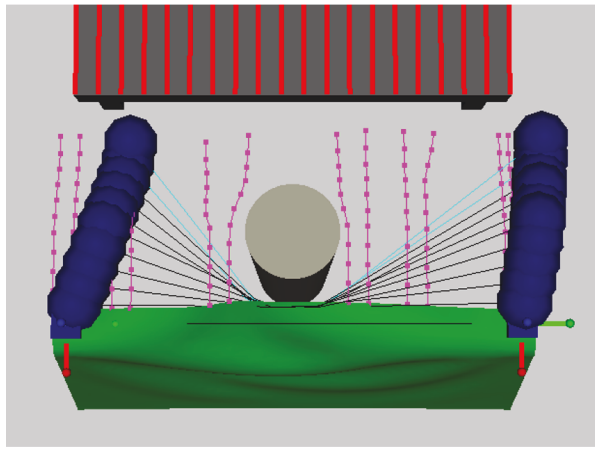

Fig. 5. Estimated gross motion of the deformable object (magenta lines) and end-effectors (blue spheres). The virtual elastic band (black lines) is forward propagated by tracking the end-effector positions, changing to cyan lines when overstretch is predicted. Note: Colour version of the figure is available online.

To make progress towards achieving the task, we want to set the goal for the global planner to be a configuration that we have not explored with the local controller. We do so in two parts; we find the set of all target points $\mathcal{T}_U$ that are contributing to task error, split these points into two clusters, and use the cluster centers to define the goal region of

the end-effectors, $q_{\text{xyz}}^{\text{goal}}$; any end-effector position within a task-specified distance δ_{goal} is considered to have reached the end-effector goal (Algorithm 4, lines 1–3). Second, we set the goal configuration of the virtual elastic band to be any configuration that is not similar to a *blacklist* of virtual elastic bands. This blacklist is the set of all band configurations from which we predicted that the local controller would be deadlocked in the future (Section 4.2).

To define similarity we use Jaillet and Siméon's *visibility deformation* definition to compare two virtual elastic bands (Jaillet and Siméon, 2008). Intuitively two virtual elastic bands are similar if you can sweep a straight line connecting the two bands from the start points to the end points of the two bands without intersecting an obstacle. Unlike the original use, we do not constrain the start and end points of each path to match, but the algorithm is identical. We use this as a heuristic to find states that are dissimilar from states where we have already predicted that the local controller would be deadlocked. Let $\text{VisCheck}(B, \text{Blacklist}) \rightarrow \{0, 1\}$ denote this visibility deformation check, returning 1 if B is similar to a band in the blacklist and 0 otherwise. Then

$$\mathbb{B}^{\text{goal}} = \{B \mid \text{VisCheck}(B, \text{Blacklist}) = 0\} \tag{5}$$

is the set of all virtual elastic bands that are dissimilar to the Blacklist.

Combined, $q_{\text{xyz}}^{\text{goal}}$, δ_{goal}, and $\mathbb{B}^{\text{goal}}$ define what it means for the planner to have found a path to the goal (Algorithm 5); the end-effectors must be in the right region, and the virtual elastic band must be dissimilar to any band in the Blacklist.

The combination of local control, deadlock prediction, and global planning are shown in the MainLoop function (Algorithm 6). Because the virtual elastic band is an approximation, we need to predict deadlock while executing the planned path. We use the same prediction method for path execution as for the local controller. To set the maximum band length $L_{\max}$ used by the global planner and the deadlock prediction algorithms, we calculate the geodesic distance between the grippers through the deformable object in its "laid-flat" state and scale it by the task-specified maximum stretching factor λ_s.

Algorithm 4 PlanPath($q_{r,t}, \mathcal{P}_t, B_t, \mathcal{T}, \delta_{\text{goal}}, L_{\max}, \delta_{BN}$, Blacklist)

1: $\mathcal{T}_U \leftarrow$ UncoveredTargetPoints($\mathcal{P}_t, \mathcal{T}$)
2: $q_{\text{xyz}}^{\text{goal}} \leftarrow$ ClusterCenters($\mathcal{T}_U$)
3: $q_{\text{xyz}}^{\text{goal}} \leftarrow$ ProjectOutOfCollision($q_{\text{xyz}}^{\text{goal}}$)
4: $\mathbb{B}^{\text{goal}} \leftarrow \{B \mid \text{VisCheck}(B, \text{Blacklist}) = 0\}$
5: Path $\leftarrow$ RRT-EB($q_{r,t}, B_t, q_{\text{xyz}}^{\text{goal}}, \delta_{\text{goal}}, \mathbb{B}^{\text{goal}}, L_{\max}, \delta_{BN}$)
6: **if** Path $\neq$ Failure **then**
7: **return** ShortcutSmooth(Path)
8: **else**
9: **return** Failure
10: **end if**

Algorithm 5 GoalCheck($\mathcal{V}, q_{\text{xyz}}^{\text{goal}}, \delta_{\text{goal}}, \mathbb{B}^{\text{goal}}$)

1: **for** $q_f = (q_r, B) \in \mathcal{V}$ **do**
2: $(p_0, p_1) \leftarrow$ ForwardKinematics(q_r)
3: **if** $\|p_0 - q_{\text{xyz},0}^{\text{goal}}\| \leq \delta_{\text{goal}}$ **and**
4: $\|p_1 - q_{\text{xyz},1}^{\text{goal}}\| \leq \delta_{\text{goal}}$ **and** $B \in \mathbb{B}^{\text{goal}}$ **then**
5: **return** 1
6: **end if**
7: **end for**
8: **return** 0

5. Global planning

The purpose of the global planner is not to find a path to a configuration where the task is complete, but rather to move the system into a state from which the local controller can complete the task. Planning directly in configuration space of the full system $\mathbb{C}_r \times \mathbb{R}^{3P}$ is not practical for two important reasons. First, this space is very high-dimensional and the system is highly underactuated. More importantly, to accurately know the state of the deformable object after a series of robot motions one would need a high-fidelity simulation that has been tuned to represent a particular task. We seek to plan paths very quickly without knowing the physical properties of a deformable object a priori. The key idea that allows us to plan paths quickly is to consider only the constraint on robot motion that is imposed by the deformable object, i.e., the robot motion shall not tear or cause excessive stretching of the deformable object. We represent this constraint using a virtual elastic band and enforce the constraint that the band's length cannot exceed $L_{\max}$.

5.1. Planning setup

Denote the planning configuration space as $\mathbb{C}_f = \mathbb{C}_r \times \mathbb{B}$. In order to split $\mathbb{C}_f$ into valid and invalid sets, we first define what it means for a band $B \in \mathbb{B}$ to be valid. A band $B \in \mathbb{B}$ is considered valid if the band is not overstretched and the path defined by B does not *penetrate* an obstacle:

$$\begin{aligned}\mathbb{B}^{valid} = \{B \mid\ & \text{Length}(B) \leq L_{\max} \text{ and} \\ & \text{Path}(B) \cap \text{Interior}(\mathcal{O}) = \emptyset\}.\end{aligned} \tag{6}$$

Then the invalid set is $\mathbb{B}^{inv} = \mathbb{B} \setminus \mathbb{B}^{valid}$. Similarly define $\mathbb{C}_f^{valid} = \mathbb{C}_r^{valid} \times \mathbb{B}^{valid}$ and $\mathbb{C}_f^{inv} = \mathbb{C}_f \setminus \mathbb{C}_f^{valid}$.

Here $\mathbb{C}_r$ and $\mathbb{C}_f$ are imbued with distance metrics $d_r(\cdot,\cdot) : \mathbb{C}_r \times \mathbb{C}_r \rightarrow \mathbb{R}^{\geq 0}$ and $d_f(\cdot,\cdot) : (\mathbb{C}_r \times \mathbb{B}) \times (\mathbb{C}_r \times \mathbb{B}) \rightarrow \mathbb{R}^{\geq 0}$, respectively. We define distances in robot configuration space and band space to be additive. That is,

$$d_f(\cdot,\cdot)^2 = d_r(\cdot,\cdot)^2 + \lambda_b d_b(\cdot,\cdot)^2 \tag{7}$$

for some scaling factor $\lambda_b > 0$. To measure distances in $\mathbb{B}$, we first upsample each band using linear interpolation to use the maximum number of points $N_b^{\max}$ for the given task, then measure the Euclidean distance between the

upsampled points when considered as a single vector (Algorithm 7).

For a given planning problem, we are given a query $(q_f^{\text{init}}, \mathbb{Q}_f^{\text{goal}})$ that describes the initial configuration of the robot and band, as well as a goal region for the system to reach. Note that $\mathbb{Q}_f^{\text{goal}}$ is defined implicitly via the GoalCheck() function and the parameters $(q_{\text{xyz}}^{\text{goal}}, \delta_{\text{goal}},$ Blacklist) rather than any explicit enumeration.

We now establish a relationship between a path in robot configuration space π_r and one in the full configuration space π_f by making the following assumption.

Assumption 1 (Deterministic propagation)**.** *Given an initial configuration in full space $q_f^{init} \in \mathbb{C}_f^{valid}$ and the corresponding robot configuration $q_r^{init} \in \mathbb{C}_r^{valid}$, a path $\pi_r : [0,1] \rightarrow \mathbb{C}_r^{valid}$ in robot configuration space with $\pi_r(0) = q_r^{init}$ uniquely defines a single path in full space π_f, where $\pi_f(0) = q_f^{init}$. Specifically, define*

$$\pi_f(t) = \begin{bmatrix} \pi_r(t) \\ \lim_{h \to 0^-} \text{ForwardPropogateBand}(B(t-h), \pi_r(t)) \end{bmatrix} \quad (8)$$

Equation (8) implicitly defines an underactuated system where the only way we can change the state of the band is by moving the robot; for a path in the full configuration space π_f to be achievable there must be a robot configuration space path π_r, which when propagated using Equation (8), produces π_f. Let FullSpace$(\pi_r, q_f^{\text{init}})$ be the function that maps a given robot configuration space path π_r and full space initial configuration q_f^{init} to the full space path defined by Equation (8).

5.2. Planning problem statement

For a given planning instance, the task is to find a path starting from q_f^{init} through $\mathbb{C}_f^{valid}$ to any point in $\mathbb{Q}_f^{\text{goal}}$, while obeying the constraints implied by Equation (8).

For a sequence of robot configurations $q_f^{\text{init}}, q_{1,r}, \ldots, q_{M,r} \in \mathbb{C}_r$, let $\pi_r = \text{Path}(q_r^{\text{init}}, q_{1,r}, \ldots, q_{M,r})$ be the path defined by linearly interpolating between each point in order. Then, formally, the problem our planner addresses is the following:

$$\begin{aligned} \text{find} \quad & \{q_{1,r}, \ldots, q_{M,r}\} \\ \text{s.t.} \quad & \pi_r = \text{Path}(q_r^{\text{init}}, q_{1,r}, \ldots, q_{M,r}) \\ & \pi_f(s) \in \mathbb{C}_f^{valid}, \ \forall s \in [0,1] \\ & \pi_f(1) \in \mathbb{Q}_f^{\text{goal}} \end{aligned} \quad (9)$$

where $\pi_f = \text{FullSpace}(\pi_r, q_f^{\text{init}})$.

5.3. RRT-EB

Our planner, RRT for Elastic Bands (RRT-EB; see Algorithm 8) is based on RRT with changes to account for a virtual elastic band in addition to the robot configuration. Lines 5–12 perform random exploration with lines 13–23 biasing the tree expansion towards the goal region. The key variations are the BestNearest function (Algorithm 9) and the goal bias method.

BestNearest is based on the selection method used by Li et al. (2016), selecting the node of smallest cost within a radius δ_{BN} if one exists, falling back to standard nearest-neighbor behavior if no node in the tree is within δ_{BN} of the random sample. We use path length in robot configuration space $\mathbb{C}_r$ as a cost function in our implementation. This helps reduce path length and ensures that we can specify lower bounds in Section 6.3. In order to avoid calculating distances in the full configuration space when it is not necessary, our method for finding the nearest neighbor is split into two parts, first searching in robot space, then searching in the full configuration space (see Figure 6). Section

Algorithm 6 MainLoop$(\mathcal{T}, \Omega, \rho, \mathcal{P}_{\text{flat}}, \lambda_s, \lambda_w, N_p, \delta_{\text{goal}}, \delta_{BN})$

```
1:  D ← GeodesicDistanceBetweenEndEffectors(P_flat)
2:  L_max ← λ_s D
3:  Blacklist ← ∅
4:  Path ← ∅
5:  t ← 0
6:  q_{r,0} ← SenseRobotConfig( )
7:  P_0 ← SensePoints( )
8:  while ¬Ω(P_t) do
9:      B_t ← InitializeBand(P_t)
10:     if PredictDeadlock(ρ, q_{r,t}, P_t, B_t, T, L_max, N_p, Path)
    then
11:         Blacklist ← Blacklist ∪ {B_t}
12:         Path ← PlanPath(q_{r,t}, P_t, B_t, T, δ_goal,
            L_max, δ_BN, Blacklist)
13:         if Path = Failure then
14:             return Failure
15:         end if
16:     end if
17:     if Path ≠ ∅ then
18:         q̇_r^cmd ← FollowPath(Path)
19:         if PathFinished(Path) then
20:             Path ← ∅
21:         end if
22:     else
23:         q̇_r^cmd ← LocalController(q_{r,t}, P_t, T, D, λ_s, λ_w)
24:     end if
25:     CommandConfiguration(q_{r,t} + q̇_r^cmd)
26:     q_{r,t+1} ← SenseRobotConfig( )
27:     P_{t+1} ← SensePoints( )
28:     t ← t + 1
29: end while
30: return Success
```

Algorithm 7 BandDistance: $d_b(B_1, B_2)$

```
1: B̃_1 ← UpsamplePoints(B_1, N_b^max)
2: B̃_2 ← UpsamplePoints(B_2, N_b^max)
3: return ‖B̃_1 − B̃_2‖
```

6.2 shows that this method is equivalent to searching in the full configuration space directly. Here δ_{BN} is an additional parameter compared with standard RRT; it controls how much focus is placed on path cost versus exploration. The smaller δ_{BN}, the less impact it has as compared with standard RRT. The larger δ_{BN} is, the harder it is to find narrow passages. We discuss further constraints on δ_{BN} in Section 6.3.1.

