Data-Efficient Robotic Manipulation of Deformable One-dimensional Objects with Unreliable Dynamics

by

Peter Mitrano

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in The University of Michigan
2024

Peter Mitrano

pmitrano@umich.edu

ORCID iD: 0000-0002-8701-9809

ACKNOWLEDGEMENTS

Thanks to all the people who made this dissertation possible – To my advisor Dmitry, to my friends and cohort-mates Tom and Johnson who have been with me every step of the way, and to all my other lab mates and peers in Michigan Robotics. Thank you to all the teachers who helped me find my way to research. Thank you to my family for the many relaxing and re-energizing trips and holidays. Finally, I would like to acknowledge my partner Andrea Sipos for her intellectual contributions, for supporting me, and for bringing me so much happiness.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

If I were to ask you to fold a towel, you would probably do it without much thought or difficulty – but how? How did you know where the towel was? Did you think about how the towel would move or deform, and how you'd react if the towel didn't move as you predicted? Or what if I asked you to fold it using your feet, could you do it?

These questions hopefully help explain why a robot, which lacks the intuition and experience of an adult human, might find this task challenging. A robot may need specific computational answers to all of these questions and more. In more technical terms, a robot needs to perceive, predict, and plan. Many methods for perceiving, predicting, and planning have been proposed, and they can be effective for some tasks in specific environments, but we have not yet achieved sufficiently fast and accurate methods to perform complex tasks like pruning a plant, sewing sutures, or installing cable harnesses – at least not outside carefully controlled settings.

This dissertation takes a step to increase the abilities of robots to manipulate deformable objects. More specifically, deformable objects like ropes, hoses, and cables. I focus on two problems, our inability to accurately predict in all scenarios how a DOO will move or deform (called unreliable dynamics), and the cost of collecting training data in robotic manipulation. I present a system that can quickly learn to manipulation the DOO despite limited data and imperfect predictions, as well as plan grasps and regrasps when the robot gets stuck.

CHAPTER I

Introduction

1.1 Introduction

One of the most common assumptions in robotic manipulation is that the object being manipulated is rigid. That is, when the object moves, the entire object moves without deforming. In contrast, a deformable object like a towel will change shape when one part of the towel is moved, such as when grasped and moved by a robot. The evolution of the state of the object over time, given the robot's actions, is called the dynamics, and planning for manipulation relies heavily on accurate dynamics models. For instance, when the robot plans how to move its arm to fold a towel, the dynamics model might predict the shape of the towel. Assuming rigidity simplifies the dynamics considerably, and so planning with rigid models is efficient and sometimes sufficient. However, many of the objects we want robots to manipulate are not actually rigid. For example: clothing, food, wires, hoses, thread, foams, composites like carbon fiber, paper products, fluids, granular materials, or even people. Even thick steel bars may not be accurately described as rigid if the task is to cut or grind them. Without rigidity, planning for manipulation becomes difficult because, at present, we do not have dynamics models that are accurate for a wide enough range of states, actions, or objects.

Instead, we have dynamics models that are reliable in some cases but not others. This is true for both physics-based analytic models, such as FEM or mass-spring-damper models, and for learned models, such as neural networks. For physics-based models, the model may have incorrect parameters for friction, damping, or stiffness. It may be completely lacking certain physical phenomena like torsional friction, air-resistance, or plastic deformation. It may also be intentionally simplified in order to make the computation faster, which can be essential for planning and control. Learning dynamics models from real world data can mitigate this issue because the learned dynamics will at least be accurate in the situations in which we have collected training data. But when there is a lack of relevant training data or poor generalization, the learned dynamics may still be unreliable. Real world data collection is costly, can lack diversity, and doesn't scale easily. Thus, the problem of unreliable dynamics models is not yet solved simply by learning the model from real world data. Therefore, this dissertation is motivated by the research question "How can robots use inaccurate models and limited real-world data to plan to manipulate deformable objects?". By studying this question, I seek to expand the capabilities of robots to new kinds of manipulation and new kinds of objects.

This thesis addresses the problems of unreliable dynamic models when real-world data collection is expensive. While most prior work handles these challenges with fast replanning and/or exhaustive training in simulation, I propose methods for learning where a dynamics model is accurate and avoiding inaccurate regions in planning. I focus specifically on deformable one-dimensional objects (DOOs), which includes objects like rope, hoses, cables, string, or potentially objects like cooked spaghetti or plant stems. These objects are, in some ways, the simplest type of deformable object, because their state can be compactly represented as a sequence of points, and they exhibit less self-occlusion than other deformable objects.

To make progress on the problem of learning and planning with unreliable dynamics, I leverage knowledge of physics, topology, and the fact that we do not need accurate dynamics models of the entire state-action space in order to accomplish a given task. To incorporate physics, I make extensive use of physics simulators as well as design heuristics and cost functions based on our knowledge of contact, gravity, and friction. To incorporate topology, I develop a topological signature that catogorizes different grasps to make planning more efficient. To avoid modeling the entire state-action space, I focus on regions of the state-action space which are useful for the desired tasks.

This thesis makes the following contributions, organized into four chapters:

- Chapter II: Learning a classifier to predict where a dynamics model is accurate, and what to do when the model is inaccurate. This work was published in Science Robotics, May 2021 [87].
- Chapter III: Data augmentation to enable online-learning of the classifier using only several hours of real world data. This work was published in RSS 2022 [86].

- Chapter IV: Online adaptation that focuses on easy-to-model dynamics, which enables fast sim2real adaptation. This work was published in ICRA 2023 [88].
- Chapter V: A topological signature and planning method for regrasping that expands DOO manipulation capabilities to new robot morphologies and new tasks. This work is under review for ICRA 2024.

1.2 Related Work

In this section, I review recent work related to this thesis. First, I briefly overview related work on robotic manipulation of deformable one-dimensional objects (DOOs), since that is the focus of this thesis. The next two sections of related work are devoted to the two challenges my work addresses – unreliable dynamics and data efficiency.

1.2.1 Manipulation of Deformable One-dimensional Objects

This thesis focuses on model-based manipulation planning, so we begin the related work with a discussion of the dynamics models used for DOO manipulation planning.

1.2.1.1 Dynamics for DOO manipulation

Before considering the dynamics, we first need to discuss the state representation for the DOO. By far the most common representation is a set of points in $\mathbb{R}^3$, where each point p_i has an index $i \in [0, N)$, and where the number of points N is chosen a priori and is fixed. However, some works use raw images as their state [33, 92, 130, 94, 33, 43]. In addition to the state representation, one must also choose an action representation, such as a delta-position of points that are grasped by the robot [76, 75, 43], delta-poses of the grippers [105], velocities of the grasped points [72], or wrenches applied to the object [11, 67]. In this thesis, we use the common point representation of DOO state, and explore both delta-position and joint velocity action representations.

Some dynamics models for DOOs are derived from first-principles [121, 67, 11], others are learned directly from data [33, 141, 131], and some are a mix of both [72, 73, 42]. For image-based prediction or for systems with high-dimensional states or complex dynamics, neural networks are a popular choice [6, 33, 92, 47, 43]. These models are often trained on datasets of random actions, using the mean squared prediction error as the training objective.

1.2.1.2 Planning for DOO manipulation

These dynamics models are then used to create plans for how to manipulate the object using a variety of methods such as A* [130], RRT [47], the cross entropy method (CEM) [40], and gradient-based optimization [116]. The choice of planner typically depends on the type of manipulation or the task being performed. For example, MPC methods like CEM or random-shooting are often paired with fast dynamics models for dexterous manipulation, since rapid replanning with inaccurate dynamics can be better than long-horizon planning [33, 93, 92]. In contrast, if the task requires large motions through regions with narrow passages, then an RRT or A* planner may be preferred. In this thesis, we use both RRT and MPC planning methods, since they complement each other well.

1.2.2 Unreliable Dynamics Models

In cases where we cannot assume our dynamics models will always be accurate, prior work has proposed several ways of quantifying the reliability of a dynamics model, as well as different ways to use that in planning.

1.2.2.1 Quantifying Model Reliability

In [64, 65] the error of the dynamics is learned using data coming from the true dynamics. [84] proposes training a classifier for model reliability, where the ground-truth label is a combination of thresholding the observed error, and comparing the first-order homotopy of the true versus predicted states. [64] introduced Model Deviation Estimators (MDE) which estimate a continuous scalar error values, and which we use in this thesis. Another approach is to quantify uncertainty in the dynamics due to lack of data [66, 23, 57], known as epistemic uncertainty. Intuitively (and sometimes literally) this means evaluating the distance to the training data, and treating states or state-action pairs which are far from any training data as unreliable. Finally, [82] proposed another notion of reliability based on the expected reduction in task-error, called utility, rather than the difference in state.

1.2.2.2 Using Model Reliability Estimates in Planning

There are several ways to use estimates of model reliability in planning. [130] and [47] both avoid regions where the learned model may have significant error. With MDEs, avoiding unreliable regions can be done by penalizing the cumulative or final error as predicted by the MDE [64], or a binary notion of reliability can be used to reject transitions in an RRT planner [84]. A related problem is also addressed in [125], which makes local adjustments in response to inaccurate predictions encountered in execution.

If a dynamics model has a known probabilistic transition model, belief-space planning can be used [54, 101]. However, meaningful predictive uncertainty distributions are difficult to estimate for novel scenarios, so these methods have not yet been applied to DOO manipulation.

1.2.3 Learning Dynamics from Limited Amounts of Data

The above works focus on quantifying and managing the unreliability of the dynamics model. Various learning techniques have been proposed to try to increase the model's reliability. However, the main limitation for these learning methods is the lack of large and diverse real world datasets. The problem of learning better models from limited data is important for robotics applications and has received significant attention as researchers have tried to apply deep learning to robotics [145, 62].

1.2.3.1 Adapting Dynamics Models

Given a class of dynamics models which have a few physical parameters, such as spring-mass or FEM models, one approach is to use real world observations to more accurately estimate system parameters (e.g. system-identification or active-learning) [108, 3, 31, 94]. For example, [91] uses system identification for deformable objects. These methods assume that there exists a set of system parameters which explain the observations. They also rely on collecting diverse data which disambiguates related parameters like mass and friction, which makes these methods sensitive to how the data is collected and may require complex data collection setups.

Another approach to improve accuracy is to first pre-train the dynamics on large datasets generated in simulation. Generating large amounts of diverse data is practical in simulation, but the challenge is that simulations may not be accurate to or representative of the real world. This difference has been called the *sim2real gap*, and many sim2real methods have been proposed to address this [100, 100, 154, 74, 21].

Curriculum learning and transfer learning methods can be used to transfer models trained in simulation to the real world [7, 115, 123]. Since transfer works best when the difference is small, curriculum learning methods create intermediate problems of increasing difficulty by changing the labels or the task. Curriculum learning has been successful in classification, reinforcement learning, and machine translation [63, 149, 146, 102]. Prior work has also used estimates of similarity between the source and target system to guide adaptation, for instance by selecting training data most similar to the source system [115] or selecting the most useful source domain for a given target domain [26].

Another method is domain randomization, which uses random variations of conditions during training to enable the model to be robust to that type of variance during test time [77]. Some methods go further, and iteratively refine the noise distribution by comparing simulations to rollouts executed in the real world [22, 68]. These methods can be used to adapt a dynamics model learned in one environment to another environment, but they require knowledge of how parameters can vary between the source and the target domains. For example, one may need to choose which parameters may vary (mass, stiffness, damping, friction) as well as mean and variances [22]. Our approach to adaptation does not adapt simulation parameters, but instead adapts a neural network dynamics model.

Existing work on adaptation, curriculum learning, and domain randomization can fail in the case where the simulation and real world dynamics differ significantly in some regions but not in others. This is often the case for unreliable dynamics of DOOs, since some regions of state-action space have very difficult to model dynamics (contact with environment, self-contact, knots, etc.), but other regions are simple (free-space). We address this specific kind of adaptation problem in Chapter IV.

1.2.3.2 Data Augmentation

One way to make better use of limited data is to use data augmentation. Data augmentation is the process of creating additional training examples by modifying existing ones. Data augmentation has been applied to many machine learning problems, from material science [97] to financial modeling [29] (see [49, 32, 110] for several surveys). It is especially common in computer vision [110, 111, 27, 69, 8], and is also popular in natural language processing [32, 78]. In these fields the data is often in standardized data types—images, text, or vectors of non-physical features (e.g. prices). Each of the data types can be used for a wide variety of tasks, and various data augmentations have been developed for various pairings of data type and task.

However, problems in robotic manipulation use other formats of data, such as point clouds or object poses, and may consist of time-series data mixed with time-invariant data. The data augmentation method I propose in Chapter III fills this gap, and is designed specifically for data of the types most prevalent in robotics.

In contrast to engineering augmentations based on prior knowledge, another body of work uses unsupervised generative models to generate augmentations [124, 97, 29]. Typically, these methods train a model like an Auto-Encoder or Generative Adversarial Network (GAN) [37]) on the data, encode the input data into the latent space, perturb the data in the latent space, then decode to produce the augmented examples. These methods can be applied to any data type, and handle both regression and classification problems. However, they do not incorporate prior knowledge, and only add small but sophisticated noise. In contrast, we embed prior knowledge about the physical and spatial nature of manipulation, and as a result can produce large and meaningful augmentations, at the cost of being less generally-applicable.

