Deformable Object Manipulation: Learning While Doing

by

Dale M^{c}Conachie

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in The University of Michigan
2020

Doctoral Committee:

Associate Professor Dmitry Berenson, Chair
Professor Jessie Grizzle
Associate Professor Chad Jenkins
Professor Leslie Pack Kaelbling, Massachusetts Institute of Technology

Dale M^{c}Conachie

dmcconac@umich.edu

ORCID iD: 0000-0002-2615-3473

ACKNOWLEDGEMENTS

Thanks to Calder Phillips-Grafflin, Brad Saund, Andrew Dobson, and Yu-Chi Lin for helpful discussions. Thank you to all my collaborators from the Autonomous Robotic Manipulation Lab. Thanks to my advisor Dmitry Berenson for his guidance and encouragement. Thanks to Chad Jenkins, Jessie Grizzle, and Leslie Pack Kaelbling for their advice. Thanks to my parents for supporting my return to university and advice throughout. Finally, special thanks to my wife Molly for everything.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

This dissertation is motivated by two research questions: (1) How can robots perform a broad range of useful tasks with deformable objects without a time consuming modelling or data collection phase? and (2) How can robots take advantage of information learned while manipulating deformable objects?

To address the first question, I propose a framework for deformable object manipulation that interleaves planning and control, enabling complex manipulation tasks without relying on high-fidelity modeling or simulation. Each part of the framework uses a different representation of the deformable object that is well suited for the specific requirements of each component. The key idea behind these techniques is that we do not need to explicitly model and control every part of the deformable object, instead relying on the object's natural compliance in many situations.

For the second question, I consider the two major components of my framework and examine what can cause failure in each. The goal then is to learn from experience gathered while performing tasks in order to avoid making the same mistake again and again. To this end I formulate the controller's task as a Multi-Armed Bandit problem, enabling the controller to choose models based on the current circumstances. For the planner, I present a method to learn when we can rely on the robot's model of the deformable object, enabling the planner to avoid generating plans that are infeasible.

This framework is demonstrated in simulation with free floating grippers as well as on a 16 DoF physical robot, where reachability and dual-arm constraints make the tasks more difficult.

CHAPTER I

Introduction

In the 1950s and 1960s George Devol, Joseph Engleberger, and many others began developing machines that were programmable manipulators of objects. Industrial manufacturers, in a desire to reduce labour costs, improve quality, and reduce delivery times were early adopters of this technology. These industrial robots were programmed to perform highly repetitive manipulation of various rigid objects in fixed environments. Key to the robot's success are the specific program instructions derived from previously measured and calculated features of the real-world objects under manipulation in a fixed environment. The pervasive use of industrial robots performing flawlessly around the globe in factories performing tasks like medical laboratory testing, automobile assembling, or electronics circuit board manufacturing demonstrate the success of programmable manipulators of objects. Complexity has increased, dexterity has improved, and a given robot may be capable of more than one task or can be applied to a different, but previously known size of object or in a different, fully described environment; but, the vast majority of the modeling and planning remains a human creation and is merely programmed into the robot in advance. The industrial robot certainly does not learn.

Indeed, 35 years ago, Michael Brady [2] argued that "Since robotics is the field concerned with the connection of perception to action, Artificial Intelligence must have a central role in Robotics if the connection is to be intelligent." Since then, there have been great strides made developing robots with intelligence; one prominent example of this is the self-driving automobile. Like industrial manipulators, a self-driving automobile knows much about its own dimensions and physics but it is constantly relying on computation-intensive processes for intelligence. The robot's task is achievable because the environment being modeled is rigid, the automobile's dynamics are known, and advancements in computing speed have soared, making it possible for the robot to execute a very large, but finite, number of calculations

quickly enough for secure and accurate control. Brady went on to say, "Robotics challenges AI by forcing it to deal with real objects in the real world." As true as that statement certainly is for robots such as self-driving automobiles, it is all the more true for robots that manipulate deformable objects possessing an infinite number of degrees of freedom and the inherently incomplete description of the object in all of its possible arrangements. An interesting example of a robot that manipulates deformable objects in current use is the da Vinci surgical robot, well known for its YouTube video demonstrating it stitching a grape back together with thread. Its relevance to this paper is simple: The surgical robot does not learn, plan, or control anything directly; those computationally intensive tasks are performed by a human who controls the robot's manipulating tools. The only way an argument could be made for declaring this surgical robot a learning, autonomous robot would be to assume that it ships from the factory, complete with a human operator who is deemed one of its components. Computational intensity is one of it's biggest hurdles. The computational challenge posed by modeling, planning, and control for deformable objects will be addressed in this dissertation; and, we describe a framework that we successfully used to obtain measurable improvements against that challenge.

Traditional motion planning techniques such as A*, probabilistic roadmaps, and rapidly-exploring random trees were designed with rigid body motion in mind. In this framework contact with the environment is explicitly disallowed. In order to manipulate an object a typical approach is to build a model of the object being grasped; this model is then used to ensure that the object does not collide with anything during a particular motion. In contrast, when working with deformable objects, interaction with the environment is common (and often required), with the deformable object complying to the environment rather than colliding with it. This raises the question "How accurate do our models need to be?" There are a broad range of deformable object manipulation tasks that robots have performed without highly accurate models ranging from surgical applications [3] to knot-tying [4]. While these methods have some success they rely on either a hand designed sequence of actions (or controllers), or a time-consuming data collection phase.

To address these limitations, this dissertation is motivated by two key research questions: (1) How can robots perform a broad range of tasks with deformable objects without high-fidelity models and simulation? and (2) How can robots take advantage of information learned while manipulating deformable objects? By answering these questions we can take a step towards a household robot that is capable of performing a broad range of tasks without any additional training, and can improve its ability

to perform the specific tasks that it commonly encounters.

Examples of deformable object manipulation range from domestic tasks like 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 involved; even approximating the configuration of a piece of cloth in 3D with a 4×4 grid results in a 48 degree of freedom configuration space. In addition, the dynamics of the deformable object are difficult to model [5]; even with high-fidelity modeling and simulation, planning for an individual task can take hours [6]. 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 "attraction basin" of the goal [7, 8]. We propose combining 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.

Two key ideas allow this framework to reliably perform tasks without a time-consuming modelling or data collection phase. First, we do not need to explicitly model and control every part of the deformable object, instead relying on the object's natural compliance in many situations. By doing so we drastically reduce the need for model fidelity, enabling the use of model approximations that do not need to be highly accurate. Second, our framework does not assume that the local controller or global planner are infallible. Instead, we assume that mistakes *will* be made, and implement learning algorithms designed to avoid making the same mistake again. To this end I formulate the controller's task as a Multi-Armed Bandit problem, enabling the controller to chose models based on the current circumstances. For planning we present a planning formulation that explicitly exposes the challenge of planning with model approximations, as well as a method for learning when we can and cannot rely on a model approximation during planning.

This dissertation makes seven contributions towards answering these research questions:

- We introduce a more accurate geometric model of how the direction of gripper motion and obstacles affect deformable objects.
- We specify a novel stretching avoidance constraint to prevent the object from

being overstretched by the robot as part of a local controller, allowing for the use of less accurate models without risking tearing the deformable object.

- We formulate the task of the local controller as a Multi-Armed Bandit problem, with each arm representing a model of the deformable object.
- We introduce a manipulation framework that interleaves planning and control, choosing each when most useful.
- We present a global motion planner to generate gross motion of the deformable object, and provide a 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 introduce a novel formulation of planning in reduced state spaces.
- We propose a method for improving the performance of the global planner as mistakes are made due to model approximations, enabling the planner to learn from experience.

CHAPTER II

Related Work

Robotic manipulation of deformable objects has been studied in many contexts ranging from surgery to industrial manipulation (see [9, 10, 11] for surveys). Below we discuss the most relevant methods to the work presented in this dissertation, starting with methods of modelling and simulating deformable objects. We then discuss visual servoing and other local control methods for performing deformable object manipulation tasks. Next we discuss related work for model selection and using multiple models for control. We then describe work relevant to combining local controllers and global planners for accomplishing tasks. We conclude with related work in motion planning for deformable objects and ways to consider topology in planning.

2.1 Modelling Deformable Objects

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 [12, 13] and cloth-like objects [14, 15]. The most common simulation methods use Mass-Spring models [16, 5], which are generally not accurate for large deformations [17], and Finite-Element (FEM) models [18, 19, 20]. 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 it's Young's modulus and friction parameters, which we do not assume are known.

Also, we seek a model that can be evaluated very quickly inside an optimal control framework, and Finite-element models, while accurate, can be computationally-expensive to simulate. While methods have been developed to track objects using FEM in real-time [21], a controller may need to evaluate the model many times to find an appropriate command, requiring speeds faster than real-time. Specialized models have also been developed, e.g., [22] and [23] focus on elastic rods that are not in contact. We seek a model that works well with rope-like and cloth-like materials that can deform as a result of contact. Finally, researchers have also investigated automatic modeling of deformable objects [24, 25]. However, these methods rely on a time-consuming training phase for each object to be modeled, which we would like to avoid.

Our work is complementary to methods that adapt the model of the object during manipulation [26, 27, 28]. Our model can serve as an initial guess and a reference for such methods so that the online adaptation process does not diverge too far from a reasonable model as a result of perception or modeling error. Our modelling methods build on the idea of *diminishing rigidity* Jacobians [7] by improving the model by considering the effects of the direction of motion and static obstacles that the deformable object interacts with.

2.2 Local Control for Manipulation Tasks

Given a model such as those above, researchers have investigated various control methods to manipulate deformable objects. 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 [29, 30, 31]. Other approaches [7, 26, 32, 27] bypass 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. [28] has enabled the use of Gaussian process regression while controlling a deformable object. Our work builds on Berenson [7], capturing overstretching and obstacle avoidance into control constraints that are more effective at preventing damage to the deformable object.

2.3 Using Multiple Models

In order to accomplish a given manipulation task, we need to determine which type of model to use at the current time to compute the next velocity command, as

well as how to set the model parameters. Frequently this selection is done manually, however, there are methods designed to make these determinations automatically. Machine learning techniques such as [33, 34] rely on supervised training data in order to intelligently search for the best regression or classification model. These methods are able to estimate the accuracy of each model as training data is processed, pruning models from the training that are unlikely to converge or otherwise outperform models that are kept. These methods are designed for large datasets rather than an online setting where we may not have any training data *a priori*. While it may be possible to adjust these methods to consider model utility instead of model accuracy, it is unclear how to acquire the needed training data for the task at hand without having already performed the task. The most directly applicable methods come from the Multi-Armed Bandit (MAB) literature [35, 36, 37]. In this framework there are multiple actions we can take, each of which provides us with some reward according to an unknown probability distribution. The problem then is to determine which action to take (which arm to pull) at each time step in order to maximize reward.

The MAB approach is well-studied for problems where the reward distributions are *stationary*; i.e. the distributions do not change over time [36, 38]. This is not the case for deformable object manipulation; consider the situation where the object is far away from the goal versus the object being at the goal. In the first case there is a possibility of an action moving the object closer to the goal and thus achieving a positive reward; however, in the second case any motion would, at best, give zero reward. In the *contextual bandits* [39, 40] variation of the MAB problem, additional contextual information or features are observed at each timestep, which can be used to determine which arm to pull. Typical solutions map the current features to estimates of the expected reward for each arm using regressions techniques or other metric-space analysis. In order to use contextual bandits for a given task a set of features would need to be engineered, however it is not clear what features to use.

Recent work [41] on non-stationary MAB problems offer promising results that utilize independent Kalman filters as the basis for the estimation of a non-stationary reward distribution for each arm. This algorithm (KF-MANB) provides a Bayesian estimate of the reward distribution at each timestep, assuming that the reward is normally distributed. KF-MANB then performs Thompson sampling [38] to select which arm to pull, choosing each in proportion to the belief that it is the optimal arm. We build on this approach in this paper to produce a method that also accounts for dependencies between arms by approximating the coupling between arms at each timestep.

For the tasks we address, the reward distributions are both non-stationary as well as *dependent*. Because all arms are operating on the same physical system, pulling one arm both gives us information about the distributions over other arms, as well as changing the future reward distributions of all arms. While work has been done on dependent bandits [42, 39], we are not aware of any work addressing the combination of non-stationary and dependent bandits using a regret-based formulation. Our method for model selection is inspired by KF-MANB, however we directly use coupling between models in order to form a joint reward distribution over all models. This enables a pull of a single arm to provide information about all arms, and thus we spend less time exploring the model space and more time exploiting useful models to perform the manipulation task.

2.4 Motion Planning for Deformable Objects

Motion planning for manipulation of deformable objects is an active area of research [10]. Saha et al. [43] present a Probabilistic Roadmap (PRM) [44] that plans for knot-tying tasks with rope. Rodriguez and Amato [45] study motion planning in fully deformable simulation environments. Their method, based on Rapidly-exploring Random Trees (RRTs) [46], applies forces directly to an object to move it through narrow spaces while using the simulator to compute the resulting deformations. Frank et al. [47] 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 [48, 49, 50, 51, 52, 53]. However, all the above 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.

Our planning method has some similarity to topological [54, 55] and tethered robot [56, 57] 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 *visiblity*

deformations from [54] 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 [58, 59, 60, 1]. 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. [53] bypass this challenge by analyzing completeness in the submanifold of quasi-static contact-free configurations of a 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. [1] present an efficient asymptotically-optimal planner which 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. [1], but we allow for the deformable object to be in contact with obstacles along a given trajectory.

2.5 Interleaving Planning and Control for Deformable Object Manipulation

The use of a local controller is not considered in the above methods, instead relying on a global planner (and thus implicitly the accuracy of the simulator) to generate a path that completes the entire task. In contrast, our framework combines the strengths of global planning with the strengths of local control in order to perform tasks.

Park et al. [61] considered interleaving planning and control for arm reaching tasks in rigid unknown environments. In their method, they assume 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.

Approaches based on learning from demonstration avoid planning and deformable

object modelling challenges entirely by using offline demonstrations to teach the robot specific manipulation tasks [4, 62]; 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 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.

2.6 Learning for Planning in Reduced State Spaces

In terms of applying machine learning to control and planning, prior work has primarily used learned dynamics models for control [63, 64, 65, 66, 67]. Recent work [68] has also explored planning in a learned reduced space, but they do not consider the error in a reduced model's prediction when planning. Visual Planning and Acting (VPA) [69] learns a latent transition model based on visual input for planning. This work uses on a classifier to prune infeasible transitions during planning. However, despite this classifier, only 15% of generated plans were visually plausible, with only 20% of the visually plausible plans being executable. When considering machine learning methods in this dissertation we do not focus on learning a reduction but rather on creating a framework that can be used to overcome limitations in a given model reduction.

CHAPTER III

Deformable Object Modelling

One of the key challenges in manipulating deformable objects is the inherent difficulty of modeling and simulating them. While there has been some progress towards online modeling of deformable objects [24, 25] these methods rely on a time-consuming training phase for each object to be modeled. This training phase typically consists of probing the deformable object with test forces in various configurations, and then fitting model parameters to the generated data. While this process can generate useful models, the time it takes to generate a model for each task can be prohibitive for some applications. Of particular interest are Jacobian-based models; in these models we assume that there is some function $F : SE(3)^G \rightarrow \mathbb{R}^N$ which maps a configuration of G robot grippers $q \in SE(3)^G$ to a parameterization of the deformable object $\mathcal{P} \in \mathbb{R}^N$, where N is the dimensionality of the parameterization of the deformable object. These models are then linearized by calculating the Jacobian of F:

$$\begin{aligned} \mathcal{P} &= F(q) \\ \frac{\partial \mathcal{P}}{\partial t} &= \frac{\partial F(q)}{\partial q}\frac{\partial q}{\partial t} \\ \dot{\mathcal{P}} &= J(q)\dot{q} \ . \end{aligned} \tag{3.1}$$

Computation of an exact Jacobian $J(q)$ at a given configuration q is often computationally intractable and requires high-fidelity models anbd simulators, so instead approximations are frequently used.

In this chapter we discuss a *diminishing-rigidity* based approximation first introduced by Berenson [7] and extensions of this model. The diminishing-rigidity model assumes that points on the deformable object that are near a gripper move "almost rigidly" with respect to the gripper while points that are further away move "less

rigidly". In addition to this Jacobian-based model, we also introduce a non-linear modification of the diminishing-rigidity Jacobian which more accurately captures the effect of the direction of gripper motion and obstacles.

3.1 Definitions

Let the robot be represented by a set of G grippers with configuration $q \in SE(3)^G$. We assume that the robot configuration can be measured exactly; in this work we assume the robot to be a set of free floating grippers; in practice we can track the motion of these with inverse kinematics on robot arms (see Sec 4.3.2.2 for an implementation). We use the Lie algebra [70] of $SE(3)$ to represent robot gripper velocities. This is the tangent space of $SE(3)$, denoted as $\mathfrak{se}(3)$. The velocity of a single gripper g is then $\dot{q}_g = \begin{bmatrix} v_g^T & \omega_g^T \end{bmatrix}^T \in \mathfrak{se}(3)$ where v_g and ω_g are the translational and rotational components of the gripper velocity. We define the velocity of the entire robot to be $\dot{q} = \begin{bmatrix} \dot{q}_1^T & \dots & \dot{q}_G^T \end{bmatrix}^T \in \mathfrak{se}(3)^G$. We define the inner product of two gripper velocities $\dot{q}_1, \dot{q}_2 \in \mathfrak{se}(3)$ to be

$$\langle \dot{q}_1, \dot{q}_2 \rangle = v^T v_2 + c\omega_1^T \omega_2 \ , \tag{3.2}$$

where $c > 0$ is scaling factor relating rotational and translational velocities. This defines the $\mathfrak{se}(3)$ norm

$$\|\dot{q}_g\|^2_{\mathfrak{se}(3)} = \langle \dot{q}_g, \dot{q}_g \rangle \ . \tag{3.3}$$

Let the configuration of a deformable object be a set of P points with configuration $\mathcal{P} = \begin{bmatrix} p_1^T & \dots & p_P^T \end{bmatrix}^T \in \mathbb{R}^{3P}$. We assume that we have a method of sensing $\mathcal{P}$. Let D be a symmetric $P \times P$ matrix where D_{ij} is the the geodesic distance (see Figure 3.1) between p_i and p_j when the deformable object is in its "natural" or "relaxed" state. To measure the norm of a deformable object velocity $\dot{\mathcal{P}} = \begin{bmatrix} \dot{p}_1^T & \dots & \dot{p}_P^T \end{bmatrix}^T \in \mathbb{R}^{3P}$ we will use a weighted Euclidean seminorm

$$\|\dot{\mathcal{P}}\|^2_W = \sum_{i=1}^{P} w_i \dot{p}_i{}^T \dot{p}_i = \dot{\mathcal{P}}^T \operatorname{diag}(W) \dot{\mathcal{P}} \tag{3.4}$$

where $W = \begin{bmatrix} w_1 & \dots & w_P \end{bmatrix}^T \in \mathbb{R}^P$ is a set of non-negative weights. The rest of the environment is denoted $\mathcal{O}$ and is assumed to be both static, and known exactly.

The current state of the deformable object is a function of the current gripper pose $\mathcal{P}$, the history of gripper motions that have been applied Q^{hist}, the object's initial

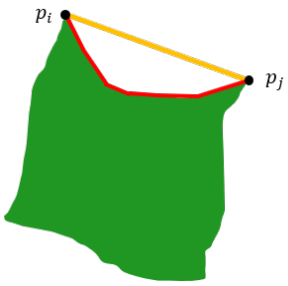


Figure 3.1: Euclidean distance measures length of the shortest path between p_i and p_j in $\mathbb{R}^3$ (gold). Geodesic distance measures the length of the shortest path, constrained to stay within the deformable object (red).

configuration $\mathcal{P}_0$, and the obstacles in the environment $\mathcal{O}$:

$$\mathcal{P} = F(q, Q^{\text{hist}}, \mathcal{P}_0, \mathcal{O}) \ . \tag{3.5}$$

Let a *deformation model* ϕ be defined as a function which takes as input the system configuration, gripper velocities, and obstacle configuration to a deformable object and returns a deformable object velocity:

$$\dot{\mathcal{P}} = \phi(q, \dot{q}, \mathcal{P}, \mathcal{O}) \ . \tag{3.6}$$

For brevity this will frequently be shortened to $\dot{\mathcal{P}} = \phi(\dot{q})$.

For Jacobian based models, we take the time derivative of Eq. (3.5) to get

$$\frac{d\mathcal{P}}{dt} = \frac{\partial F}{\partial q}\frac{\partial q}{\partial t} + \frac{\partial F}{\partial Q^{\text{hist}}}\frac{\partial Q^{\text{hist}}}{\partial t} + \frac{\partial F}{\partial \mathcal{P}_0}\frac{\partial \mathcal{P}_0}{\partial t} + \frac{\partial F}{\partial \mathcal{O}}\frac{\partial \mathcal{O}}{\partial t} \ . \tag{3.7}$$

Only the first term is non-zero, thus

$$\dot{\mathcal{P}} = \frac{\partial F(q, Q^{\text{hist}}, \mathcal{P}_0, \mathcal{O})}{\partial q}\dot{q} = J(q, \dot{q}, \mathcal{P}, \mathcal{O})\dot{q}. \tag{3.8}$$

Note that Q^{hist} and $\mathcal{P}_0$ are needed in F to compute the current state of the object, but if we can sense $\mathcal{P}$ directly (as we assume), then Q^{hist} and $\mathcal{P}_0$ are not needed to compute the Jacobian J. Thus for Jacobian based models Eq. (3.8) directly defines the deformation model ϕ

$$\phi(\dot{q}) \approx J(q, \dot{q}, \mathcal{P}, \mathcal{O})\dot{q} \ . \tag{3.9}$$

3.2 Diminishing Rigidity Jacobian

The key assumption used by this method [7] is *diminishing rigidity*: the closer a gripper is to a particular part of the deformable object, the more that part of the object moves in the same way that the gripper does (i.e. more "rigidly"). The further away a given point on the object is, the less rigidly it behaves; the less it moves when the gripper moves. In this section we refine Berenson's method by redefining $J^{\text{rot}}_{i,g}$ and introducing an extra parameter. This results in two parameters $k^{\text{trans}} \geq 0$ and $k^{\text{rot}} \geq 0$ which control how the translational and rotational rigidity scales with distance. Small values entail very rigid objects such as steel cable; high values entail very deformable objects such as fine string.

For every point i and every gripper g we construct a Jacobian $J^{\text{rigid}}(i, g)$ such that if p_i was rigidly attached to the gripper q_g then

$$\dot{p_i} = J^{\text{rigid}}_{i,g} \dot{q}_g = \begin{bmatrix} J^{\text{trans}}_{i,g} & J^{\text{rot}}_{i,g} \end{bmatrix} \dot{q}_g \ . \tag{3.10}$$

We then modify this Jacobian to account for the effects of *diminishing rigidity*. Let the indices of the set of points grasped by gripper g be $\mathcal{G}_g \subseteq \{1, \ldots, P\}$. Let $c(i, g)$ be the index of the point with minimal relaxed geodesic distance to p_i among the ones grasped by gripper g:

$$c(i, g) = \underset{j \in \mathcal{G}_g}{\text{argmin}}\, D_{ij} \ . \tag{3.11}$$

Then the translational rigidity of point i with respect to gripper g is defined as

$$w^{\text{trans}}_{i,g} = e^{-k^{\text{trans}} D_{ic(i,g)}} \tag{3.12}$$

and the rotational rigidity is defined as

$$w^{\text{rot}}_{i,g} = e^{-k^{\text{rot}} D_{ic(i,g)}}. \tag{3.13}$$

To construct an approximate Jacobian $\tilde{J}(i, g)$ for a single point and a single gripper we combine the rigid Jacobians with their respective rigidity values

$$\tilde{J}(i, g) = \begin{bmatrix} w^{\text{trans}}_{i,g} J^{\text{trans}}_{i,g} & w^{\text{rot}}_{i,g} J^{\text{rot}}_{i,g} \end{bmatrix} \ , \tag{3.14}$$

and then combine the results into a single matrix

$$\tilde{J}(q, \mathcal{P}) = \begin{bmatrix} \tilde{J}(1,1) & \tilde{J}(1,2) & \dots & \tilde{J}(1,G) \\ \tilde{J}(2,1) & \ddots & & \\ \vdots & & & \\ \tilde{J}(P,1) & & & \end{bmatrix} . \tag{3.15}$$

3.3 Constrained Directional Rigidity

While the diminishing rigidity Jacobian method has been used to do practical manipulation tasks with a deformable object [7], we observe that this rigidity does not only diminish as the distance from the gripper increases. Instead, it is a function of a larger set of variables derived from the configuration of the object. First, the rigidity also depends on the direction of gripper motion. Figure 3.2 shows an example of an object's *directional rigidity*. In addition capturing the effects of directional rigidity, in this section we seek to address contact with the environment, increasing the accuracy of the approximation.

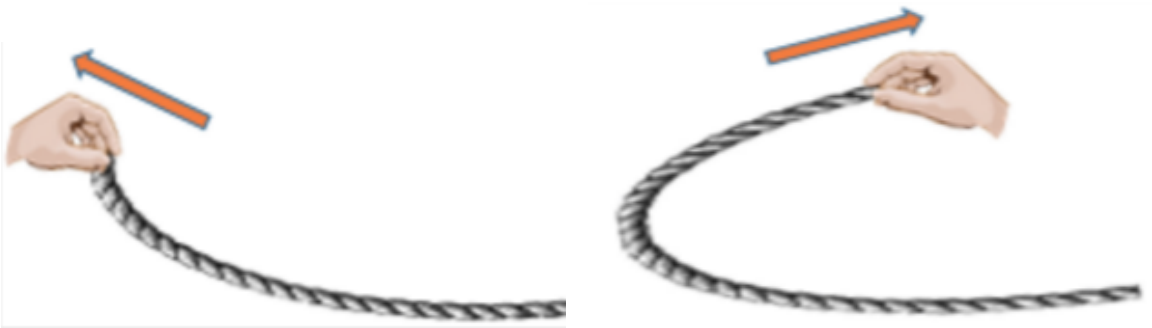


Figure 3.2: An illustrative example of directional rigidity. Left: The rope moves almost rigidly when dragging it by one end to the left. Right: The rope deforms when pulling it on the right in the opposite direction.

3.3.1 Model Overview

In Section 3.2, J is assumed to be independent of $\dot{q}$ and $\mathcal{O}$, yielding $\frac{\partial F}{\partial q} = J(q, Q^{\text{hist}}, \mathcal{P}_0) = J(q, \mathcal{P})$, which is analogous to a rigid-body Jacobian. While these assumptions allow a linear relationship between $\dot{q}$ and $\dot{\mathcal{P}}$, and thus computational convenience, they are not accurate in many situations (see Figure 3.2 for an example). In this section we augment the definition of J to include effects from $\dot{q}$ and

$\mathcal{O}$:

$$\dot{\mathcal{P}} = J(q, \dot{q}, \mathcal{P}, \mathcal{O})\dot{q} = \phi(q, \dot{q}, \mathcal{P}, \mathcal{O}) \ . \tag{3.16}$$

We now describe how J is approximated, focusing on how it accounts for directional rigidity (using $\dot{q}$) and how it enforces obstacle penetration constraints (using $\mathcal{O}$).

3.3.2 Directional Rigidity

We build on the idea proposed by Berenson [7], which approximates J based on the observation that the deformable object behaves rigidly near points grasped by the robot grippers. [7] encoded this effect through a simple function that only considered the distance of a point from the the nearest gripper. However, we find that we can exploit geometric information in the object's configuration to better predict the object's motion when we use a more complex model. We have observed that the key features of the deformable object configuration for predicting its motion are its deformability (which is determined by its material properties) and where it is slack. The deformation influences the transmission of the force from the grippers, i.e., the more stretchable the object, the more it will stretch when force is applied. However, when a region of the object is taut, regardless of how stretchable it is, it will move as if it were rigidly connected to a gripper (e.g. imagine a rope held taut by two grippers). We also must take into account that points are not influenced equally by different grippers; i.e., grippers farther away contribute less to the motion of a point than those closer to it.

To incorporate the above effects into our model, we define the following variables, which can be derived from $q, \dot{q}$, and $\mathcal{P}$:

- D_{ij}: the geodesic distance (a scalar) between points p_i and p_j on the surface of the object.
- v_{ij}: the vector starting at a point p_i and ending at the point p_j, as shown in Figure 3.3.
- q_g: the configuration of gripper g.
- $\dot{q}_g$: the velocity of gripper g.

Furthermore, let $c(i, g)$ be the index of the point with the minimal geodesic distance to p_i among the ones grasped by the g'th gripper. We address the notion of rigidity in object motion by considering the slackness of the object and reformulating

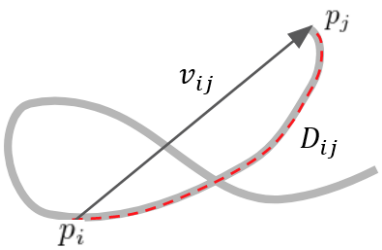


Figure 3.3: The length of the the red segment on the rope is the geodesic distance D_{ij}. v_{ij} is the vector showing the relative position of p_j with respect to p_i.

the rigidity as a function of $D_{ic(i,g)}$, $v_{ic(i,g)}$, and $\dot{q}_g$. For each point i and gripper g we compute

$$\begin{aligned} \tilde{J}(i,g) &= \theta_{i,g} \begin{bmatrix} w^{\text{trans}}_{i,g} J^{\text{trans}}_{i,g} & w^{\text{rot}}_{i,g} J^{\text{rot}}_{i,g} \end{bmatrix} \\ w^{\text{trans}}_{i,g} &= w^{\text{trans}}(D_{ic(i,g)}, v_{ic(i,g)}, \dot{q}_g) \\ w^{\text{rot}}_{i,g} &= w^{\text{rot}}(D_{ic(i,g)}) \end{aligned} \tag{3.17}$$

w^{trans} and w^{rot} are the corresponding translational and rotational diminishing rigidity factors defined by p_i and gripper g (discussed below).

Our goal is to encode the directional rigidity of the object motion into w^{trans} and w^{rot} and use $\theta_{i,g}$ to describe the *influence* of gripper g on p_i. Intuitively, w^{trans} should decrease with the increasing geodesic $D_{ic(i,g)}$ distance between p_i and $p_{c(i,g)}$. This is because the deformation of the region between p_i and $p_{c(i,g)}$ will attenuate the transmitted force of the gripper's motion unless the object is taut. Since the effects on $w^{\text{rot}}_{i,g}$ from $\dot{q}_g$ and $v_{ic(i,g)}$ are not as clear or significant as $D_{ic(i,g)}$, we keep $w^{\text{rot}}_{i,g}$ as a function of $D_{ic(i,g)}$, where

1. $w^{\text{rot}}_{i,g}$ ranges between 0 and 1.
2. $w^{\text{rot}}_{i,g}$ decreases as $D_{ic(i,g)}$ increases.

We give the definition of $w^{\text{rot}}_{i,g}$ below.

From observation, we find two key reasons related to the slackness of the object that induce the diminishing rigidity effect for translation motion, and we aim to encode these factors into $w^{\text{trans}}_{i,g}$. The first case is that the moving direction of $\dot{q}_g$ makes the region on the object between p_i and $p_{c(i,g)}$ less taut. The second case is that this region is already slack. $w^{\text{trans}}_{i,g}$ is thus a product of two terms:

$$w^{\text{trans}}_{i,g} = \alpha_{i,g}\beta_{i,g} \tag{3.18}$$

where $\alpha_{i,g}$ addresses the effect in the first case (motion reducing tension), and $\beta_{i,g}$ addresses the effect in the second case (object slackness). Both $\alpha_{i,g}$ and $\beta_{i,g}$ are functions of some of $q_g, \dot{q}_g, p_i$, or variables derived from these.