To sample $q_f^{\text{rand}} = (q_r^{\text{rand}}, B^{\text{rand}})$, we sample the robot and band configurations independently, then combine the samples. For typical robot arms q_r^{rand} is generated by sampling each joint independently and uniformly from the joint limits. To sample from $\mathbb{B}$, we draw a sequence of $N_b^{\max}$ points from the bounded workspace. For our example tasks, workspace is a rectangular prism, and we sample each axis independently and uniformly.

Owing to the fact that our system is highly underactuated, and the goal region is defined implicitly by a function call rather than an explicit set of configurations, we cannot sample from the goal set directly as is typically done for a goal bias. Instead, we precompute a finite set of robot configurations $\mathcal{Q}_r^{\text{goal}}$ such that the end-effectors of the robot are at $q_{\text{xyz}}^{\text{goal}}$. Then, as a goal bias mechanism, γ_{gb} percent of the time, we attempt to connect to a potential goal configuration starting from the last configuration created by a call to the Connect function (or the last node selected by BestNearest if $\mathcal{V}^{\text{new}} = \emptyset$). A connection is then attempted between q_f^{last} and the nearest configuration in $\mathcal{Q}_r^{\text{goal}}$. This allows us to bias exploration toward the robot component of the goal region, which we are able to define explicitly.

Algorithm 8 RRT-EB($q_{r,t}, B_t, q_{\text{xyz}}^{\text{goal}}, \delta_{\text{goal}}, \mathbb{B}^{\text{goal}}, L_{\max}, \delta_{BN}, \gamma_{gb}$)

$\mathcal{V} \leftarrow \{(q_{r,t}, B_t)\}$
$\mathcal{E} \leftarrow \emptyset$
$\mathcal{Q}_r^{\text{goal}} \leftarrow$ GetGoalConfigs($q_{\text{xyz}}^{\text{goal}}$)
while ¬MaxTimeEllapsed() **do**
  $q_f^{\text{rand}} \leftarrow$ SampleUniformConfig()
  $q_f^{\text{near}} \leftarrow$ BestNearest($\mathcal{V}, \mathcal{E}, \delta_{BN}, q_f^{\text{rand}}$)
  $\mathcal{V}^{\text{new}}, \mathcal{E}^{\text{new}} \leftarrow$ Connect($q_f^{\text{near}}, q_r^{\text{rand}}, L_{\max}$)
  $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}^{\text{new}}$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}^{\text{new}}$
  if GoalCheck($\mathcal{V}^{\text{new}}, q_{\text{xyz}}^{\text{goal}}, \delta_{\text{goal}}, \mathbb{B}^{\text{goal}}$) $= 1$ **then**
    return ExtractPath($\mathcal{V}, \mathcal{E}$)
  end if
  $\gamma \sim$ Uniform[0, 1]
  if $\gamma \leq \gamma_{gb}$ **then**
    $q_f^{\text{last}} \leftarrow$ LastConfig($q_f^{\text{near}}, \mathcal{V}^{\text{new}}$)
    $q_r^{\text{bias}} \leftarrow \text{argmin}_{q_r \in \mathcal{Q}_r^{\text{goal}}} d_r(q_r^{\text{last}}, q_r)$
    $\mathcal{V}^{\text{new}}, \mathcal{E}^{\text{new}} \leftarrow$ Connect($q_f^{\text{last}}, q_r^{\text{bias}}, L_{\max}$)
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}^{\text{new}}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}^{\text{new}}$
    if GoalCheck($\mathcal{V}^{\text{new}}, q_{\text{xyz}}^{\text{goal}}, \delta_{\text{goal}}, \mathbb{B}^{\text{goal}}$) $= 1$ **then**
      return ExtractPath($\mathcal{V}, \mathcal{E}$)
    end if
  end if
end while
Return Failure

Algorithm 9 BestNearest ($\mathcal{V}, \mathcal{E}, \delta_{BN}, q_f^{\text{rand}}$)

$\mathcal{Q}_f^{\text{near}} \leftarrow \{q_f | q_f \in \mathcal{V}, d_f(q, q_f^{\text{rand}}) \leq \delta_{BN}\}$
if $\mathcal{Q}_f^{\text{near}} \neq \emptyset$ **then**
  return $\text{argmin}_{q \in \mathcal{V}}$ Cost($q, \mathcal{V}, \mathcal{E}$)
else
  $D^2_{\text{near},r} \leftarrow \min_{q \in \mathcal{V}} d_r(q^{\text{rand}}, q_r)^2$
  $D^2_{\max,f} \leftarrow D^2_{\text{near},r} + \lambda_b D^2_{\max,b}$
  $\mathcal{Q}_f^{\text{near}} \leftarrow \{q | q \in \mathcal{V}, d_r(q_r, q_r^{\text{rand}})^2 \leq D^2_{\max,f}\}$
  return $\text{argmin}_{q_f \in \mathcal{Q}_f^{\text{near}}} d_f(q_f, q_f^{\text{rand}})$
end if

6. Probabilistic completeness of global planning

Proving probabilistic completeness in $\mathbb{C}_f$ is challenging due to the multi-modal nature of the problem. Specifically, as the virtual elastic band moves in and out of contact the dimensionality of the manifold that the system is operating in can change. In addition, the virtual elastic band forward propagation function (Algorithm 3) can allow the band to "snap tight" as the grippers move past the edge of an obstacle, changing the number of points in the band representation as it does so. By leveraging the assumptions from Section 6.1, we are able to bypass most of these challenges by focusing on the portion of $\mathbb{C}_f$ that can be analyzed, i.e., $\mathbb{C}_r$.

This section proves the probabilistic completeness of the planning approach in two major steps. First, it will show that the approach for selecting the nearest node in the tree for expansion is equivalent to performing a nearest-neighbor query in the full space. Second, it proves that our algorithm will eventually return a path that is δ_r-similar to an optimal δ-robust solution to the planning problem with probability 1 (if it exists), or it will terminate early having found an alternate path to the goal region. Recall that we do not require an optimal path, only a feasible one.

6.1. Assumptions and definitions

Our problem allows for the virtual elastic band to be in contact with the surface of an obstacle, both during execution and as part of the goal set; this means that common assumptions regarding the expansiveness (Hsu et al., 1999) of the planning problem may not hold. Instead of relying on expansiveness, we define a series of alternate definitions and assumptions that are sufficient to ensure the completeness of our method.

First, in line with prior work, we assume properties of the problem instance in regards to robustness. In particular, we assume the existence of a solution to a given query $\pi_f^{ref} : [0, 1] \rightarrow \mathbb{C}_f^{valid}$, which has several robustness properties. This solution is called a reference path.

To begin describing the properties of the reference path, we assume π_f^{ref} has robustness properties in the robot configuration space. That is, the corresponding path in robot configuration space π_r^{ref} has strong δ_r-clearance under distance metric $d_r(\cdot,\cdot)$ for some $\delta_r > 0$.

Definition 2 (Strong δ-clearance)**.** *A path $\pi : [0,1] \rightarrow \mathbb{C}^{valid}$ has strong δ-clearance under distance metric $d(\cdot,\cdot)$ if $\forall s \in [0,1]$, $d(\pi(s), \mathbb{C}^{inv}) \geq \delta$, for $\delta > 0$.*

Given our assumption about the δ_r-clearance of the reference path in robot space, there exists a set $\mathbb{T}_r$ of δ_r-similar paths to the reference path that are also collision-free.

Definition 3 (δ-similar path)**.** *Two paths π_a and π_b are δ-similar if the Fréchet distance between the paths is less than or equal to δ.*

Informally, the Fréchet distance is described as follows (Alt and Godau, 1995). Suppose a man is walking a dog. The man is walking on one curve while the dog is walking on another curve. Both walk at any speed but are not allowed to move backwards. The Fréchet distance of the two curves is then the minimum length of leash necessary to connect the man and the dog.

Given the relationship between robot-space and full-space paths, we can define a full-space equivalent to $\mathbb{T}_r$ as

$$\mathbb{T}_f = \{\pi_f \mid \ \pi_r \in \mathbb{T}_r \text{ and } \pi_f = \text{FullSpace}(\pi_r, q_f^{\text{init}})\} \quad (10)$$

Given these assumptions and definitions, we are ready to define an *acceptable δ-robust path*.

Definition 4 (Acceptable δ-robust path)**.** *A path π_f^{ref} is an acceptable δ-robust path if the following hold:*

1. *the robot-space reference path π_r^{ref} has strong δ_r-clearance for some $\delta_r > 0$;*
2. *the final state for every path $\pi_f \in \mathbb{T}_f$ is in $\mathbb{Q}_f^{goal}$.*

We assume there exists a reference path that satisfies this property and answers our given planning query.

Assumption 5 (Solvable problem)**.** *There exists some $\delta_r > 0$ such that the planning problem admits an acceptable δ-robust path.*

If a planning problem does not yield a reference path with this property, then it would be practically impossible for a sampling-based approach to solve it, as this would require sampling on a lower-dimensional manifold in robot space. Given that our planner is able to find paths, we believe this assumption is true except in special cases where the band must achieve a singular configuration to reach the goal.

While the focus of this article is not on asymptotic optimality, we make use of a cost function $\text{Cost}(\pi)$ of a path in Section 6.3.1. Our cost function is path length in robot configuration space. With a cost function of this form, we then assume from here onward that the reference path in question is optimal under the following definition.

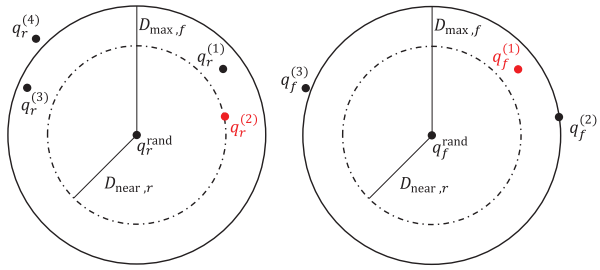

Fig. 6. Left: $q_r^{(2)}$ is the nearest node to the q^{rand} in robot space, but it my be as far as $D_{\max,f}$ away in the full configuration space. By considering all nodes within $D_{\max,f}$ in robot space, we ensure that any node (such as $q_f^{(1)}$) that is closer to q_f^{rand} than $q_f^{(2)}$ is selected as part of Q_f^{near}, while nodes such as $q_f^{(4)}$ are excluded in order to avoid the expense of calculating the full configuration space distance. Right: we then measure the distance in the full configuration space to all nodes that could possibly be the nearest to q_f^{rand}, returning $q_f^{(1)}$ as the nearest node in the tree.

Definition 6 (Optimal δ-robust path)**.** *Let $\mathbb{T}_{f,\delta}$ be the set of all acceptable δ-robust paths. A path π_f^{ref} is optimal δ-robust if*

$$Cost(\pi_f^{ref}) = \inf_{\pi_f \in \mathbb{T}_{f,\delta}} Cost(\pi_f) \quad (11)$$

Finally, we also assume that workspace is bounded. This will be true for any practical task and is rarely mentioned in the literature, but we use this assumption in our analysis in Section 6.2.