1.2.3.3 Small Models and Feature Engineering

Although adaptation and data augmentation are popular approaches to make better use of limited data, there are other methods that have been proposed for data-efficient learning, both in general and specifically for robotic manipulation. We highlight a few important ones here, but note that these are all complementary to model adaptation and/or data augmentation.

The most common technique is simply to pick a low-capacity model class, such as linear models or very small neural networks [96, 151]. Alternatively, prior work has also developed heuristics/priors specific to robotics [53] which can be used as objectives during training. Another extremely useful technique is to engineer the state or action representations to include certain known invariances. For instance, a standard technique in dynamics learning methods is to represent the input positions in a local frame as opposed to a world frame, to encode translation invariance [72, 45, 153]. There are also methods for learning these kinds of invariances [151].

1.2.4 Grasping and Regrasping

With rigid grasping, the challenge is primarily in achieving a stable [80, 79, 85, 89, 30], with grasp functionality being a secondary focus [17, 104]. With deformables, stability is less often a concern, and functionality (what you can do with that grasp) is the main challenge. Works on cloth smoothing or folding [43, 76, 138] use pick-and-place primitives with a single manipulator, which is too restrictive for many tasks. In [147], a dual arm manipulator autonomously dresses a mannequin. Their method for grasp planning is based on learned visual models of the garment, and only considers grasps near keypoints such as the elbow or shoulder. The methods in [129, 95, 141] plan grasps on the DOO, but they assume the rope is planar (flat on a table), use one manipulator, and do not consider obstacles for the manipulator. [112] describes a method that produces pick, place, and sliding paths in the configuration space without explicit task planning. However, this method does not address underactuated kinodynamic systems such as DOOs. [107, 84, 131, 114, 144, 143] all address manipulation planning for DOOs assuming the grasp is fixed, which is complementary to grasping and regrasping.

1.2.5 Topology and Knot Theory in Manipulation Planning

Topology and homotopy have been used in path planning for flying and driving robots [13, 14], as well as tethered robots [48]. [48] operates only in 2D and [50] considers an approximation of homotopy for 3D path planning. [13] introduces a simple-to-compute and exact signature for characterizing the homotopy of 3D paths with respect to 3D obstacles with holes in them, called the h-signature. We build on [13] to define the $\mathcal{G}_L$-signature presented in Chapter V.

Prior work on knot typing and untying has also applied knot theory to DOO manipulation [127, 128, 106, 132, 38, 118, 119]. These methods use planar crossing representations, which projects a curve into a specified plane and counts the sequence and type of crossings. [106] used this method for robotic knot tying, and extended this to tying around obstacles by specifying connections between obstacles and the DOO. However, these methods do not consider how the manipulator effects the topology, and fail in some cases with non-planar obstacles. A method for threading surgical needles was proposed in [133], but uses floating grippers and does not address planning for the robots' arms or obstacles, and is limited tight-tolerance insertion tasks.

CHAPTER II

Learning Where to Trust Unreliable Dynamics

This first chapter addresses the questions “When should we trust a dynamics model?” and “What do we do if the model is unreliable at the current state?”. I introduce a formal problem statement for planning with unreliable dynamics, and formalize learning reliability as a binary classification problem. I then propose methods for learning this classifier and using this classifier in planning. I also define what it means to be in need of recovery in terms of the classifier. Using this definition, I then propose a method for learning a recovery policy for these states using the data already collected for the classifier.

2.1 Introduction

Control and motion planning methods that rely on models to predict the outcomes of potential actions are ubiquitous in robotics. But whether these models are analytical, simulated, learned, or implicit within a policy, they are only as valid as the assumptions used to create them. When encountering a real-world unstructured environment, these assumptions may be violated, rendering a model unreliable [130]. What is worse, estimating how erroneous our models are is difficult, as methods that predict uncertainty distributions based on training data [23, 66] don’t account for novel scenarios, e.g. when new types of constraints are introduced. This is especially problematic if these constraints are not easily represented [55] in the model’s state space. Consequently, the inability to generate meaningful predictive uncertainty distributions for novel scenarios precludes the use of belief-space planning techniques [134, 2, 10].

Roboticists rarely address the unreliable model problem directly; instead, we often resort to high-frequency re-planning, hoping to compensate for model errors online [33, 61]. But this kind of approach assumes that the erroneous model is somehow likely to produce useful short-horizon actions. While it may not be possible to create universally-valid models of complex high-degree-of-freedom systems such as deformable objects, this does not preclude using imperfect models to perform useful tasks. For example, we do not need to know all the friction and stiffness properties of a rope to drag it along the ground. The key questions, which have, until now, received very little attention, are *1) When should we trust a model?* and *2) What do we do if the robot is in a state where the model is unreliable?* Addressing these questions is paramount if we wish to deploy robots in unstructured environments such as homes and industrial sites, where conditions change frequently, and it may not be possible to gather large datasets and re-learn accurate dynamics after every change.

This chapter tackles the above two questions in the context of learning dynamics models for the manipulation of rope-like objects among constraints such as obstacles. A key challenge in this domain is that the behavior of the rope when in contact is extremely difficult to predict, as it is heavily influenced by the rope's often non-uniform friction and stiffness properties. These properties are not only difficult to model, they are also difficult to estimate from observation. Thus, the hypothesis at the core of our work is that it is much better to learn to predict where a useful (but limited) model is reliable than to attempt to learn a model which is reliable everywhere. Specifically, in the context of rope manipulation, we propose a two-phase learning process, where we learn a useful model of rope dynamics assuming constraints such as obstacles are not present. Then, given limited data of the rope's interaction with obstacles, we can learn a classifier that predicts when the learned model is reliable. We then use this classifier in motion planning for novel tasks to bias the planner away from regions of state space where the model cannot be trusted. We emphasize that we do not attempt to learn the dynamics of the rope in contact from this limited dataset. In fact, we find that attempting to learn these dynamics yields poor results.

While the above methods allow us to perform useful tasks with rope despite being unable to predict dynamics on constraints boundaries, we are still faced with the question of what to do when the rope strays into a part of state space where the learned model is unreliable. This may occur because of error in execution, an external disturbance, or simply by starting the task in a state from which prediction is unreliable. Our approach for overcoming this problem is to first use our classifier to detect when this has occurred, and then to recover—i.e. execute a series of actions that bring us back to a region where the learned model is reliable. While random actions can eventually lead us to this region, we find that it is more efficient to use a

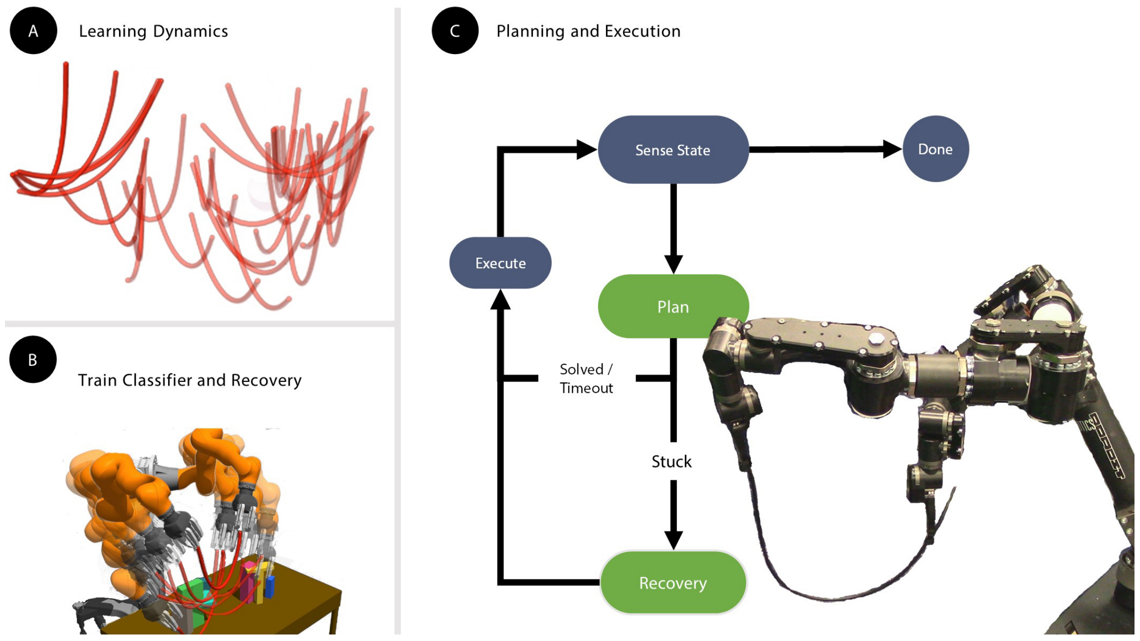

Figure 2.1: Overview of the proposed method. Two data collection phases (A,B) are used to collect training data for learning dynamics, a classifier for where these dynamics are accurate, and a recovery model what for actions to take when no accurate dynamics predictions can be made. These learned components are used in an iterative process of planning, replanning, and recovery (C). We apply this method to a number of deformable objects tasks, including a dual arm robot manipulating an automotive hose.

recovery policy (also learned from the limited dataset) to determine which actions are likely to improve model reliability. After reaching the region where our predictions are reliable, we can launch our planner to move to the goal. An overview of our learning and execution frameworks is shown in Figure 2.1.

2.2 Problem Statement

Here we present the formal problem addressed by our method. Let the state space of the system be $\mathcal{S}$ and the action space be $\mathcal{A}$. The true dynamics are $f(\mathcal{E}, s^t, a^t) \rightarrow s^{t+1}$ produces the next state s^{t+1} given the environment $\mathcal{E}$, state s^t, and action a^t. We consider the feasible discrete-time motion planning problem, which informally means finding a sequence of actions that take the system from a start configuration s^0 to a goal region $\mathcal{G} \subset \mathcal{S}$. In general, 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 with the true dynamics f.

Instead, we consider the challenge of planning with an incomplete model of the dynamics h. Since these dynamics will sometimes be inaccurate, we introduce the model-error requirement (MER) to reason about where they can be trusted. The MER is a constraint in the planning problem, to ensure that our plan only contains predictions from our dynamics h that are δ-close to the true dynamics f. The model-error is defined for a given state-action (s^t, a^t) using a distance function in state space dist, and is shown in Equation (2.1).

$$\text{dist}(\hat{s}^{t+1}, s^{t+1}) = \text{dist}(h(\mathcal{E}, \hat{s}^t, a^t), f(\mathcal{E}, s^t, a^t)) \tag{2.1}$$

Using this, we define the MER itself as $\text{dist}(\hat{s}^{t+1}, s^{t+1}) < \delta$. Thus, the planning problem is

$$\begin{aligned} \text{find} \quad & N, a^0, \dots, a^{N-1} \\ \text{subject to} \quad & \hat{s}^{t+1} = h(\mathcal{E}, \hat{s}^t, a^t) \quad t \in [0, N) \\ & \text{dist}(\hat{s}^{t+1}, s^{t+1}) < \delta \quad t \in [0, N) \\ & \hat{s}^N \in \mathcal{G} \end{aligned} \tag{2.2}$$

2.2.1 Classifier

We cannot evaluate the MER directly during planning because it requires the true future state s^{t+1}. In planning, we only know the environment, actions, and predicted states $(\mathcal{E}, \hat{s}^t, a^t, \hat{s}^{t+1})$. Consequently, we need to evaluate the MER using only the information known in planning. This can be posed as the following binary classification problem:

$$\begin{aligned} \texttt{INPUT} \quad & \mathcal{E}, \hat{s}^t, a^t, \hat{s}^{t+1} \\ \texttt{LABEL} \quad & \text{dist}(\hat{s}^{t+1}, s^{t+1}) < \delta \end{aligned} \tag{2.3}$$

Let the classifier which solves this problem be $g(\mathcal{E}, \hat{s}^t, a^t, \hat{s}^{t+1}) \rightarrow \{0, 1\}$. For training this classifier, `LABEL` can be computed using the actual s^{t+1} recorded during data collection (Section “Phase Two Data Collection”). A diagram illustrating the inputs to the classifier and its labels is shown in Figure 2.2. Finally, given dynamics h and classifier g, we can approximately solve Problem (2.2) using motion planning (see Section “Planning With a Learned Model and Classifier”).

We note that the MER is conservative in the sense that we only need the final predicted $\hat{s}^N$ and actual state s^N to be close. Unfortunately, reasoning about only the final state in the MER would be a constraint on the entire trajectory. Such a constraint is neither tractable to learn nor amenable to planning in our scenarios. Thus, we enforce the MER for every action taken by the planner.