For p_i on the object, we find $\alpha_{i,g}$ is greatly impacted by $v_{ic(i,g)}$ and v_g. Decomposing v_g into v_g^{rad}, the component in the direction of $v_{ic(i,g)}$, and v_g^{perp}, the component perpendicular to $v_{ic(i,g)}$. We observed that if v_g^{rad} is in the opposite direction to $v_{ic(i,g)}$, then it is more likely to make the intervening region slacker and thus reduce the transmission of force from the gripper to p_i. Moreover, if v_g^{rad} and $v_{ic(i,g)}$ are in the same direction when the object is not already slack, p_i can move almost rigidly with $\dot{q}_g$. Figure 3.2 shows an example of the impact of this alignment. Based on these observations, we design the function $\alpha_{i,g} = \alpha(v_{ic(i,g)}, \dot{q}_g)$ with the following properties:

1. $\alpha(v_{ic(i,g)}, \dot{q}_g)$ ranges between 0 and 1.
2. $\alpha(v_{ic(i,g)}, \dot{q}_g) > \alpha(v_{jc(j,g)}, \dot{q}_g)$ **if** $\left\langle v_{ic(i,g)}, v_g \right\rangle > \left\langle v_{jc(j,g)}, v_g \right\rangle$ **and** $D_{ic(i,g)} = D_{jc(i,g)}$.
3. $\alpha(v_{ic(i,g)}, \dot{q}_g) > \alpha(v_{jc(j,g)}, \dot{q}_g)$ **if** $\left\langle v_{ic(i,g)}, v_g \right\rangle = \left\langle v_{jc(j,g)}, v_g \right\rangle$ **and** $D_{ic(i,g)} > D_{jc(i,g)}$.

We give the definition of $\alpha(v_{ic(i,g)}, \dot{q}_g)$ below.

As mentioned above, $\beta_{i,g}$ depends on the current slackness of the intervening region. Without other external forces applied on the object, the pulling force applied by the robot will tend to unwind or unfold the object eventually (we do not consider cases where the object is tied into knots). For this reason, the part of the intervening region on the object that is not already spread out is less likely to move rigidly with gripper g. One indicator that can address this property is the ratio between the Euclidean distance between p_i and $p_{c(i,g)}$, and the geodesic distance $D_{ic(i,g)}$ between them. We denote $\mathfrak{r}_{i,g} = \frac{||v_{ic(i,g)}||}{D_{ic(i,g)}}$ to be this ratio. A larger $\mathfrak{r}_{i,g}$ indicates a tauter intervening region. A tauter intervening region is more likely to result in $\dot{p}_i$ moving more rigidly. Thus we can design the function $\beta_{i,g} = \beta(\mathfrak{r}_{i,g})$ with the following properties:

1. $\beta(\mathfrak{r}_{i,g})$ ranges between 0 and 1.
2. $\beta(\mathfrak{r}_{i,g}) = 1$ if $\mathfrak{r}_{i,g} = 1$.
3. $\beta(\mathfrak{r}_{i,g}) > \beta(\mathfrak{r}_{j,g})$ if $\mathfrak{r}_{i,g} > \mathfrak{r}_{j,g}$

Finally, $\theta_{i,g}$, which captures the influence of gripper g on p_i should have the following property (where k is the index of a different gripper on the robot):

1. $\theta_{i,g}$ ranges between 0 and 1.

2. $\theta_{i,g} < \theta_{i,k}$ if $D_{ic(i,g)} > D_{ic(i,k)}$.

3. $\sum_{m=1}^{G} \theta_{i,m} = 1$.

Through experimentation, we obtained good results with the following functions:

$$\begin{aligned} \alpha(v_{ic(i,g)}, \dot{q}_g) &= e^{k^{\text{dir}} D_{ic(i,g)} \left(\cos \angle \left(v_{ic(i,g)}, v_g\right) - 1\right)} \\ \beta(\mathfrak{r}_{i,g}) &= \left(\frac{\|v_{ic(i,g)}\|}{D_{ic(i,g)}} \right)^{k^{\text{dist}}} \\ w_{i,g}^{\text{rot}} &= e^{-k^{\text{rot}} D_{ic(i,g)}} \\ \theta_{i,g} &= \frac{x_g}{\sum_{m=0}^{G} x_m} \\ x_m &= \frac{\min\{D_{ic(i,1)}, \ldots, D_{ic(i,G)}\}}{D_m} \end{aligned} \tag{3.19}$$

where k^{dir}, k^{dist}, and k^{rot} are non-negative parameters. Specifically, a larger k^{dir} indicates a greater impact on the diminishing in the rigidity from the motion reducing tension. A larger k^{dist} indicates a greater impact on the diminishing in the rigidity from the slackness of the object in the current state. A larger k^{rot} indicates a faster decrease in rotational rigidity as the distance from p_i to the gripper increases.

3.3.2.1 Obstacle Penetration Constraints

By combining the contributions of each individual gripper using the model developed above, we get a prediction of a point's movement from

$$\tilde{p}_i = \begin{bmatrix} J(i,1) & \ldots & J(i,G) \end{bmatrix} \dot{q} = J_i(q, \dot{q}, \mathcal{P}) \dot{q} \tag{3.20}$$

However, at this stage, we haven't taken into account the effect from the obstacles $\mathcal{O}$. Thus the predicted $\tilde{p}_i$ can move p_i into an obstacle.

When the prediction of p_i enters the obstacle, we project any penetration by the predicted $\tilde{p}_i$ into the tangent space of the obstacle surface (Figure 3.4). Let $d_i < \|\tilde{p}_i\|$ be the distance to collision in direction $\tilde{p}_i$ from point p_i; let $\lambda_i = \frac{d_i}{\|\tilde{p}_i\|}$; let n_i be the unit surface normal of the obstacle in contact; and let $N_i = (\mathbf{I}_{3\times 3} - \vec{n}_i \vec{n}_i^{+})$. Then to account for obstacles we compute

$$\tilde{J}_i(q, \dot{q}, \mathcal{P}, \mathcal{O}) = \begin{cases} (\lambda_i + (1 - \lambda_i) N_i) J_i(q, \dot{q}, \mathcal{P}) & \text{if } p_i + \tilde{p}_i \text{ in collision} \\ J_i(q, \dot{q}, \mathcal{P}) & \text{otherwise} \end{cases} \tag{3.21}$$

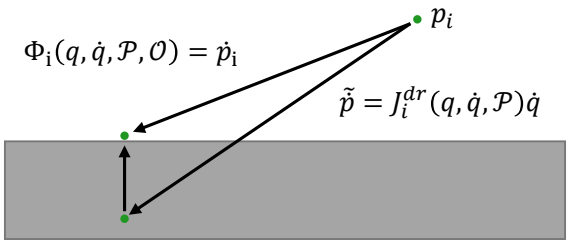


Figure 3.4: Projection process for points that are predicted to be in collision after movement.

To generate J for all the points and grippers we compute $J_i(q, \dot{q}, \mathcal{P})$ for each p_i. These matrices are modified using penetration constraints to get $J_i(q, \dot{q}, \mathcal{P}, \mathcal{O})$. These matrices are then stacked to obtain $J(q, \dot{q}, \mathcal{P}, \mathcal{O})$. Finally, we arrive at our approximate model: $\phi(q, \dot{q}, \mathcal{P}, \mathcal{O}) = J(q, \dot{q}, \mathcal{P}, \mathcal{O})\dot{q}$.

3.4 Results

Our goal for the constrained directional rigidity model is to improve the accuracy of the deformable object motion model (for use in the controller in Section 4.4), while maintaining reasonable computation speed. Our benchmark model is the diminishing rigidity model described in Section 3.2. To evaluate our method we perform experiments in simulation and on a physical robot. The simulator used is Bullet physics [71], however, we emphasize that our method has no knowledge of the simulation parameters or simulation methods used therein. The simulator is used as a "black-box," mainly to stand in for a perception system and to allow us to do repeatable experiments. The physical robot consists of two KUKA iiwa 7DoF arms with Robotiq 3-finger hands.

We ran experiments with scenarios involving both cloth and rope. The parameters we used for the benchmark method (Section 3.2) are its default best value found in [7]: $k^{\text{trans}} = k^{\text{rot}} = 10$ for rope and $k^{\text{trans}} = k^{\text{rot}} = 14$ for cloth. The parameters for the new model (Section 3.3) are set as $k^{\text{dir}} = 4$, $k^{\text{dist}} = 10$, $k^{\text{rot}} = 20$ for the new model. All experiments were run on a i7-8700K 3.7 GHz CPU with 32 GB of RAM. A video showing the experiments can be found at `https://www.youtube.com/watch?v=Y-wPsPdQVgg`.

3.4.1 Simulation Environment Model Accuracy Results

We evaluated model accuracy by pulling the rope in a straight line along the direction of the rope, then turning the gripper and pulling back towards the rope as shown in Figure 3.2. As shown in Figure 3.5, our new model is a better approximation of the true motion when the gripper is pulling the rope. When the gripper is turning, both the baseline and the new model produce comparable error, but when the gripper starts pulling again (this time in the opposite direction), the new model is a significantly better approximation.

We also evaluated model accuracy by pulling the cloth in a similar fashion; pulling the cloth one way, turning the grippers, and then pulling in the opposite direction. As shown in Figure 3.6, our new model is a better approximation of the true motion when the grippers are pulling the cloth. As in the rope test, when rotating the grippers both models produce comparable error. While the cloth is folded on itself both models produce noisy results, but when the cloth lies flat again, the new model achieves lower error.

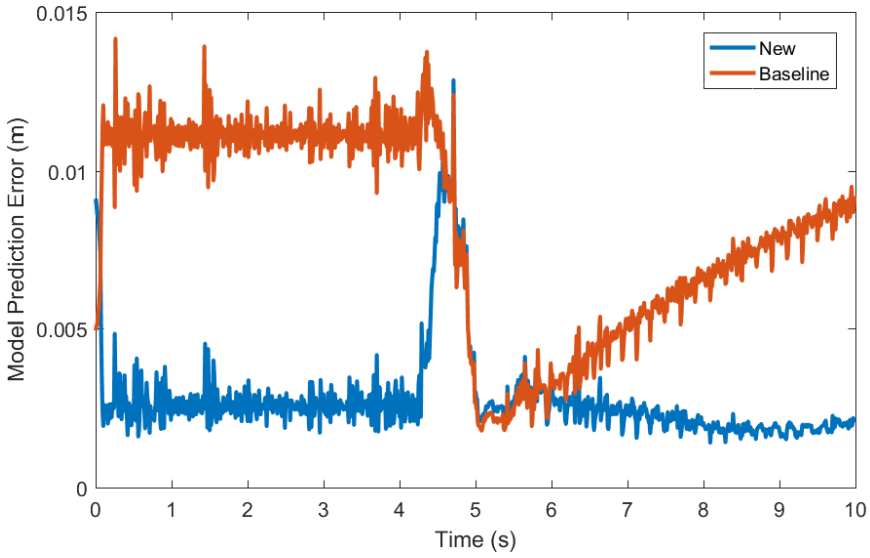


Figure 3.5: RMS model prediction error for the simulated rope model accuracy test. The gripper pulls the rope for the first 4.5 seconds, then turns for half a second, then moves in the opposite direction at the 5 second mark.

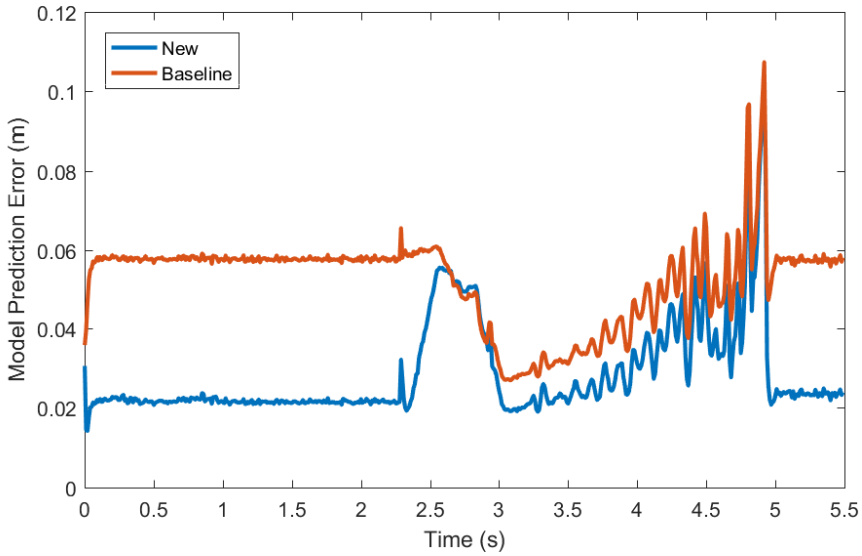


Figure 3.6: RMS model prediction error for the simulated cloth model accuracy test. The grippers pull the cloth for the first 2.3 seconds, then turn for 0.63 seconds, then move in the opposite direction at the 2.93 second mark. At the 5 second mark the cloth is no longer folded.

3.4.2 Physical Robot Experiments

To evaluate our new model on a physical system, we set up an experiment with a cloth-like object manipulated by two 7DoF KUKA iiwa arms (Figure 3.7). To sense the position of the cloth, we use the AprilTags [72] and IAI Kinect2 [73] libraries. We use the same parameters as we used for simulated experiments. This test, which evaluates model accuracy, uses a motion profile similar to the simulation accuracy tests (Figure 3.8). Similar to the simulation results, the new model improves performance when dragging the cloth (first and last sections of Figure 3.8), and is comparable during rotational motion and when the cloth is resting on edge perpendicular to the table (see video).

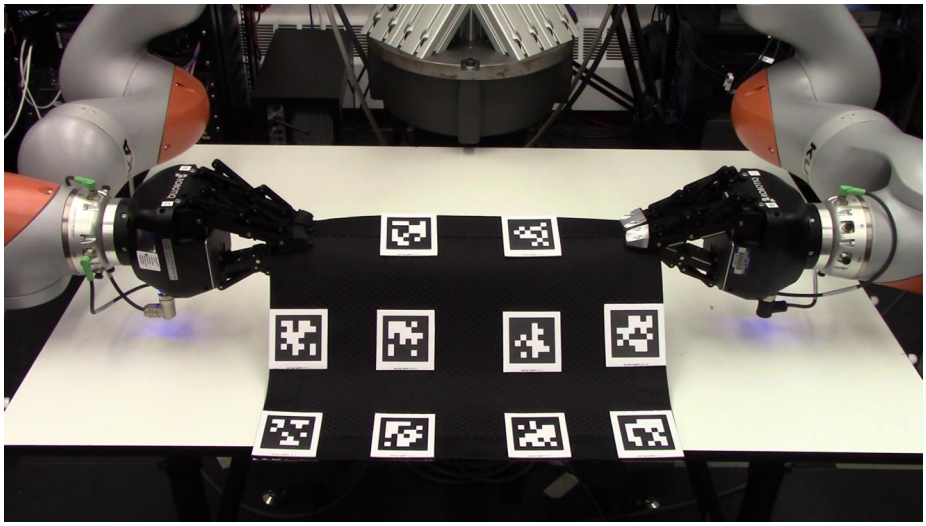


Figure 3.7: Initial setup for the physical robot model accuracy experiment.

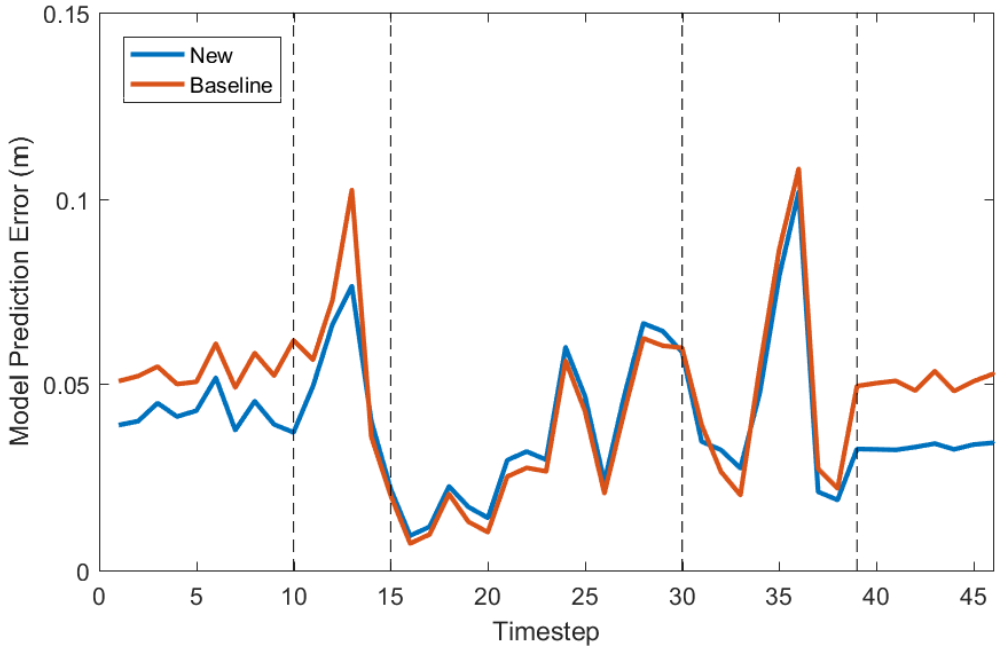
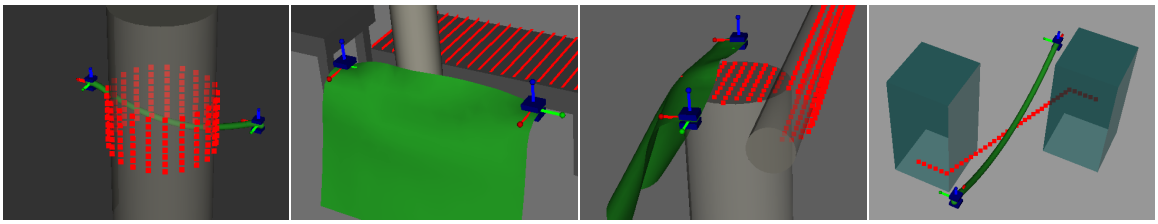


Figure 3.8: RMS model prediction error for the physical cloth accuracy test. The grippers pull the cloth toward the robot for the first 10 timesteps, upward for 5 timesteps, rotate for 15 timesteps, diagonally down and away for 9 timesteps, then directly away from the robot.

3.4.3 Computation Time

To verify the practicality of our method, we gathered data comparing its computation time to the benchmark's and to using the Bullet simulator for a variety of tasks (see Figure 3.9 and Section 4.5). Table 3.1 shows a comparison between the average time needed to evaluate the new model and the time needed to 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 to the time needed to evaluate our model, we see that the new model is indeed faster by at least an order of magnitude, in some cases by two orders of magnitude, confirming that, despite being slower than the diminishing rigidity model, our method still outperforms the simulator in terms of computation time. This is particularly important given the average number of times a model is evaluated in a control loop.



(a) (b) (c) (d)

Figure 3.9: Initial state of the four experiments, where the red points act as attractors for the deformable object. (a) Rope wrapping cylinder. (b) Cloth passing single pole. (c) Cloth covering two cylinders. (d) Rope matching zig-path

Table 3.1: Top two rows: Mean computation time (ms) per model prediction for a given gripper motion. BT: Bullet simulator; CDR: constrained directional rigidity. Bottom row: Mean number of times the model was evaluated when executing the controller in Section 4.4.

	rope-wrapping -cylinder	rope-matching -cylinder	cloth-passing -single-pole	cloth-wrapping -two-cylinder
BT	0.686	0.571	19.29	3.680
CDR	0.029	0.014	1.172	0.339
# evals	50.72	143.5	83.81	63.32

CHAPTER IV

Local Control

The previous chapter presented multiple models that approximate the effects of gripper motion on the deformable object. Next we introduce controllers that use these models as part of our framework for performing a broad range of tasks.

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 [7] and [8]. These controllers locally minimize error while avoiding robot collision and excessive stretching of the deformable object. We present two different methods for addressing overstretch in sections 4.3 and 4.4. Both of these controllers rely on the same method for computing the direction to move the deformable object in order to reduce task error.

4.1 Problem Statement

We define a task based on a set of T target points $\mathcal{T} \in \mathbb{R}^{3T}$, a function $\rho(\mathcal{T}, \mathcal{P}) \geq 0$, which measures the alignment error between $\mathcal{P}$ and $\mathcal{T}$, and a termination function $\Omega(\mathcal{T}, \mathcal{P})$ which indicates if the task is finished. The methods we present in this chapter are local, i.e. at each time t they choose an incremental movement $\dot{q}_t$ which reduces the alignment error as much as possible at time $t + 1$:

$$\min_{\dot{q}_t} \rho\left(\mathcal{T}, \mathcal{P}_{t+1}\right) \tag{4.1}$$

where $\mathcal{P}_{t+1}$ is the result of executing $\dot{q}_t$ for one unit of time. $\dot{q}_t$ must also be feasible, i.e. it should not bring the grippers into collision with obstacles and should not cause the object to stretch excessively.

4.2 Reducing Task Error

We build on previous work [7], splitting the desired deformable object movement into two parts: an error correction part and a stretching correction part. When defining the direction we want to move the deformable object to minimize error we calculate two values: which direction to move the deformable object points $\dot{\mathcal{P}}_e$ and the importance of moving each deformable object point W_e. This is analogous to computing the gradient of error, as well as an "importance factor" for each part of the gradient. We need these weights to be able to differentiate between points of the object where the error function is a plateau versus points where the error function is at a local minimum (Figure 4.1). Typically this is achieved using a Hessian, however our error function does not have a second derivative at many points.

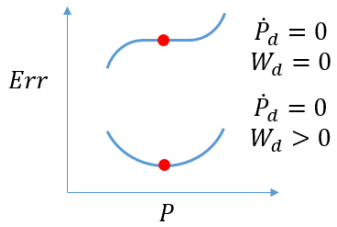


Figure 4.1: Top Line: moving the point does not change the error, thus the desired movement is zero, however, it is not important to achieve zero movement, thus $W_d = 0$. Bottom Line: error is at a local minimum; thus moving the point increases error.

In order to calculate $\dot{\mathcal{P}}_e$ and W_e, we start by defining a workspace navigation function for each target point $\mathcal{T}_k \in \mathcal{T}$ towards $\mathcal{T}_k$ 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 travelled along that path. Using these distances, at every timestep for every target point $\mathcal{T}_k$, we recalculate which point on the deformable object p_i is closest (Alg. 1). The directions each navigation function indicates are added together to define the overall direction to manipulate a point (Alg. 2 line 5). For the importance factors $W_{e,i}$, we take only the largest distance that p_i would have to move as a way to mitigate discretization effects (Alg. 2 line 6).

4.3 Stretching Avoidance Controller

An outline of how this controller functions is shown in Alg. 3; first, we calculate the error reduction direction and weight as discussed in the previous section (Lines 2 and 3). These error reduction terms are then combined with stretching avoidance terms $\dot{\mathcal{P}}_s, W_s$ to define the desired manipulation direction and importance weights

Algorithm 1 CalculateCorrespondences($\mathcal{P}, \mathcal{T}$)

1: $\mathcal{P}_C = [\emptyset]_{1\times P}$
2: **for** $k \in \{1, 2, \ldots, T\}$ **do**
3: $\quad i \leftarrow \text{argmin}_{j\in\{1,2,\ldots,P\}}\, d_{\text{Dijkstras}}(\mathcal{T}_k, p_j)$
4: $\quad d \leftarrow d_{\text{Dijkstras}}(\mathcal{T}_k, p_i)$
5: $\quad \mathcal{P}_C[i] \leftarrow \{\mathcal{P}_C[i] \cup (k, d)\}$
6: **end for**
7: **return** $\mathcal{P}_C$

Algorithm 2 FollowNavigationFunction($\mathcal{P}, \mathcal{P}_C$)

1: $\dot{\mathcal{P}}_e \leftarrow \mathbf{0}_{3P\times 1}$
2: $W_e \leftarrow \mathbf{0}_{P\times 1}$
3: **for** $i \in \{1, 2, \ldots, P\}$ **do**
4: $\quad$ **for** $(k, d) \in \mathcal{P}_C[i]$ **do**
5: $\quad\quad \dot{p}_{e,i} \leftarrow \dot{p}_{e,i}+$ DijkstrasNextStep(p_i, k)
6: $\quad\quad W_{e,i} \leftarrow \max(W_{e,i}, d)$
7: $\quad$ **end for**
8: **end for**
9: **return** $\dot{\mathcal{P}}_e, W_e$

$\dot{\mathcal{P}}_d, W_d$ at each timestep (Lines 3 and 3). We then find the best robot motion to achieve the desired deformable object motion, while preventing collision between the robot and obstacles (Line 5).

Algorithm 3 StretchingAvoidanceController($q, \mathcal{P}, \mathcal{T}$)

1: $\mathcal{P}_C \leftarrow$ CalculateCorrespondences$(\mathcal{P}, \mathcal{T})$
2: $\dot{\mathcal{P}}_e, W_e \leftarrow$ FollowNavigationFunction$(\mathcal{P}, \mathcal{P}_C)$
3: $\dot{\mathcal{P}}_s, W_s \leftarrow$ StretchingCorrection$(\mathcal{P})$
4: $\dot{\mathcal{P}}_d, W_d \leftarrow$ CombineTerms$(\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s)$
5: $\dot{q}^{\text{cmd}} \leftarrow$ FindBestRobotMotion$(q, \mathcal{P}, \dot{\mathcal{P}}_d, W_d)$

4.3.1 Stretching Correction

Our algorithm for stretching correction is similar to that found in [7], with the addition of a weighting term k_s, and a change in how we combine error correction and stretching correction. We use the StretchingCorrection() function (Alg. 4) 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 to D which contains the relaxed lengths between every pair of points. If any two neighbouring points are stretched by more than a factor

of $\gamma^{\max}$, 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 (Alg. 5) we prioritize stretching correction, accepting only the portion of the error correction that is orthogonal to the stretching correction term for each point. k_s is used to define the relative scale of the importance factors W_e and W_s

Algorithm 4 StretchingCorrection($\mathcal{P}$)

$E \leftarrow \text{EuclidianDistanceMatrix}(\mathcal{P})$
$\dot{\mathcal{P}}_s \leftarrow \mathbf{0}_{3P\times 1}$
$W_s \leftarrow \mathbf{0}_{P\times 1}$
for $i \in \{1, 2, \ldots, P\}$ **do**
    for $j \in \text{Neighbours}(i)$ **do**
        if $i < j$ **and** $E_{ij} > \gamma^{\max} D_{ij}$ **then**
            $\Delta_{ij} \leftarrow E_{ij} - D_{ij}$
            $v \leftarrow \Delta_{ij}(p_j - p_i)$
            $\dot{p}_{s,i} \leftarrow \dot{p}_{s,i} + \frac{1}{2}v$
            $\dot{p}_{s,j} \leftarrow \dot{p}_{s,j} - \frac{1}{2}v$
            $W_{s,i} \leftarrow \max(W_{s,i}, \Delta_{ij})$
            $W_{s,j} \leftarrow \max(W_{s,j}, \Delta_{ij})$
        end if
    end for
end for
return $\dot{\mathcal{P}}_s, W_s$

Algorithm 5 CombineTerms($\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s$)

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

4.3.2 Finding the Best Robot Motion and Avoiding Collisions

Given a desired deformable object velocity $\dot{\mathcal{P}}_d$ and relative importance weights W_d, we want to find the robot motion that best achieves $(\dot{\mathcal{P}}_d, W_d)$. I.e.

$$\begin{aligned} \underset{\dot{q}}{\text{argmin}} \quad & \|\phi(q, \dot{q}, \mathcal{P}, \mathcal{O}) - \dot{\mathcal{P}}_d\|_{W_d} \\ \text{subject to} \quad & \|\dot{q}\| \leq \dot{q}^{\max} \\ & (q + \dot{q}) \in \mathcal{Q}^{\text{valid}} \quad . \end{aligned} \tag{4.2}$$

In general, $\phi(\dots)$ is not known. For our stretching avoidance controller we use a Jacobian based approximation (Chapter. III):

$$\phi(q, \dot{q}, \mathcal{P}, \mathcal{O}) \approx J(q, \dot{q}, \mathcal{P}, \mathcal{O})\dot{q} \tag{4.3}$$

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

4.3.2.1 Simulated experiments:

For the simulated experiments, we first solve Eq. (4.2) using our Jacobian approximation while neglecting the collision constraints:

$$\begin{aligned} \psi_{\mathfrak{se}(3)}(\dot{\mathcal{P}}, W) = \underset{\dot{q}}{\text{argmin}} \quad & \|J\dot{q} - \dot{\mathcal{P}}\|_W \\ \text{subject to} \quad & \|\dot{q}_g\|_{\mathfrak{se}(3)} \leq \dot{q}^{\max,\text{s}}_{\mathfrak{se}(3)} \quad , \quad g = 1, \dots, G \end{aligned} \tag{4.4}$$

where $\dot{q}^{\max,\text{s}}_{\mathfrak{se}(3)}$ is the maximum velocity for each individual gripper.

In order to guarantee that the grippers do not collide with any obstacles, we use the same strategy from [7], smoothly switching between collision avoidance and other objectives (see Alg. 7). 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 Alg. 8. We then project the servoing motion from Eq. (4.4) into the null space of the avoidance motion using the null space projector $\left(\mathbf{I} - J^+_{p^g} J_{p^g}\right)$. $\beta > 0$ sets the rate at which we change between servoing and collision avoidance objectives. $\dot{q}^{\max,\text{c}}_{\mathfrak{se}(3)} > 0$ is an internal parameter that sets how quickly we move the robot away from obstacles.

Algorithm 6 FindBestRobotMotionSim$(q, \mathcal{P}, \dot{\mathcal{P}}_d, W_d)$

$\dot{q}_s \leftarrow \psi_{\mathfrak{se}(3)}(\dot{\mathcal{P}}_d, W_d)$ Eq. (4.4)
$\dot{q}^{\text{cmd}} \leftarrow \text{ObstacleRepulsion}(\dot{q}_s, \mathcal{O})$
return $\dot{q}^{\text{cmd}}$

Algorithm 7 ObstacleRepulsion$(\dot{q}_s, \mathcal{O})$

for $g \in \{1, \ldots, G\}$ **do**
    $J_{p^g}, \dot{x}_{p^g}, d_g \leftarrow \text{Proximity}(q_g, \mathcal{O})$
    $\lambda \leftarrow e^{-\beta d_g}$
    $v \leftarrow J_{p^g}^{+} \dot{x}_{p^g}$
    $\dot{q}_{g,c} \leftarrow \dot{q}_{\mathfrak{se}(3)}^{\max,\text{c}} \frac{v}{\|v\|}$
    $\dot{q}_g \leftarrow \lambda \left(\dot{q}_{g,c} + \left(\mathbf{I} - J_{p^g}^{+} J_{p^g} \right) \dot{q}_{g,s} \right) + (1 - \lambda) \dot{q}_{g,s}$
end for
return $\dot{q}$

Algorithm 8 Proximity$(q_g, \mathcal{O})$

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

4.3.2.2 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 (Alg. 9). 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 a function equivalent to Proximity() for collision checking for the points in $\mathcal{C}$ in order to maintain a minimum distance from collision:

$$\begin{aligned} \psi_{\mathbb{R}^N}(\dot{\mathcal{P}}, W) = \underset{\dot{q}}{\text{argmin}} \quad & \|J_r J\dot{q} - \dot{\mathcal{P}}\|_W^2 \\ \text{subject to} \quad & q + \dot{q} \in \mathcal{Q}^{\text{valid}} \\ & \|\dot{q}\| \leq \dot{q}_{\mathbb{R}^N}^{\max} \\ & \|J_{r,g}\dot{q}\| \leq \dot{q}_{\mathfrak{se}(3)}^{\max,\text{s}} \quad , \quad g = 1, \dots, G \\ & \dot{x}_{c_i}^T J_{c_i}\dot{q} \leq d_{c_i} + d_{\text{buffer}} \quad , \quad i = 1, \dots, |\mathcal{C}| \ . \end{aligned} \tag{4.5}$$

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

$$\|\dot{q}\| \leq \dot{q}_{\mathbb{R}^N}^{\max}$$

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

$$\|J_{r,g}\dot{q}\| \leq \dot{q}_{\mathfrak{se}(3)}^{\max,\text{s}}$$

where J_r is the Jacobian between robot motion and end effector motion for gripper all grippers, and $J_{r,g}$ is the Jacobian for gripper g.