6.2. Proof of nearest-neighbor equivalence

Lemma 7. *If the maximum distance between any two points in workspace is bounded by $D_{max,w} > 0$, then under distance metric $d_b(\cdot,\cdot)$, the maximum distance between any two points in virtual elastic band space is bounded. That is, there exists $D_{max,b} > 0$ such that $d_b(B_1, B_2) \leq D_{max,b}$ for all $B_1, B_2 \in \mathbb{B}$.*

Proof. From the definition of $\mathbb{B}$ in Section 4.2.2, the number of points used to represent a virtual elastic band is bounded by $N_b^{\max}$. Let $B_1, B_2 \in \mathbb{B}$ be two virtual elastic band configurations, and let $\tilde{B}_1 = (\tilde{b}_{1,1}, \ldots, b_{1,N_b^{\max}})$ and $\tilde{B}_2 = (\tilde{b}_{2,1}, \ldots, b_{2,N_b^{\max}})$ be their upsampled versions as described in Algorithm 7. Then

$$\begin{aligned} d_b(B_1,B_2)^2 &= \sum_{i=1}^{N_b^{\max}} \|\tilde{b}_{1,i} - \tilde{b}_{2,i}\|^2 \\ &\leq \sum_{i=1}^{N_b^{\max}} D_{\max,w}^2 = N_b^{\max} D_{\max,w}^2 = D_{\max,b}^2 \end{aligned} \quad (12)$$

□

Lemma 8. *If workspace is bounded, then lines 5–8 in Algorithm 9 are equivalent to a nearest-neighbor search in the full configuration space directly.*

Proof. The upper bound of $D_{\max,b}$ and our additive distance metric (Equation (7)) ensures that the distance between any two configurations in full space $\mathbb{C}_f$ can be bounded using only the distance in robot configuration space:

$$d_f(\cdot,\cdot)^2 \leq d_r(\cdot,\cdot)^2 + \lambda_b \cdot D^2_{\max,b} \tag{13}$$

Next, consider that in line 5 of the algorithm, the nearest neighbor to q_r^{rand} under distance metric d_r is found. Let this nearest neighbor be denoted $\tilde{q}_r^{\text{near}}$, keeping in mind that it belongs to a vertex in the tree $\tilde{q}_f^{\text{near}} = (\tilde{q}_r^{\text{near}}, \tilde{q}_b^{\text{near}})$. Let the (squared) distance between these points under d_r be $D^2_{\text{near},r}$. From (13), we can bound the distance between the random sample and $\tilde{q}_f^{\text{near}}$ under d_f as $D^2_{\max,f} \leq D^2_{\text{near},r} + \lambda_b D^2_{\max,b} = D^2_{\max,f}$.

In line 7 of the algorithm, a radius nearest-neighbor query of radius $D_{\max,f}$ is performed, returning a set Q_f^{near}. By construction if there is a node $q_f \in \mathcal{V}$ that is closer to q_f^{rand} than $\tilde{q}_f^{\text{near}}$, then $q_f \in Q_f^{\text{near}}$ (Figure 6). Then, the method selects as the true nearest neighbor in full space $q_f^{\text{select}} = \text{argmin}_{q_f \in Q_f^{\text{near}}} d_f(q_f, q_f^{\text{rand}})$. □

6.3. Construction of a δ_r-similar path

The objective here is to show with probability approaching 1 that the planner generates a δ_r-similar path to some robustly feasible solution given enough time. If an alternate path is found and the algorithm terminates before generating a δ_r-similar path, then this is still sufficient for probabilistic completeness. This analysis is similar to Li et al. (2016), and is based on a covering ball sequence of the optimal δ-robust path π_r^{ref}. The key differences are in Section 6.3.2 where we show that using a straight line to connect points in $\mathbb{C}_r$ is sufficient to get a lower bound on the probability of covering the next ball, where Li et al. used a random control action.

Definition 9 (Covering ball sequence)**.** *Given a path $\pi_r : [0,1] \to \mathbb{C}_r^{valid}$, robust clearance $\delta_r > 0$, a BestNearest distance $\delta_{BN} > 0$, and a distance value $0 < \delta_c < \delta_{BN} < \delta_r$; the covering ball sequence is defined as a set of $K+1$ hyperballs $\{\mathcal{B}_{\delta_r}(q_{0,r}), \ldots, \mathcal{B}_{\delta_r}(q_{K,r})\}$ of radius δ_r, where $q_{k,r}$ are defined such that:*

- $q_{0,r} = \pi_r(0)$;
- $q_{K,r} = \pi_r(1)$;
- $\text{PathLength}(q_{k-1,r}, q_{k,r}) = \delta_c$ for $k = 1, \ldots, K$.

We use $q^*_{k,r}$ to denote the center of the kth covering hyperball for the reference path π_r^{ref}. Figure 7 shows an example of a covering ball sequence.

The objective is to show that the vertex set of the planning tree after n iterations $\mathcal{V}_n$ probabilistically contains a node within the goal set, i.e.,

$$\liminf_{n\to\infty} \mathbb{P}(\mathcal{V}_n \cap \mathbb{Q}_f^{\text{goal}} \neq \emptyset) = 1 \tag{14}$$

To do this, the analysis examines K subsegments of the reference path π_r^{ref}, based on the covering ball sequence for the reference path. If we can generate a robot path that is δ_r similar to π_r^{ref}, then given Assumption 5 and the properties of the reference path, the corresponding full space path will be a solution to the given planning problem.

Let $A_k^{(n)}$ be the event that on the nth iteration of the algorithm, it generates a δ_r-similar path to the kth subsegment of π_r^{ref}. This of course requires two events to occur: the node generated from the prior propagation covering segment $k-1$ must be selected for expansion, and the expansion must then produce a δ_r-similar path to the current segment. Then, let $E_k^{(n)}$ be the event that for segment k, $A_k^{(n)}$ has occurred for some $i \in [1,n]$, i.e., $E_k^{(n)}$ indicates whether the algorithm has constructed the δ_r-similar edge for subsegment k. From these definitions, the goal then is to show that

$$\lim_{n\to\infty} \mathbb{P}(\text{Success}) = \lim_{n\to\infty} \mathbb{P}\left(E_K^{(n)}\right) = 1 \tag{15}$$

We start by considering the probability of failing to generate an arbitrary segment $1 \leq k \leq K$. Then

$$\begin{aligned}
\mathbb{P}\left(\neg E_k^{(n)}\right) &= \mathbb{P}\left(\neg A_k^{(1)} \cap \cdots \cap \neg A_k^{(n)}\right) \\
&= \mathbb{P}\left(\neg A_k^{(1)}\right) \mathbb{P}\left(\neg A_k^{(2)} \mid \neg A_k^{(1)}\right) \cdot \cdots \cdot \\
&\qquad \mathbb{P}\left(\neg A_k^{(n)} \mid \neg A_k^{(1)} \cap \cdots \cap \neg A_k^{(n-1)}\right) \\
&= \prod_{i=1}^{n} \mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \quad .
\end{aligned} \tag{16}$$

Note the definition of $\neg E_k^{(i-1)}$ is what allows us to collapse the product into a concise form.

The probability that $\neg A_k^{(i)}$ happens given $\neg E_k^{(i-1)}$ is equivalent to the probability that we have not yet generated a δ_r-similar path for segment $k-1$ (i.e., $\mathbb{P}(\neg E_{k-1}^{(i-1)})$) plus the probability that the previous segment has been generated, but we fail to generate the current segment:

$$\begin{aligned}
&\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \\
&\quad = \mathbb{P}\left(\neg E_{k-1}^{(i-1)}\right) + \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \\
&\quad \cdot \mathbb{P}\left(\neg A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right),
\end{aligned} \tag{17}$$

which we can rewrite in terms of $A_k^{(i)}$ instead of $\neg A_k^{(i)}$:

$$\begin{aligned}
\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) &= \mathbb{P}\left(\neg E_{k-1}^{(i-1)}\right) + \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \\
&\cdot \left(1 - \mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right)\right)
\end{aligned} \tag{18}$$

Then multiplying out the last term we obtain

$$\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) = \mathbb{P}\left(\neg E_{k-1}^{(i-1)}\right) + \mathbb{P}\left(E_{k-1}^{(i-1)}\right) - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right) \tag{19}$$

Finally, summing the first two terms, we arrive at

$$\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) = 1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right) \tag{20}$$

Two events need to happen in order to generate a path to the next hyper-ball: an appropriate node must be selected for expansion, and Connect(. . .) must generate a δ_r-similar path segment, assuming that the appropriate node has already been selected. Denote the probability of these events at iteration i as $\gamma_k^{(i)}$ and $\rho_k^{(i)}$, respectively. Then

$$\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) = 1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(i)}\rho_k^{(i)} \tag{21}$$

As we are examining this probability in the limit, we instead draw a bound on this probability to put it in a form we can easily examine the limit for. To do so, we must carefully consider the values of $\gamma_k^{(i)}$ and $\rho_k^{(i)}$. In Section 6.3.1, we show that $\gamma_k^{(i)}$ is a generally decreasing function, but converges to a finite value $\gamma_k^{(\infty)} > 0$ in the limit. Therefore, we let $\gamma_k^{(\infty)}$ be a lower bound of $\gamma_k^{(i)}$. Then, in Section 6.3.2, $\rho_k^{(i)}$ will similarly be shown to be positive and lower-bounded; in particular $\gamma_k^{(i)}\rho_k^{(i)} \leq \gamma_k^{(\infty)}$. Taking $\gamma_k^{(\infty)}$ as constant, we can bound (21) as

$$\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \leq 1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)} \tag{22}$$

Combining Equations (22) and (16) we have

$$\mathbb{P}\left(\neg E_k^{(n)}\right) \leq \prod_{i=1}^{n}\left(1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)}\right) \tag{23}$$

Denote $y_k^{(n)} = \prod_{i=1}^{n}\left(1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)}\right)$. Then

$$\mathbb{P}\left(\neg E_k^{(n)}\right) \leq y_k^{(n)} \tag{24}$$

We show using induction over k that Equation (24) tends to 0 as $n \to \infty$, and thus $\lim_{n\to\infty}\mathbb{P}(\text{ Success}) = 1$.

Base case ($k = 1$)**:**
Note that $\mathbb{P}(E_0^{(i)}) = 1$ because the start node always exists. Then

$$\begin{aligned}\lim_{n\to\infty}\mathbb{P}\left(\neg E_1^{(n)}\right) &\leq \lim_{n\to\infty}\prod_{i=1}^{n}\left(1 - \mathbb{P}\left(E_0^{(i-1)}\right)\gamma_k^{(\infty)}\right)\\ &= \lim_{n\to\infty}\prod_{i=1}^{n}\left(1 - \gamma_k^{(\infty)}\right)\\ &= \lim_{n\to\infty}\left(1 - \gamma_k^{(\infty)}\right)^n = 0\end{aligned} \tag{25}$$

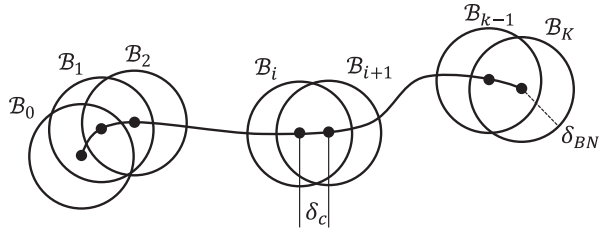

Fig. 7. Example covering ball sequence for an example reference path with a distance along the path of δ_c between each ball. Given that the path is δ_r-robust, each ball is a subset of $\mathbb{C}_r^{valid}$.

Induction hypothesis:

$$\lim_{n\to\infty}\mathbb{P}\left(\neg E_m^{(n)}\right) = 0 \text{ for } m = 1, 2, \ldots, k-1 \tag{26}$$

Note that this implies $\lim_{n\to\infty}\mathbb{P}(E_m^{(n)}) = 1$ for $m = 1, 2, \ldots, k-1$.

Induction step ($2 \leq k \leq K$)**:**
Consider the log of the bound on $\mathbb{P}\left(\neg E_k^{(n)}\right)$,

$$\log y_k^{(n)} = \sum_{i=1}^{n}\log\left(1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)}\right) \tag{27}$$

We use the notation $x = \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)}$. Given that $0 \leq x < 1$, and writing the Taylor series expansion of $\log(1-x)$ centered at $x = 0$ we have

$$\log(1-x) = -\sum_{m=1}^{\infty}\frac{x^m}{m} \tag{28}$$

Substituting (28) back into (27) we obtain

$$\log y_k^{(n)} = -\sum_{i=1}^{n}\sum_{m=1}^{\infty}\frac{\left(\mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)}\right)^m}{m} \tag{29}$$

Dropping all but the first term in the infinite sum, we get the bound

$$\log y_k^{(n)} \leq -\sum_{i=1}^{n}\mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)} \tag{30}$$

Rearranging terms yields

$$\log y_k^{(n)} \leq -\gamma_k^{(\infty)}\sum_{i=1}^{n}\mathbb{P}\left(E_{k-1}^{(i-1)}\right) \tag{31}$$

We now use the induction hypothesis. We know that $\mathbb{P}(E_{k-1}^{(n)}) \to 1$ as $n \to \infty$, thus $\sum_{i=1}^{n}\mathbb{P}(E_{k-1}^{(i-1)}) \to \infty$. Then

$$\lim_{n\to\infty}\log y_k^{(n)} \leq -\gamma_k^{(\infty)}\sum_{i=1}^{n}\lim_{n\to\infty}\mathbb{P}\left(E_{k-1}^{(i-1)}\right) = -\infty \tag{32}$$

Taking the log of (24) and combining with (32) we obtain

$$\lim_{n\to\infty}\log\mathbb{P}\left(\neg E_k^{(n)}\right) \leq \lim_{n\to\infty}\log y_k^{(n)} = -\infty \tag{33}$$

and, therefore,

$$\lim_{n\to\infty} \mathbb{P}\left(\neg E_k^{(n)}\right) = 0 \tag{34}$$

which completes the induction step.