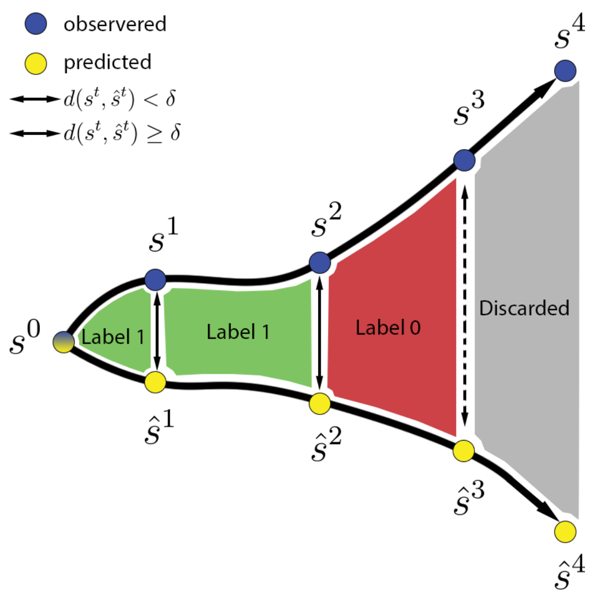

Figure 2.2: Converting a trajectory into examples for training the classifier. Each large box represents a transition. In the first transition, the final states s^1 and $\hat{s}^1$ are close, so the transition is labeled 1 (accurate). For the third transition, the initial states s^2 and $\hat{s}^2$ are close, but the final states s^3 and $\hat{s}^3$ are far, so this transition is labeled 0 (inaccurate). In the final transition, the initial states are far, so this transition is discarded.

2.2.2 Recovery

With this definition of the MER and the classifier, we also formally define what it means to be stuck, i.e. in need of recovery. This can be written as a function $r(\mathcal{E}, s^t) \rightarrow \{0, 1\}$ which determines whether a given state, environment pair can be escaped while enforcing the MER:

$$r(\mathcal{E}, s^t) = \begin{cases} 0 & \exists\, a \in \mathcal{A} \text{ for which } \mathrm{dist}(\hat{s}^{t+1}, s^{t+1}) < \delta \\ 1 & \text{otherwise} \end{cases} \tag{2.4}$$

Again, we cannot compute $r(\mathcal{E}, s^t)$ directly online, as it requires knowing the actual effect of executing actions. Instead, we will use an approximation to this function. Once we know if a state is in need of recovery, we can either launch the planner (if not) or perform recovery actions (if so).

2.2.3 State Definition

In this chapter, we focus on rope manipulation tasks with $N_{\mathrm{g}} = 1$ or $N_{\mathrm{g}} = 2$ grippers; the state of each gripper is a point in $\mathbb{R}^3$. We assume that the robot end

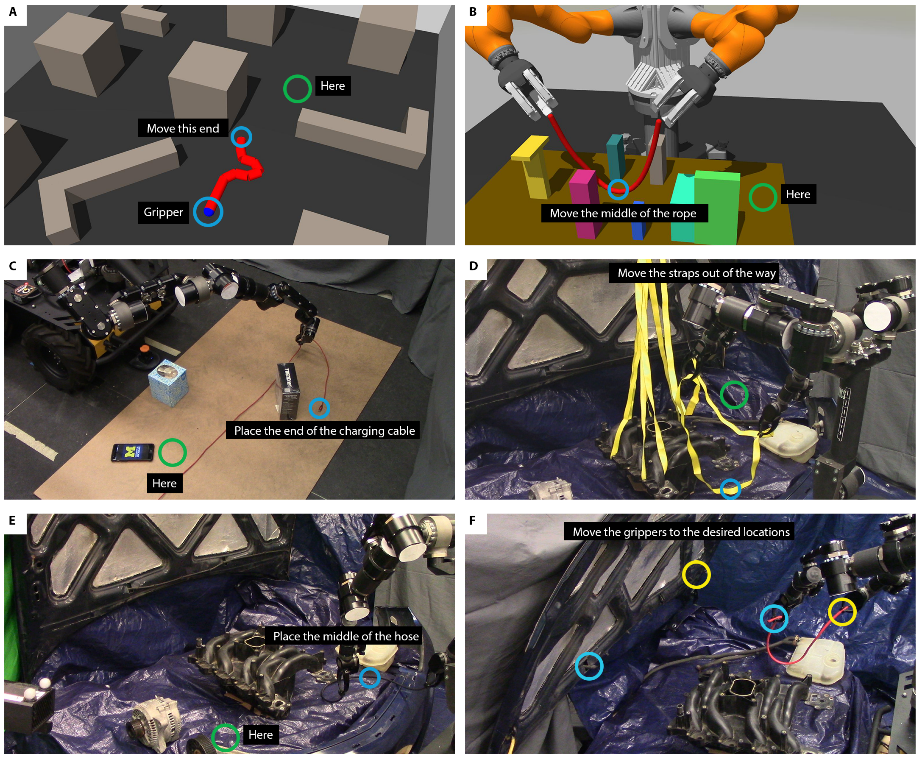
A
Here
Move this end
Gripper
B
Move the middle of the rope
Here
C
Place the end of the charging cable
Here
D
Move the straps out of the way
E
Place the middle of the hose
Here
F
Move the grippers to the desired locations

Figure 2.3: Pictures of our simulation and real robot scenarios. (A) Simulated Rope Dragging. (B) Simulated Tabletop Rope Manipulation. (C) Retrieving a Charging Cable. (D) Removing Lifting Straps. (E) Moving a Hose. (F) Preparing to Install a Hose. Annotations show examples of goals for various tasks our method can complete.

effectors are rigidly attached to the object. The configuration of the rope is a set of N_{rl} points in $\mathbb{R}^{3N_{\text{rl}}}$. The state s is then a vector of the positions of the gripper(s) and the points along the rope. The dimension of the state space is thus $N_d = 3(N_{\text{g}} + N_{\text{rl}})$.

2.3 Methods

2.3.1 Data Collection for Learning the Dynamics

Our first phase of data collection is used for training our model of the unconstrained dynamics. This involves sampling random actions and recording the observed states to form trajectories $[s^0, a^0, s^1, \dots, a^{N_t-1}, s^{N_t}]$. In all our experiments, we use $N_t = 10$. We sample actions randomly, choosing target positions which are within 0.1 m of the current gripper position(s). Put another way, we sample actions which are changes in position in a ball around the current position. We repeat the previously sampled change in position with 80% probability, which creates larger, more consistent motions and gives better coverage of our environments. During this phase, we do not reset the rope's position between trajectories.

In this phase, we also want to avoid activating physical constraints during data collection. This is done by removing obstacles, and by restricting the actions taken during data collection. For rope dragging, this only means removing any obstacles. For dual arm rope manipulation, this means removing obstacles, removing the arms entirely, and preventing the rope from overstretching by limiting the distance between the grippers. Although removing the arms would not be possible in the real world, it seems possible to design a conservative set of actions which would similarly ensure that the rope does not interact with the arms, or the arms with each other. At a high level, the purpose of this is to simplify the dynamics by avoiding activating any physical constraints during this phase.

For rope dragging, we collected 6,144 trajectories containing 10 steps, of which 1,536 are reserved for validation and testing. For dual arm rope manipulation, we collected 2,048 trajectories containing 10 steps, of which 512 are reserved for validation and testing.

2.3.2 Learning the Unconstrained Dynamics

Once we have collected our dataset of trajectories, we train our dynamics network. This network is a two layer fully connected neural network which takes in a state s^t and action a^t and predicts a change in state $\Delta\hat{s}^{t+1}$. A diagram showing this architecture is shown in Figure 2.5 and discussed in more detail in Section 2.3.7. This model can be used to make multistep predictions by feeding the predicted state back into the model. The loss is the combined prediction error for all time steps in the trajectory, and is shown in Equation (2.5).

$$\begin{aligned} MSE(s^0, a^0, \ldots, a^{N_t-1}) &= \frac{1}{N_t}\sum_{t=1}^{N_t-1} \left\|\hat{s}^t - s^t\right\|^2 \\ \hat{s}^0 &= s^0 \\ \hat{s}^{t+1} &= h(\mathcal{E}, \hat{s}^t, a^t) \end{aligned} \tag{2.5}$$

2.3.2.1 Incorporating Uncertainty in the Learned Dynamics

Prior work has shown that it can be beneficial to consider the uncertainty in the learned dynamics [109, 5, 23], even without considering constraints not seen in training. For instance, if the training data does not cover the state-action space well, then having a measure of uncertainty in the dynamics predictions makes it possible to detect out of distribution predictions. This is a measure of *epistemic uncertainty*—uncertainty due to lack of data (as opposed to inherent randomness) [46, 44].

As is done in prior work [36, 23, 66], we use an ensemble of neural networks trained on the same phase one data starting with different random seeds. When a point prediction is needed, like in planning or in constructing the classifier and recovery datasets, we take the mean of the ensemble prediction. When a measure of uncertainty is needed, we compute the sum of the standard deviations along each dimension of state across all the models in the ensemble. For simplicity we will denote this as σ^2. We use this measure of uncertainty as an input to the classifier. The intuition behind this method is that the trained networks' predictions will be similar near the training data, but will diverge far away from training data. This makes it possible for the classifier to reject or accept transitions based on the uncertainty of the learned dynamics. Although we did not find that this provided a significant improvement for our tasks, we include it nonetheless as it may be beneficial in scenarios where the unconstrained model is trained on a dataset with poor coverage of the state space.

2.3.3 Phase Two Data Collection

Once the unconstrained dynamics have been learned, we next learn where this model is accurate, and what actions to take when it is not. For this, we perform a second phase of data collection. In this phase, we collect data in the kinds of environments where we intend to perform tasks. We perform the same type of random data collection process as in the first phase, but now physical constraints are included, allowing us to gather examples of where our unconstrained dynamics are accurate and where they are not. Using the data collected in this phase, we can construct datasets to train our classifier as well as our recovery actions model. A related approach was used in [39] for learning to estimate the reachability of a quadruped robot. Our prior work has also demonstrated that this type of data can also be collected by planning and executing those plans, rather than by taking random actions [83]. However, both of the above methods used analytical/simulation models for predicting the dynamics. Furthermore, our approach is in contrast to many methods in robust control and safe reinforcement learning, which are built on the idea that predictions made outside the training distribution are unreliable [152, 36]. Those methods do not require a second data collection phase, but as a consequence would be overly conservative as the presence of any obstacle near the rope would be considered out-of-distribution.

In order to get diverse data we frequently randomize the locations of obstacles in the environment. For dual arm rope manipulation, this requires releasing the rope and moving the arms out of the way before randomizing obstacles, so that we can arrange the obstacles without permanently entangling the rope or arms.

For rope dragging, we collected 2,048 trajectories of length 50 in phase two, of which 512 were reserved for validation and testing. For dual arm rope manipulation, we collected 5,996 trajectories of length 20, of which 1,516 were reserved for validation and testing.

In total, the combined phase one and two datasets for dragging has 163,840 transitions, and the combined phase one two datasets for dual arm rope manipulation has 140,400 transitions. In comparison, [72] uses 500,000 transitions in order to accurately learn the contact dynamics of a rope amongst disc-shaped obstacles in 2D.

2.3.4 Learning the Classifier

Given the data collected in phase two, we now describe how to construct training examples for the classifier. For this we need both the predictions of the unconstrained dynamics as well the labels of whether the model-error requirement is satisfied. To compute the predictions, we take the starting state for each trajectory in the dataset and roll out the unconstrained dynamics prediction for the rest of the trajectory. This produces a tuple of $(s^t, \hat{s}^t, a^t, s^{t+1}, \hat{s}^{t+1})$ for each time step in each trajectory. As stated in Problem (2.2) the label should be 1 if $\text{dist}(\hat{s}^{t+1}, s^{t+1}) < \delta$ and 0 otherwise. To compute $\text{dist}(\hat{s}^{t+1}, s^{t+1})$, we require a distance function dist between the predicted state $\hat{s}^{t+1}$ and the actual state s^{t+1}. We selected the threshold δ by computing the 90th percentile of prediction error for the unconstrained dynamics on the unconstrained dynamics validation set. A sensitivity analysis of the threshold is given in Supplementary Materials. For rope dragging, $\delta = 0.065\,\text{m}$ and for dual arm rope manipulation $\delta = 0.025\,\text{m}$. Additionally, we discard all the transitions after the first transition labeled 0, since it is unclear whether the dynamics would have been accurate had it not diverged previously. A diagram illustrating how a trajectory is converted into examples for the classifier is shown in Figure 2.4.

The classifier network $g(\mathcal{E}, \hat{s}^t, a^t, \hat{s}^{t+1}, \sigma^2) \rightarrow p_c \in \{0, 1\}$ takes as input the environment $\mathcal{E}$ and the transition $\hat{s}^t, a^t, \hat{s}^{t+1}$. Since we use an ensemble of dynamics models, these states are the mean predictions of the ensemble. We also include the variance σ^2 of the ensemble predictions as input to the classifier. The classifier outputs a number between 0 (inaccurate) and 1 (accurate). The network architecture is shown in Figure 2.5. Since this is a binary classification problem, we use binary cross-entropy loss to train it. Furthermore, since physical constraints are spatial in nature, we convert the environment and states into multi-channel 3D voxel grids and

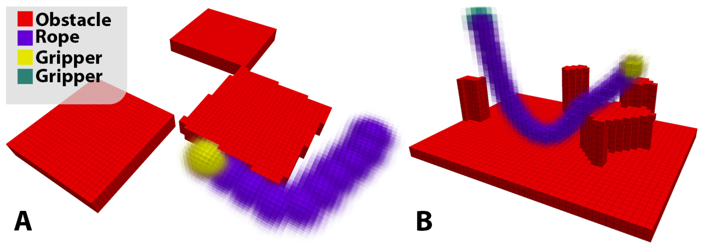

Figure 2.4: Representing the state and environment in 3D voxel grids. (A) An example for rope dragging. (B) An example dual arm rope manipulation. Different colors are used to represent different channels in the voxel grid, and alpha is used to indicate the voxel value, with 0 being fully transparent and 1 being fully opaque.

use Convolutional Neural Networks (CNN) (details in Section 2.3.7).