Algorithm 9 FindBestRobotMotionPhys$(q, \mathcal{P}, \dot{\mathcal{P}}_d, W_d)$

1: **for** $g \in \{1, 2, \dots, |\mathcal{C}|\}$ **do**
2: $\quad J_{p^g}, \dot{x}_{p^g}, d_g \leftarrow$ Proximity$(\mathcal{O}, g)$
3: **end for**
4: $\dot{q}^{\text{cmd}} \leftarrow \psi_{\mathbb{R}^N}(\dot{\mathcal{P}}_d, W_d)$ Eq. (4.5)

To solve Equations (4.4) and (4.5) we use the Gurobi optimizer [74].

4.4 Stretching Constraint Controller

While the controller in the previous section has had some success at preventing excessive stretching [7], it is not able to prevent stretching when the grippers move on opposite sides of an obstacle (see video at `https://www.youtube.com/watch?v=Y-wPsPdQVgg`). To address this, we introduce a novel geometric constraint which is able to directly address the cause of overstretch. This constraint is included directly in the optimization problem solved at each time resulting in a straightforward control algorithm (Alg. 10).

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

1: $\mathcal{P}_C \leftarrow$ CalculateCorrespondences$(\mathcal{P}, \mathcal{T})$
2: $\dot{\mathcal{P}}_e, W_e \leftarrow$ FollowNavigationFunction$(\mathcal{P}, \mathcal{P}_C)$
3: $\dot{q}^{\text{cmd}} \leftarrow$ FindBestConstrainedRobotMotion$(q, \mathcal{P}, \dot{\mathcal{P}}_e, W_e)$

4.4.1 Overstretch

The stretching avoidance of the deformable object is difficult to formulate due to the compliant and underactuated nature of deformable objects. In the previous section, a stretching correction term $\dot{\mathcal{P}}_s \in \mathbb{R}^{3P}$ is applied when the object becomes overstretched. However, this method cannot handle cases where the object is caught on an obstacle.

We detect the overstretching (i.e. excessive strain) of the object by examining the value of the stretching ratio γ, which denotes the maximum value among the ratio between the Euclidean distance $\|v_{ij}\|$ and the geodesic distance D_{ij} for every pair of points p_i and p_j:

$$\gamma = \max_{\substack{i,j \in 1,\ldots,P \\ j>i}} \frac{\|v_{ij}\|}{D_{ij}} \quad . \tag{4.6}$$

Denote $\gamma^{\max}$ as the maximum allowed stretching ratio; this controller initiates stretching-avoidance when $\gamma > \gamma^{\max}$.

We assume that the object starts in an unstretched state, so the overstretch that arises is due to the motion of the grippers. Thus if we can constrain gripper motions to a set which does not overstretch the object further than a threshold, we can prevent or reduce the overstretch at the next time step. We know that the force causing the overstretch comes from the grippers, so if we reduce the length of geodesic paths through the object between grippers, the strain on the object should decrease. When

overstretch is detected, we thus introduce a conical constraint for each gripper that shrinks the allowable $\dot{q}_g$ to reduce the length of the geodesics between the grippers.

A conical constraint is constructed for each gripper and points along the *stretching avoidance vector*, which is an estimation of the direction to move to decrease the strain. For a pair of grippers with index g and k, two stretching avoidance vectors are defined, one for each gripper. Let $\mathcal{I}_g(q_g, q_k)$ be the index of the point grasped by the gth gripper, which has the minimum geodesic distance to the set of points grasped by q_k. We define $\mathcal{I}_k(q_g, q_k)$ similarly. Let $\int_{gk}$ be the geodesic on the object from $p_{\mathcal{I}_g(q_g,q_k)}$ to $p_{\mathcal{I}_k(q_g,q_k)}$. We denote u_g^k and u_k^g as the pair of stretching avoidance vectors on grippers g and k respectively. Then u_g^k is the tangent vector of $\int_{gk}$ at $p_{\mathcal{I}_g(q_g,q_k)}$ and u_g^k is the tangent vector of $\int_{kg}$ at the point $p_{\mathcal{I}_k(q_g,q_k)}$ (as shown in Figure 4.2).

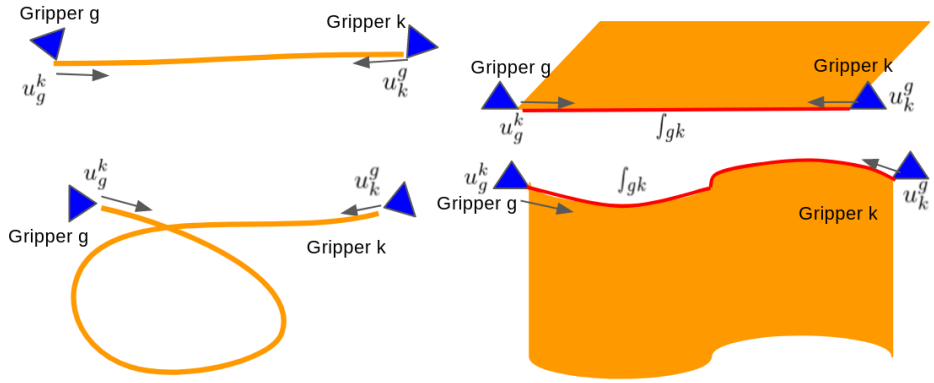


Figure 4.2: The arrows in gray show the direction of each stretching vector at the corresponding gripper with respect to the gripper pair q_g and q_k. Left: stretching vectors on the rope when the rope is at rest (above) or is deformed (below). Right: stretching vectors on the cloth when the cloth is at rest (above) or is deformed (below). The red lines denote the geodesic connecting the corresponding $p_{\mathcal{I}_g(q_g,q_k)}$ and $p_{\mathcal{I}_k(q_g,q_k)}$ on the object.

To specify the stretching constraint, we first define the function $s(\dot{q}_g, q_g, q_k, \mathcal{P})$, which specifies the constraint on gripper g defined by the interaction of grippers g and k. Correspondingly, u_g^k is the stretching avoidance vector for gripper g, which is the tangent vector of $\int_{gk}$ at $p_{\mathcal{I}_g(q_g,q_k)}$. The larger the value of s, the more we expect geodesic path length between grippers will be reduced. Thus, s should increase as $\angle(\dot{q}_g, u_g^k)$ increases. Assume we wish to have a lower bound s_s on s, then C_s is a set of constraints $C_s = \{C_s^1, \ldots, C_s^G\}$, where each constraint is:

$$C_s^g = \{\dot{q}_g \in \mathfrak{se}(3) \mid \forall k \neq g,\ s(\dot{q}_g, q_g, q_k, \mathcal{P}) \geq s_s\} \tag{4.7}$$

Many functions can satisfy the requirements of s. In our work, we specify the function

as

$$s(\dot{q}_g, q_g, q_k, \mathcal{P}) = \cos\angle(\dot{q}_g, u_g^k) = \cos\angle(v_g, u_g^k) \ . \tag{4.8}$$

4.4.2 Collision

Collision avoidance for the robot is addressed by the constraint C_c, which is the set of motions that keeps the grippers away from obstacles:

$$C_c = \left\{ \dot{q}_g = \begin{bmatrix} v_g^T & \omega_g^T \end{bmatrix}^T \in \mathfrak{se}(3) \mid d_{\text{buffer}} - l(q_g) - \frac{\mathfrak{n}(q_g)^T v_g}{\|v_g\|} v_g \Delta t < 0 \right\} \tag{4.9}$$

where $l(q_g)$ is the function returning the distance from the gripper to its closest obstacle. $\mathfrak{n}(q_g)$ returns the unit surface normal of the obstacle closest to the gth gripper. The idea is to make each gripper keep at least the safe distance away from the closest obstacle. While we consider free-flying grippers in this section, similar constraints can be imposed on the entire geometry of a robot arm to avoid collisions all along the arm as was done in Section 4.3.2.2.

4.4.3 Optimization Method

Given these constraints, we then formulate an optimization problem similar to Eq. (4.4), replacing the approximate Jacobian model with the directional rigidity model (Section 3.3), and adding the new stretching and collision constraints.

$$\begin{aligned} \psi(\dot{\mathcal{P}}, W) = \operatorname{argmin} \dot{q} \quad & \|\phi(q, \dot{q}, \mathcal{P}, \mathcal{O}) - \dot{\mathcal{P}}_e\|_{W_e} \\ \text{subject to} \quad & \dot{q} \in C_s \\ & \dot{q} \in C_c \\ & \|\dot{q}_g\|_{\mathfrak{se}(3)} \leq \dot{q}_{\mathfrak{se}(3)}^{\max,\text{s}} \ , \quad g = 1, \ldots, G \end{aligned} \tag{4.10}$$

Because our objective function is not necessarily convex, we used a custom optimization method to solve the problem specified in Eq. 4.10. Our method is a type of numerical gradient descent with an additional projection step to enforce constraints. This addresses the constraints, but we are still using a local optimization method to solve a non-convex problem. In practice this has been a significant limitation for our experiments.

Our method's outer loop computes the numerical gradient of the objective function. An inner loop then performs backtracking line search to find the gradient step size. However, the gradient step may cross a constraint boundary, thus after we compute the step size, we check if any constraint has been violated after taking the

step. If it has, we project the step back to the feasible space. A simple projection to the boundary of a violated constraint may satisfy that constraint but violate others. Instead, to perform the projection, we solve a convex optimization problem (using the Gurobi optimizer [74]) to find the nearest feasible point. This is possible because all the constraints in our problem are convex. Once such a point is found, the outer loop continues to iterate until convergence.

4.5 Results

Our goal for the stretching constraint controller is formulating a set of constraints for the controller to mitigate collision and excessive stretching issues. As mentioned in previous sections, our benchmark controller is based on [7] and described in Section 4.3, using the diminishing rigidity model from Section 3.2. To evaluate our method we perform experiments in simulation and on a physical robot. The simulator used is Bullet physics [71], however, we emphasize that our method has no knowledge of the simulation parameters or simulation methods used therein. The simulator is used as a "black-box," mainly to stand in for a perception system and to allow us to do repeatable experiments. The physical robot consists of two KUKA iiwa 7DoF arms with Robotiq 3-finger hands.

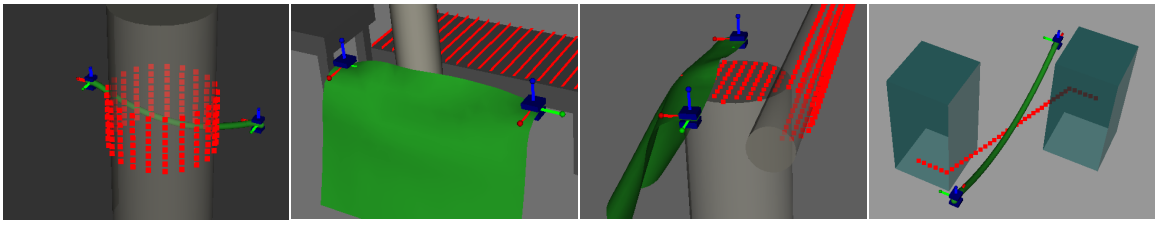
We ran experiments with scenarios involving both cloth and rope, using the same model parameters as Section 3.4. For the both controllers we set $c = 0.0025, \dot{q}^{\max,\mathrm{s}}_{\mathfrak{se}(3)} = 0.2, \gamma^{\max} = 1.1$ for rope, and $\gamma^{\max} = 1.67$ for cloth. For the benchmark controller we additionally set $\dot{q}^{\max,\mathrm{c}}_{\mathfrak{se}(3)} = 0.2, \beta = 200$ for rope, $\beta = 1000$ for cloth. For the stretching constraint controller we set $l_c = 0.023$, and $s_s = 0.4$. All experiments were run on a i7-8700K 3.7 GHz CPU with 32 GB of RAM. A video showing the experiments can be found at `https://www.youtube.com/watch?v=Y-wPsPdQVgg`.

4.5.1 Constraint Enforcement

Since the benchmark controller can already handle the collision constraint very well, and the new controller addresses the collision constraint in the similar way as the benchmark, there is not a significant difference in how the collision constraint is enforced. However, the stretching constraint shows a very clear improvement.

The metrics of stretching avoidance is the stretching ratio γ defined in Section 4.4.1. A controller with good stretching avoidance should prevent γ from increasing beyond a certain threshold.

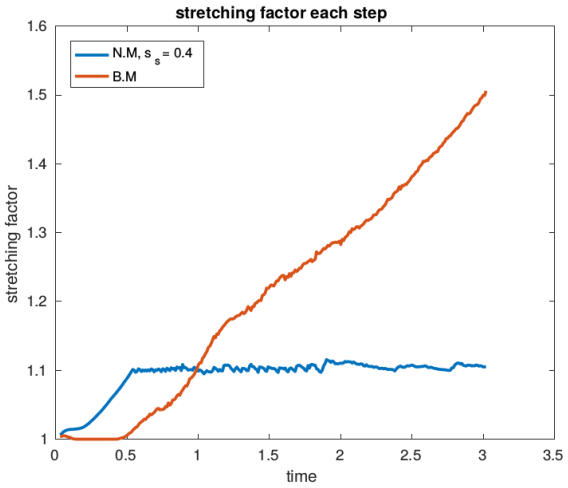
The two experiments we used for the stretching avoidance test are the rope-wrapping-cylinder and the cloth-passing-single-pole, shown in Figure 4.3 (a-b). We ran each controller separately for a fixed amount of time for each task and show γ vs. time for both controllers in Figure 4.4. In both these two setups, the desired object motion $\dot{\mathcal{P}}_e$ generated by the Dijkstra field will tear the object unless overstretching is prevented.



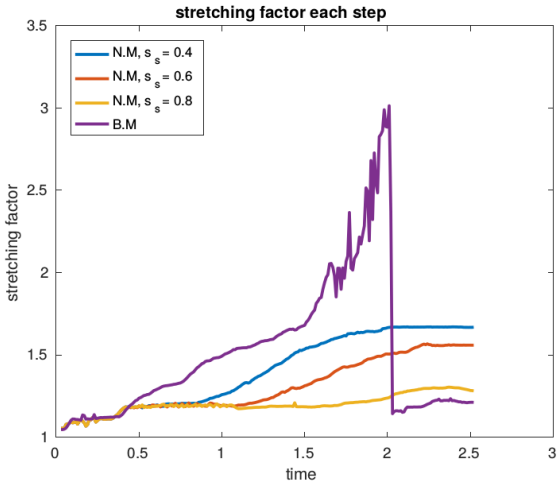
(a) (b) (c) (d)

Figure 4.3: Initial state of the four experiments, where the red points act as attractors for the deformable object. (a) Rope wrapping cylinder. (b) Cloth passing single pole. (c) Cloth covering two cylinders. (d) Rope matching zig-path

Figure 4.4 shows that the new controller is able to prevent further stretching happening when the object is taut for both the rope and the cloth. In the rope test, the new controller can prevent overstretching with $s_s = 0.4$, as defined in Eq.4.8. We can see the γ of the benchmark methods keeps growing beyond this threshold, while the γ of the new method stays close to the threshold. In the cloth test, the benchmark method's γ (in purple) increases above the threshold $\gamma^{\max} = 1.667$ for cloth, and a sudden drop in γ happens after running the test for 2 seconds. This drop is the "tearing" point in the simulator. Though we still see overstretching happened using the new method for some settings of s_s, in all cases the γ converged before tearing happened (instead of growing without bound).



(a) Rope wrapping cylinder



(b) Cloth passing single pole

Figure 4.4: (a) The red line shows the γ of the benchmark and the blue line shows the γ of the new controller with $s_s = 0.4$ throughout the simulation. (b) The purple line shows the γ of the benchmark, and the blue, red, and yellow lines each show the γ of the new controller with $s_s = 0.4$, $s_s = 0.6$, and $s_s = 0.8$, respectively.

4.5.2 Controller Task Performance

Besides the quantitative analysis of the model accuracy and stretching avoidance, we ran another two experiments, rope-matching-zig-path and cloth-covering-two-cylinder, one each with the rope or the cloth, as shown in Figure 4.3 to see how the new method performed for some coverage tasks. Both the benchmark and the new controllers are able to perform these tasks with comparable performance; reaching approximately the same configurations when forward progress stops due to a local minimum (Figure 4.5), and completing the task (Figure 4.6). This result suggests that we have not lost functionality with respect to the benchmark despite changing the model and control method used.

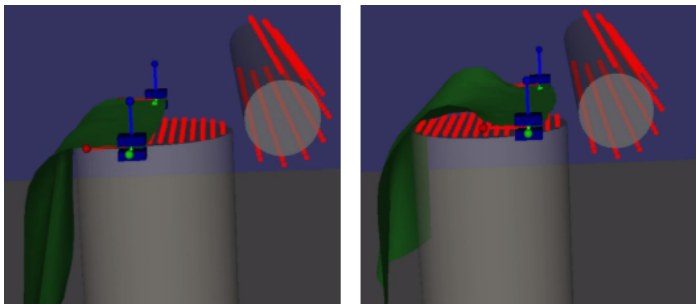


Figure 4.5: Cloth-covering-two-cylinder task start and end configurations. Both controllers are unable to make progress due to a local minima.

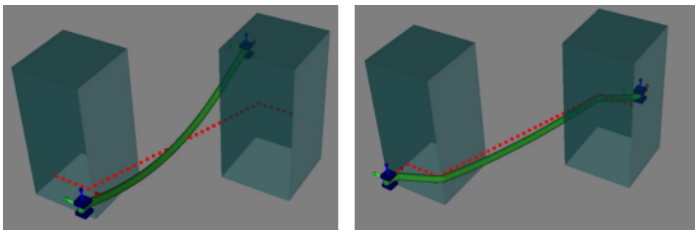


Figure 4.6: Rope-matching-zig-path start and end configurations. Both controllers are able to succeed at the task, bringing the rope into alignment with the desired path.

4.5.3 Physical Robot Experiment

To evaluate our new model and controller on a physical system, we set up an experiment with cloth-like objects manipulated by two 7DoF KUKA iiwa arms (Figure 4.7). To sense the position of the cloth, we use the AprilTags [72] and IAI Kinect2 [73] libraries. The parameters are set as $k^{\text{dir}} = 4$, $k^{\text{dist}} = 10$, $k^{\text{rot}} = 10$ for the new model, $l_c = 0.08$, and $s_s = 0.6$ for the new controller. We set up a task similar to the cloth-passing-single-pole example using a paper towel. For this task, the baseline controller tears the paper towel while the new controller avoids excessive overstretch, instead wrapping around the pole to reach a local minimum.

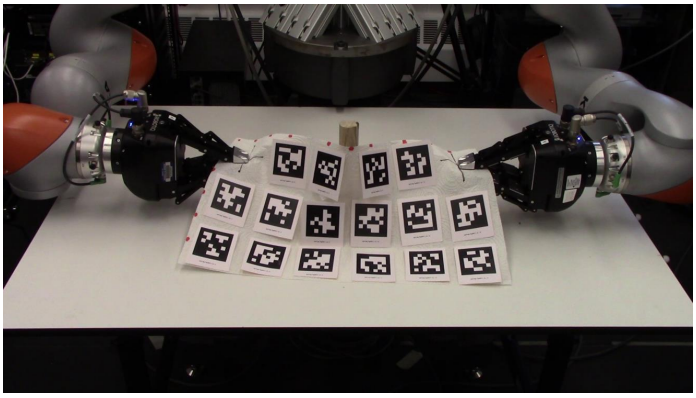


Figure 4.7: Initial setup for the physical robot stretching avoidance test.

4.5.4 Computation Time

To verify the practicality of our method, we gathered data comparing its computation time to the benchmark's and to using the Bullet simulator. Table 4.1 shows the average computation time of a call to the controller for the new method vs. the benchmark. As expected, the benchmark, which uses a linear model, is faster than the

new method. However, the computation times for the new method are still reasonable to use in a control loop.

Table 4.1: Mean computation time (s) to compute the gripper motion for a given state. BM: stretching avoidance controller; NM: stretching constraint controller.

	rope-wrapping -cylinder	rope-matching -cylinder	cloth-passing -single-pole	cloth-wrapping -two-cylinder
BM	0.0055	0.0054	0.0153	0.0037
NM	0.0342	0.0834	0.2363	0.1008

CHAPTER V

Estimating Model Utility

In the previous chapters, we have been working with a single model and a single controller for any given task. When given a new task however, a new choice needs to be made for what model and controller is most suitable. Rather than assuming we have a single high-fidelity model of a deformable object interacting with its environment, our approach in this chapter is to have multiple models available for use, any one of which may be useful at a given time. We do not assume these models are correct, we simply treat the models as having some measurable *utility* for the task. The *utility* of a given model is the expected reduction in task error when using this model to generate robot motion. As the task proceeds, the utility of a given model may change, making other models more suitable for the current part of the task. However, without testing a model's prediction, we do not know its true utility. Testing every model in the set is impractical, as all models would need to be tested at every step, and performing a test changes the state of the object and may drive it into a local minimum. The key question is then which model should be selected for testing at a given time.

The central contribution of this chapter is framing the model selection problem as a Multi-armed Bandit (MAB) problem where the goal is to find the model that has the highest utility for a given task. An arm represents a single model of the deformable object; to "pull" an arm is to use the arm's model to generate and execute a velocity command for the robot. The reward received is the reduction in task error after executing the command. In order to determine which model has the highest utility we need to explore the model space, however we also want to exploit the information we have gained by using models that we estimate to have high utility. One of the primary challenges in performing this exploration versus exploitation trade-off is that our models are inherently coupled and non-stationary; performing an action changes the state of the system which can change the utility of every model, as well as the

reward of pulling each arm. While there is work that frames robust trajectory selection as a MAB problem [75], we are not aware of any previous work which either 1) frames model selection for deformable objects as a MAB problem; or 2) addresses the coupling between arms for non-stationary MAB problems.

In our experiments, we show how to formulate a MAB problem with coupled arms for Jacobian-based models. We perform our experiments on three synthetic systems, and on three deformable object manipulation tasks in the Bullet Physics [71] simulator. We demonstrate that formulating model selection as a MAB problem is able to successfully perform all three manipulation tasks. We also show that our proposed MAB algorithm outperforms previous MAB methods on synthetic trials, and performs competitively on the manipulation tasks.

5.1 Problem Statement

Using similar notation as previous chapters, let a *deformation model* be defined as a function

$$\phi : \mathfrak{se}(3)^G \rightarrow \mathbb{R}^{3P} \tag{5.1}$$

which maps a change in robot configuration $\dot{q}$ to a change in object configuration $\dot{\mathcal{P}}$. Let $\mathcal{M}$ be a set of M deformable models which satisfy this definition (Chapter III). Each model is associated with a robot command function

$$\psi : \mathbb{R}^{3P} \times \mathbb{R}^P \rightarrow \mathfrak{se}(3)^G \tag{5.2}$$

which maps a desired deformable object velocity $\dot{\mathcal{P}}$ and weight W (Section 4.2) to a robot velocity command $\dot{q}$. The deformation model ϕ and robot command function ψ also take the object and robot configuration $(\mathcal{P}, q)$ and environment $\mathcal{O}$ as additional input, however these are frequently omitted for brevity. When a model m is selected for testing, the model generates a gripper command

$$\dot{q}_m(t) = \psi_m(\dot{\mathcal{P}}(t), W(t)) \tag{5.3}$$

which is then executed for one unit of time, moving the deformable object to configuration $\mathcal{P}(t+1)$.

The problem we address in this chapter is which model $m \in \mathcal{M}$ to select in order to move G grippers such that the points in $\mathcal{P}$ align as closely as possible with some task-defined set of T target points $\mathcal{T} \subset \mathbb{R}^3$, while avoiding gripper collision and

excessive stretching of the deformable object. Each task defines a function ρ which measures the alignment error between $\mathcal{P}$ and $\mathcal{T}$. The method we present is a local method which picks a single model m^* at each timestep to treat as the true model. This model is then used to reduce error as much as possible while avoiding collision and excessive stretching.

$$m^* = \underset{m \in \mathcal{M}}{\operatorname{argmin}} \, \rho(\mathcal{T}, \mathcal{P}(t+1)) \tag{5.4}$$

We show that this problem can be treated as an instance of the multi-arm non-stationary dependent bandit problem.

5.2 Bandit-Based Model Selection

The primary difficulty with solving Eq. (5.4) directly is that the effectiveness of a particular model in minimizing error is unknown. It may be the case that no model in the set produces the optimal option, however, this does not prevent a model from being useful. In particular the *utility* of a model may change from one task to another, and from one configuration to another as the deformable object changes shape, and moves in and out of contact with the environment. We start by defining the utility $u_m(t) \in \mathbb{R}$ of a model as the expected improvement in task error ρ if model m is used to generate a robot command at time t. If we know which model has the highest utility then we can solve (5.4). This leads to a classic exploration versus exploitation trade-off where we need to explore the space of models in order to learn which one is the most useful, while also exploiting the knowledge we have already gained. The multi-armed bandit framework is explicitly designed to handle this trade-off.

In the MAB framework, each arm represents a model in $\mathcal{M}$; to pull arm m is to command the grippers with velocity $\dot{q}_m(t)$ (Eq. 5.3) for 1 unit of time. We then define the *reward* $r_m(t+1)$ after taking action $\dot{q}_m(t)$ as the improvement in error

$$r_m(t+1) = \rho(t) - \rho(t+1) = u_m(t) + w \tag{5.5}$$

where w is a zero-mean noise term. The goal is to pick a sequence of arm pulls to minimize total expected regret $R(T_f)$ over some (possibly infinite) horizon T_f

$$E[R(T_f)] = \sum_{t=1}^{T_f} (E[r^*(t)] - E[r(t)]) \tag{5.6}$$

where $r^*(t)$ is the reward of the best model at time t. The next section describes how to use bandit-based model selection for deformable object manipulation.

5.3 MAB Formulation for Deformable Object Manipulation

Our algorithm (Alg. 11) can be broken down into four major sections and an initialization block. In the initialization block we pre-compute the geodesic distance (see Figure 3.1) between every pair of points in $\mathcal{P}$ when the deformable object is in its "natural" or "relaxed" state and store the result in D. These distances are used to construct the deformation models (Section 3.2), as well as to avoid overstretching the object (Section 4.3.1). At each iteration we:

1. pick a model to use to achieve the desired direction (Section 5.4);
2. compute the task-defined desired direction to move the deformable object (Section 4.2);
3. generate a velocity command using the chosen model (Section 4.3);
4. modify the command to avoid obstacles (Section 4.3);
5. update bandit algorithm parameters (Section 5.4).

5.4 Algorithms for MAB

Previous solutions [36, 41] to minimizing Eq. (5.6) assume that rewards for each arm are normally and independently distributed and then estimate the mean and variance of each Gaussian distribution. We test three algorithms in our experiments: Upper Confidence Bound for normally distributed bandits UCB1-Normal, Kalman Filter Based Solution to Non-Stationary Multi-arm Normal Bandits (KF-MANB), and our extension of KF-MANB, Kalman Filter Based Solution to Non-Stationary Multi-arm Normal Dependent Bandit (KF-MANDB).

5.4.1 UCB1-Normal

The UCB1-Normal algorithm [36] (Alg. 12) treats each arm (model) as independent, estimating an optimistic Upper Confidence Bound (UCB) for the utility of each model. The model with the highest UCB is used to command the robot at each

Algorithm 11 MainLoop($\mathcal{O}, \beta, \lambda$)

$t \leftarrow 0$
$D \leftarrow$ GeodesicDistanceMatrix($\mathcal{P}_{relaxed}$)
$\mathcal{M} \leftarrow$ InitializeModels(D)
InitialzeBanditAlgorithm()
$\mathcal{P}(0) \leftarrow$ SensePoints()
$q(0) \leftarrow$ SenseRobotConfig()
while true **do**
    $m \leftarrow$ SelectArmUsingBanditAlgorithm()
    $\mathcal{T} \leftarrow$ GetTargets()
    $\mathcal{P}_C \leftarrow$ CalculateCorrespondences($\mathcal{P}_t, \mathcal{T}$)
    $\dot{\mathcal{P}}_e, W_e \leftarrow$ FollowNavigationFunction($\mathcal{P}_n, \mathcal{P}_C$)
    $\dot{\mathcal{P}}_s, W_s \leftarrow$ StretchingCorrection($D, \gamma^{\max}, \mathcal{P}$)
    $\dot{\mathcal{P}}_d, W_d \leftarrow$ CombineTerms($\dot{\mathcal{P}}_e, W_e, \dot{\mathcal{P}}_s, W_s, k_s$)
    $\dot{q}_d \leftarrow \psi_m(\dot{\mathcal{P}}_d, W_d)$
    $\dot{q} \leftarrow$ ObstacleRepulsion($\dot{q}_d, \mathcal{O}, \beta$)
    CommandConfiguration($q(t) + \dot{q}$)
    $\mathcal{P}(t+1) \leftarrow$ SensePoints()
    $q(t+1) \leftarrow$ SenseRobotConfig()
    UpdateBanditAlgorithm()
    $t \leftarrow t + 1$
end while

timestep. This algorithm assumes that the utility of each model is stationary, gradually shifting from exploration to exploitation as more information is gained. While our problem is non-stationary and dependant, we use UCB1-Normal as a baseline algorithm to compare against due to its prevalence in previous work.

5.4.2 KF-MANB

The Kalman Filter Based Solution to Non-Stationary Multi-arm Bandit (KF-MANB) algorithm [41] (Alg. 13) uses independent Kalman filters to estimate the utility distribution of each model, and then uses Thompson sampling [38] to chose which model to use at each timestep. Because this algorithm explicitly allows for non-stationary reward distributions, it is able to "switch" between models much faster than UCB1-Normal.

5.4.3 KF-MANDB

We also propose a variant of KF-MANB, replacing the independent Kalman filters with a single joint Kalman filter (Alg. 14). This enables us to capture the correlations

Algorithm 12 UCB1-Normal - reproduced from [36]

for $t = 1, 2, \ldots$ **do**

- If there is an arm which has been pulled less than $\lceil 8 \log t \rceil$ times then pull this arm. If multiple arms qualify, we select the arm that has been pulled less, selecting the arm with the lower index in the case of a tie.
- Otherwise pull arm j that maximizes

$$\bar{u}_j + \sqrt{16 \cdot \frac{q_j - n_j \bar{u}_j^2}{n_j - 1} \cdot \frac{\ln(t-1)}{n_j}}$$

  where $\bar{u}_j$ is the average reward obtained from arm j, q_j is the sum of squared rewards obtained from arm j, and n_j is the number of times arm j has been pulled so far.
- Update $\bar{u}_j$ and q_j with the obtained reward r_j.

end for

between models, allowing us to learn more from each pull. We start by defining utility as a linear system with Gaussian noise with process model $u(t+1) = u(t) + v$ and observation model $r(t) = C(t)u(t) + w$ where $u(t)$ is our current estimate of the relative utility of each model, while v and w are zero-mean Gaussian noise terms. $C(t)$ is a row vector with a 1 in the column of the model we used and zeros elsewhere. The variance on w is defined as $\sigma_{obs}^2 \eta^2$. η is a tuning parameter to scale the covariance to match the reward scale of the specific task, while σ_{obs} controls how much we believe each new observation.