Thus, given that $\mathbb{P}(\neg E_k^{(n)}) \to 0$ as $n \to \infty$ for any $1 \leq k \leq K$

$$\lim_{n\to\infty} \mathbb{P}(\text{Success}) = \lim_{n\to\infty} \left(1 - \mathbb{P}\left(\neg E_K^{(n)}\right)\right) = 1 \tag{35}$$

6.3.1. Selection of an appropriate node ($\gamma_k^{(\infty)}$). First, we define the following restriction on the definition of δ_{BN}.

Definition 10 (δ_{BN} restriction)**.** *For a reference path π_r^{ref} with robustness δ_r, δ_{BN} is defined such that $\delta_\theta = \delta_r - \delta_{BN} > 0$.*

The proof that $\gamma_k^{(\infty)} > 0$ follows directly from the related work of Li et al. (2016: proof of Lemma 23). To summarize, owing to best nearest-neighbor selection, there exists a positive-measure region around the minimum cost vertex q_f^{near} that observes the optimal reference path in which its cost dominates all other nearby nodes and, therefore, when q_f^{rand} is drawn in this volume, $q_f^{\text{near}} = (q_r^{\text{near}}, q_b^{\text{near}})$ is guaranteed to be selected (Figure 8). As our approach follows an equivalent sampling and nearest-neighbor method to Li et al. (2016: Algorithm 6) (as shown in Section 6.2),

$$\gamma_k^{(\infty)} = \frac{\mu\left(\mathcal{B}_{\delta_\theta}(q_{k,f}^*) \cap \mathcal{B}_{\delta_{BN}}(q_f^{\text{near}})\right)}{\mu\left(\mathbb{C}_f\right)} > 0 \tag{36}$$

follows directly.

To show that $\gamma_k^{(\infty)} < 1$, we need only consider the case when there are at least two nodes in $\mathcal{V}$.

6.3.2. δ_r-similar propagation ($\rho_k^{(i)}$). Given that our nearest-neighbor method is non-standard, and operating in the full configuration space $\mathbb{C}_f$, we need to carefully consider how this affects the propagation probability $\rho_k^{(i)}$. Given the kinematic model of our robot system, it is straightforward to show that the system in robot space is small-time locally controllable (STLC), i.e., q_r can be instantaneously moved in any direction, barring the presence of obstacles or configuration space limits.

Then, based on the construction of the covering ball sequence and the δ_{BN} restriction, the following lemma holds.

Lemma 11. *If q_f^{rand} is within the minimum domination region as described in Li et al. (2016: Lemma 23) (Figure 8), then $q_r^{rand} \in \mathcal{B}_{\delta_r}(q_{k,r}^*)$ and Connect() will generate a segment that is δ_r-similar to segment k of the reference path.*

Proof. Assume that $q_f^{\text{rand}} \in \mathcal{B}_{\delta_\theta}(q_{k-1,f}^*)$. Then we have

$$\begin{aligned} d_r(q_r^{\text{rand}}, q_{k,r}^*) &\leq d_f(q_f^{\text{rand}}, q_{k,f}^*) \\ &\leq d_f(q_f^{\text{rand}}, q_{k-1,f}^*) + d_f(q_{k-1,f}^*, q_{k,f}^*) \\ &\leq \delta_\theta + \delta_c = \delta_r - \delta_{BN} + \delta_c \end{aligned}$$

Then by construction of the covering ball sequence, we have that $\delta_c - \delta_{BN} < 0$ and, thus, $d_r(q_r^{\text{rand}}, q_{k,r}^*) < \delta_r$. In addition, we have that the straight line between q_r^{near} as selected by q_r^{rand} is entirely contained in $\mathcal{B}_{\delta_r}(q_{k-1,r}^*)$, and thus is also in $\mathbb{C}_r^{valid}$ as the reference path is optimal δ-robust. We then have that the path generated by Connect is δ_r-similar to the kth segment of the reference path. □

Lemma 12. *The probability of covering segment k at iteration i, given that we have not yet covered segment k but we have covered segment $k-1$*

$$\mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right) = \gamma_k^{(i)} \rho_k^{(i)}$$

is lower-bounded by $\gamma_k^{(\infty)}$.

Proof. Consider two possible events. First, that q_f^{rand} is within the minimum domination region (Figure 8) of q_f^{near}. If q_f^{rand} is within the minimum domination region of q_f^{near}, then by Lemma 11, Connect() will generate a δ_r-similar segment with probability 1. Denote this event as B. Second, the event that q_f^{rand} is somewhere else. Denote this event as C. Then we can bound $\mathbb{P}(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)})$ by considering only B:

$$\begin{aligned} \mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right) &= \mathbb{P}(B) + \mathbb{P}(C) \\ &\geq \mathbb{P}(B) \geq \gamma_k^{(\infty)} \end{aligned}$$

□

7. Simulation experiments and results

We now present four example tasks to demonstrate our algorithm, two with cloth and two with rope. These tasks are designed to show that our framework is able to handle non-trivial tasks that cannot be performed using either our controller or planner alone. In Section 8, we demonstrate that our method can also be applied to a physical robot.

For these simulation tasks $\mathbb{C}_r = SE(3) \times SE(3)$, i.e., there are two free-flying grippers. In the first and second tasks, two grippers manipulate the cloth so that it covers a table. In the first task the cloth is obstructed by a pillar while in the second task the grippers must pass through a narrow passage before the table can be covered. The third and fourth scenarios require the robot to navigate a rope through a three-dimensional maze before aligning the rope with a line traced on the floor (see Figure 1). Extension 1 shows the task executions.

All experiments were conducted in the open-source Bullet simulator (Coumans, 2010), with additional wrapper code developed at UC Berkeley (Robot Learning Lab, 2012). The cloth is modeled as a triangle mesh using 1,500 vertices with a total size of 0.3 m × 0.5 m. The rope is modeled as a series of small capsules linked together by springs. In the first rope experiment we use 39 capsules for a 0.78 m long rope, and 47 capsules for a 0.94 m rope in the last

experiment. We emphasize that our method does not have access to the model of the deformable object or the simulation parameters. The simulator is used as a "black box" for testing. We set the maximum stretching factor λ_s to 1.17 for the cloth and 1.15 for the rope. All tests are performed using an i7-8700K 3.7 GHz CPU with 32 GB of RAM. We use the same deadlock prediction and planner parameters for all tasks, as listed in Tables 1 and 2. For the purpose of the planner we treat the grippers as spheres, reducing the planning space from $SE(3) \times SE(3) \times \mathbb{B}$ to $\mathbb{R}^6 \times \mathbb{B}$. To lift the planned path back into $SE(3) \times SE(3) \times \mathbb{B}$ we copy the starting orientation of the grippers to each gripper configuration in the plan.

To smooth the path returned by the planner, at each iteration we randomly select either a single gripper or both grippers and two configurations in the path. To smooth between the configurations we use the same forward-propagation method for the virtual elastic band as used in the planning process. If we have selected only one gripper for smoothing, we do not change the configuration of the second gripper during that smoothing iteration. We also forward-propagate the virtual elastic band to the end of the path to ensure that the band at the end of the smoothed path is dissimilar from the blacklist. We perform 500 smoothing iterations for experiments 1, 2, and 4; and 1,500 for experiment 3 owing to the larger environment.

7.1. Single pillar

In the first example task, the objective is to spread the cloth across a table that is on the far side of a pillar (see Figure 9). We uniformly discretize the surface of the table to create the target points $\mathcal{T}$, with each discretized point creating a navigation function that pulls the closest point on the deformable object towards the target. These target points are set slightly above the surface to allow for collision margins within the simulator. A single point on the cloth can have multiple "pulls" or none. Task error ρ is defined as the sum of the Dijkstra's distances from each target point to the closest point on the cloth. If a target point in $\mathcal{T}$ is within a small-enough threshold of their nearest neighbors in $\mathcal{P}$, then these points are considered "covered" and do not affect task error or any other calculation. Our results show that even though the global planner is only planning using the gripper positions and a virtual elastic band between them, it is able to find the correct neighborhood for the local controller to complete the task. On average we are able to find and smooth a path in 3.0 seconds (Table 3), with the majority of the planning time spent on forward propagation of the virtual elastic band as part of the validity check for a potential movement of the grippers. In all 100 trials the global planner is only invoked once, with the local controller completing the task after the plan finishes.

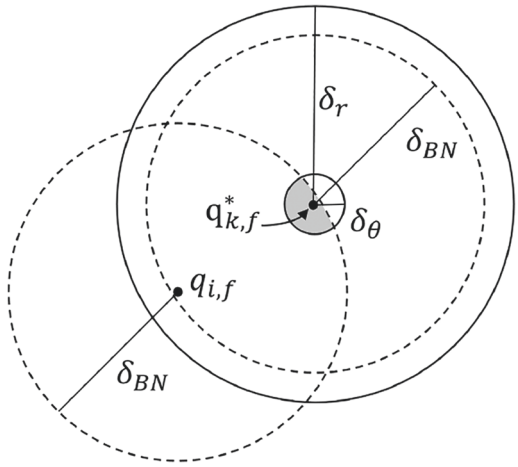

Fig. 8. Minimum domination region for a node $q_{i,f}$, adapted from Li et al. (2016: Lemma 23). Sampling q_f^{rand} in the shaded region guarantees that a node $q_f^{\text{near}} \in \mathcal{B}_{\delta_r}(q^*_{k,f})$ is selected for propagation so that either $q_f^{\text{near}} = q_{i,f}$ or $\text{Cost}(q_f^{\text{near}}) < \text{Cost}(q_{i,f})$.

Table 1. Deadlock prediction parameters.

Prediction horizon	N_p	10
Band annealing factor	α	0.3
History window	N_h	100
Error improvement threshold	β_e	1
Configuration distance threshold	β_m	0.03

Table 2. Distance and planner parameters.

Goal bias	γ_{gb}	0.1
Workspace goal radius	δ_{goal}	0.02
Best nearest radius	δ_{BN}	0.001
Band distance scaling factor	λ_b	10^{-6}
Maximum band points	$N_b^{\max}$	500

7.2. Double slit

The second experiment uses the same setup as the first, with the only change being that the single pillar obstacle is replaced by a wide wall with two narrow slits (Figure 10). This adds a narrow passage problem and also demonstrates the utility of the progress detection filter. In this example the local controller is trying to move the deformable object straight forward, but with the wall in the way it is unable to make progress; the local controller cannot explicitly go around obstacles. This experiment shows comparable planning time, but it takes longer to smooth the resulting path (as expected given that the virtual elastic band forward propagation takes longer near obstacles). The local controller is again able to complete the task after invoking the planner a single time on all 100 trials.