We also note that depending on the environments and specific parameters for how actions are sampled, the resulting dataset of transitions may be imbalanced. In our experiments, our classifier datasets contained more positive than negative examples, ranging between 65% and 95% positive. To mitigate bias in our classifier, we balance each minibatch by oversampling examples from the underrepresented class.

2.3.5 Planning With a Learned Model and Classifier

Once we have learned our unconstrained dynamics and classifier, these models are used for planning. Although they could be applied to a number of different planning methods, including the cross entropy method (CEM) [58], probabilistic roadmaps (PRM) [56], or trajectory optimization [103], we chose to use them in a kinodynamic RRT [70], as implemented in the Open Motion Planning Library (OMPL) [117]. Kinodynamic RRT is a sampling-based tree-search algorithm, and is well suited for our tasks as they contain local minima and narrow passages. Graph-based methods like PRMs could be used instead if we expect to plan repeatedly in the same environment, and trajectory optimization could be used if there are criteria such as path length which should be minimized.

In our Kinodynamic RRT, we sample a single random action a^t and attempt to extend using that action. Whenever we attempt to extend from state $\hat{s}^t$ with action a^t to the state $\hat{s}^{t+1}$, we first check the transition $(\hat{s}^t, a^t, \hat{s}^{t+1})$ by feeding it through the classifier. The extension is added to the tree only if the classifier output is greater than 0.5. We chose this type of planner for its simplicity, and many algorithmic and implementation optimizations could be made to decrease planning times.

2.3.6 Evaluating Stuck States and Learning Recovery

The formal definition of being stuck (Equation (2.4)) evaluates every possible action from a given state. In practice, checking whether a state is stuck consists of sampling N_{rs} actions and checking whether any of them are accepted by the classifier. Since the classifier is trained to approximate the MER, this is a quick and effective method for determining whether recovery is needed. Additionally, this procedure is done naturally by the RRT at the start of planning, which means that checking for recovery can be easily integrated into planning without any redundant calls to sampling, dynamics, or classification.

Given the unconstrained dynamics and the classifier, we can also use the data collected in phase two to learn recovery actions. As defined in our problem statement, recovery actions are needed when the robot is in a state where the classifier rejects all proposed actions. In this case, we would like the robot to take *recovery actions* which bring it back to regions of state space where the unconstrained dynamics are accurate.

For this, we propose training a neural network to evaluate the probability that an action is recovering. This network takes in a state, action, and the environment $(\mathcal{E}_l, s^t, a^t)$ and outputs the probability p_r that the action is recovering. Specifically, when we detect recovery is needed, we sample N_{rs} actions randomly and use this learned recovery model to assign each action a probability of recovering. The highest probability action is then selected and executed; this sampling, recovery probability evaluation, and execution process repeats until the system is no longer stuck. This approach requires training a model which estimates the probability that an action is recovering. To construct a dataset of recovering actions and labels of the recovery probability, we use a similar approach to the one used to construct the classifier dataset.

This process considers each observed transition s^t, a^t, s^{t+1} and environment $\mathcal{E}$ in the dataset and first determines whether recovery is needed at s^t. We sample N_{rs} random actions from that state, predict according to the unconstrained dynamics, and feed this into the classifier. If all N_{rs} sampled transitions are rejected, then the state s^t needs recovery. Next we perform the same test starting at the next state s^{t+1}. Here we record the proportion of the sampled actions which were accepted, and this is used as the label for probability of recovery p_r. For example, if at s^t none of the random samples were accepted, but from s^{t+1} all N_{rs} were accepted, then this is an excellent example of recovery, and we would like our recovery model to predict that taking action a^t from s^t in environment $\mathcal{E}$ is likely to lead to recovery. On the

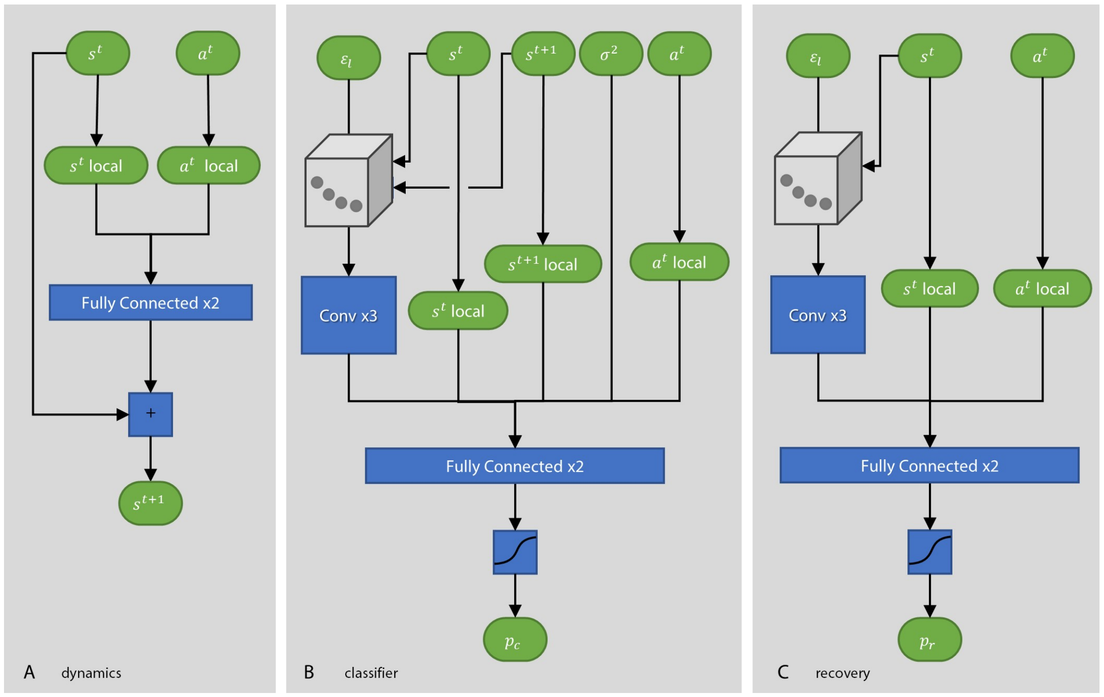

Figure 2.5: Network architectures. (A) dynamics, (B) classifier, and (C) recovery. *local* refers to the translations used to make the states and actions invariant to the position in the world frame, and is described in Sections 2.3.2, 2.3.4, and 2.3.7. Green capsules shapes indicate vectors, and blue boxes indicate functions. The gray 3D box represents the 3D voxel grid representation of state (Section 2.3.7.2).

other hand, if when checking s^{t+1} we find that still no actions are accepted by the classifier, then this is a poor example of recovery. We use all transitions for which recovery is needed (i.e. $r(\mathcal{E}, s^t) = 1$ as the dataset for training our recovery model, and train the network using binary-cross entropy to predict the recovery probability. For our experiments, we set N_{rs} to 32. More details about the neural network and its structure can be found in Section 2.3.7.

Approaches to recovery like those from the safe exploration community [60, 12, 34] are focused on staying within a region of the state space from which the agent can guarantee safety during the learning process, or the existence of a sequence of control actions which will move the system into a predefined safe set. In contrast, our method is targeting the case where the model is already too unreliable for use at the given state and any predictions made cannot be trusted.

2.3.7 Network architectures

Our method uses three neural networks, one for the dynamics model, one for the classifier, and one for recovery. Architectures are shown in Figure 2.5.

2.3.7.1 Dynamics Model Architecture

The dynamics network is a two layer fully connected network with 1,024 hidden units in each layer and ReLU activation. This model takes in the state of the rope and grippers concatenated with the actions and outputs the change in state, which is then added to the input state to produce the final predicted state. Furthermore, because we assume that the unconstrained dynamics are invariant to the global position in space, we first translate the state and actions into a local frame before feeding them into the dynamics network. For rope dragging, states and actions are relative to the position of the gripper, and for dual arm manipulation they are relative to the average of all the points of the rope.

2.3.7.2 Classifier Architecture

The classifier network takes in the environment $\mathcal{E}$ and a transition $\hat{s}^t, a^t, \hat{s}^{t+1}$ and outputs the probability of the MER being satisfied. Since we experimented with performing classification on multiple transitions we use a Long Short Term Memory network (LSTM) with a CNN encoder, although in this work we only consider a single transition as input. As shown in Figure 2.5, a single time step is passed through a CNN encoder that maps a single state $\hat{s}^t$ and action a^t into a latent vector. The state and environment are first represented in a 3D voxel grid and passed through three convolutional layers. The output is flattened and concatenated with the vector representations of the state and action. The LSTM outputs a scalar prediction of the probability of the MER being satisfied for each time step. Since we use a single transition, this means there are two outputs, one for time t and one for $t + 1$. The output for time t is ignored, because it is not a function of $\hat{s}^{t+1}$. Instead, we use the output for time $t + 1$ as the probability for the entire transition $\hat{s}^t, a^t, \hat{s}^{t+1}$.

In order to represent the environment and states in a voxel grid of a fixed size, we take a crop of the full environment occupancy grid centered around the local origin (as defined above for dynamics). As with the local representation of states and actions, this assumes invariance to the absolute position. Additionally, using a fixed size local environment has the benefit of allowing the size of the full environment to change from task to task without any retraining. In order to make it easier for the classifier to reason over the 3D input, we also include a 3D representation of the input states. For each component of the state (grippers, rope) we construct a 3D voxel grid of the same size and location as the local environment and each voxel's value is proportional to the inverse-log of its distance to the nearest point in that component of the state.

This results in a smoothed version of simply drawing the points into the voxel grid, and examples are shown in Figure 2.4. We stack these representations along with the local environment to get a multi-channel voxel grid.

The vector representations of state and action are the same as is described for the dynamics, however we use both the original state vector (not translated) and the local state vector (translated). This is because the classifier should be able to learn reachability and kinematic constraints, e.g. in our dual-arm manipulation scenario.

2.3.7.3 Recovery Architecture

Finally, the recovery network has the same encoder structure as the classifier, and we use the learned parameters, without fine-tuning, of convolution layers from the classifier in the recovery network. This network takes in a proposed transition, this time consisting of the 3D local environment, a single state, and a proposed action, and outputs the probability of recovery. Unlike the classifier, we do not pass in the predicted result of the proposed action, since recovery is only used when accurate predictions cannot be made. After encoding, two fully connected layer with ReLU activation followed by a layer with sigmoid activation are used to map down to the output probability.

2.3.7.4 Full Dynamics Architecture

Our full dynamics baseline uses a different network. We use the network proposed in [93], but extend to 3D convolution. The state representations used are the same as in our unconstrained dynamics network. Graph neural network architectures for predicting physics in 3D may provide more accurate predictions [72, 90], however these networks assume a graphical model of the world is known, whereas our dynamics learning method does not.

2.3.8 Simulation Environments

We use the Gazebo simulator with ODE physics [59, 113] for our quantitative experiments. We emphasize that our method does not have access to the simulator's model of the rope or the simulation parameters. For our rope dragging experiment, the rope is modeled with 10 rigid links, and the state consists of the positions of the links and the position of the gripper $s = [x_g, y_g, z_g, x_1, y_1, z_1, \ldots, x_{10}, y_{10}, z_{10}]$, which has 33 dimensions. We considered including gripper orientation, however this would make the dynamics dependent on gripper geometry and friction properties (because the rope could wind around the gripper) without necessarily enabling significant new capabilities, so we chose not to include orientation. The gripper is attached to one end of the rope, and the point at the other end of the rope (which we will call the tail) is the point which we wish to place at a goal position. Task error is measured is the Euclidean distance in 3D between the tail and the goal point. The actions are target gripper positions $a = [x, y, z]$, and a local controller is used to attempt to reach these goals. When commanded into an obstacle, the gripper will stop or potentially slide as it applies force into the obstacle. An action completes when the commanded position is reached or a timeout of 1 second occurs. The nominal joint velocity of the controller is tuned to reduce jerk and keep the simulation quasi-static.

For the dual arm rope manipulation experiments, the rope is modeled with 25 rigid links, and the state consists of the positions of the links and the positions of the two grippers $s = [x_{g1}, y_{g1}, z_{g1}, x_{g2}, y_{g2}, z_{g2}, x_1, y_1, z_1, \dots, x_{25}, y_{25}, z_{25}]$, which has 81 dimensions. The grippers attach to opposite ends of the rope. The actions are target positions for each gripper $a = [x_1, y_1, z_1, x_2, y_2, z_2]$. A Jacobian-based inverse kinematics controller is used to track the target position for each gripper. This controller stops when the target position is reached or just before any collisions between the arms or between the arms and obstacles.

2.4 Discussion

2.4.1 On the Specialization to Deformable Objects

Planning with inaccurate models has many potential applications, so it would be interesting to explore a broader range of tasks in future work. However, deformable object manipulation is particularly well-suited for our framework. Specifically because 1) Compliance allows us to make mistakes, stop, and replan; 2) Dynamics are more complex in some regions than in others; and 3) Much of the state space and dynamics are irrelevant for doing useful tasks. We discuss each of these points below:

First, our method relies on taking actions for which we have no accurate model, which means we must be able to take actions safely, despite their outcome being uncertain. The compliance afforded by deformable objects allows us to safely collect data, and thus to learn where our model is wrong.