To define the process noise v we want to leverage correlations between models; if two model commands are similar at the current time, the utility of these models is likely correlated. To measure the similarity between two models i and j we use the angle between their gripper velocity commands $\dot{q}_i$ and $\dot{q}_j$. This similarity is then used to directly construct a covariance matrix for each arm pull:

$$\begin{aligned} v &\sim \mathcal{N}(0, \Sigma_{tr}) \\ \Sigma_{tr} &= \sigma_{tr}^2 \eta^2 (\xi \Sigma_{sim} + (1 - \xi)\, \mathbf{I}) \\ \Sigma_{sim,i,j} &= \frac{\langle \dot{q}_i, \dot{q}_j \rangle}{\|\dot{q}_i\| \|\dot{q}_j\|} = \cos \theta_{i,j} \ . \end{aligned} \tag{5.7}$$

σ_{tr} is the standard Kalman Filter transition noise factor tuning parameter. $\xi \in [0, 1]$ is the correlation strength factor; larger ξ gives more weight to the arm correlation, while smaller ξ gives lower weight. When ξ is zero then KF-MANDB will have the

Algorithm 13 KF-MANB - reproduced from [41]

Input: Number of bandit arms M; Observation noise σ_{obs}^2; Transition noise $\sigma_{tr}^2\eta^2$.
Initialization: $\bar{u}_1(1) = \bar{u}_2(1) = \cdots = \bar{u}_M(1) = A$; $\sigma_1(1) = \sigma_2(1) = \cdots = \sigma_M(1) = B$; *# Typically, A can be set to* 0*, with B being sufficiently large*
for $t = 1, 2, \ldots$ **do**

1. For each arm $j \in \{1, \ldots, M\}$, draw a value x_j randomly from the associated *normal* distribution $f(x_j; \bar{u}_j(t), \sigma_j(t))$ with the parameters $(\bar{u}_j(t), \sigma_j(t))$.
2. Pull the arm i whose drawn x_i is the largest one:

$$i = \underset{j \in \{1,\ldots,M\}}{\text{argmax}}\ x_j.$$

3. Receive reward $\tilde{r}_i$ from pulling arm i, and update parameters as follows:
   - Arm i:

$$\bar{u}_i(t+1) = \frac{(\sigma_i^2(t) + \sigma_{tr}^2\eta^2) \cdot \tilde{r}_i + \sigma_{obs}^2 \cdot \bar{u}_i(t)}{\sigma_i^2(t) + \sigma_{tr}^2\eta^2 + \sigma_{obs}^2}$$
$$\sigma_i^2(t+1) = \frac{(\sigma_i^2(t) + \sigma_{tr}^2\eta^2)\sigma_{obs}^2}{\sigma_i^2(t) + \sigma_{tr}^2\eta^2 + \sigma_{obs}^2}$$

   - Arm $j \neq i$:

$$\bar{u}_j(t+1) = \bar{u}_j(t)$$
$$\sigma_j^2(t+1) = \sigma_j^2(t) + \sigma_{tr}^2$$

end for

same update rule as KF-MANB, thus we can view KF-MANDB as a generalization of KF-MANB, allowing for correlation between arms.

After estimating the utility of each model and the noise parameters at the current timestep, these values are then passed into a Kalman filter which estimates a new joint distribution. The next step is the same as KF-MANB; we draw a sample from the resulting distribution, then use the model that yields the largest sample to generate the next robot command. In this way we automatically switch between exploration and exploitation as the system evolves; if we are uncertain of the utility of our models then we are more likely to choose different models from one timestep to the next. If we believe that we have accurate estimates of utility, then we are more likely to choose the model with the highest utility.

Algorithm 14 KF-MANDB

Input: Number of bandit arms M; Observation noise σ_{obs}^2; Transition noise $\sigma_{tr}^2\eta^2$.
Initialization: $\bar{u}(1) = A \in \mathbb{R}^M$; $\Sigma(1) = B \in \mathbb{R}^{M\times M}$; *# Typically, A can be set to* 0, *with* $B \succ 0$ and sufficiently large
for $t = 1, 2, \ldots$ **do**

1. For each arm $j \in \{1, \ldots, M\}$, generate a gripper velocity command $\dot{q}_j$.
2. Draw a value $x = \begin{bmatrix} x_1 & \ldots & x_M \end{bmatrix}^T$ randomly from the joint *normal* distribution $f(x; \bar{u}(t), \Sigma(t))$ with the parameters $(\bar{u}(t), \Sigma(t))$.
3. Pull the arm i whose drawn x_i is the largest one:

$$i = \underset{j\in\{1,\ldots,M\}}{\text{argmax}}\ x_j.$$

4. Receive reward $\tilde{r}_i$ from pulling arm i, and perform a standard Kalman filter prediction and update step:
   - Compute *a priori* covariance estimate and Kalman gain:

$$\begin{aligned} \Sigma_{tr} &\text{ is calculated using Eq. 5.7} \\ \hat{\Sigma} &= \Sigma(t) + \Sigma_{tr} \\ S &= C(t)\hat{\Sigma}C(t)^T + \sigma_{obs}^2 \\ K &= \hat{\Sigma}C(t)^T S^{-1} \end{aligned}$$

   - Compute *a posteriori* utility and covariance estimates:

$$\begin{aligned} \bar{u}(t+1) &= \bar{u}(t) - K\left(C(t)\bar{u}(t) - \tilde{r}_i\right) \\ \Sigma(t+1) &= \hat{\Sigma} - KC(t)\hat{\Sigma} \end{aligned}$$

end for

5.5 Experiments and Results

We test our method on three synthetic tests and three deformable object manipulation tasks in simulation. The synthetic tasks show that the principles we use to estimate the coupling between models are reasonable; while the simulated tasks show that our method is effective at performing deformable object manipulation tasks. Table 5.1 shows the parameters used by the Jacobian-based controller, while Table 5.2 shows the parameters used by the the bandit algorithms for all experiments. We chose these parameters by comparing performance across noise factors σ_{obs}^2 and σ_{tr}^2 from $\{0.01, 0.1, 1, 10\}$ and correlation strength factor ξ from $\{0.1, 0.5, 0.9, 0.99\}$. While performance on individual experiments could be marginally improved by using

different values, we found that $\sigma_{obs}^2 = 0.01$, $\sigma_{tr}^2 = 0.1$, and $\xi = 0.9$ resulted in robust performance for all of our manipulation tasks. η is set dynamically and discussed in Section 5.5.1.

Table 5.1: Controller parameters

		Synthetic Trials	Rope Winding	Table Coverage	Two Stage Coverage
$\mathfrak{se}(3)$ inner product constant	c	-	0.0025	0.0025	0.0025
Servoing max gripper velocity	$\dot{q}^{\max,s}_{\mathfrak{se}(3)}$	0.1	0.2	0.2	0.2
Obstacle avoidance max gripper velocity	$\dot{q}^{\max,c}_{\mathfrak{se}(3)}$	-	0.2	0.2	0.2
Obstacle avoidance scale factor	β	-	200	1000	1000
Stretching correction scale factor	k_s	-	0.005	0.03	0.03

Table 5.2: KF-MANB and KF-MANDB parameters

		Synthetic Trials	Rope Winding	Table Coverage	Two Stage Coverage
Correlation strength factor (KF-MANDB only)	ξ	0.9	0.9	0.9	0.9
Transition noise factor	σ_{tr}^2	1	0.1	0.1	0.1
Observation noise factor	σ_{obs}^2	1	0.01	0.01	0.01

5.5.1 Synthetic Tests

For the synthetic tests, we set up an underactuated system that is representative of manipulating a deformable object with configuration $y \in \mathbb{R}^n$ and control input $\dot{x} \in \mathbb{R}^m$ such that $m < n$ and $\dot{y} = J\dot{x}$. To construct the Jacobian of this system we start with $J = \begin{bmatrix} \mathbf{I}_{m\times m} \\ \mathbf{0}_{(n-m)\times m} \end{bmatrix}$ and add uniform noise drawn from $[-0.1, 0.1]$ to each element of J. The system configuration starts at $\begin{bmatrix} 10 & \dots & 10 \end{bmatrix}^T$ with the target configuration set to the origin. Error is defined as $\rho(t) = \|y(t)\|$, and the desired direction to move the system at each timestep is $\dot{y}_d(t) = -y(t)$. These tasks have

no obstacles or stretching, thus β, k_s, and $\dot{q}^{\max,c}_{\mathfrak{se}(3)}$ are unused. Rather than setting the utility noise scale η *a priori*, we use an annealing filter

$$\eta(t+1) = \max(10^{-10}, 0.9\eta(t) + 0.1|r(t+1)|) \ . \tag{5.8}$$

This enables us to track the changing available reward as the system gets closer to the target.

To generate a model for the model set we start with the true Jacobian J and add uniform noise drawn from $[-0.025, 0.025]$ to each element of J. For an individual trial, each bandit algorithm uses the same J and the same model set. Each bandit algorithm receives the same random number stream during a trial, ensuring that a more favourable stream doesn't bias results. We ran one small test using a 3×2 Jacobian with 10 arms in order to yield results that are easily visualised. The second and third tests are representative of the scale of the simulation experiments, using the same number of models and similar sizes of Jacobian as are used in simulation. A single trial consists of 1000 pulls (1000 commanded actions); each test was performed 100 times to generate statistically significant results. Our results in Table 5.3 show that KF-MANDB clearly performs the best for all three tests.

Table 5.3: Synthetic trial results showing total regret with standard deviation in brackets for all bandit algorithms for 100 runs of each setup.

# of Models	# of rows in J	# of cols in J	UCB1-Normal	KF-MANB	KF-MANDB
10	3	2	4.41 [1.65]	3.62 [1.73]	2.99 [1.40]
60	147	6	5.57 [1.37]	4.89 [1.32]	4.53 [1.42]
60	6075	12	4.21 [0.64]	3.30 [0.56]	2.56 [0.54]

5.5.2 Simulation Trials

We now demonstrate the effectiveness of multi-arm bandit techniques on three example tasks, show how to encode those tasks for use in our framework, and discuss experimental results. The first task shows how our method can be applied to a rope, with the goal of winding the rope around a cylinder in the environment. The second and third tasks show the method applied to cloth. In the second task, two grippers manipulate the cloth so that it covers a table. In the third task, we perform a two-stage coverage task, covering portions of two different cylinders. In all three tasks, the alignment error $\rho(\mathcal{P}, \mathcal{T})$ is measured as the sum of the distances between every

point in $\mathcal{T}$ and the closest point in $\mathcal{P}$ in meters. Figure 5.1 shows the target points in red, and the deformable object in green. A video showing the experiments can be found at `https://www.youtube.com/watch?v=d6ma_Kg8QlQ`.

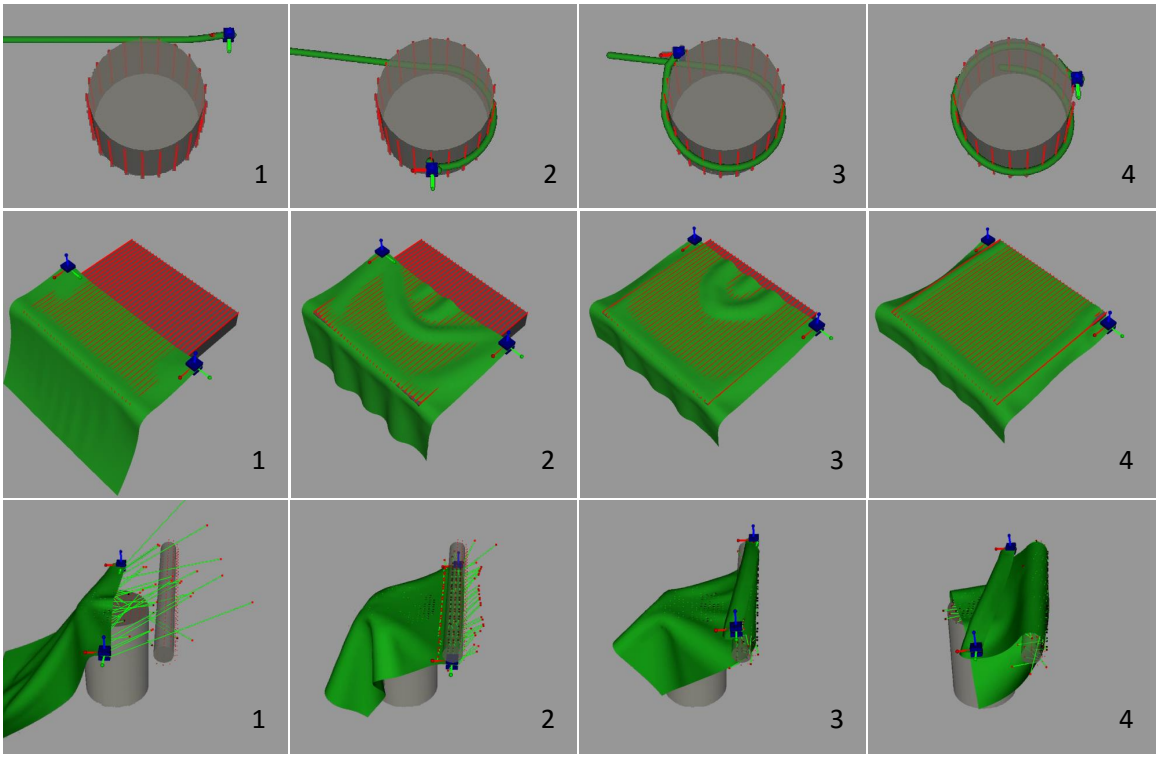


Figure 5.1: Sequence of snapshots showing the execution of the simulated experiments using the KF-MANDB algorithm. The rope and cloth are shown in green, the grippers is shown in blue, and the target points are shown in red. The bottom row additionally shows $\dot{\mathcal{P}}_d$ as green rays with red tips.

All experiments were conducted in the open-source Bullet simulator [71], with additional wrapper code developed at UC Berkeley. The rope is modeled as a series of 49 small capsules linked together by springs and is 1.225m long. The cloth is modeled as a triangle mesh of size 0.5m $\times$ 0.5m for the table coverage task, and size 0.5m $\times$ 0.625m for the two-stage coverage task. 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.[1]

In addition to the diminishing rigidity model introduced in Section 3.2 we will also use *adaptive Jacobian* models based on the work of Navarro-Alarcon et al. [26]. This formulation uses an online estimation method to approximate $J(q, \mathcal{P})$. First we with some estimate of the Jacobian $\tilde{J}(0)$ at time $t = 0$ and then use the Broyden

[1]Our code is available at `https://github.com/UM-ARM-Lab/mab_ms`.

update rule [76] to update $\tilde{J}(t)$ at each timestep t

$$\tilde{J}(t) = \tilde{J}(t-1) + \Gamma \frac{\left(\dot{\mathcal{P}}(t) - \tilde{J}(t-1)\dot{q}(t)\right)}{\dot{q}(t)^T \dot{q}(t)} \dot{q}(t)^T \quad . \tag{5.9}$$

This update rule depends on a update rate $\Gamma \in (0, 1]$ which controls how quickly the estimate shifts between timesteps.

We use models generated using the same parameters for all three tasks with a total of 60 models: 49 diminishing rigidity models with rotation and translational deformability values k^{trans} and k^{rot} ranging from 0 to 24 in steps of 4, as well as 11 adaptive Jacobian models with learning rates Γ ranging from 1 to 10^{-10} in multiples of 10. All adaptive Jacobian models are initialized with the same starting values; we use the diminishing rigidity Jacobian for this seed with $k^{\text{trans}} = k^{\text{rot}} = 10$ for the rope experiment and $k^{\text{trans}} = k^{\text{rot}} = 14$ for the cloth experiments to match the best model found in [7]. We use the same strategy for setting η as we use for the synthetic tests.

We evaluate results for the MAB algorithms as well as using each of the models in the set for the entire task. To calculate regret for each MAB algorithm, we create copies of the simulator at every timestep and simulate the gripper command, then measure the resulting reward $r_m(t)$ for each model. The reward of the best model $r^*(t)$ is then the maximum of individual rewards. As KF-MANB and KF-MANDB are not deterministic algorithms, each task is performed 10 times for these methods. All tests are run on an Intel Xeon E5-2683 v4 processor with 64 GB of RAM. UCB1-Normal and KF-MANB solve Eq. (4.4) once per timestep, while KF-MANDB solves it for every model in $\mathcal{M}$. Computation times for each test are shown in their respective sections.

Winding a Rope Around a Cylinder: In the first example task, a single gripper holds a rope that is lying on a table. The task is to wind the rope around a cylinder which is also on the table (see Figure 5.1). Our results (Figure 5.2) show that at the start of the task all the individual models perform nearly identically, starting to split at 2 seconds (when the gripper first approaches the cylinder) and again at 6 seconds. Despite our model set containing models that are unable to perform the task, our formulation is able to successfully perform the task using all three bandit algorithms. Interestingly, while KF-MANDB outperforms UCB1-Normal and KF-MANB in terms of regret, all three algorithms produce very similar results. Solving Eq. (4.4) at each iteration requires an average of 17.3 ms (std. dev. 5.5 ms) for a single model, and 239.5 ms (std. dev. 153.7 ms) for 60 models.

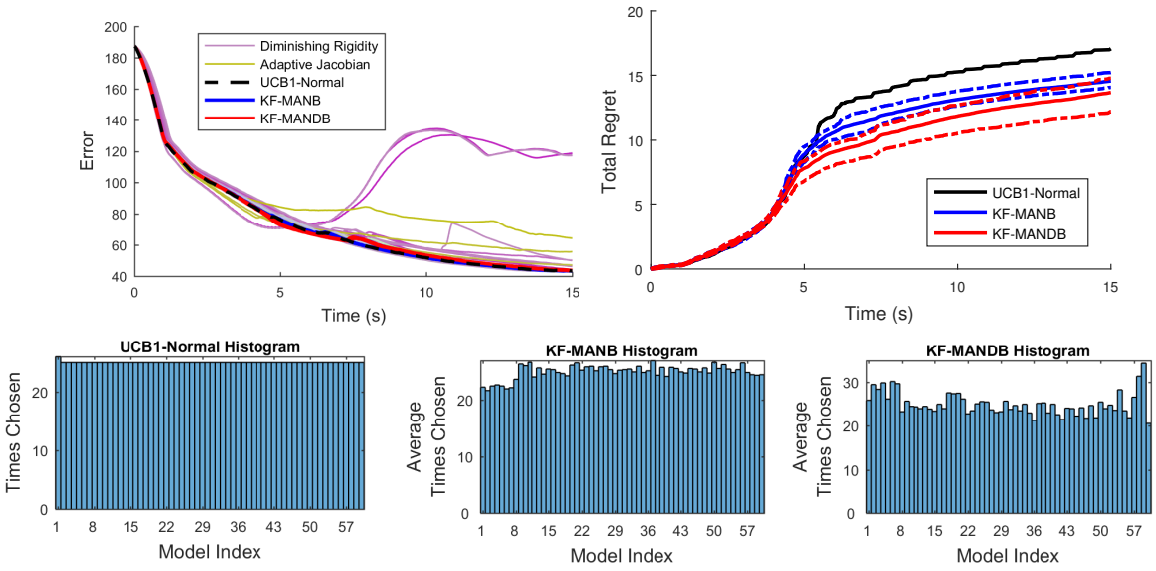


Figure 5.2: Experimental results for the rope-winding task. Top left: alignment error for 10 trials for each MAB algorithm, and each model in the model set when used in isolation. UCB1-Normal, KF-MANB, KF-MANDB lines overlap in the figure for all trials. Top right: Total regret averaged across 10 trials for each MAB algorithm with the minimum and maximum drawn in dashed lines. Bottom row: histograms of the number of times each model was selected by each MAB algorithm; UCB1-Normal (bl), KF-MANB (bm), KF-MANDB (br).

Spreading a Cloth Across a Table: The second scenario we consider is spreading a cloth across a table. In this scenario two grippers hold the rectangular cloth at two corners and the task is to cover the top of the table with the cloth. All of the models are able to perform the task (see Figure 5.3), however, many single-model runs are slower than the bandit methods at completing the task, showing the advantage of the bandit methods. When comparing between the bandit methods, both error and total regret indicate no performance difference between the methods. Solving Eq. (4.4) at each iteration requires an average of 89.5 ms (std. dev. 82.4 ms) for a single model, and 605.1 ms (std. dev. 514.3 ms) for 60 models.

Two-Part Coverage Task: In this experiment, we consider a two-part task. The first part of the task is to cover the top of a cylinder similar to our second scenario. The second part of the task is to cover the far side of a second cylinder. For this task the `GetTargets` function used previously pulls the cloth directly into the second cylinder. The collision avoidance term then negates any motion in that direction causing the grippers to stop moving. To deal with this, we discretize the free space using a voxel grid, and then use Dijkstra's algorithm to find a collision free path between each cover point and every point in free space. We use the result from Dijkstra's algorithm to

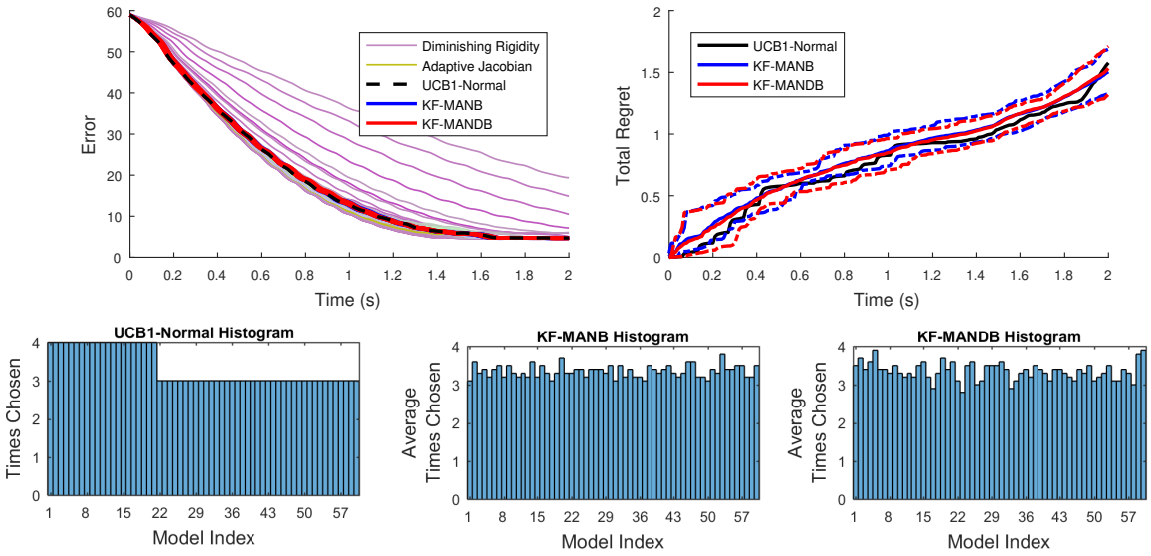


Figure 5.3: Experimental results for the table coverage task. See Figure 5.2 for description.

define a vector field that pulls the nearest (as defined by Dijkstra's) deformable object point p_k along the shortest collision free path to the target point. This task is the most complex of the three (see Figure 5.4); many models are unable to perform the task at all, becoming stuck early in the task. We also observe that both KF-MANB and KF-MANDB show a preference for some models over others. Two interesting trials using KF-MANDB stand out; in the first the grippers end up on opposite sides of the second cylinder, in this configuration the physics engine has difficulty resolving the scene and allows the cloth to be pulled straight through the second cylinder. In the other trial the cloth is pulled off of the first cylinder, however KF-MANDB is able to recover, moving the cloth back onto the first cylinder. KF-MANDB and UCB1-Normal are able to perform the task significantly faster than KF-MANB, though all MAB methods complete the task using our formulation. Solving Eq. (4.4) at each iteration requires an average of 102.6 ms (std. dev. 30.6 ms) for a single model, and 565.5 ms (std. dev. 389.8 ms) for 60 models.

5.6 Discussion

One notable result we observe is that finding and exploiting the best model is less important than avoiding poor models for extended periods of time; in all of the experiments UCB1-Normal never leaves its initial exploration phase, however it is able to successfully perform each task and remains competitive with the other

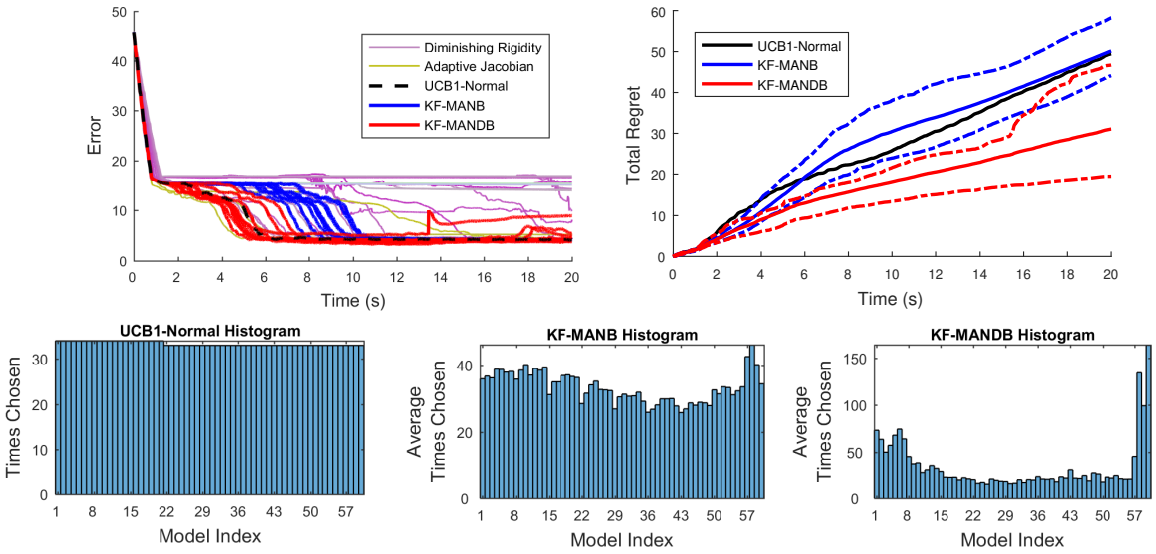


Figure 5.4: Experimental results for the two-part coverage task. See Figure 5.2 for description.

bandit-based methods. We believe this is due to many models being able to provide commands that have a positive dot-product with the correct direction of motion. Intuitively, this means that even inaccurate models can be used to generate useful motion. Indeed, in our experience, models do not need to be particularly accurate to be able to succeed at straightforward tasks; for example using a set of Jacobian models with randomly generated elements, we were still able to successfully cover the table with the cloth, albeit requiring much longer to complete the task.

While avoiding bad models is important for being able to succeed at the task, we can see the effect of choosing good models in the two-part coverage task (Figure 5.4). In this task, the grippers can become stuck for a significant period of time. By taking advantage of model coupling, KF-MANDB is able to explore the model space much more efficiently, quickly finding a model that is able to complete the task. In contrast, the table coverage task (Figure 5.3) does not benefit from significant model switching, and thus all bandit algorithms show very similar performance.

Another benefit of our approach is the ability to escape local minima of individual models, without any explicit detection of local minima. If a given model is unable to improve task error then our estimate of its utility will decrease, leading to other models being more likely to be selected for testing, which will in turn generate different gripper commands which may be able to escape the local minima.

CHAPTER VI

Interleaving Planning and Control

The previous chapters have focused on local methods for solving tasks. While we've shown that these methods are capable of performing interesting tasks, they are unable to escape from local minima due to their very design. This chapter is focused on a method to overcome this limitation by adding planning to the set of tools that we can apply to a given task.

One of the challenges in planning for deformable object manipulation is the high number of degrees of freedom involved; even approximating the configuration of a piece of cloth in 3D with a 4×4 grid results in a 48 degree of freedom configuration space. In addition, the dynamics of the deformable object are difficult to model [5]; even with high-fidelity modeling and simulation, planning for an individual task can take hours [6]. 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 "attraction basin" of the goal as seen in Chapter IV.

The central question we address in this chapter 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 [54, 55]. By answering the question "Will the local controller get stuck?" we can predict if the local 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 neighbourhood

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 6.2.1, and tasks suited to these controllers. Examples of these types of tasks are shown in Figure 6.1. 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.

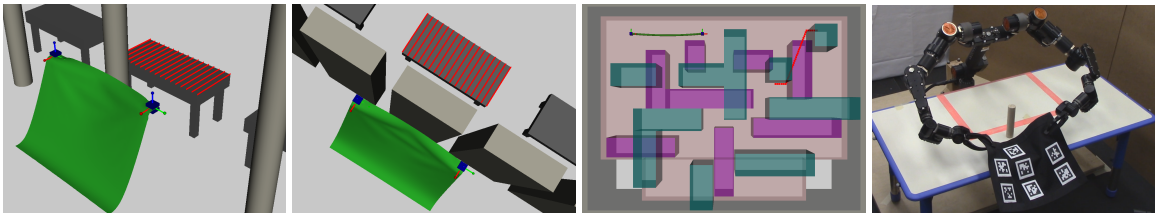


Figure 6.1: Four example manipulation tasks for our framework. In the first two tasks, the objective is to cover the surface of the table (indicated by the red lines) with the cloth (shown in green). In the first task, the grippers (shown in blue) can freely move however the cloth is obstructed by a 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.

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. 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.

6.1 Problem Statement

Define the robot configuration space to be $\mathcal{Q}$. We assume that the robot configuration can be measured exactly. Denote an individual robot configuration as $q \in \mathcal{Q}$. This set can be partitioned into a valid and invalid set. The valid set is referred to as $\mathcal{Q}^{\text{valid}}$, 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 $\mathcal{Q}^{\text{invalid}} = \mathcal{Q} \setminus \mathcal{Q}^{\text{valid}}$.

We assume that our model of the robot is purely kinematic, with no higher order dynamics. Previous chapters assumed an arbitrary number of grippers; in this chapter we restrict the problem to cases where two end-effectors are rigidly attached to the object. We assume that the robot moves slowly enough that we can treat the combined robot and deformable object as quasi-static. Let the function $\phi(q, \mathcal{P}, \dot{q})$ map the system configuration $(q, \mathcal{P})$ and robot movement $\dot{q}$ to the corresponding deformable object movement $\dot{\mathcal{P}}$.

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

The problem we address in this chapter is how to find a sequence of N_e robot commands $\left\{\dot{q}_0^{\text{cmd}}, \ldots, \dot{q}_{N_e-1}^{\text{cmd}}\right\} = \dot{Q}^{\text{cmd}}$ 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 $\dot{q}^{\max}$. Let these feasibility constraints be represented by $A(\dot{Q}^{\text{cmd}}) = 0$. Then the problem we

[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.

seek to solve is:

$$\begin{aligned} \text{find} \quad & N_e, \dot{Q}^{\text{cmd}} \\ \text{s.t.} \quad & \Omega(\mathcal{T}, \mathcal{P}_{N_e}) = \texttt{true} \\ & A(\dot{Q}^{\text{cmd}}) = 0 \end{aligned} \tag{6.1}$$

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

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 which does not rely on high-fidelity modelling 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.