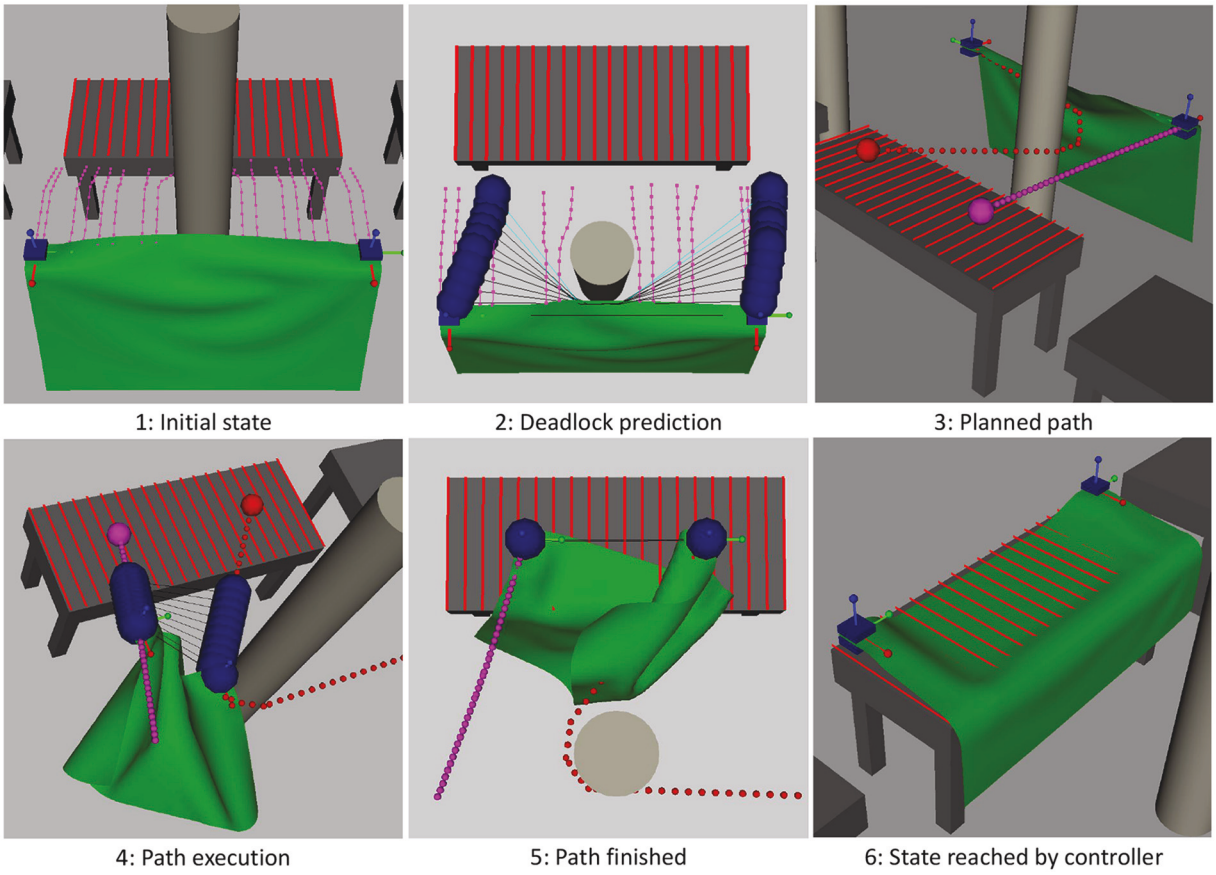

Fig. 9. Sequence of snapshots showing the execution of the first experiment. The cloth is shown in green, the grippers are shown in blue, and the target points are shown as red lines. (1) The approximate integration of the navigation functions from error reduction over N_p timesteps, shown in magenta, pull the cloth to opposite sides of the pillar. (2) A sequence of *virtual elastic bands* between the grippers is shown in black and teal, indicating the predicted gripper configuration over the prediction horizon as the local controller follows the navigation functions. The elastic band changes to teal as the predicted motion of the grippers moves the cloth into an infeasible configuration. (3–5) The resulting plan by the RRT, shown in magenta and red, moves the system into a new neighborhood. (6) Final system state when the task is finished by the local controller.
Note: Colour version of the figure is available online.

Table 3. Planning statistics for the first plan for each example task in simulation, averaged across 100 trials. Standard deviation is shown in brackets.

	RRT planning					**Smoothing**			
	Samples	States	NN time (s)	Validity checking time (s)	Total time (s)	Iterations	Validity checking time (s)	Visibility deformation time (s)	Total time (s)
Single pillar	158 [121]	1,182 [804]	∼0.0 [∼0.0]	0.6 [0.5]	0.6 [0.5]	500	0.8 [1.2]	1.6 [0.2]	2.4 [1.2]
Double slit	478 [353]	2,124 [1,428]	∼0.0 [∼0.0]	0.7 [0.8]	0.7 [0.8]	500	2.5 [2.6]	∼ 0.0 [∼0.0]	2.5 [2.6]
Rope maze	4,796 [1,613]	9,926 [3,760]	0.1 [∼0.0]	4.0 [1.7]	4.2 [1.8]	1,500	6.4 [3.9]	∼ 0.0 [∼0.0]	6.5 [3.9]
Repeated planning	54 [46]	153 [147]	∼0.0 [∼0.0]	0.1 [0.1]	0.1 [0.1]	500	1.4 [0.9]	∼ 0.0 [∼0.0]	1.4 [0.9]

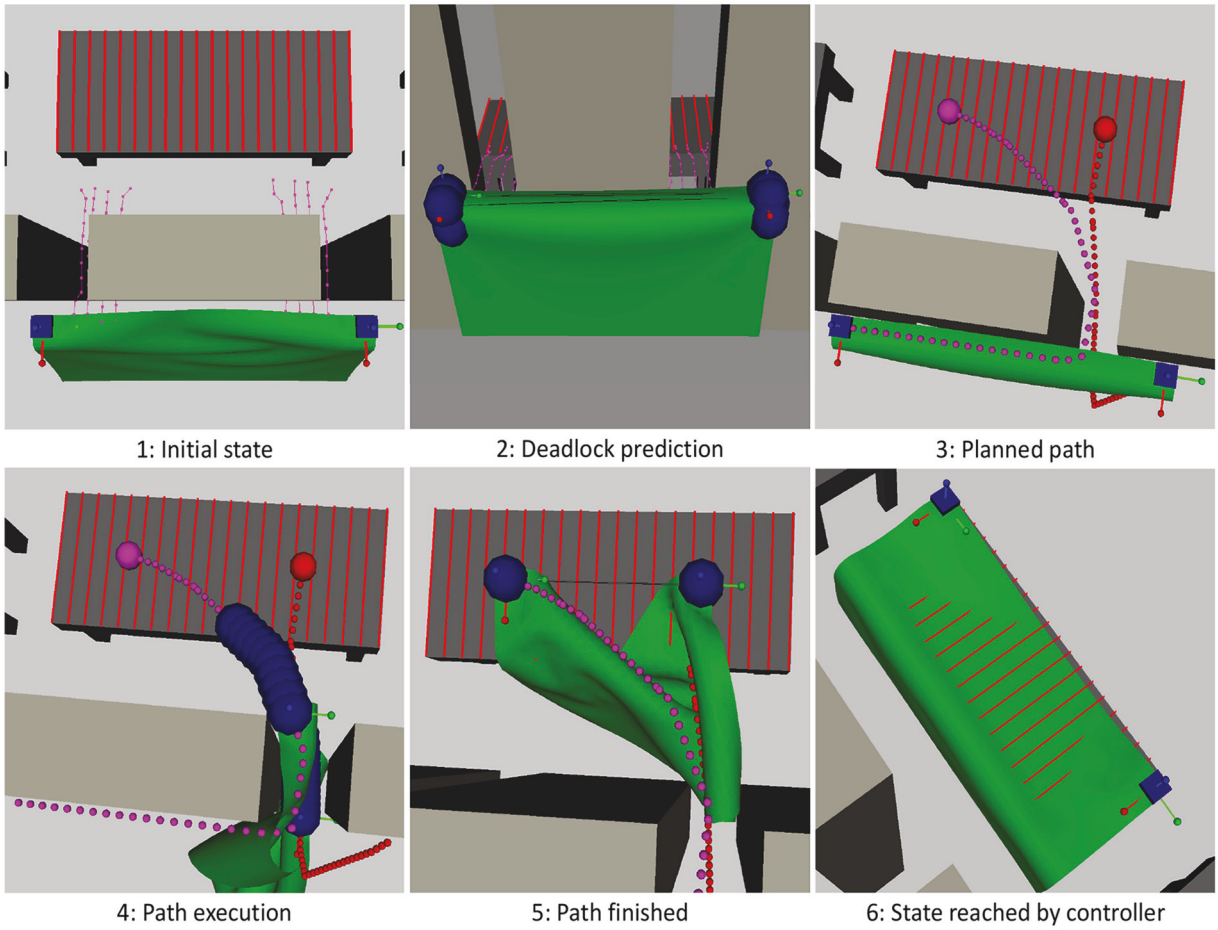

Fig. 10. Sequence of snapshots showing the execution of the second experiment. We use the same colors as the previous experiment (Figure 9), but in this example instead of detecting future overstretch in panel (2), we detect that the system is stuck in a bad local minimum and unable to make progress.

7.3. Moving a rope through a maze

In the third task, the robot must navigate a rope through a three-dimensional maze before aligning the rope with a line traced on the floor (Figure 11). This scenario is meant to represent tasks such as moving a heavy cable through a construction zone without crane access. In this task, the correspondences between the target points $\mathcal{T}$ and the deformable object points $\mathcal{P}$ are fixed in advance, thus the CalculateCorrespondences() function does not have to do any work, as shown in Table 4. Task error ρ is defined in the same way as in the first two experiments. Again the planner is invoked a single time per trial, but planning and smoothing times are longer than the previous tasks. This is a function of the size of the environment rather than any particular difference in the difficulty of performing the planning or smoothing. The planner finds a feasible path in 4.2 s on average, suggesting that our method can maintain fast planning times, even in larger environments with many more obstacles.

7.4. Repeated planning

The fourth task is a variant of the third, with the start configuration of the rope moved near the goal region on the top layer of the maze and a longer rope. This task has the most potential for a planned path to move the deformable object into a configuration from which the local controller cannot finish the task by wrapping the rope around an obstacle near the goal. For this experiment, we reduce the size of the planning arena to only the goal area, and the immediate surroundings on the top layer (Figure 12). From this starting position, the planner is more likely to find the incorrect neighborhood for the local controller, which corresponds to placing the rope into the wrong homotopy class, on the first attempt. We emphasize that the correct

Table 4. Local controller and deadlock prediction average computation time per iteration for each type of deformable object, averaged across all trials.

	Calculate Correspondences() time (s)	Predict Deadlock() time (s)	Local controller time (s)
Cloth	0.0114	0.0077	0.0126
Rope	0	0.0119	0.0023

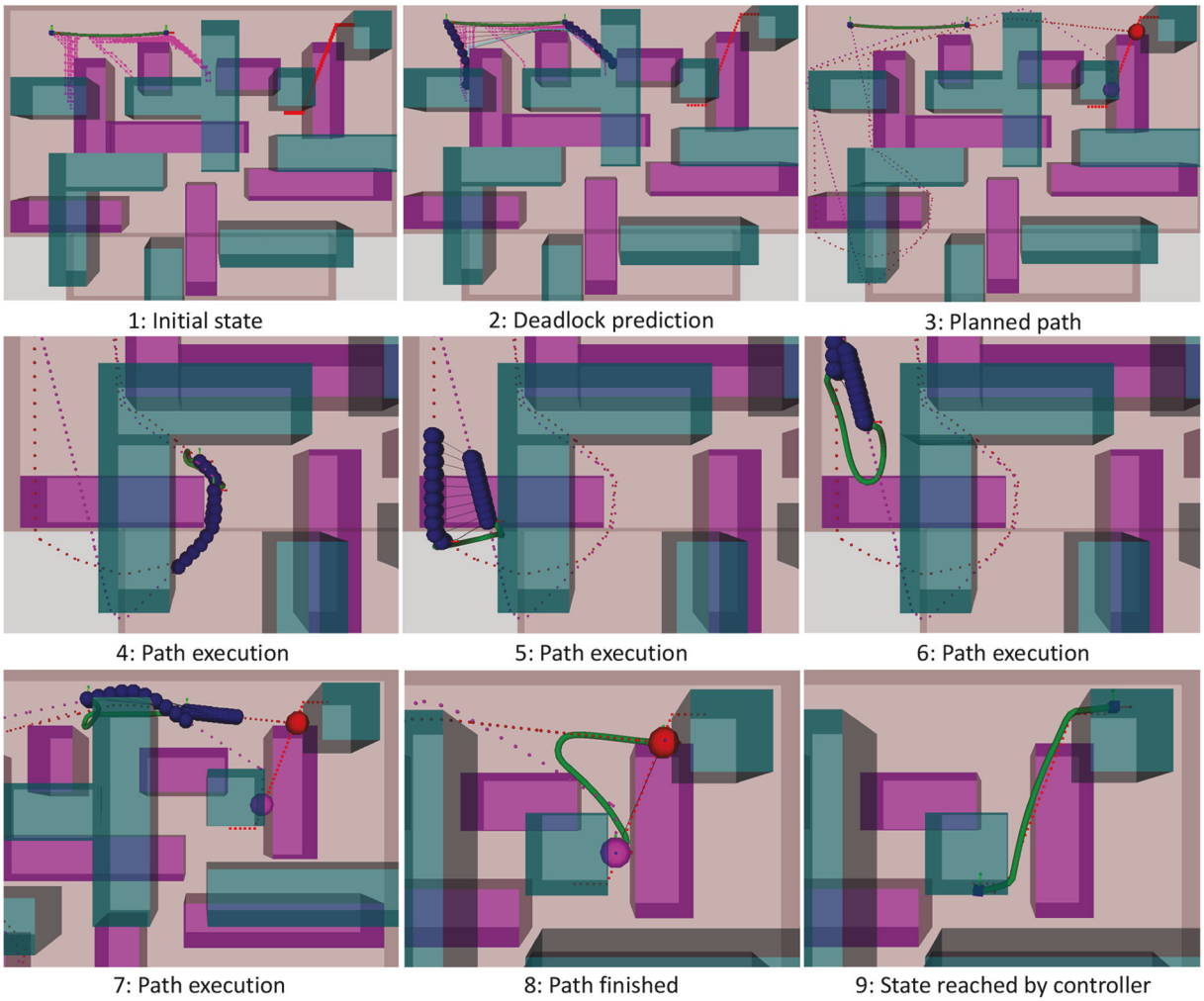

Fig. 11. Sequence of snapshots showing the execution of the third experiment. The rope is shown in green starting in the top left corner, the grippers are shown in blue, and the target points are shown in red in the top right corner. The maze consists of top and bottom layers (green and purple, respectively). The rope starts in the bottom layer and must move to the target on the top layer through an opening (bottom left or bottom right).
Note: Colour version of the figure is available online.

homotopy class is unknown, as we assume no information is given about the connectivity of the target points. Thus, our method must discover the correct homotopy class by trail-and-error, invoking the planner when the deadlock prediction determines the controller will be stuck.