Second, our method assumes that there is some subset of the dynamics which we can learn accurately, and which is sufficiently useful. Such an assumption is particularly well-suited to deformable object manipulation, where the full dynamics are much more difficult to learn than the unconstrained dynamics, yet interesting and practical tasks can still be done without learning these dynamics.

Third, deformable objects have high-dimensional state-action spaces. However, only a small region of state-action space is either reachable or useful for practical tasks (i.e. we need not consider the many different crumpled or knotted states). Because of this, it is often acceptable to avoid large portions of this space. Our method takes advantage of this in many ways, including 1) only learning the dynamics for the subset of state-action space covered in phase 1 data collection, and 2) only learning the classifier for the relatively small subset of state-action space covered in phase 2 data collection.

2.4.2 Limitations

In this work we present significant progress on planning with unreliable models and addressing their inaccuracies, however many open questions remain. In this work we do not address challenges in state estimation and tracking, control and precise manipulation, or in describing and defining complex tasks.

There are also many avenues in which our proposed methods might be improved or extended. For instance, we define which simplified dynamics should be learned by defining the phase one setting and data collection procedures. This assumes that we know which dynamics will be tractable to learn, but still useful for planning in more constrained scenarios. In future work, it would be interesting to explore how to make this decision automatically, e.g. we can search over various simplifications of the dynamics based on the performance of learned models. Finally, we plan to extend these methods to incorporate real world data based on potentially unreliable perception and tracking.

Although we show that our method can be used for several interesting tasks, we are limiting the tasks our method can do by choosing to learn only the unconstrained dynamics. Our method assumes that the goal is reachable while remaining in the part of state-action space where the unconstrained dynamics are accurate. While this is a reasonable assumption, it would be interesting to incorporate our method into a framework which uses it to get as close as possible to a given goal and then switching to a feedback-based local method (e.g. [96, 51, 139]) to finalize the task (as is done in [83]).

In terms of recovery, we note that although our learned recovery actions dramatically improved our performance for the dual arm manipulation task, the learned recovery policy still fails in some cases. The learned recovery policy tends to raise the grippers, as this is an effective strategy in the training data. However, while this will work well when the rope is draped on the table or obstacles, it leads to being caught on a protrusion if the rope starts below it. A better policy would likely be learned by collecting phase two data in environments where getting caught and escaping is more likely.

In this work, we treat the simulator as a black-box proxy for the real world. However, these simulations can differ from real world physics, and so at best they provide a starting point from which sim2real methods can be used to transfer either a learned dynamics model [35, 24] or a learned policy [16, 99] to the real world. For instance, it would be useful to adapt online to different stiffnesses of lengths of rope without re-collecting large datasets. Sim2real has been demonstrated for a number of other robots and tasks [71, 98], and incorporating these techniques into our proposed methods is an interesting direction for future work.

2.4.3 Learning Performance

In addition to reporting the performance of our complete method on various tasks, here we also report the training and validation accuracies of our learned models. Further details on the training of these models can be found in Sections "Learning the Unconstrained Dynamics", "Learning the Classifier", and "Evaluating Stuck States and Learning Recovery".

For dynamics learning, we report learning error as the Euclidean distance between every predicted and true point on the rope, and average over all the points, time steps, and examples in the dataset. For rope dragging, our unconstrained dynamics model has an error of 0.0081 m in training and 0.0097 m in testing. These numbers are small in comparison to the length of the rope (0.5 m) and the size of the environment (2x2m). FD achieves an error of 0.0090 m in training and 0.0117 m in testing. For dual arm rope manipulation, our unconstrained dynamics model has an error of 0.0025 m in training in 0.0030 m in testing. Here the rope has a length of 0.8 m, indicating that we are able to learn the unconstrained dynamics very accurately. The full dynamics baseline achieves an error of 0.0194 m in training and 0.0218 m in testing. Learning accurate dynamics over long horizons is critical for planning, and by learning only the unconstrained dynamics, our method is able to do so with higher accuracy than the full dynamics baseline.

We report learning metrics on the phase two dataset (see Section "Learning the Classifier" for details). For rope dragging, the classifier achieved an accuracy of 0.890, precision of 0.959 and recall of 0.822 on the training set. On the testing set, it has an accuracy of 0.835, precision of 0.888, and recall of 0.789. For dual arm rope manipulation, the classifier achieved an accuracy of 0.904, precision of 0.914 and recall of 0.927 on the training set. On the testing set, it has an accuracy of 0.890, precision of 0.895, and recall of 0.927. Throughout our experimentation we observed that, as prior work has noted [83], the accuracy of this classifier is a poor indicator of its usefulness in planning.

For recovery, we use the binary cross entropy loss to measure learning performance. For rope dragging, the training loss (unitless) is 0.021 and the testing loss is 0.023. For rope dragging, the training loss is 0.139 and the testing loss is 0.146.

2.4.4 Physical Robot Demonstrations

To demonstrate the practicality of our method, we designed real-world mockups of domestic and automotive tasks. For dual arm manipulation, we demonstrate three tasks done under the hood of a car, where the robot manipulates hoses and straps. For rope dragging, we show an example where the robot retrieves a phone charging cable by sliding it. For clarity and simplicity, we demonstrate only the parts of these tasks where our method applies and forgo the use of sophisticated local controllers to, for example, plug the charging cable into the phone.

Perception of deformable objects remains a difficult open problem [140], much less in cluttered environments where the object is partially-occluded. Online perception of the object was not used in this demonstration, is used in future chapters. To demonstrate our methods despite not having such perception algorithms, we manually constructed the scenes in a simulator and planned our actions there before executing them on the real robot.

In both scenes, we re-use the unconstrained dynamics models learned in simulation directly. For rope dragging, we also re-use the classifier and recovery models as-is. For dual arm manipulation however, because a different robot is used, and the scene geometry differs significantly from our simulation, we use classifier and recovery models trained on scenes similar to the one we test on in the real world. For more information, see the video in our supplemental materials.

2.4.5 Experiment Design

To compare these methods quantitatively, we consider the task error over 150 trials per method in two types of rope manipulation tasks. The obstacle configurations are randomized before every trial, and each trial is allowed 180 seconds. During these 180 seconds, the method alternates between action selection and execution, where action selection is either planning or recovery. The trial is terminated if the goal is reached or the time limit is reached (see Figure 2.1, and Supplementary Materials for more details).

At the end of the trial, the final state is used to determine task error, and we report statistics of this error across the trials. A trial is a success if the final state error is below the goal threshold. Because tasks are generated randomly, some tasks will be impossible to achieve (e.g. the object cannot reach the goal because of a barrier), thus the absolute success rate is less informative than the difference in success rates between methods. When claims of statistical significance are made, a one-sided T-test is used and p-values are reported.

2.5 Results

To rigorously evaluate our approach, we perform statistical comparisons of our method vs. ablations and baselines in simulation over two types of rope manipulation tasks in 150 randomly-generated environments. We show that our approach greatly improves performance over learning the full dynamics as well as simply trusting the model learned in a simplified setting. We then demonstrate the practicality of our method for performing tasks on a real robot in domestic and automotive scenarios.

2.5.1 Baselines

In this work we argue that learning the dynamics for deformable objects in environments with constraints such as obstacles is difficult, and that we should instead learn only the unconstrained dynamics and a classifier to predict when those dynamics are valid. To support this claim we compare to a method which plans with a model of the full dynamics. We term this baseline Full Dynamics (FD), and we learn the dynamics using an approach similar to [92] (see Section "Learning the Unconstrained Dynamics"). In FD, we learn the dynamics from a dataset collected in the same way as we collect data for our classifier (Section "Phase Two Data Collection"). Our method uses two phases of data collection, the data in each phase is used differently. Therefore, to make a fair comparison, the full dynamics baseline is trained on a dataset whose size is equal to our method's two datasets combined. Once trained, we plan with FD using the same Rapidly-exploring Random Tree (RRT) planner as in our method. However, unlike our method, FD requires no constraint checker (learned or otherwise), as anything we might consider a constraint is subsumed by the dynamics.

Additionally, we remove various components of our method to demonstrate the benefits they provide. We compare to a version called *No Classifier*, which plans using our unconstrained dynamics but no constraint checker. Comparisons to this baseline show the benefit of using the classifier over simply trusting the unconstrained dynamics everywhere. We also compare to a version without recovery (*Classifier*), and a method which takes random actions as its recovery policy (*Classifier + Random Recovery*).

2.5.2 Scenario 1: Rope Dragging

Here we describe our results for the task of dragging a rope-like object along a surface among obstacles, as is shown in Figure 2.1a. Rope manipulation has numerous important applications including suturing and managing wires or hoses, and the rope dragging task requires long horizon planning for which our method is well-suited. The task is to place one end of a rope at a point while dragging the rope by the other end. This task is difficult because the system is highly under-actuated and the environment is cluttered. Task error is the Euclidean distance between the end of the rope and the goal point. The goal region is a sphere about the goal point with radius 5 cm. This task illustrates the challenges of long horizon planning for deformable objects in cluttered scenes, and is challenging for existing methods. The practicality of our method is illustrated in our physical robot demonstrations.

The task error across 50 trials is shown in Figure 2.6a. Our complete method reached the goal 76% of the time with a mean task error of 4.48 cm. This is better than FD (32% success, 20.32 cm error) and No Classifier (60% success, 8.20 cm error) and is significant at $p < 0.001$. Additional numerical results are shown in Table 2.1. For this task, recovery was never needed, meaning that for all states from which the planner was run there was at least one action from that state which our classifier accepted as yielding an accurate prediction. The benefit of recovery actions is shown in the dual arm manipulation task below.

To show the ability of our classifier to learn difficult prediction functions, we also compare our results to a manually-engineered solution for this scenario. By inspecting the data of where the model tended to make errors, we found that, unsurprisingly, the model could not predict the effect of pushing/pulling the rope into obstacles but was fairly accurate when not in contact with obstacles or sliding along them (See Supplementary Materials for examples of sliding motion accepted by our classifier). To capture this predictive behavior with a rough approximation we used a collision checker to measure if the predicted state was penetrating into the obstacle in place of

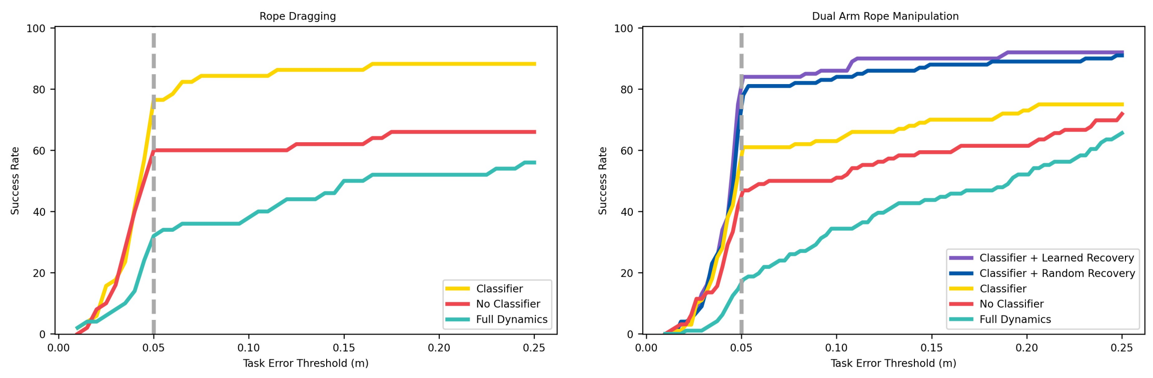

Figure 2.6: Comparing success across methods. (left) The success rate as a function of the success threshold on task error for our simulated rope dragging experiments. The dashed line indicates the size of the goal region used. For example, this tells us that if our goal region was 0.1 m, the "Classifier" method would achieve about 84% success. (right) The success rate as a function of the success threshold on task error for our simulated dual arm rope manipulation experiments.

our learned classifier (as is done in [67]). Note that this is a scenario-specific method derived by using human intuition and an engineered collision-checker. While this intuition and hand engineered solution can be an effective solution for some tasks, changes in the task or object being manipulated often lead to additional tuning and engineering. For example, if the rope was very thick and the points defining the rope state (which are along the central axis of the rope) did not enter obstacles even when the rope was pressing into an obstacle, collision boundaries would need to be tuned. Likewise, if the rope is very thin, numerical error in predictions could lead to erroneous collision-checking results when sliding along the surface of an obstacle. To the credit of our method, we found that using the collision checker gave a 78% success rate and a mean task error of 12.74 cm, with $p = 0.498$ for the hypothesis that our method outperforms the collision checking method. This is an encouraging result, as it demonstrates that our method can perform on-par with a scenario-specific human-engineered solution, at least in this scenario.