6.2 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 6.2 shows how 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.

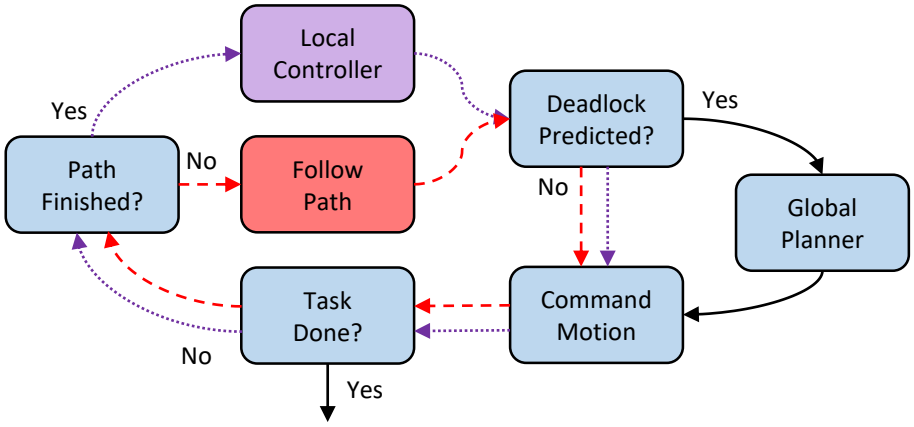


Figure 6.2: Block diagram showing the major components of our framework. On each cycle we use either the local controller (dotted purple arrows) or a planned path (dashed red arrows) to predict if the system will be deadlocked in the future, planning a new path is needed to avoid deadlock.

6.2.1 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 Section 4.3. These controllers locally minimize error ρ while avoiding robot collision and excessive stretching of the deformable object.

An important limitation of this approach is that the individual navigation functions used by these controllers 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 6.3 shows our motivating example of this type of situation. Other examples of this kind of situation are shown in Section 6.5. In addition, while this local controller prevents collision between the robot and obstacles, it does not explicitly have any ability to *go around* obstacles.

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

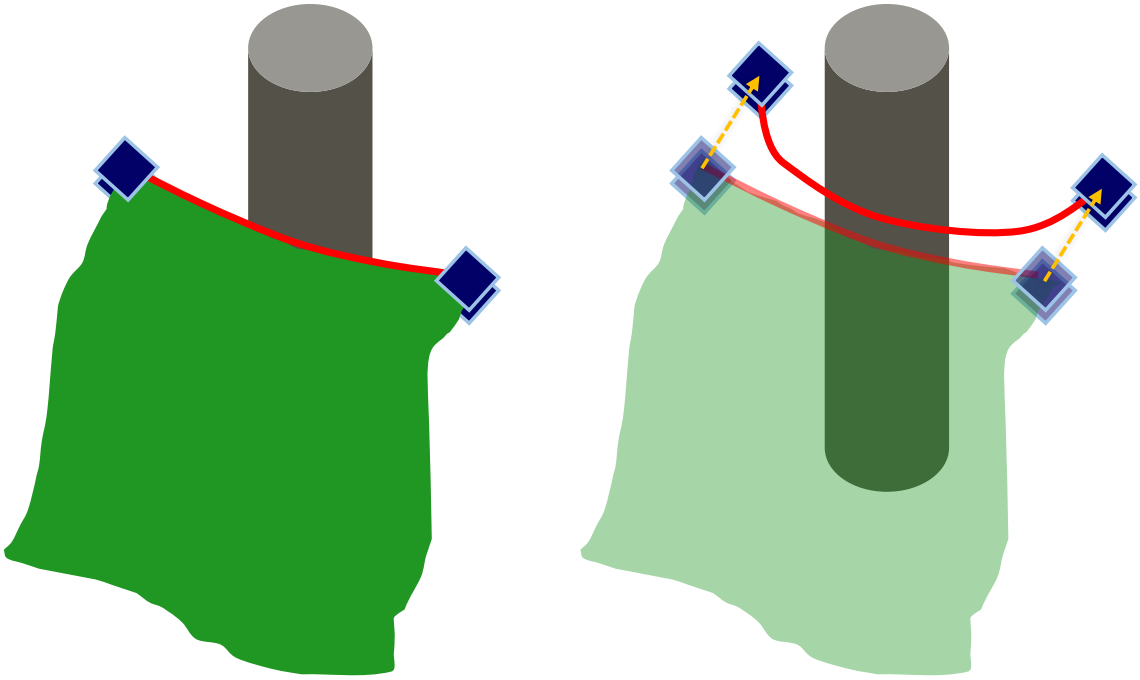


Figure 6.3: Motivating example for deadlock prediction. The local controller moves the grippers on opposite sides of an obstacle, while the geodesic between the grippers (red line) cannot move past the pole, eventually leading to overstretch or tearing of the deformable object if the robot does not stop moving towards the goal.

6.2.2 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 6.3 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 due to an obstacle directly in the path of the desired motion of the robot.

Let $E(q, \mathcal{P}, \dot{q}^{\text{cmd}}) = \dot{q}^{\text{act}}$ be the true motion of the robot when $\dot{q}^{\text{cmd}}$ is executed for unit time; in this section we will be predicting the future state of the system, thus it is not sufficient to consider only $\dot{q}^{\text{cmd}}$, we must also consider $\dot{q}^{\text{act}}$. Modelling inaccuracies as well as the deformable object being in contact can lead to meaningful differences between $\dot{q}^{\text{cmd}}$ and $\dot{q}^{\text{act}}$. Specifically, when a deformable object is in contact with the environment, tracking $\dot{q}^{\text{cmd}}$ 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:

$$\begin{aligned} \|\dot{q}_t^{\text{act}}\| &\approx 0 \\ \Omega(\mathcal{T}, \mathcal{P}_t) &= \texttt{false}. \end{aligned} \tag{6.2}$$

In general we cannot predict if the system will get stuck in the limit; to do so would require a very accurate simulation of the deformable object. Instead we predict if 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.

6.2.2.1 Estimating Gross Motion

The idea central to our prediction (Alg. 15) 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 if 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 $p_i \in \mathcal{P}$. Examples of this approximation are shown in Figure 6.4.

Next we use a simplified version of LocalController() which omits the stretching avoidance terms (Alg. 3 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.

6.2.2.2 Predicting Overstretch

Next we introduce the notion of a *virtual elastic band* (VEB) between the robot's end-effectors. This VEB 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 VEB will be too long, and thus the deformable object is stretched beyond a task-specified maximum stretching factor $\gamma^{\max}$. Similarly, if the VEB 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

Algorithm 15 PredictDeadlock($\rho, q_t, \mathcal{P}_t, v_t, \mathcal{T}, N_p$, Path)

ConfigHistory $\leftarrow$ [ConfigHistory, q_t]
ErrorHistory $\leftarrow$ [ErrorHistory, $\rho(\mathcal{P}_t)$]
BandPredictions $\leftarrow$ []
$\mathcal{P}_C \leftarrow$ CalculateCorrespondences($\mathcal{P}_t, \mathcal{T}$)
for $n = t, \ldots, t + N_p - 1$ **do**
    if Path $\neq \emptyset$ **then**
        $\dot{\mathcal{P}}_e, W_e \leftarrow$ FollowNavigationFunction($\mathcal{P}_n, \mathcal{P}_C$)
        $\mathcal{P}_{n+1} \leftarrow \mathcal{P}_n + \dot{\mathcal{P}}_e$
        $\dot{q}_n^{\text{cmd}} \leftarrow$ FindBestRobotMotion($q_n, \mathcal{P}_n, \dot{\mathcal{P}}_e, W_e$)
        $q_{n+1} \leftarrow q_n + \dot{q}_n^{\text{cmd}}$
    else
        $q_{n+1} \leftarrow q_n +$ FollowPath(Path)
    end if
    $v_{n+1} \leftarrow$ ForwardPropagateBand(v_n, q_{n+1})
    BandPredictions $\leftarrow$ [BandPredictions, v_{n+1}]
end for
if PredictOverstretch(BandPredictions) **or**
   NoProgress(ConfigHistory, ErrorHistory) **then**
    return `true`
else
    return `false`
end if

rest of deformable object will comply to the environment, and does not need to be considered when predicting overstretch. The VEB 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 Chapter VII. This VEB is based on Quinlan's path deformation algorithm [77] and is used both in deadlock prediction as well as global planning (Section 6.2.3 and Section 6.3)

Denote the configuration of an VEB at time t as a sequence of $N_{v,t}$ points $v_t \subset \mathbb{R}^3$. The number of points used to represent an VEB can change over time, but for any given environment and deformable object there is an upper limit $N_v^{\max}$ on the number of points used. Define Path(v) to be the straight line interpolation of all points in v. Define the length of a band to be the length of this straight line interpolation. At each timestep the VEB 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 [77] Section 5, and a hard constraint for collision avoidance. The endpoints of the band track the predicted translation of the end effectors (Alg. 16). 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 6.7).

Algorithm 16 ForwardPropagateBand(v, q)

1: $(p_0, p_1) \leftarrow$ ForwardKinematics(q)
2: $v \leftarrow [p_0, v, p_1]$
3: $v \leftarrow$ InterpolateBandPoints(v)
4: $v \leftarrow$ RemoveExtraBandPoints(v)
5: $v \leftarrow$ PullTight(v)
6: **return** v

Let L_{t+n} be the length of the path defined by the VEB v_{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:

$$\begin{aligned} \tilde{L}_{t+1} &= L_{t+1} \\ \tilde{L}_{t+n} &= \alpha \tilde{L}_{t+n-1} + (1-\alpha) L_{t+n} \ \ , n = 2, \ldots, N_p \ . \end{aligned} \tag{6.3}$$

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 which will prevent overstretch in this case in general. Last 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 6.5, where the local controller will wrap the cloth around the pole, eventually becoming deadlocked in the process.

6.2.2.3 Progress Detection

Last, we track the progress of the robot and task error to estimate if the controller C is making progress towards the task goal. This is designed to detect cases when the robot is trapped against an obstacle. Naively we could look for instances when $\dot{q}^{\text{act}} = 0$ however due 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

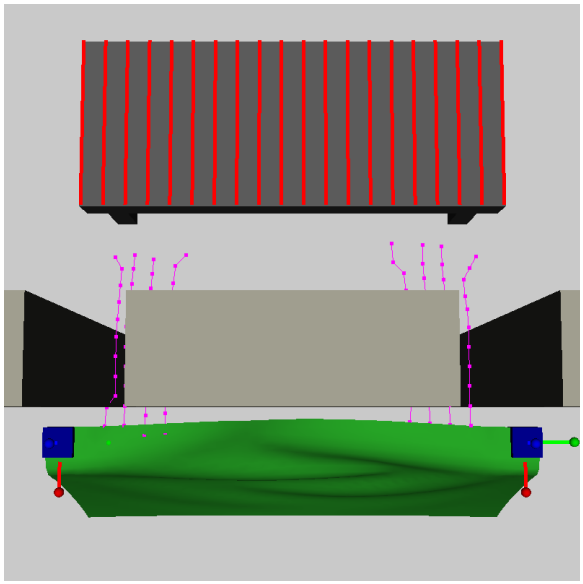


Figure 6.4: Example of estimating the gross motion of the deformable object for a prediction horizon $N_p = 10$. The magenta lines start from the points of the deformable object that are closest to the target points (according to the navigation function). These lines show the paths those points would follow to reach the target when following the navigation function.

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 ErrorrHistory) 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.

6.2.3 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 6.2.2.2, but the interpretation in this use is slightly different; the VEB is a proxy for the leading edge of the deformable object. 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.

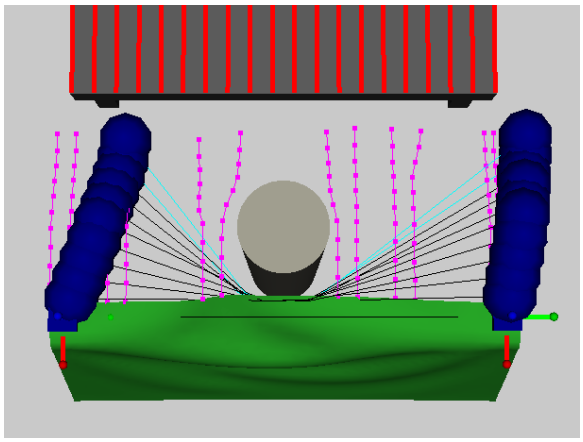


Figure 6.5: Estimated gross motion of the deformable object (magenta lines) and end effectors (blue spheres). The VEB (black lines) is forward propagated by tracking the end effector positions, changing to cyan lines when overstretch is predicted.

In order 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 $\mathcal{T}_U$ which 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 (Alg. 17 lines 1-3). Second, we set the goal configuration of the VEB to be any configuration that is not similar to a *blacklist* of VEBs. This blacklist is the set of all band configurations from which we predicted that the local controller would be deadlocked in the future (Section 6.2.2).

Algorithm 17 PlanPath$(q_t, \mathcal{P}_t, v_t, \mathcal{T}, \text{Blacklist})$

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

To define similarity we use Jaillet and Siméon's *visibility deformation* definition to compare two VEBs ([54]). Intuitively two VEBs 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 $\text{VisCheck}(v, \text{Blacklist}) \rightarrow \{0, 1\}$ denote this visibility deformation check, returning 1 if v is similar to a band in the blacklist and 0 otherwise. Then

$$\mathbb{V}^{\text{goal}} = \{v \mid \text{VisCheck}(v, \text{Blacklist}) = 0\} \tag{6.4}$$

is the set of all VEBs that are dissimilar to the Blacklist.

Combined, $q^{\text{goal}}_{\text{xyz}}, \delta^{\text{goal}}$, and $\mathbb{V}^{\text{goal}}$ define what it means for the planner to have found a path to the goal (Alg. 18); the end-effectors must be in the right region, and the VEB must be dissimilar to any band in the Blacklist.

Algorithm 18 GoalCheck($\mathcal{N}, q^{\text{goal}}_{\text{xyz}}, \mathbb{V}^{\text{goal}}$)

```
for b = (q, v) ∈ N do
    q_xyz ← ForwardKinematics(q)
    if ‖q_xyz,0 − q^goal_xyz,0‖ ≤ δ^goal and ‖q_xyz,1 − q^goal_xyz,1‖ ≤ δ^goal and v ∈ V^goal then
        return true
    end if
end for
return false
```

The combination of local control, deadlock prediction, and global planning are shown in the MainLoop function (Alg. 19). Because the VEB 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 $\gamma^{\max}$.

Algorithm 19 MainLoop($\mathcal{T}, \Omega, \rho, \mathcal{P}_{\text{relaxed}}, \gamma^{\max}$)

$D \leftarrow$ GeodesicDistanceBetweenEndEffectors($\mathcal{P}_{\text{relaxed}}$)
$L^{\max} \leftarrow \gamma^{\max} D$
Blacklist $\leftarrow \emptyset$
Path $\leftarrow \emptyset$
$t \leftarrow 0$
$q_0 \leftarrow$ SenseRobotConfig()
$\mathcal{P}_0 \leftarrow$ SensePoints()
while $\neg\Omega(\mathcal{T}, \mathcal{P}_t)$ **do**
    $v_t \leftarrow$ InitializeBand($\mathcal{P}_t$)
    if PredictDeadlock($\rho, q_t, \mathcal{P}_t, v_t, \mathcal{T}$, Path) **then**
        Blacklist $\leftarrow$ Blacklist $\cup\{v_t\}$
        Path $\leftarrow$ PlanPath($q_t, \mathcal{P}_t, v_t, \mathcal{T}$, Blacklist)
        if Path = Failure **then**
            return Failure
        end if
    end if
    if Path $\neq \emptyset$ **then**
        $\dot{q}^{\text{cmd}} \leftarrow$ FollowPath(Path)
        if PathFinished(Path) **then**
            Path $\leftarrow \emptyset$
        end if
    else
        $\dot{q}^{\text{cmd}} \leftarrow$ LocalController($q_t, \mathcal{P}_t, \mathcal{T}, D, \gamma^{\max}$)
    end if
    CommandConfiguration($q_t + \dot{q}^{\text{cmd}}$)
    $q_{t+1} \leftarrow$ SenseRobotConfig()
    $\mathcal{P}_{t+1} \leftarrow$ SensePoints()
    $t \leftarrow t + 1$
end while
return Success

6.3 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 $\mathcal{Q} \times \mathbb{R}^{3P}$ 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 $L^{\max}$.

6.3.1 Planning Setup

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

$$\mathbb{V}^{\text{valid}} = \{v \mid \text{Length}(v) \leq L^{\max} \textbf{ and } \text{Path(v)} \cap \text{Interior}(\mathcal{O}) = \emptyset\} \ . \tag{6.5}$$

Then the invalid set is $\mathbb{V}^{\text{inv}} = \mathbb{V} \setminus \mathbb{V}^{\text{valid}}$. Similarly define $\mathbb{B}^{\text{valid}} = \mathcal{Q}^{\text{valid}} \times \mathbb{V}^{\text{valid}}$ and $\mathbb{B}^{\text{inv}} = \mathbb{B} \setminus \mathbb{B}^{\text{valid}}$. An individual element is then $b = (b_q, b_v) = (q, v) \in \mathbb{B}$.

$\mathcal{Q}$ and $\mathbb{B}$ are imbued with distance metrics $d_q(\cdot,\cdot) : \mathcal{Q} \times \mathcal{Q} \to \mathbb{R}^{\geq 0}$ and $d_b(\cdot,\cdot) : (\mathcal{Q} \times \mathbb{V}) \times (\mathcal{Q} \times \mathbb{V}) \to \mathbb{R}^{\geq 0}$, respectively. We define distances in robot configuration space and band space to be additive; i.e.,

$$d_b(\cdot,\cdot)^2 = d_q(\cdot,\cdot)^2 + \lambda_v d_v(\cdot,\cdot)^2 \tag{6.6}$$

for some scaling factor $\lambda_v > 0$. To measure distances in $\mathbb{V}$, we first upsample each band using linear interpolation to use the maximum number of points $N_v^{\max}$ for the given task, then measure the Euclidean distance between the upsampled points when considered as a single vector (Alg. 20).

For a given planning problem, we are given a query $(b^{\text{init}}, \mathbb{B}^{\text{goal}})$ which describes

Algorithm 20 BandDistance: $d_v(v_1, v_2)$

1: $\tilde{v}_1 \leftarrow \text{UpsamplePoints}(v_1, N_v^{\max})$
2: $\tilde{v}_2 \leftarrow \text{UpsamplePoints}(v_2, N_v^{\max})$
3: **return** $\|\tilde{v}_1 - \tilde{v}_2\|$

the initial configuration of the robot and band, as well as a goal region for the system to reach. Note that $\mathbb{B}^{\text{goal}}$ 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_q and one in the full configuration space π_b by making the following assumption.

Assumption VI.1 (Deterministic Propagation)**.** *Given an initial configuration in full space $b^{init} \in \mathbb{B}^{valid}$ and the corresponding robot configuration $b_q^{init} \in \mathcal{Q}^{valid}$, a path $\pi_q : [0,1] \to \mathcal{Q}^{valid}$ in robot configuration space with $\pi_q(0) = b_q^{init}$ uniquely defines a single path in full space π_b, where $\pi_b(0) = b^{init}$. Specifically, define*

$$\pi_b(t) = \begin{bmatrix} \pi_q(t) \\ \lim_{h \to 0^-} \text{ForwardPropogateBand}(v(t-h), \pi_q(t)) \end{bmatrix} . \tag{6.7}$$

Eq. (6.7) 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_b to be achievable there must be a robot configuration space path π_q, which when propagated using Eq. (6.7), produces π_b. Let $\text{FullSpace}(\pi_q, b^{\text{init}})$ be the function that maps a given robot configuration space path π_q and full space initial configuration b^{init} to the full space path defined by Eq. (6.7).

6.3.2 Planning Problem Statement

For a given planning instance, the task is to find a path starting from b^{init} through $\mathbb{B}^{\text{valid}}$ to any point in $\mathbb{B}^{\text{goal}}$, while obeying the constraints implied by Eq. (6.7).

For a sequence of robot configurations $b_q^{\text{init}}, b_{1,q}, \ldots, b_{M,q} \in \mathcal{Q}$, let

$$\pi_q = \text{Path}(b_q^{\text{init}}, b_{1,q}, \ldots, b_{M,q}) \tag{6.8}$$

be the path defined by linearly interpolating between each point in order. Then,

formally, the problem our planner addresses is the following:

$$\begin{aligned} \text{find} \quad & M, \{b_{1,q}, \ldots, b_{M,q}\} \\ \text{s.t.} \quad & \pi_q = \text{Path}(b_q^{\text{init}}, b_{1,q}, \ldots, b_{M,q}) \\ & \pi_b = \text{FullSpace}(\pi_q, b^{\text{init}}) \\ & \pi_b(s) \in \mathbb{B}^{\text{valid}}, \ \forall s \in [0,1] \\ & \pi_b(1) \in \mathbb{B}^{\text{goal}} \ . \end{aligned} \tag{6.9}$$

6.3.3 RRT-EB

Our planner, RRT for Elastic Bands (RRT-EB), (Alg. 21) is based on an 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 (Alg. 22) and the goal bias method.

BestNearest is based on the selection method used by [1], selecting the node of smallest cost within a radius δ_{BN} if one exists, falling back to standard nearest neighbour behaviour if no node in the tree is within δ_{BN} of the random sample. We use path length in robot configuration space $\mathcal{Q}$ as a cost function in our implementation. This helps reduce path length and ensures that we can specify lower bounds in Section 6.4.3. 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 6.6). Section 6.4.2 shows that this method is equivalent to searching in the full configuration space directly. δ_{BN} is an additional parameter compared to a 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 to a standard RRT. The larger δ_{BN} is, the harder it is to find narrow passages. We discuss further constraints on δ_{BN} in Section 6.4.3.1.

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

Due to the fact that our system is highly underactuated, and the goal region

Algorithm 21 RRT-EB($q_t, v_t, q^{\text{goal}}_{\text{xyz}}, \delta^{\text{goal}}, \mathbb{V}^{\text{goal}}$)

$\mathcal{N} \leftarrow \{(q_t, v_t)\}$
$\mathcal{E} \leftarrow \emptyset$
$\mathcal{Q}^{\text{goal}} \leftarrow$ GetGoalConfigs($q^{\text{goal}}_{\text{xyz}}$)
while $\neg$MaxTimeEllapsed() **do**
    $b^{\text{rand}} = (q^{\text{rand}}, v^{\text{rand}}) \leftarrow$ SampleUniformConfig()
    $b^{\text{near}} \leftarrow$ BestNearest($\mathcal{N}, \mathcal{E}, \delta_{BN}, b^{\text{rand}}$)
    $\mathcal{N}^{\text{new}}, \mathcal{E}^{\text{new}} \leftarrow$ Connect($b^{\text{near}}, q^{\text{rand}}, L^{\max}$)
    $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}^{\text{new}}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}^{\text{new}}$
    if GoalCheck($\mathcal{N}^{\text{new}}, q^{\text{goal}}_{\text{xyz}}, \delta^{\text{goal}}, \mathbb{V}^{\text{goal}}$) $= 1$ **then**
        return ExtractPath($\mathcal{N}, \mathcal{E}$)
    end if
    $\gamma \sim$ Uniform$[0, 1]$
    if $\gamma \leq \gamma_{gb}$ **then**
        $b^{\text{last}} = (q^{\text{last}}, v^{\text{last}}) \leftarrow$ LastConfig($b^{\text{near}}, \mathcal{N}^{\text{new}}$)
        $q^{\text{bias}} \leftarrow \text{argmin}_{q \in \mathcal{Q}^{\text{goal}}} d_q(q^{\text{last}}, q)$
        $\mathcal{N}^{\text{new}}, \mathcal{E}^{\text{new}} \leftarrow$ Connect($b^{\text{last}}, q^{\text{bias}}, L^{\max}$)
        $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}^{\text{new}}$
        $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}^{\text{new}}$
        if GoalCheck($\mathcal{N}^{\text{new}}, q^{\text{goal}}_{\text{xyz}}, \delta^{\text{goal}}, \mathbb{V}^{\text{goal}}$) **then**
            return ExtractPath($\mathcal{N}, \mathcal{E}$)
        end if
    end if
end while
Return Failure

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 pre-compute a finite set of robot configurations $\mathcal{Q}^{\text{goal}}$ such that the end-effectors of the robot are at $q^{\text{goal}}_{\text{xyz}}$. 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 $\mathcal{N}^{\text{new}} = \emptyset$). A connection is then attempted between b^{last} and the nearest configuration in $\mathcal{Q}^{\text{goal}}$. This allows us to bias exploration toward the robot component of the goal region, which we are able to define explicitly.

Algorithm 22 BestNearest($\mathcal{N}, \mathcal{E}, \delta_{BN}, b^{\text{rand}}$)

$V^{\text{near}} \leftarrow \{b \mid b \in \mathcal{N}, d_b(b, b^{\text{rand}}) \leq \delta_{BN}\}$
if $V^{\text{near}} \neq \emptyset$ **then**
    return $\text{argmin}_{b \in \mathcal{N}} \text{Cost}(b, \mathcal{N}, \mathcal{E})$
else
    $D^2_{\text{near},q} \leftarrow \min_{(q,v) \in \mathcal{N}} d_q(q^{\text{rand}}, q)^2$
    $D^2_{\text{max},b} \leftarrow D^2_{\text{near},q} + \lambda_v D^2_{\text{max},v}$
    $V^{\text{near}} \leftarrow \{b \mid b \in \mathcal{N}, d_q(q, q^{\text{rand}})^2 \leq D^2_{\text{max},b}\}$
    return $\text{argmin}_{b \in V^{\text{near}}} d_b(b, b^{\text{rand}})$
end if

6.4 Probabilistic Completeness of Global Planning

Proving probabilistic completeness in $\mathbb{B}$ 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 (Alg. 16) 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 6.4.1, we are able to bypass most of these challenges by focusing on the portion of $\mathbb{B}$ that can be analyzed; i.e. $\mathcal{Q}$.

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_q-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.

6.4.1 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 [78] of the planning problem may not hold. Instead of relying on expansiveness, we will define a series of alternate definitions and assumptions which are sufficient to ensure the completeness of our method.

First, in line with prior work, we will be assuming properties of the problem

instance in regards to robustness. In particular, we will be assuming the existence of a solution to a given query $\pi_b^{ref} : [0,1] \to \mathbb{B}^{\text{valid}}$ which has several robustness properties. This solution is called a reference path.

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

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

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

Definition VI.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 [79]: Suppose a man is walking a dog. The man is walking on one curve while the dog 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 $\mathbb{T}_q$ as

$$\mathbb{T}_b = \{\pi_b \mid \pi_q \in \mathbb{T}_q \text{ and } \pi_b = \text{FullSpace}(\pi_q, b^{\text{init}})\} \ . \tag{6.10}$$

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

Definition VI.4 (Acceptable δ-Robust Path)**.** *A path π_b^{ref} is acceptable δ-robust if the following hold:*

1. *The robot-space reference path π_q^{ref} has strong δ_q-clearance for some $\delta_q > 0$;*
2. *The final state for every path $\pi_b \in \mathbb{T}_b$ is in $\mathbb{B}^{goal}$.*

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

Assumption VI.5 (Solvable Problem)**.** *There exists some $\delta_q > 0$ 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 paper is not on asymptotic optimality, we will make use of a cost function $\text{Cost}(\pi)$ of a path in Section 6.4.3.1. 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.

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

$$Cost(\pi_b^{ref}) = \inf_{\pi_b \in \mathbb{T}_{b,\delta}} Cost(\pi_b) \ . \tag{6.11}$$

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 will use this assumption in our analysis in Section 6.4.2.

6.4.2 Proof of Nearest-Neighbors Equivalence

Lemma VI.7. *If the maximum distance between any two points in workspace is bounded by $D_{max,w} > 0$, then under distance metric $d_v(\cdot,\cdot)$, the maximum distance between any two points in virtual elastic band space is bounded. I.e. $\exists D_{max,v} > 0$ such that $d_v(v_1, v_2) \leq D_{max,v}\ \forall v_1, v_2 \in \mathbb{V}$.*

Proof. From the definition of $\mathbb{V}$ in Section 6.2.2.2, the number of points used to represent a virtual elastic band is bounded by $N_v^{\max}$. Let $v_1, v_2 \in \mathbb{V}$ be two virtual elastic band configurations, and let $\tilde{v}_1 = (\tilde{b}_{1,1}, \ldots, b_{1,N_v^{\max}})$ and $\tilde{v}_2 = (\tilde{b}_{2,1}, \ldots, b_{2,N_v^{\max}})$ be their upsampled versions as described in Alg. 20. Then

$$\begin{aligned} d_v(v_1, v_2)^2 &= \sum_{i=1}^{N_v^{\max}} \left\| \tilde{b}_{1,i} - \tilde{b}_{2,i} \right\|^2 \\ &\leq \sum_{i=1}^{N_v^{\max}} D_{\max,w}^2 = N_v^{\max} D_{\max,w}^2 = D_{\max,v}^2 \end{aligned} \tag{6.12}$$

□

Lemma VI.8. *If workspace is bounded, then lines 5-8 in Alg. 22 are equivalent to a nearest neighbor search in the full configuration space directly.*

Proof. The upper bound of $D_{\max,v}$ and our additive distance metric (Eq. (6.6)) ensures that the distance between any two configurations in full space $\mathbb{B}$ can be bounded using only the distance in robot configuration space:

$$d_b(\cdot,\cdot)^2 \leq d_q(\cdot,\cdot)^2 + \lambda_v \cdot D_{\max,v}^2 \ . \tag{6.13}$$

Next, consider that in Line 5 of the algorithm, the nearest neighbor to q^{rand} under distance metric d_q is found. Let this nearest neighbor be denoted $\tilde{q}^{\text{near}}$, keeping in mind that it belongs to a vertex in the tree $\tilde{b}^{\text{near}} = (\tilde{q}^{\text{near}}, \tilde{v}^{\text{near}})$. Let the (squared) distance between these points under d_q be $D_{\text{near},q}^2$. From Eq. (6.13), we can bound the distance between the random sample and $\tilde{b}^{\text{near}}$ under d_b as $D_{\max,b}^2 \leq D_{\text{near},q}^2 + \lambda_v D_{\max,v}^2 = D_{\max,b}^2$.

In Line 7 of the algorithm, a radius nearest-neighbors query of radius $D_{\max,b}$ is performed, returning a set V^{near}. By construction if there is a node $b \in \mathcal{N}$ that is closer to b^{rand} than $\tilde{b}^{\text{near}}$, then $b \in V^{\text{near}}$ (Figure 6.6). Then, the method selects as the true nearest neighbor in full space $b^{\text{select}} = \text{argmin}_{b \in V^{\text{near}}} d_b(b, b^{\text{rand}})$. □

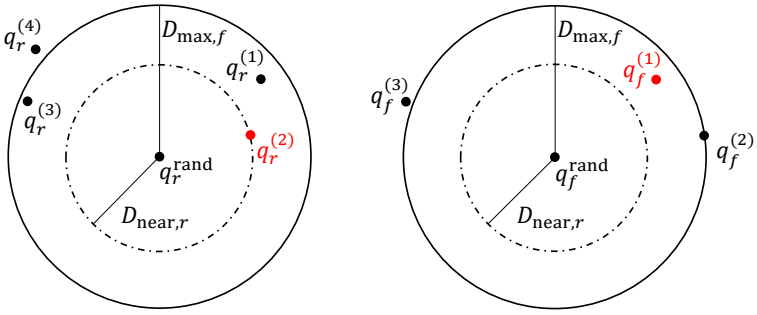


Figure 6.6: Left: $q^{(2)}$ is the nearest node to the b^{rand} in robot space, but it my be as far as $D_{\max,b}$ away in the full configuration space. By considering all nodes within $D_{\max,b}$ in robot space, we ensure that any node (such as $b^{(1)}$) that is closer to b^{rand} than $b^{(2)}$ is selected as part of V^{near}, while nodes such as $b^{(4)}$ 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 b^{rand}, returning $b^{(1)}$ as the nearest node in the tree.