In 71 of the 100 trials, the planner was invoked twice, in 13 other trials it was invoked three times, and in 2 trials it was invoked four times. These additional planning and smoothing stages took on average an additional 6.6 seconds, but the task was completed successfully in all 100 trials. This experiment suggests that our framework is able to effectively explore different band neighborhoods until the correct one is found, enabling the local controller to finish the task, even when the initial configuration is adversarial.

7.5. Computation time

To verify the practicality of our deadlock prediction algorithm and virtual elastic band approximation, we gathered data comparing computation time for these components to the local controller by itself, and to using the Bullet simulator. Table 4 shows the average times per iteration for the local controller and deadlock prediction algorithms, averaged across all trials of all experiments. As expected, adding in the deadlock prediction step does increase computation time, but the overall control loop is still fast enough for practical use.

Table 5. Average computation time to compute the effect of a gripper motion.

	Bullet simulation time (ms)	Virtual elastic band propagation time (ms)
Cloth	36.12	0.19
Rope	3.19	0.58

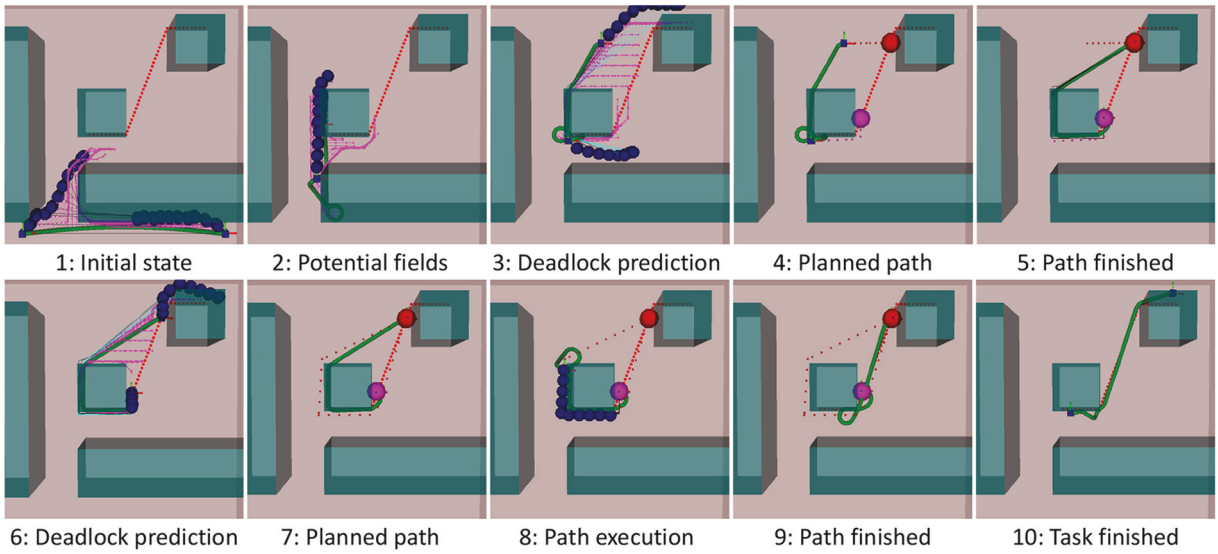

Fig. 12. Sequence of snapshots for the fourth experiment. We use the same colors as the previous experiment (Figure 11), but in this example the local controller gets stuck twice, in panels 3 and 6. In panel 7 the global planner finds a new neighborhood that is distinct from previously tried neighborhoods.

Table 5 shows a comparison between the average time needed to compute the virtual elastic band propagation for a gripper motion and the time needed to reliably simulate a gripper motion with the Bullet simulator. Note that the amount of time required for the simulator to converge to a stable estimate depends on many conditions, including what object is being simulated. Through experimentation we determined that 4 simulation steps were adequate for rope and 10 for cloth. Comparing the time needed to do this simulation with the time needed to forward propagate a virtual elastic band, we see that our approximation is indeed faster by an order of magnitude for rope, and by two orders of magnitude for cloth. This result reinforces the importance of using a simplified model, such as the virtual elastic band, within the planner: this model, while not as accurate as a simulation, allows us to evaluate motions much faster.

8. Physical robot experiment and results

In order to show that our method is practical for a physical robotic system, not only free floating end-effectors, we set up a task similar to the single pillar task (Section 7.1) with a dual-arm robot. It also shows that while our methods make strong assumptions about the ability to perceive the deformable object in Section 3 (in particular, no occlusions and no sensor noise), our framework is still able to perform meaningful tasks when those assumptions are violated. In this task, the robot must align a cloth placemat inside of the pink rectangle, going around an obstacle in the process (Figure 13).

8.1. Experiment setup

8.1.1. Robotic platform. Val is a stationary robotic platform with a 2-DoF torso, two 7-DoF arms, and a rotary pincer per arm. As in the simulated environments it is assumed that Val is already holding the cloth, leaving 16 DoFs to be controlled and planned for ($\mathbb{C}_r = \mathbb{R}^{16}$).

8.1.2. Cloth perception. The placemat is 0.33 m × 0.46 m that we discretize into a 3 × 3 grid. As tracking of deformable objects is a difficult problem, and out of scope of this article, we instead use fiducials to track the configuration of the cloth. Two of the points are tracked using the position of the grippers; the other seven points are tracked with AprilTags (Olson, 2011) and a Kinect V2 RGB-D sensor (Wiedemeyer, 2014–2015).

In order to address occlusions and noisy data, we filter the raw observations using a set of objective terms, and a set of constraints (see Figure 14). We use z_i to denote the last observed position of point i, and use t_i to denote the last time point i was observed. Then we add objective terms to pull the cloth estimate towards the observations, combined with constraints between each pair of points to ensure that the estimate is plausible:

$$\begin{aligned} \mathcal{P}(t) = \underset{\{p_i\}}{\text{argmin}} \quad & \sum_i e^{-K_T(t-t_i)} \|p_i - z_i\|^2 \\ \text{subject to} \quad & \|p_i - p_j\|^2 \leq K_L d_{ij}^2 \;\; \forall i,j \text{ s.t. } i \neq j \end{aligned} \tag{37}$$

where K_T and K_L are task-defined scale factors that we set to 1.5 and 1.0001, respectively, for this task.

8.2. Experiment results

We use the same deadlock, distance, and planner parameters as used in the simulation experiments, performing 500 smoothing iterations once a path is found. We constrain the rotation of the end-effectors to stay within 1.6 rad of their

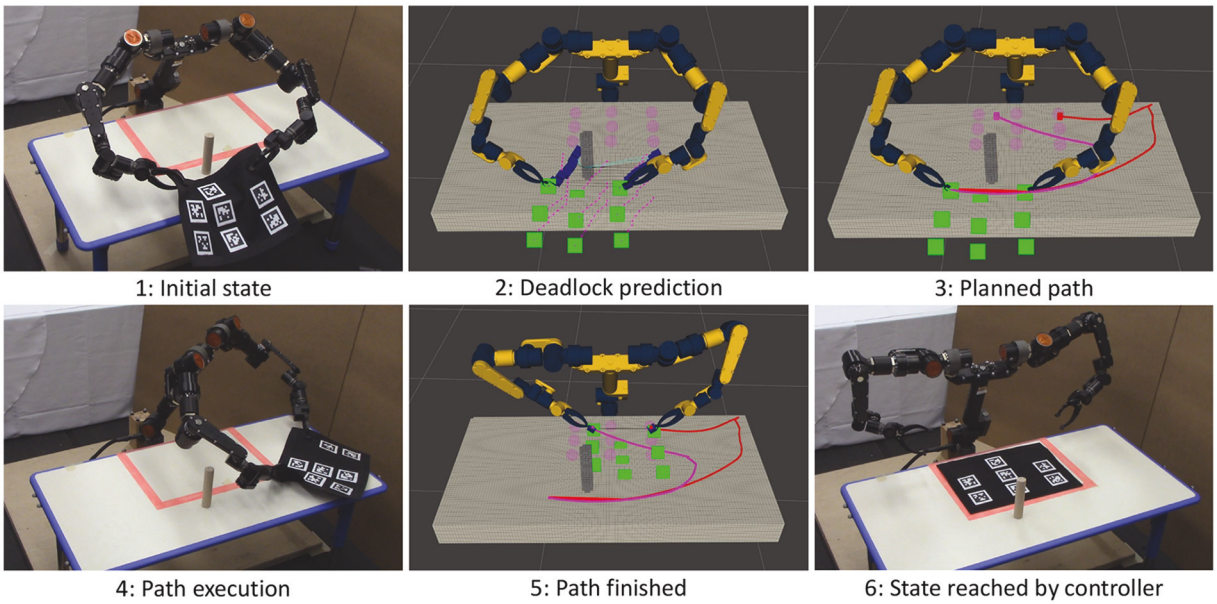

Fig. 13. Cloth placemat task. The placemat starts on the far side of an obstacle and must be aligned with the pink rectangle near the robot.

starting orientation during the planning process as well as constrain the grippers to stay close to the table. This forces the planner to move the placemat around the obstacle rather than over the obstacle. Finally, we also introduce planning restarts (Wedge and Branicky, 2008) into the planning process in order to address the greater complexity added by using a 16-DoF robot and the relatively strict workspace constraints; the restart timeout we set is 60 seconds.

Table 6 shows the planning statistics across 100 planning trials with identical starting configurations, but different random seeds. On average planning and smoothing takes less than 60 seconds, with forward kinematics and collision checking dominating the planning time. The restart timeout was unused in 68 out of 100 trials, with the other 32 trials requiring a total of 50 restarts between them. Figure 15 shows that the planning time follows a "heavy tail" distribution typical of sampling-based planners.

Our overall framework is able to complete this task as shown in Figure 13. As in the simulated version of this task, we are able to predict deadlock before the robot gets stuck, plan and execute a path to a new neighborhood, and then use the local controller to finish the task.

9. Discussion and conclusion

We have presented a method to interleave global planning and local control for deformable object manipulation that does not rely on high-fidelity modeling or simulation of the object. Our method combines techniques from topologically based motion planning with a sampling-based planner to generate gross motion of the deformable object. The purpose of this gross motion is not to achieve the task alone, but rather to move the object into a position from which the local controller is able to complete the task. This division of labor enables each component to focus on their strengths rather than attempt to solve the entire problem directly. We also presented a probabilistic completeness proof for our planner which does not rely on either a steering function or choosing controls at random, and addresses our underactuated system. As part of our framework, we introduced a novel deadlock prediction algorithm to determine when to use the local controller and when to use the global planner.

Our experiments demonstrate that our framework is able to be applied to several interesting tasks for rope and cloth, including an adversarial case where we set up the planner to fail on the first attempt. For the simulated tasks, our framework is able to succeed at each task 100/100 times, with average planning and smoothing time under 4 seconds for 3 tasks, and under 11 seconds for the larger environment. The physical robot experiment shows that our framework can be used for practical tasks in the real world, with planning and smoothing taking less than 60 seconds on average. This experiment also shows that our methods can function despite noisy and occluded perception of the deformable object.

9.1. Parameter selection

There are several parameters in both the local controller and the global planner that can have a significant effect on the performance of our method. In particular, if the local controller is prone to oscillations (Appendix B.4), this can cause the deadlock prediction algorithm to incorrectly predict that the local controller will get stuck, leading to an unnecessary planning phase. In the worse case, this can cause the global planner to be unable to find an acceptable

Table 6. Planning statistics for the cloth placemat example, averaged across 100 trials. Standard deviation is shown in brackets.

RRT planning						**Smoothing**			
Samples	States	NN time (s)	Validity checking time (s)	Random restarts	Total time (s)	Iterations	Validity checking time (s)	Visibility deformation time (s)	Total time (s)
83,041 [83,677]	8,438 [6,182]	4.5 [4.9]	44.1 [44.5]	0.5 [0.9]	50.0 [50.9]	500	3.6 [1.1]	0.1 [∼0.0]	3.6 [1.1]

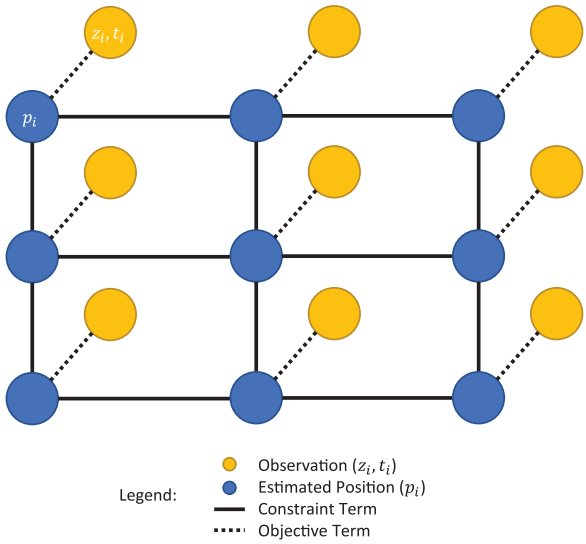

Fig. 14. Constraint and objective graph for (37). Note that not all constraints are shown to avoid clutter; every estimated position has a constraint between itself and every other estimated position.