2.5.3 Scenario 2: Dual Arm Rope Manipulation

Here we describe our results for using a robot for dual arm manipulation of rope among obstacles, as is shown in Figure 2.3b. Using two arms to manipulate deformable objects allows more control of the object than one arm and introduces additional interesting challenges in coordinating the two arms. The task we use for evaluation is to place the midpoint of the rope at a point in 3D while only hold-

Name	Dynamics	Classifier	max (m)	mean (m)	median (m)	std. dev. (m)
Classifier	Unconstrained	Learned	1.133	0.128	0.044	0.247
No Classifier	Unconstrained	None	1.408	0.325	0.045	0.425
Full Dynamics	Full	None	1.519	0.438	0.156	0.450

Table 2.1: Task error statistics for simulated rope dragging.

ing the ends. In addition to obstacles, this scenario imposes the constraints of not overstretching the rope, not colliding the arms, and staying within the reachability limits of the robot. Task error is the Euclidean distance between the midpoint of the rope and the goal point. The goal region is defined as a sphere about the goal point with radius 5 cm. This type of manipulation, while difficult for existing methods, is a prerequisite for many practical tasks such as cable harnessing.

The task error across 100 trials is shown in Figure 2.6b. Our complete method (*Classifier + Learned Recovery*) reached the goal 84% of the time with a mean task error of 7.29 cm. This is statistically significantly ($p < 0.001$) better than FD, which reached the goal 17.7% of the time with a mean task error of 20.32 cm, and No Classifier, which reached the goal 47% of the time with a mean task error of 16.97 cm. We also compare to our method without recovery actions, and to our method with random recovery actions. In this task, recovery actions are critical, and without them our method performs statistically significantly ($p < 0.001$) worse with a success rate of 61% and a mean task error of 14.69 cm. When compared to random recovery actions, which reaches the goal 78% of the time, our method has similar task error ($p = 0.281$ for the hypothesis that our method outperforms random recovery), however our method needs only a third as many recovery actions to achieve this task error. Over all 100 trials, random recovery used 613 recovery actions whereas our method used only 177. Additional numerical results are shown in Table 2.2.

2.5.4 Physical Robot Demonstrations

We demonstrate potential applications of our method on real-world mock-ups of several domestic and automotive tasks. The first set of tasks are performed under the hood of a car, and require the robot to manipulate straps or hoses. These tasks include moving the midpoint of a hose to a specific location (Figure 2.3e), positioning the ends of a wiper fluid hose for installation (Figure 2.3f), and removing lifting straps from an engine (Figure 2.3d).

Name	Dynamics	Classifier	max (m)	mean (m)	median (m)	std. dev. (m)
Classifier Learned Recovery	Unconstrained	Learned	0.450	0.073	0.045	0.094
Classifier Random Recovery	Unconstrained	Learned	0.629	0.081	0.046	0.106
Classifier	Unconstrained	Learned	0.630	0.147	0.047	0.167
No Classifier	Unconstrained	None	0.741	0.170	0.082	0.165
Full Dynamics	Full	None	0.621	0.203	0.191	0.142

Table 2.2: Task error statistics for simulated dual arm rope manipulation.

We also perform the task of fetching a charging cable (Figure 2.3c). In this example, the robot cannot reach the end of the cable directly because it is blocked by an obstacle. Instead, the robot grasps the cable elsewhere, and must drag the end of the cable towards the phone.

Notably, these tasks use several different types of goals described in the state space of the rope and grippers, all of which can be handled by our planner. This is in contrast to policy learning methods [81, 118, 139] and methods which use goal images [95, 33, 148]. More details on how we perform these tasks can be found in Section "Physical Robot Demonstrations".

2.6 Conclusion

In summary, the proposed method is able to complete a variety of difficult rope manipulation tasks in clutter. We are able to learn the unconstrained dynamics accurately, and by using our learned classifier, the planner successfully avoids the regions of state-action space where these unconstrained dynamics are incorrect. Using the classifier outperforms both using the full dynamics and simply trusting the unconstrained model by a wide margin. Last, recovery makes the method more resilient, allowing the robot to act even when it cannot trust its dynamics model. In our tabletop manipulation simulations, we demonstrated that these recovery actions statistically significantly improve the success rate of our method.

CHAPTER III

Data Augmentation for Learning Reliability

The previous chapter introduced the Model-Error Requirement RRT and a method for learning a recovery policy. In simulation, we used these methods to learn the reliability of a dynamics model for bimanual rope manipulation. We also executed trajectories planned in simulation in the real world to illustrate what it would look like to apply these methods in the real world. This was needed because the proposed learning methods did not consider data efficiency, and required datasets that would be impractical to collect in the real world.

This chapter addresses this problem by proposing a data augmentation method for manipulation. We first formalize data augmentation as an optimization problem, where we attempt to maximize diversity, validity, and relevance. We propose loss functions that formalize these abstract concepts, and apply our proposed method to learning dynamics and learning reliability. Finally, we use the proposed data augmentation method to learn reliability for dual-arm rope manipulation – online, from scratch, and in the real world.

3.1 Introduction

In recent years, interest in applying deep learning to robotic manipulation has increased. However, the lack of cheap data has proven to be a significant limitation [145]. To enable applications such as smart and flexible manufacturing, logistics, and care-giving robots [137], we must develop methods that learn from smaller datasets, especially if the learning is done online on real robots.

One of the simplest and most effective ways to mitigate the problem of small datasets is to use data augmentation. While data augmentation has been shown to significantly improve generalization performance in tasks like image classification, it is not straightforward to extend existing data augmentation methods to the types of

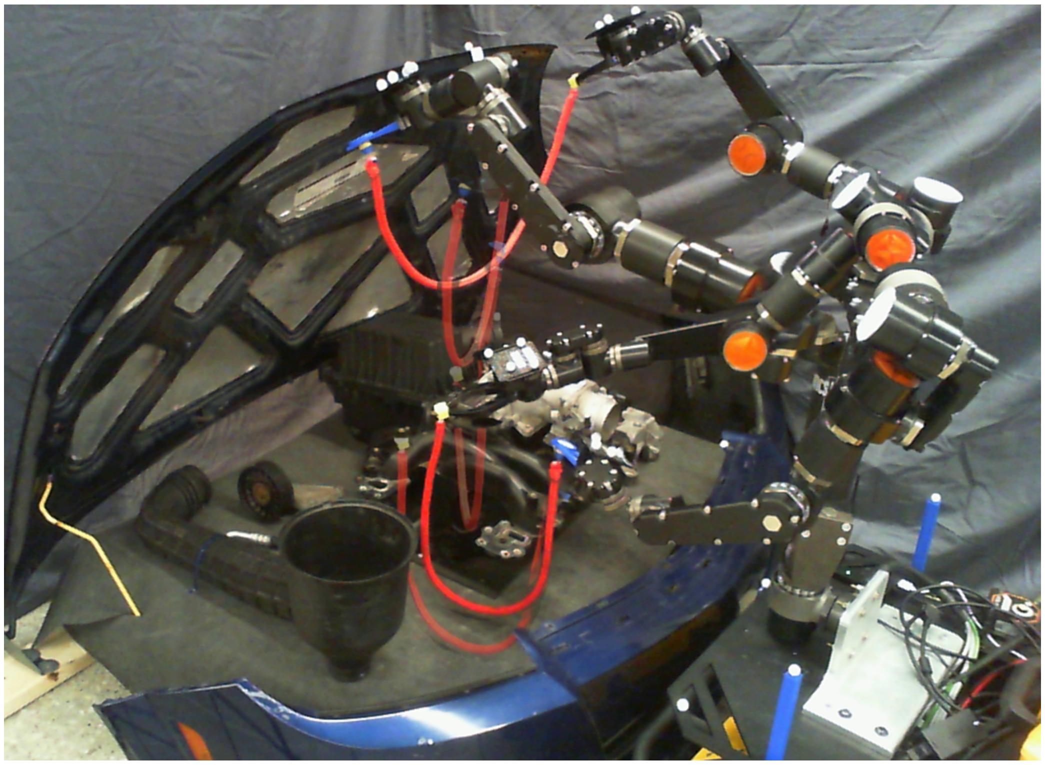

Figure 3.1: A mock-up of a car engine bay. The robot must move the rope and place it under the engine without snagging it to set up for lifting the engine. We use data augmentation to improve task success rate during online learning for this task.

data used in robotic manipulation. Furthermore, most existing augmentation methods fall into one of two categories, and both have severe limitations:

In the first category, augmentations are defined by a set of transformations, sampled independently for each example. Most image augmentation methods fall into this category, where rotations or crops are sampled randomly for each example [8, 27, 111]. By making augmentations independent of the example being augmented, we are restricted to operations which are valid on all examples. In the second category, there are methods which learn a generative model (VAE, GAN, etc.) of the data, and then sample new training examples from that model [124, 97, 29]. This approach assumes a useful generative model can be learned from the given dataset, but we found these methods did not perform well when the dataset is small.

It is not trivial to define a coherent framework for data augmentation that encompasses many domains and many types of learning problems (e.g. classification and regression). Thus, the first contribution of this chapter is a formalization of the data augmentation problem. In our problem statement (Section 3.2), we formalize data augmentation as an optimization problem based on three key criteria: *validity*,

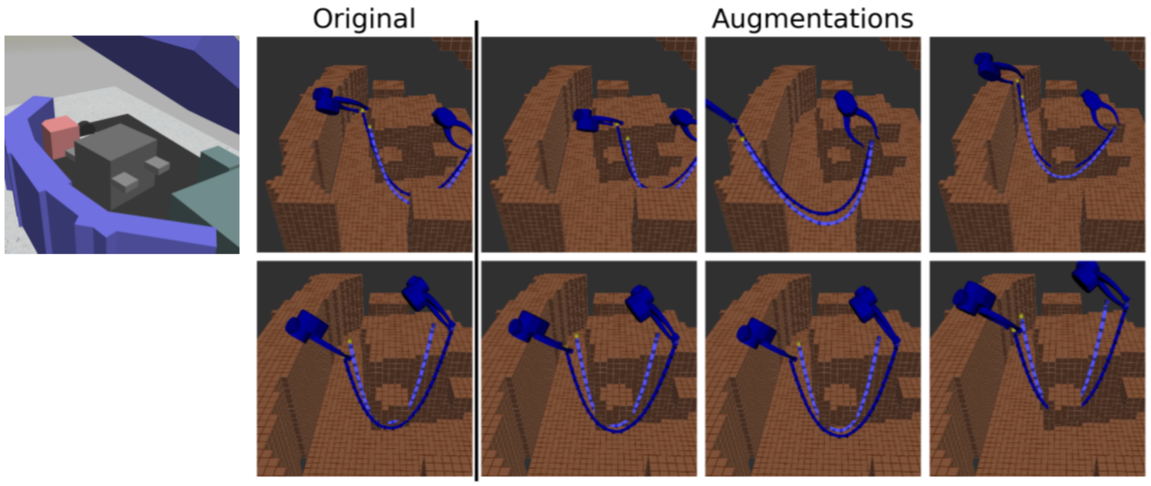

Figure 3.2: Examples of augmentations of rope generated by our method. On the left is a picture of the scene in simulation from a zoomed out viewpoint. The simplified engine block model is in the center. The rope start (dark blue) and end (light blue) states are shown, with the grippers shown at the start state. The static environment geometry is shown in brown. The first row shows a transition in free space, where the resulting augmentations are particularly diverse. The final augmentation shows how our method found a transformation to move the rope underneath the hook while remaining in free space. The second row shows a transition which involves contact between the rope and the environment. The augmentations preserve this contact.

relevance, and *diversity*. We define an augmented example as *valid* if it obeys the laws of physics and is possible in the real world. Augmentations are *relevant* if they are similar to data that would be seen when performing the target task. *Diversity* encourages the augmentations to be as varied as possible, i.e. the transformations applied to the data should be uniformly distributed to maximize diversity. Producing diverse augmentations for each original example is key to improving the generalization of the trained network.

The general definitions of validity, relevance, and diversity we propose depend on information that is intractable to compute for many manipulation problems, and therefore we also present approximations to these definitions. We do not claim that this formulation is useful for all manipulation problems, and clearly define the physical assumptions behind this formulation in Section 3.2.

Our second contribution is a method for solving this approximated optimization problem. Our method operates on trajectories of object poses and velocities, and searches for rigid-body transformations to apply to the moving objects in the scene to produce augmentations. Our method encourages validity by preserving contacts and the influence of gravity. Additionally, we encourage relevance by initializing the

Algorithm 1: $\phi(x)$

```
1 $T = \texttt{sample}(x)$
2 $\tilde{x} = \texttt{apply}(x, T)$
3 return $\tilde{x}$
```

augmentations nearby the original examples and preserving near-contacts. Finally, we encourage diversity by pushing the augmentations towards randomly sampled targets.

Our results demonstrate that training on our augmentations improves downstream task performance for a simulated cluttered planar-pushing task and a simulated bimanual rope manipulation task. The learning problems in these tasks include classification and regression, and have high-dimensional inputs and outputs. Lastly, we demonstrate our augmentation in an online learning scenario on real-robot bimanual rope manipulation using noisy real-world perception data (Figure 3.1). In this scenario, augmentation increased the success rate from 27% to 50% in only 30 trials. Additional materials such as code and video can be found on our project website.