6.4.3 Construction of a δ_q-similar Path

The objective here is to show with probability approaching 1, the planner generates a δ_q-similar path to some robustly-feasible solution given enough time. If an alternate path is found and the algorithm terminates before generating a δ_q-similar path then this is still sufficient for probabilistic completeness. This analysis is similar to [1], and is based on a covering ball sequence of the optimal δ-robust path π_q^{ref}.

Definition VI.9 (Covering Ball Sequence)**.** *Given a path $\pi_q : [0,1] \to \mathcal{Q}^{valid}$, robust clearance $\delta_q > 0$, a BestNearest distance $\delta_{BN} > 0$, and a distance value $0 < \delta_s < \delta_{BN} < \delta_q$; the covering ball sequence is defined as a set of $K+1$ hyper-balls $\{\mathcal{B}_{\delta_q}(q_0), \ldots, \mathcal{B}_{\delta_q}(q_K)\}$ of radius δ_q, where q_k are defined such that:*

- $q_0 = \pi_q(0)$*;*
- $q_K = \pi_q(1)$*;*
- $\text{PathLength}(q_{k-1}, q_k) = \delta_s$ *for* $k = 1, \ldots, K$.

Denote q_k^* to be the center of the k^{th} covering hyper-ball for the reference path π_q^{ref}. Figure 6.7 shows an example of a covering ball sequence.

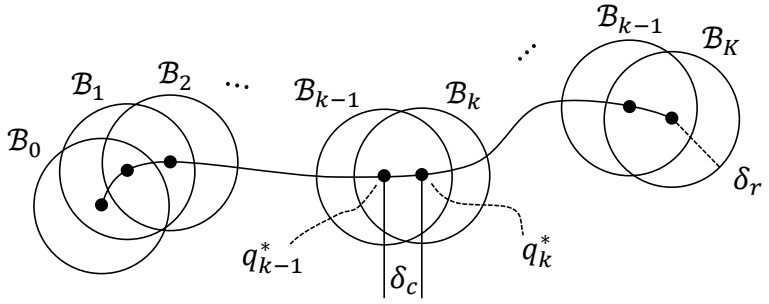


Figure 6.7: Example covering ball sequence for an example reference path with a distance along the path of δ_s between each ball. Given that the path is δ_q-robust, each ball is a subset of $\mathcal{Q}^{\text{valid}}$.

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

$$\liminf_{n\to\infty} \mathbb{P}(\mathcal{N}_n \cap \mathbb{B}^{\text{goal}} \neq \emptyset) = 1 \ . \tag{6.14}$$

To do this, the analysis examines K subsegments of the reference path π_q^{ref}, based on the covering ball sequence for the reference path. If we can generate a robot

path that is δ_q similar to π_q^{ref}, then given Assumption VI.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 n^{th} iteration of the algorithm, it generates a δ_q-similar path to the k^{th} subsegment of π_q^{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_q-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_q-similar edge for subsegment k. From these definitions, the goal then is to show that

$$\lim_{n\to\infty} \mathbb{P}(\text{Success}) = \lim_{n\to\infty} \mathbb{P}\left(E_K^{(n)}\right) = 1 \ . \tag{6.15}$$

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

$$\begin{aligned}
&\mathbb{P}\left(\neg E_k^{(n)}\right) \\
&\quad = \mathbb{P}\left(\neg A_k^{(1)} \cap \cdots \cap \neg A_k^{(n)}\right) \\
&\quad = \mathbb{P}\left(\neg A_k^{(1)}\right) \mathbb{P}\left(\neg A_k^{(2)} \mid \neg A_k^{(1)}\right) \cdot \ldots \\
&\qquad\qquad \cdot \mathbb{P}\left(\neg A_k^{(n)} \mid \neg A_k^{(1)} \cap \cdots \cap \neg A_k^{(n-1)}\right) \\
&\quad = \prod_{i=1}^{n} \mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \ .
\end{aligned} \tag{6.16}$$

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_q-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:

$$\begin{aligned}
&\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \\
&\quad = \mathbb{P}\left(\neg E_{k-1}^{(i-1)}\right) + \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \\
&\qquad\qquad \cdot \mathbb{P}\left(\neg A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right),
\end{aligned} \tag{6.17}$$

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

$$\begin{aligned}
&\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \\
&\quad = \mathbb{P}\left(\neg E_{k-1}^{(i-1)}\right) + \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \\
&\qquad\qquad \cdot \left(1 - \mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right)\right) \quad .
\end{aligned} \tag{6.18}$$

Then multiplying out the last term we get

$$\begin{aligned}
&\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \\
&\quad = \mathbb{P}\left(\neg E_{k-1}^{(i-1)}\right) + \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \\
&\qquad - \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right) \quad .
\end{aligned} \tag{6.19}$$

Finally, summing the first two terms, we arrive at

$$\begin{aligned}
&\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \\
&\quad = 1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \mathbb{P}\left(A_k^{(i)} \mid E_{k-1}^{(i-1)} \cap \neg E_k^{(i-1)}\right) \quad .
\end{aligned} \tag{6.20}$$

Two events need to happen in order to generate a path to the next hyperball; an appropriate node must be selected for expansion, and Connect(...) must generate a δ_q-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

$$\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) = 1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \gamma_k^{(i)} \rho_k^{(i)} \quad . \tag{6.21}$$

As we are examining this probability in the limit, we will 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 6.4.3.1, it will be shown 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 6.4.3.2, $\rho_k^{(i)}$ 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 Eq. (6.21) as

$$\mathbb{P}\left(\neg A_k^{(i)} \mid \neg E_k^{(i-1)}\right) \leq 1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \gamma_k^{(\infty)} \quad . \tag{6.22}$$

Combining equations (6.22) and (6.16) we have

$$\mathbb{P}\left(\neg E_k^{(n)}\right) \leq \prod_{i=1}^{n} \left(1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \gamma_k^{(\infty)}\right) \ . \tag{6.23}$$

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

$$\mathbb{P}\left(\neg E_k^{(n)}\right) \leq y_k^{(n)} \ . \tag{6.24}$$

We will show using induction over k, that Eq. (6.24) 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

$$\begin{aligned} \lim_{n\to\infty} \mathbb{P}\left(\neg E_1^{(n)}\right) &\leq \lim_{n\to\infty} \prod_{i=1}^{n} \left(1 - \mathbb{P}\left(E_0^{(i-1)}\right) \gamma_k^{(\infty)}\right) \\ &= \lim_{n\to\infty} \prod_{i=1}^{n} \left(1 - \gamma_k^{(\infty)}\right) \\ &= \lim_{n\to\infty} \left(1 - \gamma_k^{(\infty)}\right)^n = 0 \ . \end{aligned} \tag{6.25}$$

Induction hypothesis:

$$\lim_{n\to\infty} \mathbb{P}\left(\neg E_m^{(n)}\right) = 0 \text{ for } m = 1, 2, \ldots, k-1 \ . \tag{6.26}$$

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 $\mathbb{P}\left(\neg E_k^{(n)}\right)$:

$$\log y_k^{(n)} = \sum_{i=1}^{n} \log \left(1 - \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \gamma_k^{(\infty)}\right) \ . \tag{6.27}$$

Denote $x = \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \gamma_k^{(\infty)}$. Given that $0 \leq x < 1$, and writing the Taylor series

expansion of $\log(1-x)$ centered at $x=0$ we have

$$\log(1-x) = -\sum_{m=1}^{\infty} \frac{x^m}{m} \ . \tag{6.28}$$

Substituting Eq. (6.28) back into Eq. (6.27) we get

$$\log y_k^{(n)} = -\sum_{i=1}^{n} \sum_{m=1}^{\infty} \frac{\left(\mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)}\right)^m}{m} \ . \tag{6.29}$$

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

$$\log y_k^{(n)} \leq -\sum_{i=1}^{n} \mathbb{P}\left(E_{k-1}^{(i-1)}\right)\gamma_k^{(\infty)} \ . \tag{6.30}$$

Rearranging terms yields

$$\log y_k^{(n)} \leq -\gamma_k^{(\infty)} \sum_{i=1}^{n} \mathbb{P}\left(E_{k-1}^{(i-1)}\right) \ . \tag{6.31}$$

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

$$\lim_{n\to\infty} \log y_k^{(n)} \leq -\gamma_k^{(\infty)} \sum_{i=1}^{n} \lim_{n\to\infty} \mathbb{P}\left(E_{k-1}^{(i-1)}\right) = -\infty \ . \tag{6.32}$$

Taking the log of Eq. (6.24) and combining with Eq. (6.32) we get

$$\lim_{n\to\infty} \log \mathbb{P}\left(\neg E_k^{(n)}\right) \leq \lim_{n\to\infty} \log y_k^{(n)} = -\infty \tag{6.33}$$

and therefore

$$\lim_{n\to\infty} \mathbb{P}\left(\neg E_k^{(n)}\right) = 0, \tag{6.34}$$

which completes the induction step.

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

$$\lim_{n\to\infty} \mathbb{P}(\text{Success}) = \lim_{n\to\infty} \left(1 - \mathbb{P}\left(\neg E_K^{(n)}\right)\right) = 1 \ . \tag{6.35}$$

6.4.3.1 Selection of an appropriate node ($\gamma_k^{(\infty)}$):

First, we define the following restriction on the definition of δ_{BN}:

Definition VI.10 (δ_{BN} Restriction)**.** *For a reference path π_q^{ref} with robustness δ_q, δ_{BN} is defined such that $\delta_\theta = \delta_q - \delta_{BN} > 0$.*

The proof that $\gamma_k^{(\infty)} > 0$ follows directly from the related work of [1] (proof of Lemma 23). To summarize, due to best-nearest neighbors selection, there exists a positive-measure region around the minimum cost vertex b^{near} which observes the optimal reference path in which its cost dominates all other nearby nodes, and therefore, when b^{rand} is drawn in this volume, $b^{\text{near}} = (q^{\text{near}}, v^{\text{near}})$ is guaranteed to be selected (Figure 6.8). Since our approach follows an equivalent sampling and nearest neighbor method to [1] (as shown in Section 6.4.2),

$$\gamma_k^{(\infty)} = \frac{\mu\left(\mathcal{B}_{\delta_\theta}\left(b_k^*\right) \cap \mathcal{B}_{\delta_{BN}}\left(b^{\text{near}}\right)\right)}{\mu\left(\mathbb{B}\right)} > 0 \tag{6.36}$$

follows directly.

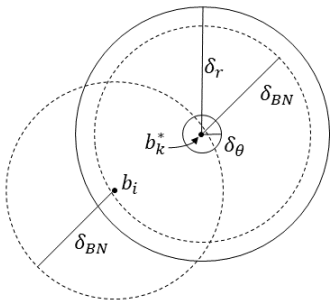


Figure 6.8: Minimum domination region for a node b_i, adapted from Li et al. [1] Lemma 23. Sampling b^{rand} in the shaded region guarantees that a node $b^{\text{near}} \in \mathcal{B}_{\delta_q}(b_k^*)$ is selected for propagation so that either $b^{\text{near}} = b_i$ or $\text{Cost}(b^{\text{near}}) < \text{Cost}(b_i)$.

To show that $\gamma_k^{(\infty)} < 1$, we need only consider the case when there are at least 2 nodes in $\mathcal{N}$.

6.4.3.2 δ_q-similar Propagation ($\rho_k^{(i)}$):

Given that our nearest neighbor method is non-standard, and operating in the full configuration space $\mathbb{B}$, 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 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 VI.11. *If b^{rand} is within the minimum domination region as described in [1] Lemma 23 (Figure 6.8), then $q^{rand} \in \mathcal{B}_{\delta_q}(q_k^*)$ and Connect() will generate a segment that is δ_q-similar to segment k of the reference path.*

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

$$\begin{aligned} d_q(q^{\text{rand}}, q_k^*) &\leq d_b(b^{\text{rand}}, b_k^*) \\ &\leq d_b(b^{\text{rand}}, b_{k-1}^*) + d_b(b_{k-1}^*, b_k^*) \\ &\leq \delta_\theta + \delta_s = \delta_q - \delta_{BN} + \delta_s \ . \end{aligned}$$

Then by construction of the covering ball sequence, we have that $\delta_s - \delta_{BN} < 0$ and thus $d_q(q^{\text{rand}}, q_k^*) < \delta_q$. In addition, we have that the straight line between q^{near} as selected by q^{rand} is entirely contained in $\mathcal{B}_{\delta_q}(q_{k-1}^*)$, and thus is also in $\mathcal{Q}^{\text{valid}}$ as the reference path is optimal δ-robust. We then have that the path generated by Connect is δ_q-similar to the k^{th} segment of the reference path. □

Lemma VI.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 b^{rand} is within the minimum domination region (Figure 6.8) of V^{near}. If b^{rand} is within the minimum domination region of V^{near}, then by Lemma VI.11, Connect() will generate a δ_q-similar segment with probability 1. Denote this event as B. Second, the event that b^{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:

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

□

6.5 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 which cannot be performed using either our controller or planner alone. In Section 6.6 we demonstrate that our method can also be applied to a physical robot.

For these simulation tasks $\mathcal{Q} = SE(3)^G$ – 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 6.1). A video showing the experiments can be found at `https://www.youtube.com/watch?v=O9w0qpbev6U`.

All experiments were conducted in the open-source Bullet simulator [71], with additional wrapper code developed at UC Berkeley [80]. The cloth is modeled as a triangle mesh using 1500 vertices with a total size of 0.3m × 0.5m. 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 0.78m long rope, and 47 capsules for a 0.94m 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 $\gamma^{\max}$ 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, shown in Tables 6.1 and 6.2. For the purpose of the planner we treat the grippers as spheres, reducing the planning space from $SE(3)^G \times \mathbb{V}$ to $\mathbb{R}^6 \times \mathbb{V}$.

Table 6.1: 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

To smooth the path returned by the planner, at each iteration we randomly select

Table 6.2: 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_v	10^{-6}
Maximum Band Points	$N_v^{\max}$	500

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 VEB 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 VEB 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 1500 for experiment 3 due to the larger environment.

6.5.1 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 6.9). 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 influence task error or any other calculation. Our results show that even though the global planner is only planning using the gripper positions and a VEB between them, it is able to find the correct neighbourhood for the local controller to complete the task. On average we are able to find and smooth a path in 3.0 seconds (Table 6.3, Table 6.4), with the majority of the planning time spent on forward propagation of the VEB 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.

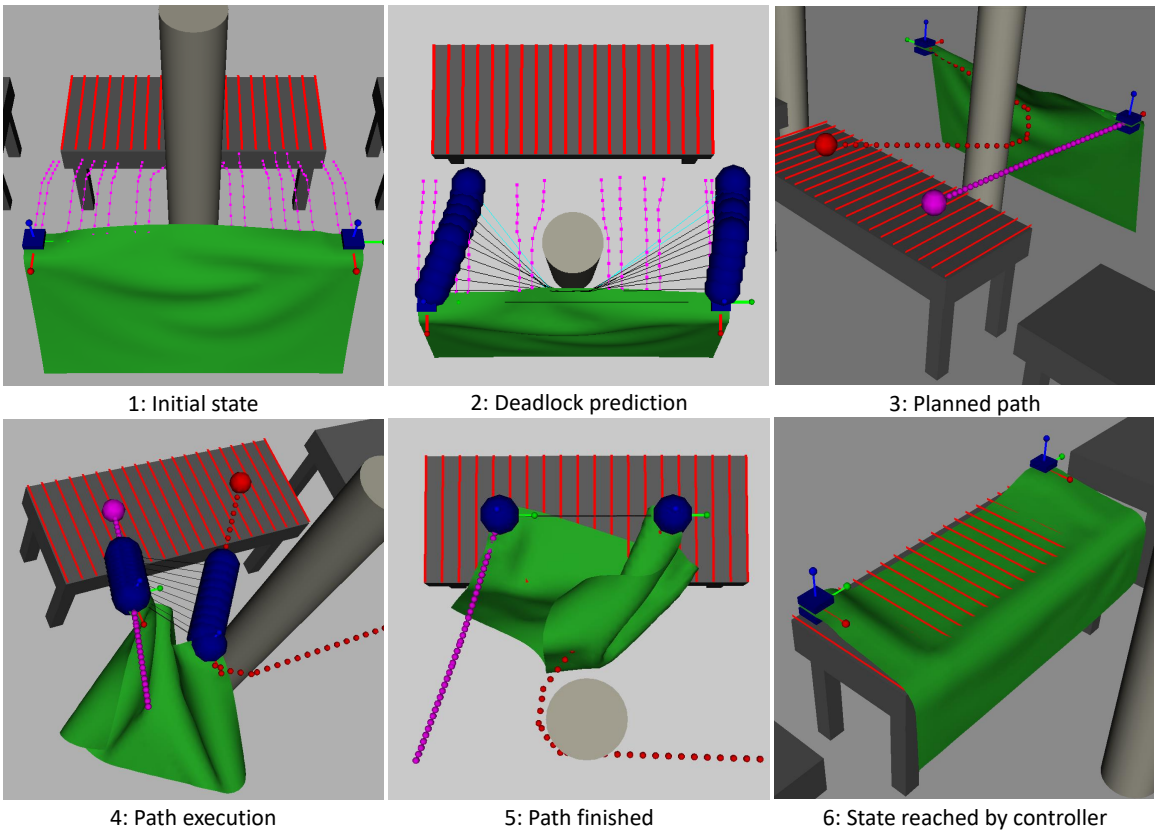


Figure 6.9: Sequence of snapshots showing the execution of the first experiment. The cloth is shown in green, the grippers are shown in blue, and the target points are shown as red lines. (1) The approximate integration of the navigation functions from error reduction over N_p timesteps, shown in magenta, pull the cloth to opposite sides of the pillar. (2) A sequence of VEBs 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 neighbourhood. (6) Final system state when the task is finished by the local controller.

6.5.2 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 6.10). 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 VEB forward propagation takes longer near obstacles). The local controller is again able to complete the task after invoking the planer a single time on all 100 trials.

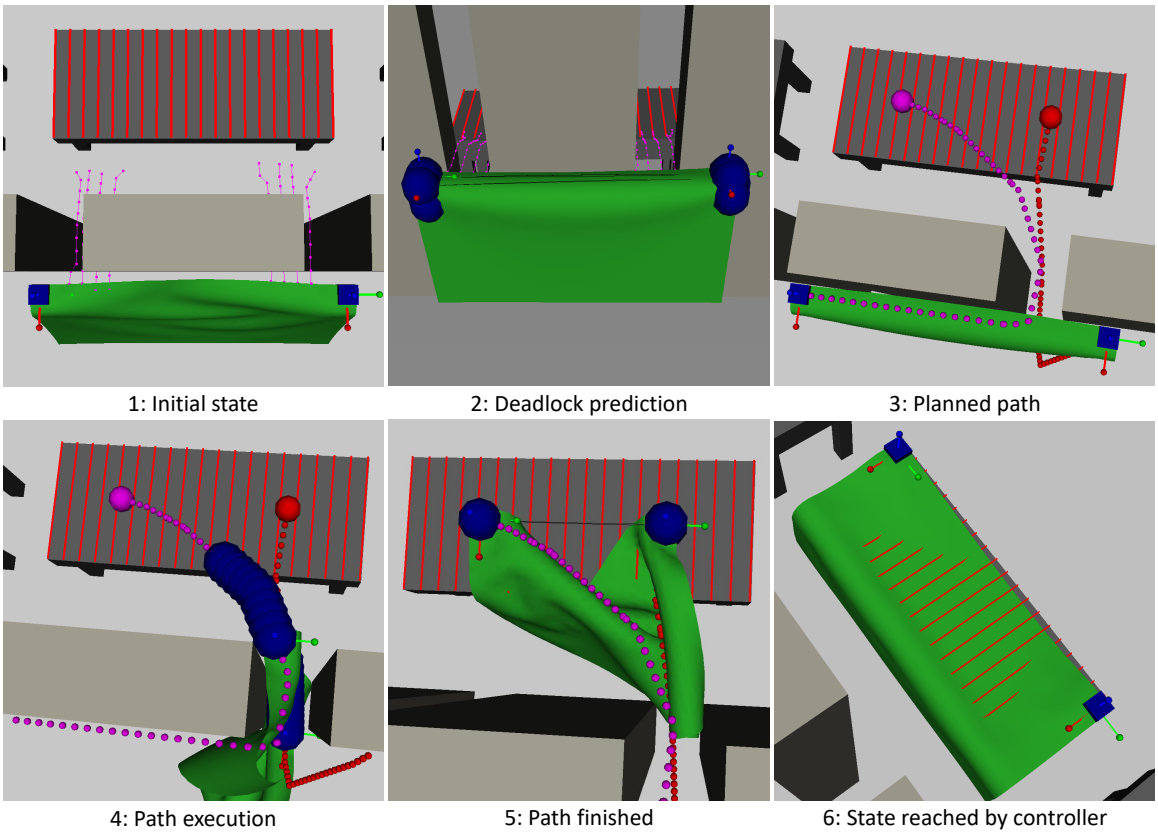


Figure 6.10: Sequence of snapshots showing the execution of the second experiment. We use the same colors as the previous experiment (Figure 6.9), but in this example instead of detecting future overstretch in panel (2), we detect that the system is stuck in a bad local minimum and unable to make progress.

6.5.3 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 6.11). 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 6.5. 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 4.2s on average, suggesting that our method can maintain fast planning times, even in larger environments with many more obstacles.

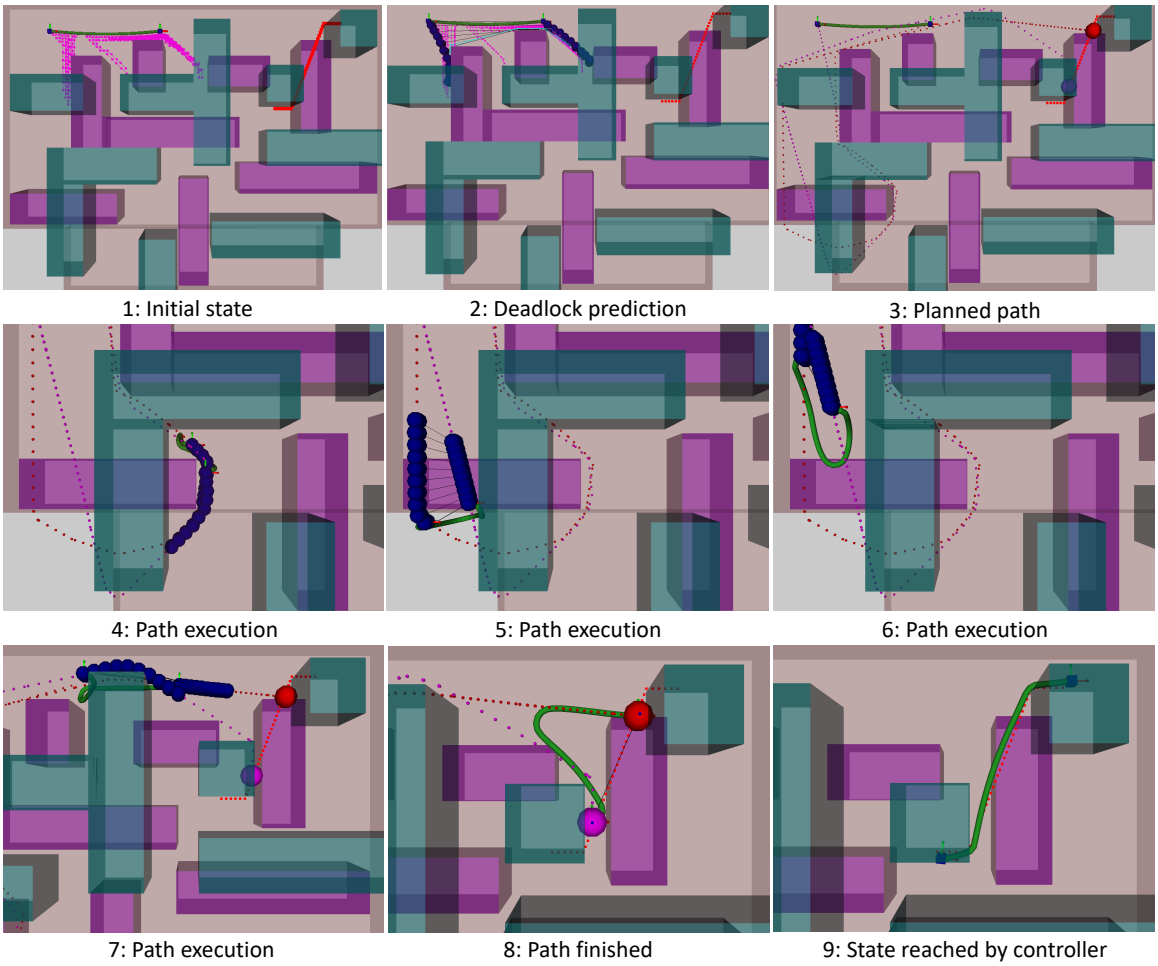


Figure 6.11: Sequence of snapshots showing the execution of the third experiment. The rope is shown in green starting in the top left corner, the grippers are shown in blue, and the target points are shown in red in the top right corner. The maze consists of top and bottom layers (green and purple, respectively). 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).

6.5.4 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 6.12). 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 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.

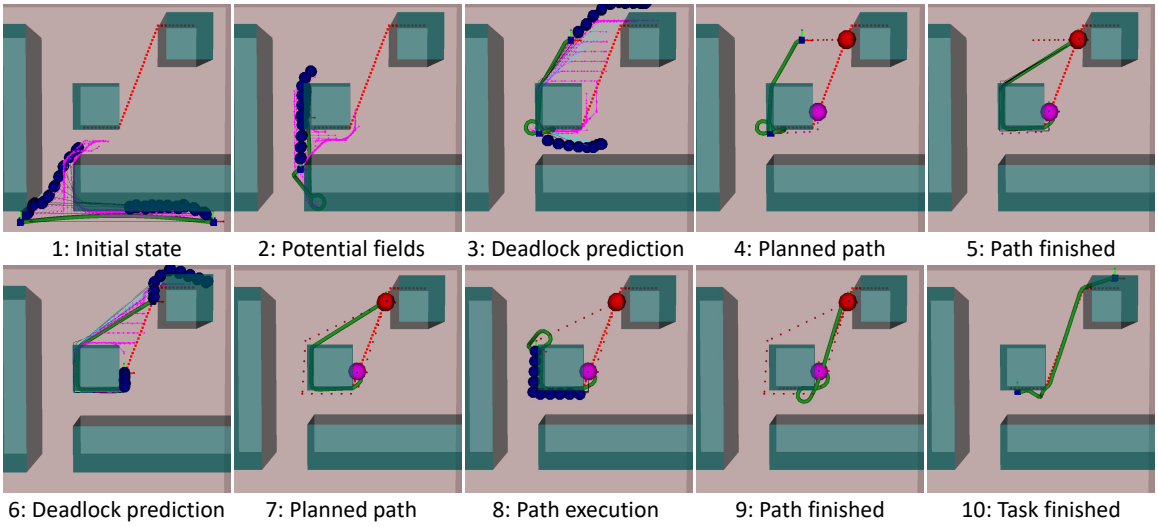


Figure 6.12: Sequence of snapshots for the fourth experiment. We use the same colors as the previous experiment (Figure 6.11), but in this example the local controller gets stuck twice, in panels 3 and 6. In panel 7 the global planner finds a new neighbourhood that is distinct from previously-tried neighbourhoods.

Table 6.3: Planning statistics for the first plan for each example task in simulation, averaged across 100 trials. Standard deviation is shown in brackets.

	Samples	States	NN Time (s)	Validity Checking Time (s)	Total Time (s)
Single Pillar	158 [121]	1182 [804]	∼0.0 [∼0.0]	0.6 [0.5]	0.6 [0.5]
Double Slit	478 [353]	2124 [1428]	∼0.0 [∼0.0]	0.7 [0.8]	0.7 [0.8]
Rope Maze	4796 [1613]	9926 [3760]	0.1 [∼0.0]	4.0 [1.7]	4.2 [1.8]
Repeated Planning	54 [46]	153 [147]	∼0.0 [∼0.0]	0.1 [0.1]	0.1 [0.1]

Table 6.4: Smoothing statistics for the first plan for each example task in simulation, averaged across 100 trials. Standard deviation is shown in brackets.

	Iterations	Validity Checking Time (s)	Visibility Deformation Time (s)	Total Time (s)
Single Pillar	500	0.8 [1.2]	1.6 [0.2]	2.4 [1.2]
Double Slit	500	2.5 [2.6]	∼ 0.0 [∼0.0]	2.5 [2.6]
Rope Maze	1500	6.4 [3.9]	∼ 0.0 [∼0.0]	6.5 [3.9]
Repeated Planning	500	1.4 [0.9]	∼ 0.0 [∼0.0]	1.4 [0.9]

6.5.5 Computation Time

To verify the practicality of our deadlock prediction algorithm and VEB approximation, we gathered data comparing computation time for these components to the local controller by itself, and to using the Bullet simulator. Table 6.5 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 6.5: Local controller and deadlock prediction avg. 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

Table 6.6: Average computation time to compute the effect of a gripper motion.

	Bullet Simulation Time (ms)	VEB Propagation Time (ms)
Cloth	36.12	0.19
Rope	3.19	0.58

Table 6.6 shows a comparison between the average time needed to compute the VEB 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 to the time needed to forward propagate a VEB, 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 VEB, within the planner—this model, while not as accurate as a simulation, allows us to evaluate motions much faster.

6.6 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 6.5.1) with a dual-arm robot. It also shows that while our methods strong assumptions about the ability to perceive the deformable object in Section 6.1 (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 6.13).

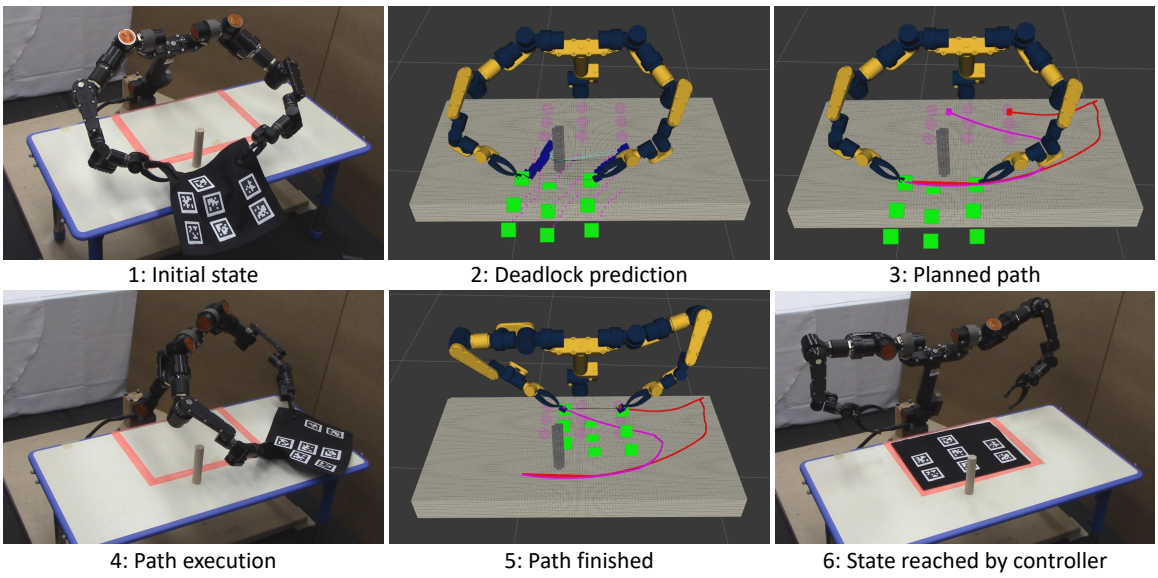


Figure 6.13: Cloth placemat task. The placemat starts on the far side of an obstacle and must be aligned with the pink rectangle near the robot.