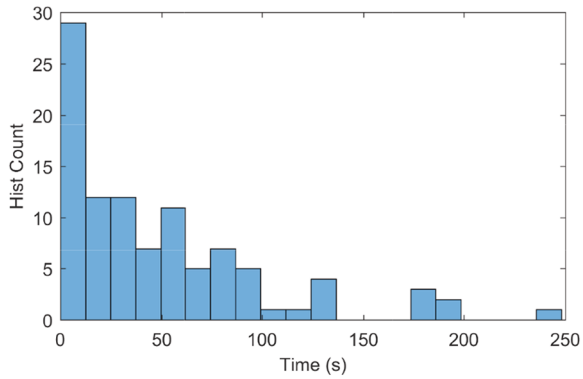

Fig. 15. Histogram of planning times across 100 trials for the cloth placemat experiment.

path owing to the blacklisting procedure. One interesting direction of future research is how to perform reachability analysis for deformable objects in general, in particular when a high-fidelity model of the deformable object is not available. In practice, we found that increasing the prediction horizon N_p and prediction annealing factor α was not useful as the prediction accuracy degrades quickly. We did have to tune the history window N_h and thresholds β_e, β_m against each other. Error improvement threshold β_e needs to be set relative to the definition of task error ρ, while β_m is more sensitive to oscillations. If β_m is too small, then the system will fail to detect that the controller is stuck in a poor local minima. If these thresholds are too high or N_h is too low, then false positives were common near the end of the table coverage tasks.

For the global planner, we found that the goal bias γ_{gb} has a similar effect on planning time as a standard RRT; values in the range $[0.05, 0.15]$ produced similar planning times for our experiments. In addition, if λ_b is not small, then nearest-neighbor checks can become very expensive. In practice, distances in band space are used to disambiguate between nodes that are at nearly identical configurations in robot configuration space. This happens when multiple nodes connect to the position goal $q_{\text{xyz}}^{\text{goal}}$, but their bands are similar to a blacklisted band. One potential way to make distances in band space more informative would be to develop a way to sample interesting band configurations.

9.2. *Limitations*

We made a choice to favor speed over model accuracy. As a consequence, there are several issues that our method does not address. In particular, environments with "hooks" can cause problems owing to our approximation methods; the virtual elastic band we use for constraint checking and planning assumes that: (1) there is no minimum length of the deformable object and (2) there are no holes in the deformable object. These assumptions mean that our planner cannot detect cases where the slack material or a hole can get snagged on corners or hooks, preventing the motion plan from being executed. One way this can be mitigated is by using a more accurate model (at the cost of speed and task-specific tuning). Other potential solutions include online modeling methods such as Hu et al. (2018), or learning which features of the workspace can lead to highly inaccurate approximations and planning paths that avoid those areas. In addition, we have no explicit method to avoid twisting or knot-tying behavior. While shortcut smoothing can potentially mitigate the worst effects, avoiding such cases is not something that is within the scope of this work.

Similarly, we do not have any explicit consideration for achieving a task that requires knot-tying or twisting; while some other local controller may be able to perform these tasks from a suitable starting state, we have not investigated this option. Finally, we cannot guarantee that we can achieve any given task in general; while our blacklisting method is designed to encourage exploration of the state space, it also has the potential to block regions of the state space from which the local controller can achieve the task. Defining a set of tasks that our framework can successfully perform is not practical given the limited set of assumptions we are making about the deformable object. Despite these limitations we find that our framework is able to reliably perform complex tasks where neither planning nor control alone are sufficient. In future work, we plan to address these weaknesses, in particular the snagging and twisting limitations, which are artifacts of our approximation methods. We also seek to extend our framework to a broader range of tasks, beyond coverage and point matching applications.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported in part by NSF grants IIS-1750489 and IIS-1656101, and ONR grants N000141712050 and N000141712303.

ORCID iD

Dale McConachie http://orcid.org/0000-0002-2615-3473

References

Alt H and Godau M (1995) Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications* 05(01–02): 75–91.

Anshelevich E, Owens S, Lamiraux F and Kavraki L (2000) Deformable volumes in path planning applications. In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2290–2295.

Bai Y, Yu W and Liu CK (2016) Dexterous manipulation of cloth. *Computer Graphics Forum* 35(2): 523–532.

Baraff D and Witkin A (1998) Large steps in cloth simulation. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. New York: ACM Press, pp. 43–54.

Berenson D (2013) Manipulation of deformable objects without modeling and simulating deformation. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 4525–4532.

Bergou M, Wardetzky M, Robinson S, Audoly B and Grinspun E (2008) Discrete elastic rods. *ACM Transactions on Graphics* 27(3): 1.

Bhattacharya S, Likhachev M and Kumar V (2012) Topological constraints in search-based robot path planning. *Autonomous Robots* 33(3): 273–290.

Brass P, Vigan I and Xu N (2015) Shortest path planning for a tethered robot. *Computational Geometry* 48(9): 732–742.

Burchan Bayazit O, Jyh-Ming Lien and Amato N (2002) Probabilistic roadmap motion planning for deformable objects. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 2, pp. 2126–2133.

Coumans E (2010) Bullet physics library. http://bulletphysics.org.

Essahbi N, Bouzgarrou BC and Gogu G (2012) Soft material modeling for robotic manipulation. *Applied Mechanics and Materials* 162: 184–193.

Frank B, Stachniss C, Abdo N and Burgard W (2011) Efficient motion planning for manipulation robots in environments with deformable objects. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 2180–2185.

Gayle R, Lin M and Manocha D (2005) Constraint-based motion planning of deformable robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1046–1053.

Gibson SFF and Mirtich B (1997) A survey of deformable modeling in computer graphics. Technical Report, Mitsubishi Electric Research Laboratories.

Goldenthal R, Harmon D, Fattal R, Bercovier M and Grinspun E (2007) Efficient simulation of inextensible cloth. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 26(3): 49–es.

Gurobi (2016) Gurobi optimization library. http://www.gurobi.com .

Hirai S and Wada T (2000) Indirect simultaneous positioning of deformable objects with multi-pinching fingers based on an uncertain model. *Robotica* 18(1): 3–11.

Hsu D, Latombe J and Motwani R (1999) Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications* 09(04–05): 495–512.

Hu Z, Sun P and Pan J (2018) Three-dimensional deformable object manipulation using fast online Gaussian process regression. *IEEE Robotics and Automation Letters* 3(2): 979–986.

Huang SH, Pan J, Mulcaire G and Abbeel P (2015) Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 878–885.

Irving G, Teran J and Fedkiw R (2004) Invertible finite elements for robust simulation of large deformation. In: *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 131–140.

Jaillet L and Siméon T (2008) Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *The International Journal of Robotics Research* 27(11–12): 1175–1188.

Jiménez P (2012) Survey on model-based manipulation planning of deformable objects. *Robotics and Computer-Integrated Manufacturing* 28(2): 154–163.

Karaman S and Frazzoli E (2013) Sampling-based optimal motion planning for non-holonomic dynamical systems. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5041–5047.

Kaufmann P, Martin S, Botsch M and Gross M (2008) Flexible simulation of deformable models using discontinuous Galerkin FEM. In: *Proceedings of SIGGRAPH*.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

Khalil F and Payeur P (2010) Dexterous robotic manipulation of deformable objects with multi-sensory feedback – a review. In: *Robot Manipulators, Trends and Development*. InTech, pp. 587–621.

Kim S and Likhachev M (2015) Path planning for a tethered robot using Multi-Heuristic A* with topology-based heuristics. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 4656–4663.

Kunz T and Stilman M (2015) Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete. In: *Algorithmic foundations of robotics XI*. Cham: Springer, pp. 233–244.

Lamiraux F and Kavraki LE (2001) Planning paths for elastic objects under manipulation constraints. *The International Journal of Robotics Research* 20(3): 188–208.

LaValle S and Kuffner J (2001) Randomized kinodynamic planning. *International Journal of Robotics Research* 20(5): 378–400.

LaValle SM (2006) *Planning Algorithms*. Cambridge: Cambridge University Press.

Li Y, Littlefield Z and Bekris KE (2016) Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research* 35: 528–564.

Maris B, Botturi D and Fiorini P (2010) Trajectory planning with task constraints in densely filled environments. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 2333–2338.

Moll M and Kavraki LE (2006) Path planning for deformable linear objects. *IEEE Transactions on Robotics* 22(4): 625–636.

McConachie D and Berenson D (2018) Estimating model utility for deformable object manipulation using multiarmed bandit methods. *IEEE Transactions on Automation Science and Engineering* 15(3): 967–979.

M^{c}Conachie D, Ruan M and Berenson D (2017) Interleaving planning and control for deformable object manipulation. In: *Proceedings of the International Symposium on Robotics Research (ISRR)*.

Müller M, Dorsey J, McMillan L, Jagnow R and Cutler B (2002) Stable real-time deformations. In: *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 49–54.

Navarro-Alarcon D and Liu Y (2018) Fourier-based shape servoing: A new feedback method to actively deform soft objects into desired 2-D image contours. *IEEE Transactions on Robotics* 34(1): 272–279.

Navarro-Alarcon D, Liu Yh, Romero JG and Li P (2014) On the visual deformation servoing of compliant objects: Uncalibrated control methods and experiments. *The International Journal of Robotics Research* 33(11): 1462–1480.

Navarro-Alarcon D, Yip HM, Wang Z, et al. (2016) Automatic 3-D manipulation of soft objects by robotic arms with an adaptive deformation model. *IEEE Transactions on Robotics* 32(2): 429–441.

Olson E (2011) AprilTag: A robust and flexible visual fiducial system. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407.

Park D, Kapusta A, Hawke J and Kemp CC (2014) Interleaving planning and control for efficient haptically-guided reaching in unknown environments. In: *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 809–816.

Quinlan S (1994) *Real-time Modification of Collision-free Paths*. PhD Thesis, Department of Computer Science, Stanford University.

Robot Learning Lab (2012) Simulation environment with Bullet physics. https://github.com/rll/bulletsim (accessed 2 July 2012).

Rodriguez S and Amato N (2006) An obstacle-based rapidly-exploring random tree. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 895–900.

Roussel O, Borum A, Taïx M and Bretl T (2015) Manipulation planning with contacts for an extensible elastic rod by sampling on the submanifold of static equilibrium configurations. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3116–3121.

Rungjiratananon W, Kanamori Y, Metaaphanon N, Bando Y, Chen BY and Nishita T (2011) Twisting, tearing and flicking effects in string animations. In: Allbeck JM and Faloutsos P (eds.) *Motion in Games* (*Lecture Notes in Computer Science*, Vol. 7060). Berlin: Springer, pp. 192–203.

Saha M, Isto P and Latombe JC (2008) Motion planning for robotic manipulation of deformable linear objects. In: *Proceedings of the International Symposium on Experimental Robotics (ISER)*. Berlin: Springer, pp. 23–32.

Sanchez J, Corrales JA, Bouzgarrou BC and Mezouar Y (2018) Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey. *The International Journal of Robotics Research* 37(7): 688–716.

Schulman J, Ho J, Lee C and Abbeel P (2016) Learning from demonstrations through the use of non-rigid registration. In: *Robotics Research: The 16th International Symposium ISRR* (*Springer Tracts in Advanced Robotics*, Vol. 114). Berlin: Springer, pp. 339–354.

Smolen J and Patriciu A (2009) Deformation planning for robotic soft tissue manipulation. In: *2009 Second International Conferences on Advances in Computer–Human Interactions*, pp. 199–204.

Wada T, Hirai S, Kawarnura S and Karniji N (2001) Robust manipulation of deformable objects by a simple PID feedback. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 85–90.

Wedge NA and Branicky MS (2008) On heavy-tailed runtimes and restarts in rapidly-exploring random trees. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 127–133.

Wiedemeyer T (2014–2015) IAI Kinect2. https://github.com/code-iai/iai_kinect2 (accessed 1 July 2018).

Appendix A. Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extension

Extension	Media type	Description
1	Video	Manipulating deformable objects by interleaving prediction, planning, and control

Algorithm 10 CalculateCorrespondences($\mathcal{P}, \mathcal{T}$)

$\mathcal{T}_c = [\emptyset]_{1\times T}$
for $i \in \{1, 2, \ldots, T\}$ **do**
  $j \leftarrow \text{argmin}_{j\in\{1,2,\ldots,P\}} d_{\text{Dijkstras}}(\mathcal{T}_i, \mathcal{P}_j)$
  $d \leftarrow d_{\text{Dijkstras}}(\mathcal{T}_i, \mathcal{P}_j)$
  $\mathcal{T}_c[j] \leftarrow \{\mathcal{T}_c[j] \cup (j, d)\}$
end for
return $\mathcal{T}_c$

Algorithm 11 FollowNavigationFunction($\mathcal{P}, \mathcal{T}_c$)

$\dot{\mathcal{P}}_e \leftarrow \mathbf{0}_{3P\times 1}$
$W_e \leftarrow \mathbf{0}_{P\times 1}$
for $j \in \{1, 2, \ldots, P\}$ **do**
  for $(j, d) \in \mathcal{T}_c[j]$ **do**
    $\dot{\mathcal{P}}_{e,i} \leftarrow \dot{\mathcal{P}}_{e,i} +$ DijkstrasNextStep($\mathcal{P}_i, j$)
    $W_{e,j} \leftarrow \max(W_{e,j}, d)$
  end for
end for
return $\dot{\mathcal{P}}_e, W_e$

Appendix B. Local controller

This appendix provides the details of each component of the local controller (Algorithm 1). The three main sections determine: (1) which direction to manipulate the deformable object in order to reduce task error; (2) adjustments to help avoid overstretch of the deformable object; and (3) Determining the best direction to move the robot to achieve (1) and (2). We discuss each section in turn.