3.2 Problem Statement

In this section, we formally define the form of data augmentation studied in this chapter. We define a dataset $\mathcal{D}$ as a list of examples $x \in \mathcal{X}$ and, optionally, labels $\ell(x) = w$, where ℓ is a task-specific labeling function. We assume the space $\mathcal{X}$ is a metric space with a distance function dist. Augmentation is a stochastic function $\phi : \mathcal{X} \rightarrow \mathcal{X}$ which takes in an example x and produces the augmented example $\tilde{x}$. The general form is shown in Algorithm 1. Internally, augmentation will call `sample` to generate a vector of parameters, which we call T. We also define $\tilde{x}_{1:k}$ as a set of k augmented examples produced by calling `sample` then `apply` k times. The parameters T describe the transformation which will be applied to the example in the `apply` procedure. We focus on augmentation functions that are stochastic, thus ϕ will sample new augmented examples each time it is called. If the dataset contains labels w, we assume that the labels should not change when the example is augmented.

We propose that useful augmentations should be *valid*, *relevant*, and *diverse*. Let the valid set $\mathcal{X}_v$ be the set of examples which are physically possible. Let the relevant set $\mathcal{X}_r$ be the set of examples likely to occur when collecting data for or executing a specific set of tasks in a specific domain. We define validity$(\tilde{x}) = 1$ if $\tilde{x} \in \mathcal{X}_v$ and validity$(\tilde{x}) = 0$ otherwise. We also define relevance$(\tilde{x}) := e^{-\text{dist}(\tilde{x}, \mathcal{X}_r)}$

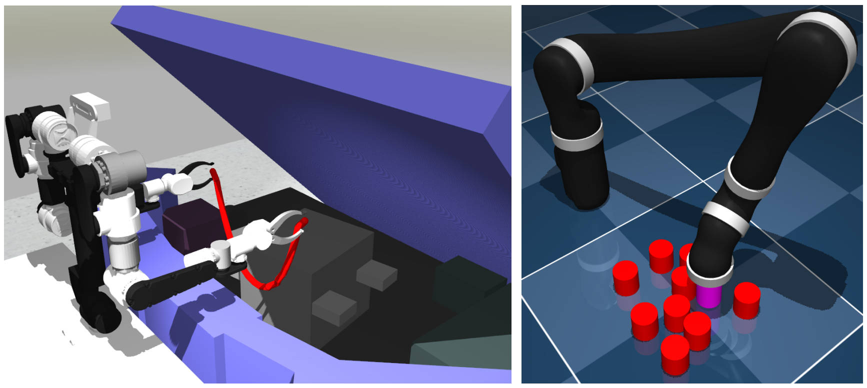
Figure 3.3: (left) The environment for bimanual rope manipulation, in simulation. (right) The environment for cluttered planar pushing of cylinders, in simulation.

and diversity$(\tilde{x}_{1:k}) := e^{-D_{KL}(p_{\tilde{x}_{1:k}}(T)\,||\,\mathbb{U}[T^-,T^+])}$, where D_{KL} is the Kullback–Leibler divergence and $p_{\tilde{x}_{1:k}}(T)$ is the distribution of the parameters for a set of augmented examples $\tilde{x}_{1:k}$. Diversity is maximized when the augmentation transformations are uniformly distributed in the range $[T^-, T^+]$. With these concepts defined, we define data augmentation as the following optimization problem, the solution to which is a set of augmentations $\tilde{x}_{1:k}$:

$$\begin{array}{cc} \max\limits_{\tilde{x}_{1:k}} & \text{diversity}(\tilde{x}_{1:k}) + \beta \sum_{\tilde{x}_i} \text{relevance}(\tilde{x}_i) \\ \text{subject to} & \text{validity}(\tilde{x}_i) \quad \forall\, \tilde{x}_i \in \tilde{x}_{1:k} \\ & \ell(\tilde{x}) = \ell(x) \end{array} \tag{3.1}$$

where β is a positive scalar.

This optimization problem can be solved directly if $\mathcal{X}_v$, $\mathcal{X}_r$, and ℓ are known. However, in manipulation tasks, that is rarely the case. Instead, we will formulate an approximation to this problem using measures of relevance, diversity, and validity that are derived from physics and useful for a variety of robotic manipulation tasks and domains.

3.2.1 Assumptions

Most augmentation algorithms rely on some expert knowledge or heuristics to define what is a valid augmentation. For instance, rotating an image for image classification makes an assumption that rotation does not change the label, and this is not always true. Similarly, the efficacy or correctness of our algorithm is also subject to certain assumptions. Here, we define the key assumptions:

- The geometry of the robot and all objects is known.

- The scene can be decomposed into objects which can be assigned or detected as either moving or stationary.
- Examples are time-series, consisting of at least two states.
- All possible contacts between stationary vs. moving objects have the same friction coefficient.
- Contacts between the robot and objects/environment (e.g. grasps) can be determined from the data.
- A rigid-body transformation of an object preserves internal forces arising from its material properties.
- Objects only move due to contact or under the force of gravity. We do not handle movement due to magnetism or wind, for example.

Notably, the assumption that a rigid-body transformation preserves internal, material forces is what allows us to handle cluttered scenes with many moving objects, as well as deformable or articulated objects. While it could be valuable to augment the deformation or relative motion of the objects, doing so in a way that is valid would be challenging. Instead, we transform them all rigidly (See examples in Figures 3.2,3.5).

The assumption of having a common friction coefficient between all moving versus stationary objects is in-line with much manipulation research. For example, work on planar pushing assumes friction is uniform across the plane [154, 142]. Note that we make no assumption on the coefficients of friction between two moving objects.

Naturally, there are scenarios where these assumptions do not hold and thus where our algorithm may not perform well. However, our experiments demonstrate significant improvement on two very different manipulation scenarios, and we expect these assumptions extend to other scenarios as well.

3.3 Methods

We first describe an approximation to the augmentation problem (3.1), which is specialized for manipulation. Next, we decompose this problem and describe each component in detail.

3.3.1 Algorithm Overview

Since robotic manipulation is interested specifically in moving objects, we focus on augmenting trajectories of poses and velocities of moving objects. A key insight is that objects in the scene can be categorized as either robots, moved objects, or stationary objects, and that these should be considered differently in augmentation. We denote the moved objects state as s, the robot state as q, the robot action as a, and the stationary objects as $\mathcal{E}$ (also called environment). Our method augments the moved object states, the robot state, and the actions, but not the stationary objects. We do not assume any specific representation for states or actions, and examples of possible representations include sets of points, joint angles, poses, or joint velocities. Since we operate on trajectories, we bold the state ($\boldsymbol{s}, \boldsymbol{q}$) and action ($\boldsymbol{a}$) variables to indicate time-series (e.g $s_{1:T} = \boldsymbol{s}$). With this categorization, we can write $x = \{\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{a}, \mathcal{E}\}$ and $\tilde{x} = \{\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{a}}, \mathcal{E}\}$.

We choose the parameters T to be rigid body transformations, i.e. either $SE(2)$ or $SE(3)$. We parameterize T as a vector with translation and rotation components, with the rotation component with Euler angles bounded from $-\pi/2$ to $\pi/2$, which gives uniqueness and a valid distance metric. These rigid body transforms are applied to moved objects in the scene, and augmented robot state and action are computed to match. We choose rigid body transforms because we can reasonably assume that even for articulated or deformable objects, augmenting with rigid body transforms preserve the internal forces, and therefore the augmentations are likely to be valid.

It may seem that an effective method to generate augmentations is then to randomly sample transforms independent of the data. However, this is not an effective strategy because it is highly unlikely to randomly sample valid and relevant transformations. We confirm this in our ablations studies (included in the Appendix 1.A). Instead of sampling transforms randomly, we formulate an approximation to Problem 3.1:

$$\begin{aligned}\min_{T} \quad & \mathcal{L}_{\mathbb{U}}(T, T^{\text{target}}) + \beta_1 \mathcal{L}_{\text{bbox}}(\tilde{\boldsymbol{s}}) + \beta_2 \mathcal{L}_{\text{valid}}(T) + \\ & \beta_3 \mathcal{L}_{\text{occ}}(\tilde{\boldsymbol{s}}, e) + \beta_4 \mathcal{L}_{\Delta d^-}(\tilde{\boldsymbol{s}}, \mathcal{E}) + \\ & \mathcal{L}_{\text{robot}}(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{a}}, \mathcal{E}) \qquad (3.2) \\ \text{subject to} \quad & \{\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{a}}, \mathcal{E}\} = \texttt{apply}(\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{a}, \mathcal{E}, T) \\ & T^{\text{target}} \sim \mathbb{U}[T^-, T^+]\end{aligned}$$

The decision variable is now the parameters T, and the validity constraint is moved into the objective. We propose that diversity should be maximized by the transforms being uniformly distributed, and therefore $\mathcal{L}_{\mathbb{U}}$ penalizes the distance to a target transform T^{target} sampled uniformly within $[T^-, T^+]$. The relevance and validity terms (which are intractable to compute) are replaced with four objective functions, which are specialized to manipulation. The magnitudes of different terms are balanced by $\beta_1, \beta_2, \beta_3, \beta_4$, which are defined manually. We define each objective function below:

3.3.1.1 Bounding Box Objective

First is the bounding-box objective $\mathcal{L}_{\text{bbox}}$, which keeps the augmented states $\tilde{\boldsymbol{s}}$ within the workspace/scene bounds defined by $[s^-, s^+]$. The bounding box objective encourages relevance, since states outside the workspace are unlikely to be relevant for the task.

$$\mathcal{L}_{\text{bbox}} = \sum_{i=1}^{|s|} \max(0, \tilde{\boldsymbol{s}}_i - s_i^+) + \max(0, s_i^- - \tilde{\boldsymbol{s}}_i) \tag{3.3}$$

3.3.1.2 Transformation Validity Objective

The transformation validity objective $\mathcal{L}_{\text{valid}}$ assigns high cost to transformations that are always invalid or irrelevant for the particular task or domain. It is defined by function f_{valid}, which takes in only the transformation. For example, in our rope manipulation case, it is nearly always invalid to rotate the rope so that it floats sideways. In our cluttered pushing task, in contrast, this term has no effect. This term can be chosen manually on a per-task basis, but we also describe how a transformation validity objective can be learned from data in section 3.3.3.

$$\mathcal{L}_{\text{valid}} = f_{\text{valid}}(T) \tag{3.4}$$

3.3.1.3 Occupancy Objective

The occupancy objective $\mathcal{L}_{\text{occ}}$ is designed to ensure validity by preventing objects that were separate in the original example from penetrating each other and ensuring that any existing penetrations are preserved. In other words, we ensure that the occupancy $O(p)$ of each point $\tilde{p}_{s,i} \in \tilde{p}_s$ in the augmented object state matches the occupancy of the corresponding original point $p_{s,i} \in p_s$. For this term, we directly define the gradient, which moves $\tilde{p}_{s,i}$ in the correct direction when the occupancies do not match. This involves converting the environment $\mathcal{E}$ into a signed-distance field

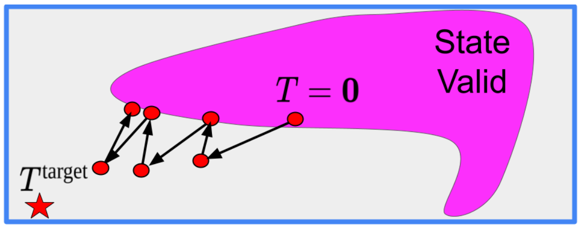

Figure 3.4: Illustration of `aug_state` within Algorithm 2. All points and sets are in the space of T. The path of T is shown in red with black arrows. The pink set, State Valid, is the set where *state_valid* is true. T begins at the origin, and alternates between moving towards T^{target} and projecting back into the set *state_valid* (by solving Equation (3.8)).

(SDF) and the moved objects states $\boldsymbol{s}$ into points p_s. This objective assumes that the environment has uniform friction, so that a contact/penetration in one region of the environment can be moved to another region.

$$\mathcal{L}_{\text{occ}} = \sum_{\substack{p_{s,i} \in p_s \\ \tilde{p}_{s,i} \in \tilde{p}_s}} \text{SDF}(\tilde{p}_{s,i})\big(O(p_{s,i}) - O(\tilde{p}_{s,i})\big) \tag{3.5}$$

3.3.1.4 Delta Minimum Distance Objective

The delta minimum distance objective is designed to increase relevance by preserving near-contact events in the data. We preserve near-contact events because they may signify important parts of the task, such as being near a goal object or avoiding an obstacle. We define the point among the moved object points p_s which has the minimum distance to the environment $p_{d^-} = \text{argmin}_{p_{s,i}} \text{SDF}(p_{s,i})$. The corresponding point in the augmented example we call $\tilde{p}_{d^-}$.

$$\mathcal{L}_{\Delta d^-} = ||\text{SDF}(p_{d^-}) - \text{SDF}(\tilde{p}_{d^-})||_2^2 \tag{3.6}$$

3.3.1.5 Robot Contact Objective

The robot contact objective $\mathcal{L}_{\text{robot}}$ ensures validity of the robot's state and the action. This means that contacts involving the moved objects which existed in the original example must also exist in the augmented example. Let the contact points on the robot be p_q^c and the contact points on the moved objects' state be p_s^c.

$$\mathcal{L}_{\text{robot}} = \sum_i (||p_{q,i}^c - p_{s,i}^c||_2^2) \tag{3.7}$$

Algorithm 2: $\phi(\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{a}, \mathcal{E})$

```
    // aug_state
1   $T^{\text{target}} \sim \mathbb{U}[T^-, T^+]$
2   $T = \mathbf{0}$  (identity)
3   for $i \in N_p$ do
4       $T_{\text{old}} = T$
5       $T =$ step_towards$(T, T^{\text{target}})$
6       $T =$ solve Equation (3.8)
7       if $\text{dist}(T, T_{old}) < \delta_p$ then
8           break
    // aug_robot
9   $\tilde{\boldsymbol{s}}, \textit{state\_valid} =$ apply_state$(\boldsymbol{s}, \boldsymbol{a}, T)$
10  $\tilde{\boldsymbol{q}}, \tilde{\boldsymbol{a}}, \textit{ik\_valid} \leftarrow \text{IK}(\boldsymbol{q}, \boldsymbol{a}, \tilde{\boldsymbol{s}}, \mathcal{E})$
11  if $!\textit{state\_valid}$ or $!\textit{ik\_valid}$ then
12      return $\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{a}, \mathcal{E}$
13  else
14      return $\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{a}}, \mathcal{E}$
```

Finally, we note that other objective functions can be added for the purpose of preserving task-specific labels, i.e. so that $\ell(x) = \ell(\tilde{x})$. However, for our experiments, no additional functions were necessary.