6.6.1 Experiment Setup

6.6.1.1 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 DoF to be controlled and planned for ($\mathcal{Q} = \mathbb{R}^{16}$).

6.6.1.2 Cloth Perception:

The placemat is 0.33m $\times$ 0.46m which we discretize into a 3×3 grid. As tracking of deformable objects is a difficult problem, and out of scope of this paper, 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 7 points are tracked with AprilTags [72] and a Kinect V2 RGB-D sensor [73].

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 6.14). Denote z_i as the last observed position of point i, and denote t_i as 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:

$$\begin{aligned} \mathcal{P}(t) = \underset{\{p_i\}}{\operatorname{argmin}} \quad & \sum_i e^{-K_T(t-t_i)} \|p_i - z_i\|^2 \\ \text{subject to} \quad & \|p_i - p_j\|^2 \leq K_L d_{ij}^2 \;\; \forall i,j \text{ s.t. } i \neq j \;\; . \end{aligned} \tag{6.37}$$

K_T and K_L are task defined scale factors which we set to 1.5 and 1.0001 respectively for this task.

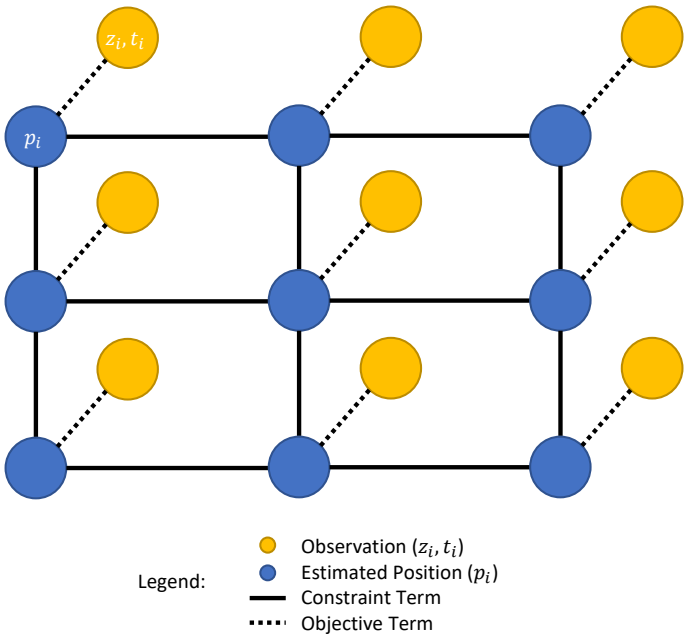


Figure 6.14: Constraint and objective graph for Eq. (6.37). Note that not all constraints are shown to avoid clutter; every estimated position has a constraint between itself and every other estimated position.

6.6.2 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 radians of their 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. Last, we also introduce planning restarts [81] 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.

Tables 6.7 and 6.8 show the planning and smoothing 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 6.15 shows that the planning time follows a "heavy tail" distribution typical of sampling-based planners.

Table 6.7: Planning statistics for the cloth placemat example, averaged across 100 trials. Standard deviation is shown in brackets.

Samples	States	NN Time (s)	Validity Checking Time (s)	Random Restarts	Total Time (s)
83041 [83677]	8438 [6182]	4.5 [4.9]	44.1 [44.5]	0.5 [0.9]	50.0 [50.9]

Table 6.8: Smoothing statistics for the cloth placemat example, averaged across 100 trials. Standard deviation is shown in brackets.

Iterations	Validity Checking Time (s)	Visibility Deformation Time (s)	Total Time (s)
500	3.6 [1.1]	0.1 [∼0.0]	3.6 [1.1]

Our overall framework is able to complete this task as shown in Figure 6.13. 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 neighbourhood, and then use the local controller to finish the task.

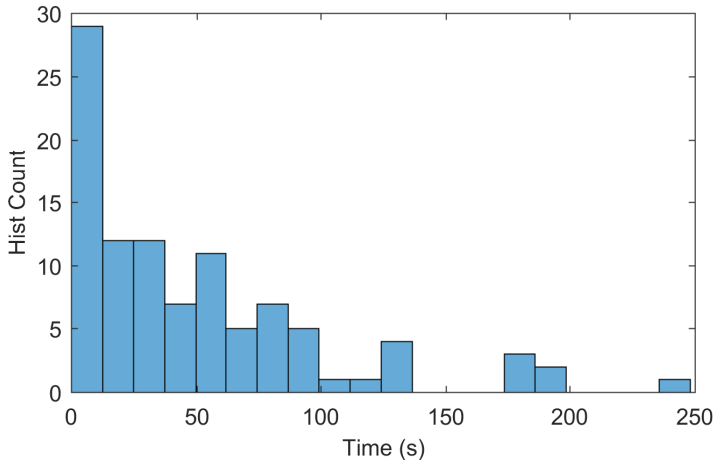


Figure 6.15: Histogram of planning times across 100 trials for the cloth placemat experiment.

6.7 Discussion

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.

6.7.1 Parameter Selection

There are several parameters in both the local controller and the global planner that can have a large impact on the performance of our method. In particular, if the local controller is prone to oscillations, 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 path due 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_v is not small, then nearest neighbour 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.

6.7.2 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 due to our approximation methods; the virtual elastic band we use for constraint checking and planning assumes (1) that 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 [28], 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 don't 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. Last, 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 which 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.

CHAPTER VII

Learning When To Trust Your Model

7.1 Introduction

Robot motion planning algorithms have been extremely successful for systems where the dynamics can be easily specified and efficiently evaluated. However, for tasks such as manipulation of deformable objects, the dynamics are very difficult to model [5] and usually require numerical simulation to evaluate. This simulation can be time-consuming and/or inaccurate. Including such simulations inside a planner can result in plans that take hours to compute [6].

Motivated by tasks where dynamics are difficult to specify and evaluate, we present a framework to plan in a reduced state space with simplified dynamics while biasing the planner to find plans that are likely to be feasible under the true dynamics. To do this, we define a function that maps from the true state space to a reduced state space as well as a dynamics model in the reduced space. We can then generate plans in the reduced space and roll them out on the true system offline to gather data on how the reduced and true dynamics correspond. Specifically, we find which transitions (i.e. state-action pairs) in the reduced state space produce reliable predictions as compared to rolling out the given action with the true dynamics.

After gathering a dataset where transitions are labeled as reliable or unreliable, we train a classifier to predict the reliability of a given transition. We then incorporate this classifier into an RRT-based planner by biasing the planner to reject transitions that are classified as unreliable. The resulting planner tends to find sequences of transitions that are likely to produce the desired outcome when the true dynamics of the system are applied, even though the planner plans with no explicit knowledge of the true dynamics.

This chapter presents both an abstract formulation of the problem of planning in a reduced state space with a classifier and how to apply this formulation to two

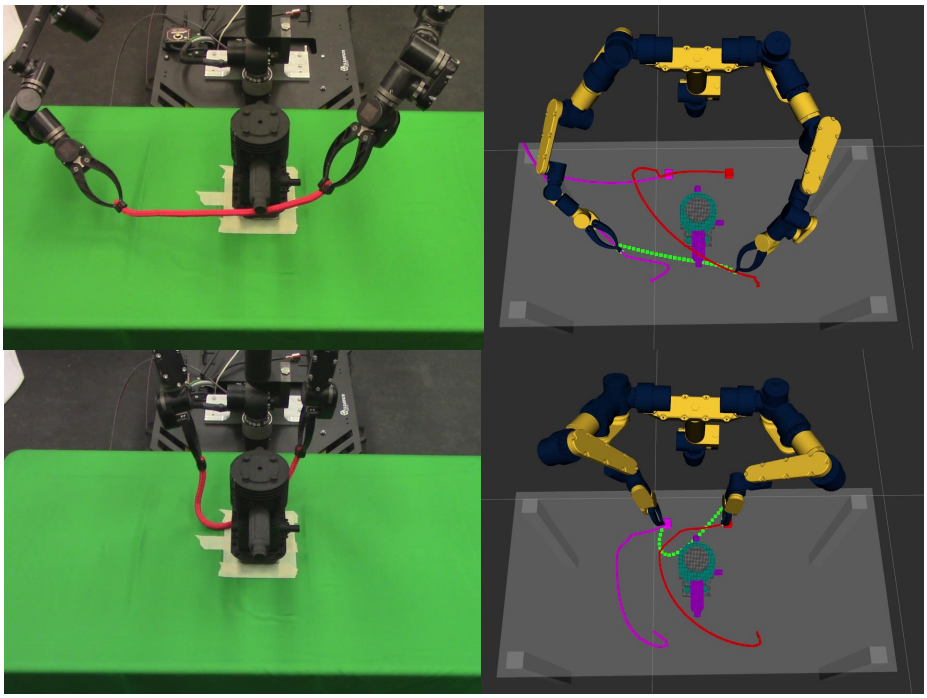


Figure 7.1: Top: a plan generated without using a classifier moves the rope under a hook and gets caught. Bottom: a plan generated with a classifier avoids this mistake, and reaches the goal.

systems. First, to illustrate our framework on a straightforward example, we consider a planar three-link arm with limited joint torque. Using a learned classifier for this system allows us to plan in configuration space (not considering dynamics) while avoiding transitions that cannot be accomplished with the limited torque. The second system focuses on rope manipulation tasks; we use a Virtual Elastic Band (VEB) (Section. 6.2.2.2 as the reduced dynamics model of the rope, as this has been shown to allow fast planning in difficult scenarios. However, this model assumes that there is no minimum length of the rope, which entails that the planner cannot detect cases where the slack material is caught on corners or hooks, preventing the motion plan from being completed because the caught object can overstretch (see Figure 7.1). Thus, we learn a classifier to bias the planner away from states where the object can be caught on an obstacle. The contributions of this chapter are:

1. A novel formulation of planning in reduced state spaces
2. A method to generate and label data for classification of transition reliability
3. Experiments demonstrating the efficacy of our framework for both the planar arm reaching and rope manipulation tasks

Our experiments suggest that we can learn a classification function for the reliability of transitions which improves the success rate of planning with the reduced model for both the planar arm and rope manipulation. Our simulation experiments considering rope manipulation in several challenging environments containing hooks demonstrate the classifier's ability to bias the planner away from unreliable transitions and to generalize over environments and rope lengths. Finally, to show a practical application of our work, we demonstrate our method running on a 16 DoF robot manipulating a rope near an engine assembly.

7.2 General Problem Formulation

We begin by formulating our problem in a system-agnostic way and then describe how to apply this formulation to planning for rope manipulation. Let the true system operate in a state space $\mathcal{X}$ with dynamics $x_{t+1} = f(x_t, u^x, \mathcal{O})$, where u^x is a command given to the system and $\mathcal{O}$ is the environment. We assume that the true state space has Markovian dynamics.

The problem we address in this work is how to find a sequence of N_e commands $\{u^x_1, \ldots, u^x_{N_e}\}$ to move from a start state x_0 to a goal state. The goal set is specified by the function $\texttt{Goal}_x : \mathcal{X} \rightarrow \{0, 1\}$, which returns 1 if a state is a goal and 0 otherwise. We thus to seek to solve the following:

$$\begin{aligned} \text{find} \quad & N_e, \{u^x_1, \ldots, u^x_{N_e}\} \\ \text{s.t.} \quad & \texttt{Goal}_x(x_{N_e}) = 1 \\ & x_t = f(x_{t-1}, u^x_t, \mathcal{O}),\ t = 1, \ldots, N_e \ . \end{aligned} \tag{7.1}$$

However, f may not be known in closed-form or it may be expensive to evaluate within a planner. Thus we cannot solve this problem by planning in $\mathcal{X}$ with the true dynamics.

To create a more tractable planning problem we define $\mathbb{B}$ to be a *reduced state space* and define a reduction function: $b = r(x, \mathcal{O})$. We do not assume that r is invertible. Dynamics in $\mathbb{B}$ are defined as $b_{t+1} = g(b_t, u^b, \mathcal{O})$ (note that u^b and u^x may be in different spaces). We treat the dynamics in this reduced state space as Markovian. $\mathbb{B}$, r, and g are user-defined. There is then an analogous goal function

for reduced states $\texttt{Goal}_b : \mathbb{B} \to \{0, 1\}$. The planning problem then becomes:

$$\begin{aligned} \text{find} \quad & N_e, \{u_1^b, \ldots, u_{N_e}^b\} \\ \text{s.t.} \quad & b_0 = r(x_0, \mathcal{O}) \\ & \texttt{Goal}_b(b_{N_e}) = 1 \\ & b_t = g(b_{t-1}, u_t^b, \mathcal{O}),\ t = 1, \ldots, N_e \ . \end{aligned} \tag{7.2}$$

Rather than making explicit guarantees on the relationship between f and g, we assume we have access to a rollout function $x_{t+1} = \Gamma(x_t, u^b, \mathcal{O})$, which outputs the next state when attempting to perform an action u^b given some controller for the system. We assume that Γ has built-in safety limits, so it will stop before violating a constraint (e.g. stopping before colliding with an obstacle). If Γ reaches a constraint boundary it will output the state on the boundary and will not violate the constraint. Γ is treated as a black box. The form of Γ may be known but even if it is, we assume it is too expensive to evaluate within the planner, otherwise there would likely be no need for the reduction. We assume we are able to gather data by executing Γ.

We will solve the problem in Eq. (7.2) using a motion planner that plans in $\mathbb{B}$. However, the plan we generate may not lead to the goal in execution because we may have lost information in r and/or g. Our approach is thus to bias our planner so that it avoids taking actions for which r and g are not reliable approximations of the behavior of the system. See Figure 7.2 for an overview of our framework.

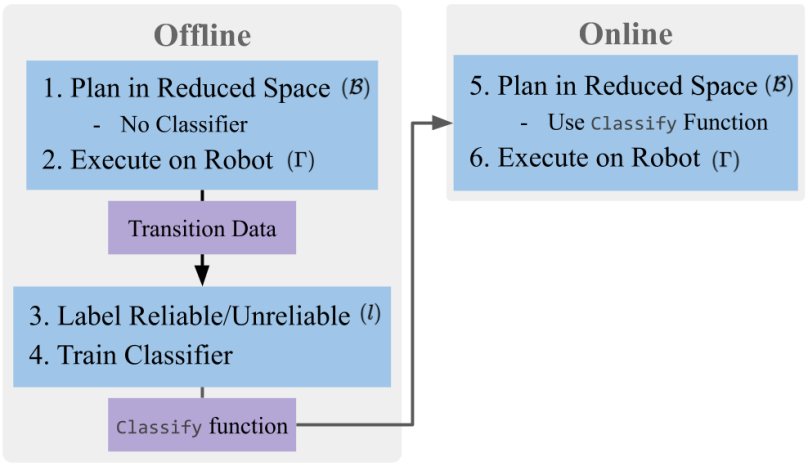


Figure 7.2: An outline of our framework. First, we plan and execute many control sequences to gather a dataset of transitions. These transitions are labeled according to a function l and used to train a classifier which predicts whether a transition is reliable given the model reduction. This classifier is used to bias the planner away from unreliable transitions.

7.3 Learning Transition Reliability

To bias the planner that plans in $\mathbb{B}$ we will learn a classifier that attempts to predict if a given transition $T^b = \langle b_t, u^b, \mathcal{O} \rangle$ will reliably succeed (e.g. not be stopped by a constraint boundary) when executed in environment $\mathcal{O}$. We thus wish to learn a function $\texttt{Classify} : \{T^b\} \rightarrow \{\texttt{Reliable}, \texttt{Unreliable}\}$, which outputs `Reliable` when performing this transition reliably succeeds under various starting x_0s and previous command sequences $u^b_{0:t-1}$, and `Unreliable` otherwise, in which case the planner should be biased not to use T^b.

Ideally, we would include x_0 and $u^b_{0:t-1}$ as input to the classifier. However, $\mathcal{X}$ may be arbitrarily high-dimensional (e.g. for a deformable object) and there may be an arbitrary number of previous commands before T^b, thus making the classifier very difficult to learn with a realistically-sized dataset. Instead we only consider T^b as input.

7.3.1 Data Generation and Labeling

To train our classifier, we need to gather a dataset of transitions and label them by whether the model reduction produces a reliable prediction. We would like to generate a training dataset of transitions in a similar way to how a planner would generate transitions, since this avoids distribution mismatch problems when planning. We therefore collect and label data from executed plans which we generate without a classifier. To do this, we run the planner without using `Classify` starting from some x_0. Planning generates a transition sequence $\mathbb{T}^b = \{T^b_t \mid t = 1, 2, ...\}$. We then execute the plan, which gives a ground-truth sequence of states $\tau^x = \{x_0, x_t = \Gamma(x_{t-1}, \mathbb{T}^b_t.u^b, \mathcal{O}) \mid t = 1, 2, 3, ..., |\mathbb{T}^b|\}$. We then reduce τ^x to the reduced state space producing $\tau^b = \{\tilde{b}_t = r(\tau^x_t, \mathcal{O}) \mid t = 1, 2, ...|\tau^x|\}$. For time t, let the *reduced dynamics prediction* be $\hat{b}_t = \mathbb{T}^b_t.b$ and the *rollout result* be $\tilde{b}_t = \tau^b_t$. Figure 7.3 summarizes the computation required to produce these variables.

To label the data we require a function that evaluates if the two predictions are meaningfully similar for the given system. Let the function $\texttt{Close} : \mathbb{B} \times \mathbb{B} \rightarrow \{0, 1\}$ return 1 when two reduced states are meaningfully similar and 0 otherwise. The environment $\mathcal{O}$ is also an input to `Close` but we omit it for brevity. We label the tth

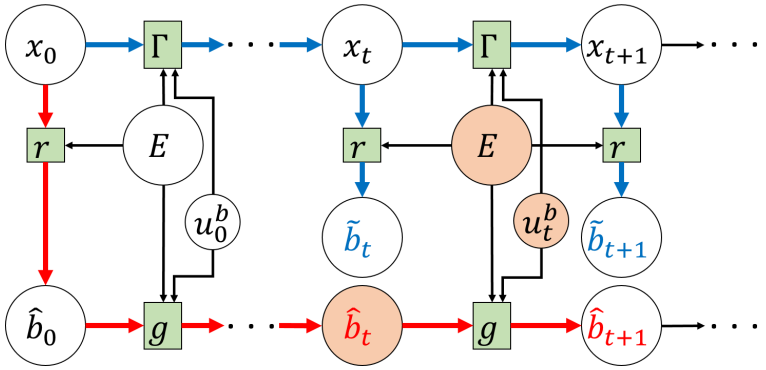


Figure 7.3: Circles represent variables and boxes represent functions. Orange: variables defining the tth transition. Red path: reduced dynamics prediction; Blue path: rollout result.

transition in $\mathbb{T}^b$ using the function l:

$$l(t, \mathbb{T}^b, \tau^b) = \begin{cases} \texttt{Reliable} & \textbf{if } \texttt{Close}(\hat{b}_t, \tilde{b}_t) \textbf{ and} \\ & \quad \texttt{Close}(\hat{b}_{t+1}, \tilde{b}_{t+1}) \\ \texttt{Unreliable} & \text{otherwise} \end{cases} \tag{7.3}$$

Intuitively, this function first checks if $\hat{b}_t$ $\tilde{b}_t$ are close. If they are, and $\hat{b}_{t+1}$ and $\tilde{b}_{t+1}$ are not close, then the transition is labeled `Unreliable` because the reduced dynamics prediction was inaccurate. If the starting states $\hat{b}_t$ and $\tilde{b}_t$ *are not* similar, then the rollout and reduced dynamics predictions have already diverged and we do not have a meaningful ground-truth label for this transition. To be conservative in our prediction, we label this transition `Unreliable`. If both $\hat{b}_t$, $\tilde{b}_t$ are close and $\hat{b}_{t+1}$, $\tilde{b}_{t+1}$ are close then r and g have performed well and we label the transition `Reliable`.

7.3.2 An Illustrative Navigation Example

To clarify the above framework and learning problem formulation, we describe an example system where the various functions and spaces can be easily visualized. Consider a car with state $x = [q, \dot{q}]$, where $q = [q_x, q_y, q_\theta]$ and control inputs $u^x = [u_a^x, u_\phi^x]$, which correspond to the acceleration and steering angle. $f(x_t, u^x, \mathcal{O})$ is the standard second-order car dynamics.

We define a reduced state using the function $b = r(x, \mathcal{O}) = x_q$ (i.e. we only consider position variables in the reduced space) and the controls to be $u^b = [\Delta x, \Delta y, \Delta\theta]$. The reduced dynamics are $b_{t+1} = g(b_t, u^b, \mathcal{O}) = b_t + u^b$.

Let the rollout function $x_{t+1} = \Gamma(x_t, u^b, \mathcal{O})$ be a method that uses a controller to

drive the car toward $r(x_t, E) + u^b$. Γ also checks if the car reaches the boundary of an obstacle in $\mathcal{O}$ and will return the state on the boundary if it does. $\texttt{Close}(\hat{b}_1, \tilde{b}_1)$ outputs 1 if the two reduced states are within a small Euclidean distance and 0 otherwise.

The task for the car is to drive to a given location while maintaining low speed and not colliding with obstacles. If we gather training data from this task domain we will find that when u^b drives the car toward an obstacle that is nearby, depending on the velocity at x_0, the car can hit the obstacle even though the lines between the planned b_t and b_{t+1} are collision-free for all t. Using only the planner, we would accept *all* transitions where the line from b_t to b_{t+1} is collision-free. However, the classifier will learn that it is better to avoid transitions that entail driving past nearby obstacles. Using the classifier's output to further prune transitions will restrict the planner to transitions that are more likely to succeed when executing Γ (see Figure 7.4).

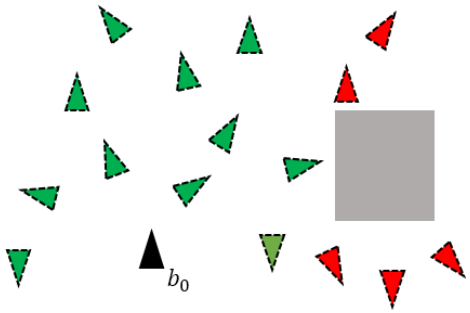


Figure 7.4: Illustration of desired prediction from a classifier. Dotted triangles indicate $\hat{b}_1$s from different u_0^b inputs. Green: Classifier predicts these transitions are `Reliable`. Red: Classifier predicts these transitions are `Unreliable`. Note that the line between b_0 and $\hat{b}_1$ is collision-free for all examples shown.

7.3.3 What can be learned

While we may produce a useful classifier for planning, it is important to know that there is a fundamental limitation on what can be learned by this approach because of the loss of information that may happen in r and/or g. Consider the following scenario: Let $b_0 = r(x_0^a, \mathcal{O}) = r(x_0^b, \mathcal{O}) = r(x_0^c, \mathcal{O})$, e.g. there are multiple states where the car is at a certain position but with a different velocity. If we apply reduced dynamics prediction for some u^b, we obtain $\hat{b}_1 = g(b_0, u^b, \mathcal{O})$. However, we if do rollouts we obtain three resulting states: $x_1^a = \Gamma(x_0^a, u^b, \mathcal{O}), x_1^b = \Gamma(x_0^b, u^b, \mathcal{O}), x_1^c = \Gamma(x_0^c, u^b, \mathcal{O})$, and then three reduced states: $\tilde{b}_1^a = r(x_1^a, \mathcal{O}), \tilde{b}_1^b = r(x_1^b, \mathcal{O}), \tilde{b}_1^c = r(x_1^c, \mathcal{O})$. It may be the case that $\texttt{Close}(\hat{b}_1, \tilde{b}_1^k)$ does not produce the

same result for $k = a, b, c$ (an example for the car system is shown in Figure 7.5). In terms of classification, this is a case of noisy labeling, and many methods have been devised to address this problem (e.g. SVM with slack variables).

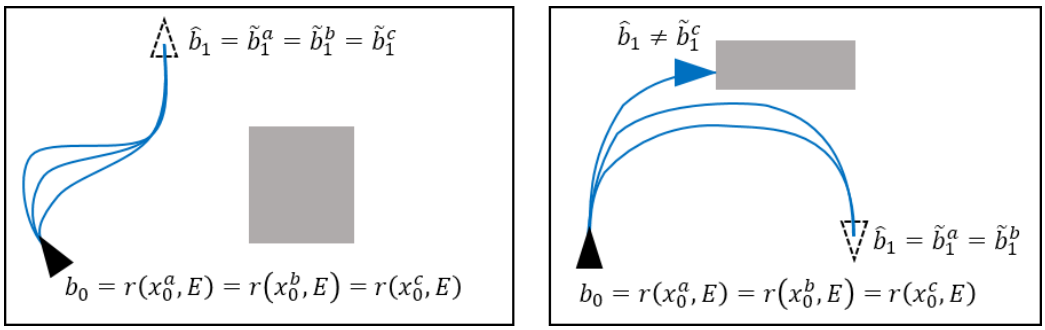


Figure 7.5: Illustration of the effect of information loss on the predictability of a transition. In both scenarios states with different velocities reduce to the same b_0. Left: A case where rolling out the same u^b from different initial velocities (blue) produces the same $\tilde{b}_1$ values, since the controller is robust to initial velocity in this case. Right: A case where rolling out the same u^b with different initial velocities produces different $\tilde{b}_1$ values. At high initial velocity (c) the controller cannot turn before reaching the obstacle.

However, if there are many noisy labels in the data, it means that r and g are not useful for this task domain. As a result the classifier will not make meaningful predictions and we would expect that it would provide no benefit over simply planning in $\mathbb{B}$. However, in our experiments with a planar arm and with rope manipulation we found that we do indeed see a benefit when using the classifier.

7.3.4 Using the Classifier in Planning

While it is possible to query every transition considered by the planner and reject all those that are classified as `Unreliable`, such a strategy would likely be overly optimistic about what the classifier has learned. In difficult scenarios the classifier may erroneously reject a set of transitions which is necessary to reach the goal, thus causing the planner to fail. Thus we accept transitions that are classified as `Unreliable` with a small probability determined by parameters k, a manually-specified constant, and p_{acc}, the validation accuracy of the classifier (see Algorithm 23). p_{acc} is included because we wish to be more permissive about accepting transitions when the classifier performs more poorly in terms of generalization.

While the above approach of incorporating classification can be applied to a wide range of planners, in this paper we focus on using RRT-based planners. An advantage of this approach for RRT-based planners is that we maintain the probabilistic

Algorithm 23 `CheckTransition`(T^b)

$b' \leftarrow g(T^b.b, T^b.u^b, \mathcal{O})$
if `Classify`(T^b) == `Reliable` **then**
    return b'
$a \sim U[0, 1]$
if $a < e^{-kp_{\text{acc}}}$ **then**
    return b'
return $\emptyset$

completeness properties of the planner by guaranteeing that any transition will be accepted with a non-zero probability (although the probability is small for transitions classified as `Unreliable`).

7.4 Application to a Torque-Limited Planar Arm

First, we demonstrate our framework on a 3-link arm that moves in the X-Z plane with gravity in the $-z$ direction. For this system we focus on the effects of including a classifier in the planner without using a reduction function. This allows us to disentangle the effects of model reduction and inaccurate dynamics.

For this experiment we use the MuJoCo simulator [82] as the ground truth dynamics. Each joint is controlled using the default position servo actuator available in MuJoCo. We convert the MuJoCo simulation to a quasistatic system by waiting for the arm to settle after a configuration is commanded (this is f). These experiments do not have any obstacles, so we omit $\mathcal{O}$ for brevity.

7.4.1 Problem Statement

Let $x \in \mathbb{R}^3$ be the true state of the system. Let $u^x = x_{des} \in \mathbb{R}^3$ be the commanded position of the arm at each timestep. Let $f(x_t, u_t)$ be the quasistatic dynamics defined by MuJoCo. We set torque limits τ_1, τ_2, τ_3 so that $\tau_1 \ll \tau_2 = \tau_3$. This means the first joint cannot support the weight of the arm when extended horizontally. As we are not doing a model reduction, $b = r(x) = x$. Commands in both spaces are also the same: $u^b = u^x$. The dynamics in $\mathbb{B}$ are purely kinematic: $b_{t+1} = g(b_t, u_t^b) = u_t^b$. These dynamics are fast to evaluate and therefore efficient for planning, but can result in plans which do not reach the goal configuration when executed (Figure 7.6). As there are no obstacles and $u^b = u^x$, the rollout function $\Gamma(x_t, u_t^b, \mathcal{O}) = f(x_t, u_t^x)$.

The planning problem is for the arm to reach a goal end-effector position. Using the true dynamics, this corresponds to Problem (7.1). However, since we assume the

true dynamics are not available, we seek to solve Problem (7.2) given the definitions of r and g above.

7.4.2 Data Collection, Labelling, and Training

To collect training data we initialized the system from random start configurations, planning to random goal configurations using RRT-Connect [83]. In practice these are straight lines in configuration space after smoothing the path. This generated a total of 231,815 transitions to use in training and validation. A randomly selected 20% of the data is held out for validation. We define $\texttt{Close}(\hat{b}_t, \tilde{b}_t)$ based on the Euclidean distance between configurations:

$$\texttt{Close}(\hat{b}_t, \tilde{b}_t) = \|\hat{b}_t - \tilde{b}_t\| < \alpha \tag{7.4}$$

with $\alpha = 0.075$.

For this planar arm, our classifier is a neural network that takes b and b' as input. The network has three hidden layers with 256, 128, and 64 hidden units respectively and a single output neuron. The hidden units use a Leaky ReLu activation [84], and the output uses a sigmoid activation. With this network we achieve 99% training accuracy and 99% validation accuracy within 4 epochs.

7.4.3 Planning and Results

To test the effect of using the learned classifier, we randomly generate 100 planning queries and evaluate how well generated plans perform when executed. Each planning query consists of a random start configuration in $\mathbb{R}^3$ and goal IK solutions for a random target point in $\mathbb{R}^2$. We only consider queries for which there is at least one IK solution where the controller can maintain the configuration of the arm within $\alpha = 0.075$ of the IK solution. For planning we use the OMPL [85] implementation of RRT-Connect, setting the probability scale factor k to 1, and p_{acc} to 0.99. The resulting path is post-processed using the default `simplifyMax` options. A plan is considered successful if the distance between the final configuration and any IK solution is less than a threshold β (Figure 7.6). Tests are performed on an i5-3570K @ 4.3 GHz. Planning and simplification takes approximately 2 ms without a classifier, and approximately 7 seconds with our classifier. For small β, using the classifier does not improve performance due to the steady state error inherent in the system. As β increases, we see that the planner that uses a classifier is able to successfully find

paths to all queries while the baseline is unable to succeed at some queries (see video at `https://www.youtube.com/watch?v=esgWD8Iqi34`).