B.1. Reducing error

Determining which direction to manipulate the deformable object in order to reduce task error is done in three steps (Algorithms 10 and 11).

Each task defines a navigation function for every target point $\mathcal{T}_i$ using Dijkstra's algorithm. In general, there is no one-to-one mapping between $\mathcal{T}$ and $\mathcal{P}$; at every timestep, for every target point $\mathcal{T}_i$, we recalculate which point on the deformable object $\mathcal{P}_j$ is closest, using the results from Dijkstra's algorithm to measure distance (Algorithm 10). These individual results are then aggregated in Algorithms 11 to define the best direction to manipulate the deformable object in order to reduce error, and the relative importance of doing so for each point on the deformable object. The directions each navigation function indicates are added together to define the overall direction to manipulate a point (Algorithm 11, line 5). For the importance factors W_j, we take only the largest distance that $\mathcal{P}_j$ would have to move as a way to mitigate discretization effects (Algorithm 11, line 6).

Algorithm 12 StretchingCorrection($D, \lambda_s, \mathcal{P}$) (adapted from McConachie and Berenson (2018))

$E \leftarrow$ EuclidianDistanceMatrix($\mathcal{P}$)
$\dot{\mathcal{P}}_s \leftarrow \mathbf{0}_{3P\times 1}$, $W_s \leftarrow \mathbf{0}_{P\times 1}$
for $i \in \{1, 2, \ldots, P\}$ **do**
  for $j \in \{i + 1, \ldots, P\}$ **do**
    if $E_{i,j} > \lambda_s D_{i,j}$ **then**
      $\Delta_{i,j} \leftarrow E_{i,j} - D_{i,j}$
      $v \leftarrow \Delta_{i,j}(\mathcal{P}_j - \mathcal{P}_i)$
      $\dot{\mathcal{P}}_{s,i} \leftarrow \dot{\mathcal{P}}_{s,i} + \frac{1}{2}v$
      $\dot{\mathcal{P}}_{s,j} \leftarrow \dot{\mathcal{P}}_{s,j} - \frac{1}{2}v$
      $W_{s,i} \leftarrow \max(W_{s,i}, \Delta_{i,j})$
      $W_{s,j} \leftarrow \max(W_{s,j}, \Delta_{i,j})$
    end if
  end for
end for
return $\dot{\mathcal{P}}_s, W_s$

Algorithm 13 CombineTerms($\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s, \lambda_w$) (adapted from McConachie and Berenson (2018))

for $i \in \{1, 2, \ldots, P\}$ **do**
  $\dot{\mathcal{P}}_{d,i} \leftarrow \dot{\mathcal{P}}_{s,i} + \left(\dot{\mathcal{P}}_{e,i} - \text{Proj}_{\dot{\mathcal{P}}_{s,i}} \dot{\mathcal{P}}_{e,i}\right)$
  $W_{d,i} \leftarrow \lambda_w W_{s,i} + W_{e,i}$
end for
return $\dot{\mathcal{P}}_d, W_d$

B.2. Stretching correction

Our algorithm for stretching correction is similar to that found in Berenson (2013), with the addition of a weighting term λ_w, and a change in how we combine error correction and stretching correction. We use the StretchingCorrection() function (Algorithm 12) to compute $\dot{\mathcal{P}}_s$ and W_s based on a task-defined stretching threshold $W_s \geq 0$. First we compute the distance between every two points on the object and store the result in E. We then compare E with D, which contains the relaxed lengths between every pair of points. If any two points are stretched by more than a factor of W_s, we attempt to move the points closer to each other. We use the same strategy for setting the importance of this stretching correction W_s as we use for error correction. When combining stretching correction and error correction terms (Algorithm 13) we prioritize stretching correction, accepting only the portion of the error correction that is orthogonal to the stretching correction term for each point. Here λ_w is used to define the relative scale of the importance factors W_e and W_s.

B.3. Finding the best robot motion

Given a desired deformable object velocity $\dot{\mathcal{P}}_d$ and relative importance weights W_d, we want to find the robot motion

Algorithm 14 FindBestRobotMotionSim($q_r, \mathcal{P}, \dot{\mathcal{P}}_d, W_d$)

$\dot{q}_r^{cmd} \leftarrow \psi_{\mathfrak{se}(3)}(\dot{\mathcal{P}}, W_d)$ Equation (40)
$\dot{q}_r^{cmd} \leftarrow$ ObstacleRepulsion($\dot{q}_r^{cmd}, \mathcal{O}, \beta$)
return $\dot{q}_r^{cmd}$

that best achieves ($\dot{\mathcal{P}}_d, W_d$), that is,

$$\begin{aligned} \underset{\dot{q}_r}{\text{argmin}} \quad & \| f(q_r, \mathcal{P}, \dot{q}_r) - \dot{\mathcal{P}}_d \|_{W_d} \\ \text{subject to} \quad & \|\dot{q}_r\| \leq \dot{q}_r^{\max} \\ & (q_r + \dot{q}_r) \in \mathbb{C}_r^{valid} \end{aligned} \tag{38}$$

In general, $f(\dots)$ is not known. For our controllers we use a Jacobian-based approximation

$$f(q_r, \mathcal{P}, \dot{q}_r) \approx J_d \dot{q}_r \tag{39}$$

from McConachie and Berenson (2018: section V-C).

Our method for ensuring the robot stays in $\mathbb{C}_r^{valid}$ is different, depending on which robot we are using.

B.3.1. Simulated experiments. For the simulated experiments, we first solve Equation (38) using our Jacobian approximation:

$$\begin{aligned} \psi_{\mathfrak{se}(3)}(\dot{\mathcal{P}}, W) = \underset{\dot{q}_r}{\text{argmin}} \quad & \|J_d \dot{q}_r - \dot{\mathcal{P}}\|_W^2 \\ \text{subject to} \quad & \|\dot{q}_r\|^2 \leq \dot{q}_{r,\max,e}^2 \end{aligned} \tag{40}$$

where $\dot{q}_{r,\max,e}$ is the maximum velocity for each individual end-effector (Algorithm 14).

In order to guarantee that the grippers do not collide with any obstacles, we use the same strategy from Berenson (2013), smoothly switching between collision avoidance and other objectives (see Algorithm 15). For every gripper g and an obstacle set $\mathcal{O}$ we find the distance d_g to the nearest obstacle, a unit vector $\dot{x}_{p^g}$ pointing from the obstacle to the nearest point on the gripper, and a Jacobian J_{p^g} between the gripper's DoF and the point on the gripper as shown in Algorithm 16. We then project the servoing motion from Equation (40) into the null space of the avoidance motion using the null space projector ($\mathbf{I} - J_{p^g}^+ J_{p^g}$). Here $\beta > 0$ sets the rate at which we change between servoing and collision avoidance objectives and $\dot{q}_{r,\max,o} > 0$ is an internal parameter that sets how quickly we move the robot away from obstacles.

B.3.2. Physical experiments. For the physical robot, instead of handling collision avoidance in a post-processing step, we build the collision constraints directly into the optimization function (Algorithm 17). To do so, we define a set of points $\mathcal{C} = \{c_1, c_2, \dots\}$ on the robot that must stay at least d_{buffer} away from obstacles. In our implementation, this is the end-effectors, wrists, and elbows of each arm of the robot. We then use the same Proximity() function (Algorithm 16) as the simulated robot to define an extra constraint that must be satisfied:

$$\begin{aligned} \psi_{\mathbb{R}16}(\dot{\mathcal{P}}, W) = \underset{\dot{q}_r}{\text{argmin}} \quad & \|J_d \dot{q}_r - \dot{\mathcal{P}}\|_W^2 \\ \text{subject to} \quad & q_r + \dot{q}_r \in \mathbb{C}_r \\ & \|\dot{q}_r\|^2 \leq \dot{q}_{r,\max}^2 \\ & \|J_r \dot{q}_r\|^2 \leq \dot{q}_{r,\max,e}^2 \\ & \dot{x}_{p^g}^T J_{p^g} \dot{q}_r \leq d_g + d_{\text{buffer}} \end{aligned} \tag{41}$$

In addition, we constrain the velocity of the robot both in joint configuration space

$$\|\dot{q}_r\|^2 \leq \dot{q}_{r,\max}^2$$

and the velocity of the end-effectors in $SE(3)$

$$\|J_r \dot{q}_r\|^2 \leq \dot{q}_{r,\max,e}^2$$

To solve Equations (40) and (41) we use the Gurobi optimizer (Gurobi, 2016). Table 7 shows the parameters we use for each experiment.

Algorithm 15 ObstacleRepulsion($\mathcal{O}, \beta$) (adapted from McConachie and Berenson (2018))

for $g \in \{1, 2\}$ **do**
  $J_{p^g}, \dot{x}_{p^g}, d_g \leftarrow$ Proximity($\mathcal{O}, g$)
  $\gamma \leftarrow e^{-\beta d_g}$
  $\dot{q}_{r,g,c} \leftarrow J_{p^g}^+ \dot{x}_{p^g}$
  $\dot{q}_{r,g,c} \leftarrow \frac{\dot{q}_{r,\max,o}}{\|\dot{q}_{r,g,c}\|} \dot{q}_{r,g,c}$
  $\dot{q}_{r,g,c} \leftarrow \gamma \left(\dot{q}_{r,g,c} + \left(\mathbf{I} - J_{p^g}^+ J_{p^g} \right) \dot{q}_{r,g,c} \right) + (1 - \gamma) \dot{q}_{r,g}$
end for
return $\dot{q}_r$

Algorithm 16 Proximity($g, \mathcal{O}$) (adapted from McConachie and Berenson (2018))

$d_g \leftarrow \infty$
for $o \in \{1, 2, \dots, |\mathcal{O}|\}$ **do**
  $p^g, p^o \leftarrow$ ClosestPoints(g, o)
  $v \leftarrow p^g - p^o$
  if $\|v\| < d_g$ **then**
    $d_g \leftarrow \|v\|$
    $\dot{x}_{p^g} \leftarrow \frac{v}{\|v\|}$
    $J_{p^g} \leftarrow$ RobotPointJacobian(g, p^g)
  end if
end for
return $J_{p^g}, \dot{x}_{p^g}, d_g$

Table 7. Controller parameters

		Simulated cloth trials	Simulated rope trials	Physical robot
Servoing max gripper velocity	$\dot{q}_{r,\max,e}$	0.2	0.2	0.3
Obstacle avoidance max gripper velocity	$\dot{q}_{r,\max,o}$	0.2	0.2	—
Max robot velocity	$\dot{q}_{r,\max}$	—	—	1.5
Obstacle avoidance scale factor	β	200	1,000	—
Max stretching factor	λ_s	1.15	1.17	1.01
Stretching correction weight factor	λ_w	2,000	2,000	2,000
Obstacle avoidance buffer	d_{buffer}	—	—	0.08
Workspace discretization (m)		0.02	0.05	0.02

Algorithm 17 FindBestRobotMotionPhys($q_r, \mathcal{P}, \dot{\mathcal{P}}_d, W_d$)

1: **for** $g \in \{1, 2, \ldots, |\mathcal{C}|\}$ **do**
2: $J_{p^g}, \dot{x}_{p^g}, d_g \leftarrow$ Proximity($\mathcal{O}, g$)
3: **end for**
4: $\dot{q}_r^{cmd} \leftarrow \psi_{\mathbb{R}16}(\dot{\mathcal{P}}, W_d)$ Eq. (41)

B.4. Parameter selection

While this controller is able to perform multiple coverage tasks successfully, it can be prone to oscillations in three circumstances in particular. First, If the gripper velocity is too high, or the obstacle avoidance scale factor is too small, the grippers can oscillate between servoing to decrease task error, and moving away from obstacles. This effect is most pronounced when the linearizations used inside the controller do not model the local environment well. The second case is when the task is nearly done; in this case, if the discretization level of the deformable object or the target points is too coarse, this can lead to rapid changes in the task error gradient, which can cause the controller to oscillate. Finally, if stretching correction is directly opposing task progress, this will lead to oscillation as the controller switches between the two objectives. The choice of workspace discretization is not critical as long as it is sufficient to capture any relevant details of obstacle geometry.