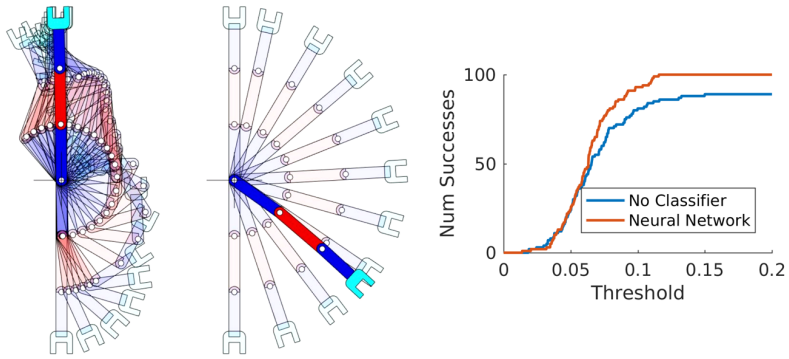


Figure 7.6: Left: Plan generated using the learned classifier to go from $[\frac{-pi}{2}, 0, 0]$ to $[\frac{pi}{2}, 0, 0]$. The plan avoids transitions which move the arm toward a horizontal position and successfully completes the task. Center: Plan generated without the classifier. The plan takes the arm to the horizontal position where it fails due to the torque limit. Right: Number of successes as success threshold β varies

7.5 Application to Rope Manipulation

We now present the application of our framework to rope manipulation, where we use both a reduction of the state space and an approximate dynamics model.

7.5.1 Problem Statement

Let the robot be represented by a pair of grippers with configuration $q \in SE(3)^G$. We assume that the robot configuration can be measured exactly. In this work we assume the robot to be a set of free floating grippers; in practice we can track the motion of these with inverse kinematics.

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 rope. The configuration of a rope 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 $\mathcal{O}$ 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 rope as quasi-static. The true state of the system is then $x = [q, \mathcal{P}]$ and $u^x = \Delta q$. Let f be a joint-space controller for the robot that stops when any of the following occur: 1) the grippers contact an object; or 2) the object stretches by more than a factor λ. Due to the difficulty of simulating rope physics, we do not assume we can execute f within a motion planner.

We wish to find a sequence of N_e commands $\{u_1^x, \ldots, u_{N_e}^x\}$ to move from a start state x_0 to a goal gripper configuration $q[g]$ such that each motion is feasible (this corresponds to Problem (7.1)). Note that this planning problem does not require bringing the rope to a specific configuration, which can often be done using local control *after* bringing the object to a desired area (as in the previous chapter). Because we do not have access to f, we cannot solve this problem by planning in $\mathcal{X}$ directly.

To make planning tractable we will perform a reduction and learn a classifier from data to predict when the reduction can be trusted. That classifier will then be used in a motion planner to bias it away from transitions that are not likely to be feasible under Γ.

7.5.2 Reduction

Chapter VI introduced the idea of a *virtual elastic band* (VEB) between the robot's end-effectors. This VEB represents the shortest path through the rope between the end-effectors. The band approximates the constraint imposed by the rope on the motion of the robot; if the end-effectors move too far apart, then the VEB will be too long, and thus the rope is stretched beyond a task-specified maximum stretching factor. Similarly, if the VEB gets caught on an obstacle and becomes too long, then the rope is also overstretched. By considering only the geodesic between the end-effectors, we are assuming that the rest of the rope will comply to the environment, and does not need to be considered when predicting overstretch. The VEB representation allows us to use a fast prediction method when planning, but does not account for the part of the material that is slack. Denote the configuration of a VEB (i.e. a sequence of points) as v. Then $b = [q, v]$ and can be generated by $b = r(x, \mathcal{O})$ for the reduction function defined in Chapter VI. The dynamics of a VEB, $g(b, u^b, \mathcal{O})$, are based on Quinlan's path deformation algorithm as presented in [77] (see Section 6.2.2.2). We also augment g to return $\emptyset$ when u^b causes the object to become overstretched. b is then propagated using g and $u^b = u^x$. Since the commands are the same, our rollout function Γ, which uses u^b, is equivalent to f. To find a path in $\mathbb{B}$ we must solve Problem (7.2).

We use the planner described in Chapter VI to solve this problem; this is an RRT-based planner designed for use with virtual elastic bands as part of the planning configuration space. This planner searches for a feasible path for the robot between a given start and goal configuration, while ensuring the VEB is never overstretched.

The VEB approximation choice favors speed over model accuracy; as a consequence, there are several issues that it does not address. Specifically, environments

with "hooks" can cause problems due to the approximation methods: The virtual elastic band assumes that there is no minimum length of the rope. This assumption means that the planner cannot detect cases where the slack material can get caught on corners or hooks, preventing the motion plan from being completed because the caught object can overstretch. To reduce the chances of this occurring we learn a classifier for T^b to predict if a given transition is either `Reliable` or `Unreliable` and bias the planner away from `Unreliable` transitions. We bias the planner using the `CheckTransition` function shown in Algorithm 23, which is used as an edge validity check in addition to the collision and overstretching checks described in Chapter VI.

7.5.3 Learning the Classifier

To learn the classifier we define the `Close` function for our domain to be:

$$\texttt{Close}(\hat{b}_t, \tilde{b}_t) = \Big(D(\hat{b}_t.v, \tilde{b}_t.v) < \alpha\Big) \wedge \texttt{FOH}(\hat{b}_t.v, \tilde{b}_t.v, \mathcal{O})$$

Where D computes the sum of the point-wise distances between the two virtual elastic bands, α is a constant, and `FOH` evaluates if the two bands are in the same first-order homotopy class [54]. We then apply labeling function l shown in Eq. (7.3) for each transition. We generate many plans using our planner (without the classifier) to produce the dataset (see Section 7.6.2).

For our classifier we use a neural network based on *VoxNet* [86], a network for classifying objects from a voxel grid. The network consists of two 3D convolutional layers (filter size 5, 3 and stride 2, 1 respectively) with max pooling followed by two fully-connected layers. All layers have ReLU activations except the output layer, which has a sigmoid activation.

The input for our classifier consists is a three-channel binary voxel grid, with channels $\langle \mathcal{O}, b, b' \rangle$, where $b' = g(b, u^b, E)$. Each voxel grid is $32 \times 32 \times 32$, and is constructed by checking for occupancy at every cell center. An example of the voxelized representation is shown in Figure 7.7.

7.6 Rope Manipulation Experiments

To characterize the planner performance with the classifier, we designed seven simulation scenarios where the rope must be moved from one side of the scene to the other, along with one physical experiment for real-world validation. All simulation experiments were conducted in the open-source Bullet simulator [71], with additional

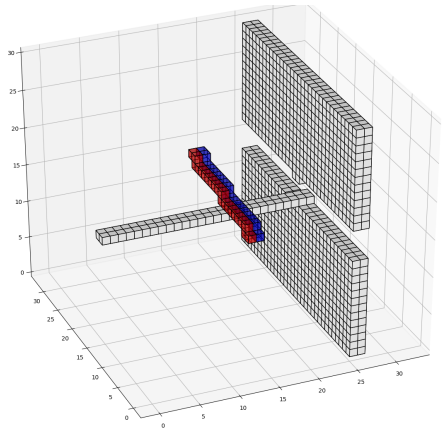


Figure 7.7: Input to the VoxNet classifier is a 3-channel voxel grid, where white is the local environment, red is the pre-transition band, and blue is the post transition band. Positions outside the bounds of the environment are marked as occupied in the local environment channel.

wrapper code developed at UC Berkeley [80]. The rope is modeled as a series of small capsules linked together by springs. We emphasize that our method does not have access to the model of the rope or the simulation parameters. The simulator is used as a "black box" for testing. All tests are performed using an i7-7700K 4.2 GHz CPU with 32 GB of RAM. For all experiments we set $\gamma^{\max}$ to 1.15, and α to 0.5.

7.6.1 Scenarios

Each scenario involves moving the rope past one (or more) hooks, while the grippers traverse a narrow slit (Figure 7.8).

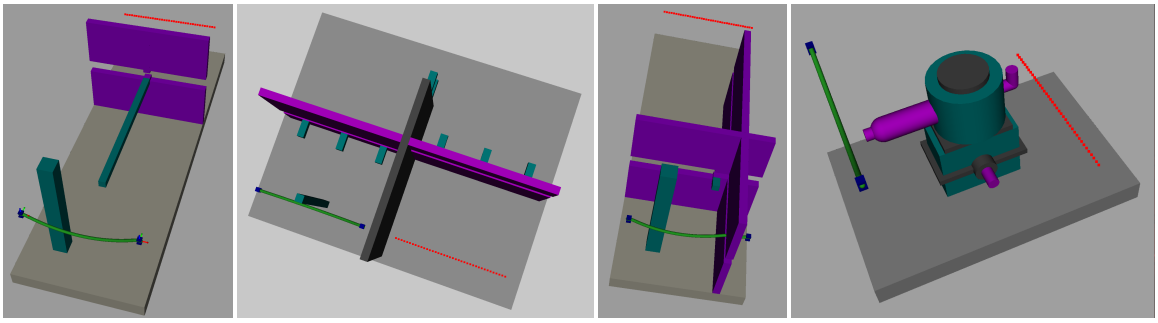


Figure 7.8: The rope is shown in green, with the grippers shown in blue. The target area for the grippers is shown in red. Walls with narrow slits for the grippers are shown in purple. Hooks and other obstacles are shown in dark cyan. Left: Simple Hook; Center Left: Multi Hook; Center Right: Complex Hook; Right: Engine Assembly

Simple Hook: In the Simple Hook environment, the end of the hook is not near

any obstacles, thus the planner does not need to deal with the end of the hook and the narrow slit at the same time. We test four variants of the simple hook environment: Short, Regular, Long, Very Long corresponding to the lengths of the rope which are 0.55m, 0.87m, 1.13m, and 1.59m respectively.

Multi Hook: In the Multi Hook environment, the rope must pass through three series of hooks, and two narrow slots before reaching the red region on the far side of a solid wall. The rope has length 0.87m.

Complex Hook: In the Complex Hook environment, the grippers are forced to pass on opposite sides of a small hook, while also passing through a narrow slit. The rope has length 0.87m.

Engine Assembly: In the Engine Assembly environment, we seek to move the grippers from one side of an engine model [87] to the others avoiding two hooks on the front and back of the engine. The rope has length 0.87m. The engine assembly environment is shown in Figure 7.8.

Physical Robot: In the Physical Robot environment, we attempt the engine assembly task on a physical 16 DoF robot shown in Figure 7.1 with a 3D printed model of the engine and a rope of length 0.46m.

7.6.2 Data Collection

To collect training data, we ran the planner without any classifier on the Simple Hook Regular scene repeatedly, generating a total of 4190 plans from many different starting locations. This generated a total of 562,177 transitions to use in training and validation. This training set is generated only from the Simple Hook environment using the Regular length rope. We emphasize that we use the classifier trained on this data for planning in *all* test environments.

7.6.3 Training the Classifier

VoxNet is trained using the Adam optimizer [88] with initial learning rate of 5×10^{-4}. We use a learning rate decay of 0.8 every 4 epochs. Since the dataset set is imbalanced, with 32% of the examples labelled as unreliable, and 68% labelled as reliable, we use a weighted random sampler to make all minibatches balanced. A randomly selected 10% of the data is held out for validation. Our minibatch size is 32, and we train for 100 epochs. We use the binary cross-entropy loss function during training. Training took approximately 16 hours on a Tesla V100-SMX2 GPU. The VoxNet classifier achieved 99% accuracy on the training set and 91% accuracy on the

validation set.

7.6.4 Planning Results

To evaluate the planning performance when using the classifier, we generated 30 plans using the classifier on each test environment and compare the success rate and planning time to planning without a classifier. A success is when executing the plan results in a final gripper configuration which is within a small tolerance of the goal gripper configuration. If, for example, the rope gets caught on a hook and prevents the grippers from reaching the goal, the trial is marked as a failure. Results are shown in Table 7.1. We set k to 10 and p_{acc} to 0.91. Our results show that using a classifier improves the success rate of the planner over not using a classifier in all tested scenarios, but the effect is less prominent on the Multi Hook environment. The use of a classifier does lead to longer planning time, partially due to extra computation when checking each edge, as well as making the planning problem harder to solve. Section 7.7 discusses this in more detail. Example results can be found in the video at `https://www.youtube.com/watch?v=esgWD8Iqi34`.

7.7 Discussion

We found that the use of a neural network classifier to evaluate the reliability of a model approximation can be an effective way to improve the success rate of a planner for rope manipulation. When using the classifier in the planner as a hard constraint ($k = \infty$), some starting configurations would cause the classifier to reject all transitions from that state, leading to planning failure. An interesting approach to setting k would be to treat it similar to temperature as done in T-RRT [89]. Despite training the rope manipulation classifier on only a single rope and environment (the Simple Hook with Regular length rope), we found that the classifier was able to generalize and lead to improved planning performance with both different lengths of rope and more complex environments.

It is important to note that while our method can find more feasible plans than not using a classifier, we may be making the planning problem more difficult. In the planar arm example, a straight line in configuration space between start and goal solves the planning problem under g but may not be a feasible plan under Γ. To avoid this mismatch, the classifier restricts the set of transitions that can be used, but doing so may induce a narrow passage effect, which leads to longer planning times.

A major challenge in this work is determining how to label the data in a way

Table 7.1: Planning statistics, averaged over 30 trials

Environment	Metric	Classifier	
		None	VoxNet
Simple Hook - Short	Success rate	18/30	**30/30**
	Planning time (s)	0.6	14.6
	Smoothing time (s)	1.0	5.4
Simple Hook - Regular	Success rate	20/30	**29/30**
	Planning time (s)	3.7	17.3
	Smoothing time (s)	4.0	6.5
Simple Hook - Long	Success rate	23/30	**29/30**
	Planning time (s)	6.8	51.9
	Smoothing time (s)	6.4	8.0
Simple Hook - Very Long	Success rate	18/30	**27/30**
	Planning time (s)	12.4	15.4
	Smoothing time (s)	15.6	11.4
Multi Hook	Success rate	9/30	**13/30**
	Planning time (s)	11.5	44.1
	Smoothing time (s)	22.0	30.0
Complex Hook	Success rate	0/30	**20/30**
	Planning time (s)	3.7	28.7
	Smoothing time (s)	27.0	23.4
Engine	Success rate	1/30	**10/30**
	Planning time (s)	4.8	2.0
	Smoothing time (s)	0.3	4.1

that will lead to good performance. In particular, by including transitions where the reduced dynamics predictions have already diverged in our dataset and labelling these transitions as `Unreliable`, the classifier learns that interaction with objects is frequently poorly modelled by the VEB. This leads the planner to avoid contact with the environment when possible, but many interesting tasks involving deformable objects will explicitly require interaction with other objects.

One interesting point is that increased classification accuracy does not necessarily lead to better planning performance. We experimented with different classifiers during development, and although VoxNet had the best classification accuracy on the validation set and usually the best performance, other classifiers performed equivalently or better for planning in some environments.

CHAPTER VIII

Discussion and Future Work

This dissertation presented a framework for autonomously manipulating deformable objects, focusing on methods that can be used without a time-consuming modelling phase, and without high-fidelity simulation. We have focused on techniques and algorithmic choices that enable the robot to learn from performing tasks, improving the robot's ability to manipulate deformable objects and perform interesting tasks. Our contributions in modelling and control do not involve learning directly, but they form part of the basis from which the robot is able to acquire experience from which to learn. These methods could also be combined with other techniques such as residual physics [90] to integrate analytical methods with learning based approaches.

Our initial goal for integrating learning into the planning process was to do so in an online fashion; i.e., after making a plan that leads to getting caught on a hook we would like to avoid repeating a similar mistake when generating the next plan. The key challenge here is how to evaluate similarity; what does it mean for two transitions to be similar, particularly when accounting for the local environment? By using a classifier directly in state transition space we were able to avoid making the same mistake on a given hook, but this approach does not generalize to different obstacle configurations. Being able to learn quickly from a small number of mistakes and to generalize that experience to new scenarios is a skill that will enable robots to move from the lab to unstructured environments like the home.

Chapter V introduced the idea of using multiple models for control, switching between them as the task progressed; we experimented with a similar idea during the planning process. There are three key challenges that need to be overcome to make this approach workable. First, we need a way to acquire multiple models that might be useful in various situations. The representations used by these models would need to be *interchangeable*, i.e. in order to switch to a different model at a later timestep we need to be able to convert between the representations used by

each model. Second, we need a way to evaluate the performance of a model in a way that allows us to make decisions during planning about which model to use. If we are using multiple models during the planning processes, disentangling the effects of choices made early in the planning process from their long term effects later in the plan is difficult. In Chapter VII we addressed this credit assignment problem for a single model by introducing a conservative model-accuracy based supervised classification problem; this approach has the potential to apply to a multi-model approach, but the effects of switching between models while planning needs investigating. Third, it's not clear how and when to use predictions from multiple models during planning; Phillips-Grafflin and Berenson [91] introduced a clustering approach to managing the exponential growth of the planning tree if multiple predictions are considered at each timestep which, could be a promising direction of research.

One of the limiting assumptions of this dissertation is the representation of the deformable object as a set of points, accurately and directly sensed. By making these assumptions we are requiring far more from our perception systems than they are generally capable of. For example if we consider a pile of clothes, being able to disambiguate between each clothing item that the robot can see, as well as estimating the configuration of every item is prohibitive. Instead, we may want to consider task specific representations that enable a higher level of reasoning over actions rather than a point-wise representation. It is an open question how to come by these representations, or how to know when to use a given representation for the task at hand. In this work, we used two representations, one for local control purposes and another for global planning. By using these different representations we were able to write algorithms focused on different parts of an overall task, and then combine these algorithms together into a single framework. Other representations may allow for a different way of thinking about deformable object manipulation that may be better suited to other tasks.

In this dissertation we hand-designed the decisions of when to use each representation and the associated method for manipulating the deformable object. Extending this framework to a broader set of abilities and representations naturally leads to approaches such as task and motion planning techniques which reason over discrete choices (for example, which representation/algorithm to use) along with continuous variables (for example, where to grasp). A great deal more work is needed in this area as we work towards integrating deformable object manipulation tools into a general robotics solution.

Some of the limitations imposed by our interleaving framework may be addressed

by closing the loop on the execution of plans generated by RRT-EB. By monitoring the deformable object while a plan is executed we can quickly determine when the deformable object has diverged from the planned path, but it is an open question how to recover from a divergence. Can we define a local controller that reasons about how close our VEB model reduction is representing the true configuration of the deformable object and close the loop at this level? While this may be possible in some circumstances, in general we believe that something like our interleaving framework will always be needed in order to respond to unmodelled aspects of problems and continue with a given task.

One of the key differences between rigid objects verses deformable object manipulation is that deformable objects are pliable; we explicitly took advantage of this when designing our VEB model reduction and planner. That said, we have not taken advantage of that compliance as an explicit manipulation tool; the directional rigidity model in Section 3.3 can enable a controller to take advantage of interactions with the environment, but it does so without explicitly reasoning about it. It's not clear how to take advantage of such contact in a principled fashion; is including contact in the modelling process enough? Is it better to use contact as a way of creating funnels for chaining actions or controllers? Can we exploit contact in a way that helps reduce the scope of the problem that a task and motion planning framework needs to solve?

In this dissertation, we have presented a framework and series of algorithms, supported by successful simulations and lab robot performance, that successfully answers two primary questions that motivated this research. Firstly, our robot successfully reduced the amount of time-consuming modelling and data collection necessary to perform tasks with rope and cloth by interleaving planning and control. By using different representations for each component, we are able to perform these tasks without high-fidelity modelling or simulation. Secondly, we integrated ideas from decision theory and machine learning into classical motion planning and control approaches in order to overcome the limitations of using low-fidelity modelling and approximation methods by learning from the robot's experience gained while using these approximations to perform manipulation tasks.

However, some questions remain arising from the inherent difference between rigid body manipulation and deformable object manipulation, generating a number of worthwhile future research endeavors, as presented above. Loosely grouped as related to planning and online learning, they include topics such as learning to avoid repeating similar mistakes, evaluating and defining "similar", integrated task planning and motion planning, and how to achieve a modicum of intuition such that the

robot is able to understand what success looks like when encountering new tasks or environments. Topics grouped around the idea of modelling include improved use of a wider variety of types of models, real-world sensing of the deformable object and how it is represented, and how to take better advantage of the compliant nature of the object. Control-related questions include how to better incorporate interleaving, quicker discovery of divergence from the desired path, and of course the ongoing challenges of dexterous manipulation.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *International Journal of Robotics Research (IJRR)*, vol. 35, pp. 528–564, Apr 2016.

[2] M. Brady, "Artificial intelligence and robotics," *Artificial Intelligence*, vol. 26, no. 1, pp. 79 – 121, 1985.

[3] Z. Wang, X. Li, D. Navarro-Alarcon, and Y.-h. Liu, "A Unified Controller for Region-reaching and Deforming of Soft Objects," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 472–478.

[4] S. H. Huang, J. Pan, G. Mulcaire, and P. Abbeel, "Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 878–885.

[5] N. Essahbi, B. C. Bouzgarrou, and G. Gogu, "Soft Material Modeling for Robotic Manipulation," in *Applied Mechanics and Materials*, vol. 162, April 2012, pp. 184–193.

[6] Y. Bai, W. Yu, and C. K. Liu, "Dexterous Manipulation of Cloth," *Computer Graphics Forum*, vol. 35, no. 2, pp. 523–532, 2016.

[7] D. Berenson, "Manipulation of deformable objects without modeling and simulating deformation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 4525–4532.

[8] D. M^{c}Conachie and D. Berenson, "Estimating model utility for deformable object manipulation using multiarmed bandit methods," *IEEE Transactions on Automation Science and Engineering (T-ASE)*, vol. 15, no. 3, pp. 967–979, July 2018.

[9] F. Khalil and P. Payeur, "Dexterous robotic manipulation of deformable objects with multi-sensory feedback – a review," in *Robot Manipulators, Trends and Development*. InTech, 2010, ch. 28, pp. 587–621.

[10] P. Jiménez, "Survey on model-based manipulation planning of deformable objects," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 154–163, Apr. 2012.

[11] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey," *International Journal of Robotics Research (IJRR)*, vol. 37, no. 7, pp. 688–716, 2018.

[12] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun, "Discrete elastic rods," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, vol. 27, no. 3, p. 1, Aug. 2008.

[13] W. Rungjiratananon, Y. Kanamori, N. Metaaphanon, Y. Bando, B.-Y. Chen, and T. Nishita, "Twisting, Tearing and Flicking Effects in String Animations," in *Motion in Games*, ser. Lecture Notes in Computer Science, J. M. Allbeck and P. Faloutsos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 7060, pp. 192–203.

[14] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of SIGGRAPH*. ACM Press, Jul. 1998, pp. 43–54.

[15] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun, "Efficient Simulation of Inextensible Cloth," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, vol. 26, no. 3, 2007.

[16] S. F. F. Gibson and B. Mirtich, "A survey of deformable modeling in computer graphics," Mitsubishi Electric Research Laboratories, Tech. Rep., 1997.

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

[18] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pp. 49–54, 2002.

[19] G. Irving, J. Teran, and R. Fedkiw, "Invertible finite elements for robust simulation of large deformation," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pp. 131–140, 2004.

[20] P. Kaufmann, S. Martin, M. Botsch, and M. Gross, "Flexible simulation of deformable models using discontinuous Galerkin FEM," in *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2008.

[21] A. Petit, V. Lippiello, G. A. Fontanelli, and B. Siciliano, "Tracking elastic deformable objects with an rgb-d sensor for a pizza chef robot," *Robotics and Autonomous Systems*, vol. 88, no. C, pp. 187–201, Feb. 2017.

[22] A. Borum, D. Matthews, and T. Bretl, "State estimation and tracking of deforming planar elastic rods," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4127–4132.

[23] T. Bretl and Z. McCarthy, "Quasi-static manipulation of a kirchhoff elastic rod based on a geometric analysis of equilibrium configurations," *International Journal of Robotics Research (IJRR)*, vol. 33, no. 1, pp. 48–68, 2014.

[24] J. Lang, D. Pai, and R. Woodham, "Acquisition of elastic models for interactive simulation," *International Journal of Robotics Research (IJRR)*, vol. 21, no. 8, pp. 713–733, 2002.

[25] A. M. Cretu, P. Payeur, and E. M. Petriu, "Neural network mapping and clustering of elastic behavior from tactile and range imaging for virtualized reality applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 9, pp. 1918–1928, Sept 2008.

[26] D. Navarro-Alarcon, Y.-h. Liu, J. G. Romero, and P. Li, "On the visual deformation servoing of compliant objects: Uncalibrated control methods and experiments," *International Journal of Robotics Research (IJRR)*, vol. 33, no. 11, pp. 1462–1480, jun 2014.

[27] D. Navarro-Alarcon and Y. Liu, "Fourier-based shape servoing: A new feedback method to actively deform soft objects into desired 2-d image contours," *IEEE Transactions on Robotics (T-RO)*, vol. 34, no. 1, pp. 272–279, Feb 2018.

[28] Z. Hu, P. Sun, and J. Pan, "Three-dimensional deformable object manipulation using fast online gaussian process regression," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 2, pp. 979–986, April 2018.

[29] S. Hirai and T. Wada, "Indirect simultaneous positioning of deformable objects with multi-pinching fingers based on an uncertain model," *Robotica*, vol. 18, no. 1, pp. 3–11, Jan. 2000.

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

[31] J. Smolen and A. Patriciu, "Deformation Planning for Robotic Soft Tissue Manipulation," in *2009 Second International Conferences on Advances in Computer-Human Interactions*, Feb. 2009, pp. 199–204.

[32] D. Navarro-Alarcon, H. M. Yip, Z. Wang, Y. Liu, F. Zhong, T. Zhang, and P. Li, "Automatic 3-d manipulation of soft objects by robotic arms with an adaptive deformation model," *IEEE Transactions on Robotics (T-RO)*, vol. 32, no. 2, pp. 429–441, April 2016.

[33] O. Maron and A. W. Moore, "Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 1994.

[34] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, "Automating model search for large scale machine learning," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2015.

[35] P. Whittle, "Restless Bandits: Activity Allocation in a Changing World," *Journal of Applied Probability*, vol. 25, p. 287, 1988.

[36] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2/3, pp. 235–256, 2002.

[37] J. Gittins, K. Glazebrook, and R. Weber, *Multi-armed Bandit Allocation Indices*. John Wiley & Sons, 2011.

[38] S. Agrawal and N. Goyal, "Analysis of Thompson Sampling for the multi-armed bandit problem," *Conference on Learning Theory*, 2012.

[39] J. Langford and T. Zhang, "The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2008.

[40] A. Slivkins, "Contextual bandits with similarity information," *Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 2533–2568, Jan. 2014.

[41] O.-C. Granmo and S. Berg, "Solving Non-Stationary Bandit Problems by Random Sampling from Sibling Kalman Filters," in *Trends in Applied Intelligent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 199–208.

[42] S. Pandey, D. Chakrabarti, and D. Agarwal, "Multi-armed bandit problems with dependent arms," in *International Conference on Machine Learning (ICML)*. New York, NY, USA: ACM, 2007, pp. 721–728.

[43] M. Saha, P. Isto, and J.-C. Latombe, "Motion planning for robotic manipulation of deformable linear objects," in *Proceedings of the International Symposium on Experimental Robotics (ISER)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 23–32.

[44] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[46] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

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

[48] E. Anshelevich, S. Owens, F. Lamiraux, and L. Kavraki, "Deformable volumes in path planning applications," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 2290–2295.

[49] F. Lamiraux and L. E. Kavraki, "Planning Paths for Elastic Objects under Manipulation Constraints," *International Journal of Robotics Research (IJRR)*, vol. 20, no. 3, pp. 188–208, Mar. 2001.

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

[51] R. Gayle, M. Lin, and D. Manocha, "Constraint-Based Motion Planning of Deformable Robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 1046–1053.

[52] M. Moll and L. E. Kavraki, "Path Planning for Deformable Linear Objects," *IEEE Transactions on Robotics (T-RO)*, vol. 22, no. 4, pp. 625–636, 2006.

[53] O. Roussel, A. Borum, M. Taïx, and T. Bretl, "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)*, May 2015, pp. 3116–3121.

[54] L. Jaillet and T. Siméon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *International Journal of Robotics Research (IJRR)*, vol. 27, no. 11-12, pp. 1175–1188, 2008.

[55] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, no. 3, pp. 273–290, Jun. 2012.

[56] P. Brass, I. Vigan, and N. Xu, "Shortest path planning for a tethered robot," *Computational Geometry*, vol. 48, no. 9, pp. 732–742, 2015.

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

[58] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research (IJRR)*, vol. 20, May 2001.

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

[60] T. Kunz and M. Stilman, "Kinodynamic RRTs with Fixed Time Step and Best-Input Extension Are Not Probabilistically Complete," in *Algorithmic foundations of robotics XI*. Springer, Cham, 2015, pp. 233–244.

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

[62] J. Schulman, J. Ho, C. Lee, and P. Abbeel, "Learning from demonstrations through the use of non-rigid registration," in *Springer Tracts in Advanced Robotics*, vol. 114. Springer International Publishing, 2016, pp. 339–354.

[63] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[64] E. Banijamali, R. Shu, M. Ghavamzadeh, H. Bui, and A. Ghodsi, "Robust Locally-Linear Controllable Embedding," *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, Oct 2017.

[65] B. Jia, Z. Hu, J. Pan, and D. Manocha, "Manipulating highly deformable materials using a visual feedback dictionary," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[66] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine, "SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning," *International Conference on Machine Learning (ICML)*, pp. 7444–7453, 2019.

[67] G. Sutanto, N. Ratliff, B. Sundaralingam, Y. Chebotar, Z. Su, A. Handa, and D. Fox, "Learning Latent Space Dynamics for Tactile Servoing," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[68] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 3, pp. 2407–2414, 2019.

[69] A. Wang, T. Kurutach, K. Liu, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," in *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[70] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994, vol. 29.

[71] E. Coumans, "Bullet physics library," *Open source: bulletphysics.org*, 2010.

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

[73] T. Wiedemeyer, "IAI Kinect2," https://github.com/code-iai/iai_kinect2, Institute for Artificial Intelligence, University Bremen, 2014 – 2015, accessed July 1, 2018.

[74] Gurobi, "Gurobi optimization library," *Proprietary: gurobi.com*, 2016.

[75] M. C. Koval, J. E. King, N. S. Pollard, and S. S. Srinivasa, "Robust trajectory selection for rearrangement planning as a multi-armed bandit problem," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[76] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of Computation*, vol. 19, no. 92, pp. 577–593, 1965.

[77] S. Quinlan, "Real-time modification of collision-free paths," Ph.D. dissertation, Department of Computer Science, Stanford University, 1994.

[78] D. Hsu, J. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry & Applications*, vol. 09, no. 04n05, pp. 495–512, August 1999.

[79] H. Alt and M. Godau, "Computing the Fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 05, no. 01-02, pp. 75–91, mar 1995.

[80] Robot Learning Lab, "Simulation environment with Bullet physics," https://github.com/rll/bulletsim, University of California, Berkeley, 2012, accessed July 2, 2012.

[81] N. A. Wedge and M. S. Branicky, "On heavy-tailed runtimes and restarts in rapidly-exploring random trees," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2008, pp. 127–133.

[82] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.

[83] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 995–1001.

[84] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *International Conference on Machine Learning (ICML)*, 2013.

[85] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics and Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[86] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[87] J. Tumber, "Engine assembly," https://grabcad.com/library/engine-assembly-11, 2016, accessed August 28, 2019.

[88] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[89] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based rrt for path planning in continuous cost spaces," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 2145–2150.

[90] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," in *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[91] C. Phillips-Grafflin and D. Berenson, "Planning and resilient execution of policies for manipulation in contact with actuation uncertainty," in *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.