3.3.2 Solving the Augmentation Optimization Problem

This section describes how we solve Problem (3.2), and the procedure is detailed in Algorithm 2. First, we split the problem into two parts, `aug_state` and `aug_robot`.

In `aug_state`, we optimize the transform T to produce the moved objects' state $\tilde{\boldsymbol{s}}$ while considering environment $\mathcal{E}$. To achieve diversity, we uniformly sample a target transform T^{target} and step towards it iteratively. This stepping alternates with optimizing for validity and relevance. We visualize this procedure in Figure 3.4, as well as in the supplementary video. The innermost optimization problem is

$$\underset{T}{\text{argmax}} \quad \beta_1 \mathcal{L}_{\text{bbox}} + \beta_2 \mathcal{L}_{\text{valid}} + \beta_3 \mathcal{L}_{\text{occ}} + \beta_4 \mathcal{L}_{\Delta d^-} \tag{3.8}$$

We solve Problem (3.8) using gradient descent, terminating after either M_p steps or until the gradient is smaller than some threshold ϵ_p.

Note that we start `aug_state` in Algorithm 2 with T at the identity transformation, rather than initially sampling uniformly. This has two benefits. First, the identity transform gives the original example, which is always in the relevant set.

Algorithm 3: Data Collection for Learning Valid Transformations

```
Input: $Q_{\text{valid}}, n_{\text{valid}}$
Output: $\mathcal{D}_{\text{valid}}$
1  $y^{-}_{\text{valid}} = \infty$
2  for $i \in [1, n_{valid}]$ do
3      for $(s_t, q_t, a_t, \mathcal{E}) \in Q_{valid}$ do
4          $\alpha_{\text{valid}} = i/n_{\text{valid}}$
5          $T \sim \mathbb{U}[\alpha_{\text{valid}}T^{-}, \alpha_{\text{valid}}T^{+}]$
6          $s_{t+1}, q_{t+1} = \texttt{simulate}(s_t, q_t, a_t, \mathcal{E})$
7          $\tilde{s}_{t,t+1}, \tilde{q}_{t,t+1}, \tilde{a}_t = \texttt{apply}(s_{t,t+1}, q_{t,t+1}, a_t, T)$
8          $\tilde{s}'_{t+1}, \tilde{q}'_{t+1} = \texttt{simulate}(\tilde{s}_t, \tilde{q}_t, \tilde{a}_t, \mathcal{E})$
9          $y_{\text{valid}} = ||\tilde{s}_{t+1} - \tilde{s}'_{t+1}||$
10         if $y_{valid} < y^{-}_{valid}$ then
11             $y^{-}_{\text{valid}} = y_{\text{valid}}$, $T_{\min} = T$, $y^{-}_{\text{valid}} = y_{\text{valid}}$
12     add $(T_{\min}, y^{-}_{\text{valid}})$ to $\mathcal{D}_{\text{valid}}$
13 return $\mathcal{D}_{\text{valid}}$
```

Second, it is unlikely that a uniformly sampled transformation is valid or relevant, so starting at a random transformation would make solving Problem (3.8) more difficult.

In aug_robot, we are optimizing $\mathcal{L}_{\text{robot}}$. This corresponds to computing the augmented robot states $\tilde{\boldsymbol{q}}$ and actions $\tilde{\boldsymbol{a}}$ given the augmented states $\tilde{\boldsymbol{s}}$ and the environment $\mathcal{E}$. Minimizing $\mathcal{L}_{\text{robot}}$ means preserving the contacts the robot makes with the scene, which we do with inverse kinematics (Line 10 in Algorithm 2).

3.3.3 Learning the Valid Transforms Objective

As discussed above, we include a term $\mathcal{L}_{\text{valid}}$ based only on the transformation T. In some cases, such as our rope manipulation example, it may not be obvious how to define this objective manually. Our rope is very flexible, and therefore rotating the rope so that it floats in a sideways arc is invalid, but it may be valid for a stiff rope or cable. To address this, we offer a simple and data efficient algorithm for learning the transformation validity function f_{valid}.

Our method for learning f_{valid} is given in Algorithm 3. This algorithm repeatedly samples augmentations of increasing magnitude, and tests them on the system (lines 6 and 8). This generates ground truth states starting from an input state and action. The result is a dataset $\mathcal{D}_{\text{valid}}$ of examples (T, y_{valid}). We then train a small neural network $f_{\text{valid},\theta}(T)$ to predict the error y_{valid} and use the trained model as our transformation validity objective. We collect $n_{\text{valid}} = \sqrt{10^d}$ examples, where d is the dimensionality of the space of the transformation T.

This method owes its efficiency and simplicity to a few key assumptions about the system/data. First, we assume that we can collect a few (< 1000) examples from the system and test various transformations. This could be performed in a simulator, as we do in our experiments. Because the transformation validity objective is not a function of state, action, or environment, we can make simplifications to this simulation by picking states and environments which are easy to simulate. We denote this set of states and actions as Q_{valid}. Second, because the transformation parameters are low-dimensional (3 and 6 in our experiments) the trained model generalizes well with relatively few examples.

3.3.4 Application to Cluttered Planar Pushing

In this section, we describe how we apply the proposed method to learning the dynamics of pushing of 9 cylinders on a table (Figure 3.3). The moved object state s consists of the 2D positions and velocities of the cylinders. The robot state q is a list of joint positions, and the actions a are desired end effector positions in 2D. There is no w in this problem. The parameters T used are $SE(2)$ transforms. In this problem, any individual trajectory may include some moved cylinders and some stationary ones. In our formulation, the stationary cylinders are part of $\mathcal{E}$ and are not augmented, whereas the moved ones are part of s and are augmented. The robot's end effector (also a cylinder) is also augmented, and IK can be used to solve for joint configurations which match the augmented cylinders' state and preserve the contacts between the robot and the moved cylinders.

3.3.5 Application to Bimanual Rope Manipulation

In this section, we describe how we apply the proposed method to a bimanual rope manipulation problem (Figure 3.3). In this problem, there is a binary class label, so $w \in \{0, 1\}$, which is preserved under our augmentation (last constraint in Problem (3.1)). The rope is the moved object, and its state s is a set of 25 points in 3D. The robot state q is a list of the 18 joint positions, and the actions a are desired end effector positions in the robot frame. In this problem, we know that the only contacts the robot makes with the objects or environment are its grasps on the rope. Therefore, we preserve these contacts by solving for a robot state and action that match the augmented points on the rope. The parameters T used are $SE(3)$ transforms.

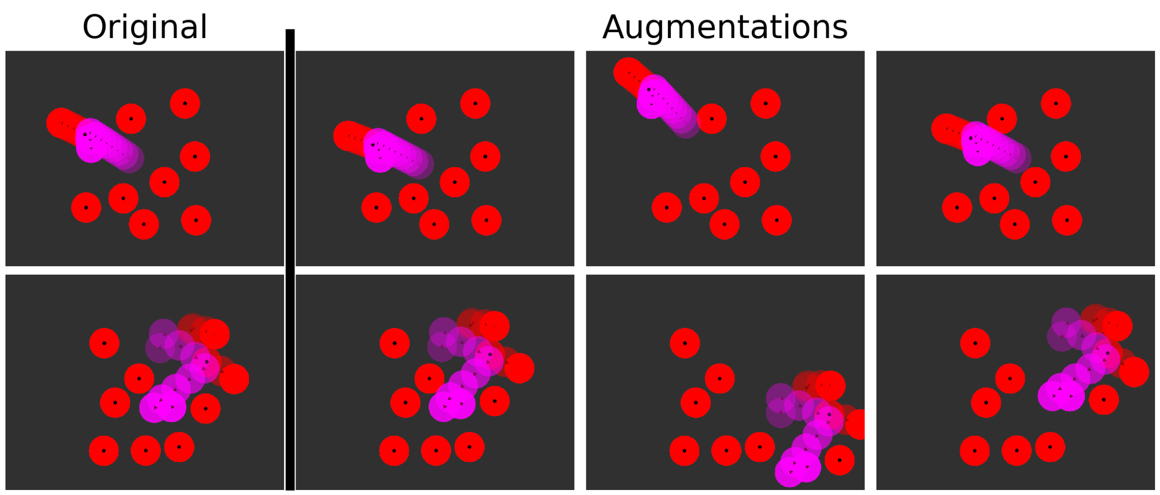

Figure 3.5: Examples of augmentations generated for learning the dynamics of planar pushing of 9 cylinders. The pink cylinder is the robot. Time is indicated by transparency. Augmentation transforms the positions and velocities of the cylinders that moved, including the robot. All moved objects are transformed together, rigidly. Despite the clutter, we are able to find relatively large transformations that still preserve existing contacts but do not create any new ones.

3.4 Results

We start by describing the tasks and our experimental methodology, then we present our results. These experiments are designed to show that training on augmentations generated by our method improves performance on a downstream task. We perform two simulated experiments, where we run thousands of evaluations, including several ablations (see Appendix 1.A). We also perform a real robot experiment (Figure 3.1) where we run 30 iterations of online validity classifier learning, with augmentation and without. In all experiments, we train until convergence or for a fixed number of training steps. This ensures a fair comparison to training without augmentation, despite the differing number of unique training examples. In all experiments, we generate 25 augmentations per original example (See Appendix 1.B). We define key hyperparameters of our method in Appendix 1.C. A link to our code is available on the project website.

3.4.1 Cluttered Planar Pushing

The cluttered planar pushing environment consists of a single robotic arm pushing 9 cylinders around on a table. The task is to learn the dynamics, so that the motion of the cylinders can be predicted given initial conditions and a sequence of robot

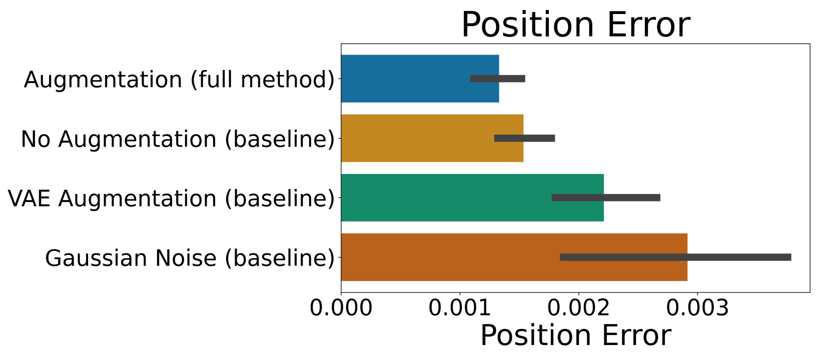

Figure 3.6: Mean position error (meters) for learning the dynamics of cluttered planar pushing.

actions. For this, we use PropNet [72], and our task is inspired by the application of PropNet to planar pushing in [120]. The inputs to PropNet are an initial state s_0 and a sequence of actions $\boldsymbol{a}$, and the targets are the future state $(s_1, \ldots, s_T)$. All trajectories are of length 50. We evaluate the learned dynamics by computing the mean and maximum errors for position and velocity on a held-out test set. Example augmentations for this scenario are shown in Figure 3.5.

This is an interesting application of our augmentation for several reasons. First, it is a regression task, which few augmentation methods allow. Second, the output of the dynamics network is high-dimensional (900 for a prediction of length 50), which normally means large datasets are needed and engineering invariances into the data or network is difficult. Finally, the trajectories contain non-negligible velocities and are not quasi-static.

The original dataset contained 60 trajectories of length 50, or 3000 time steps in total. For comparison, previous work on the same dynamics learning problem used over 100 000 time steps in their datasets [72, 120]. This is similar to the number of training examples we have *after* augmentation, which is 75 000 time steps. Finally, we measured the performance of our implementation and found that for the planar pushing scenario we generate 4.5 augmentations per second on average.

The primary results are shown in Figure 3.6. Augmentation reduces the average position error from 0.001 54 m to 0.001 33 m, a decrease of 14%. Additionally, we include two baselines, one which adds Gaussian noise to the state, robot, action, and environment data, and one which uses a VAE to generate augmentations as in [97]. The magnitude of the Gaussian noise was chosen manually to be small but visually noticeable. Our proposed augmentation method is statistically significantly better than the baseline without augmentation ($p < 0.0362$), the Gaussian noise baseline

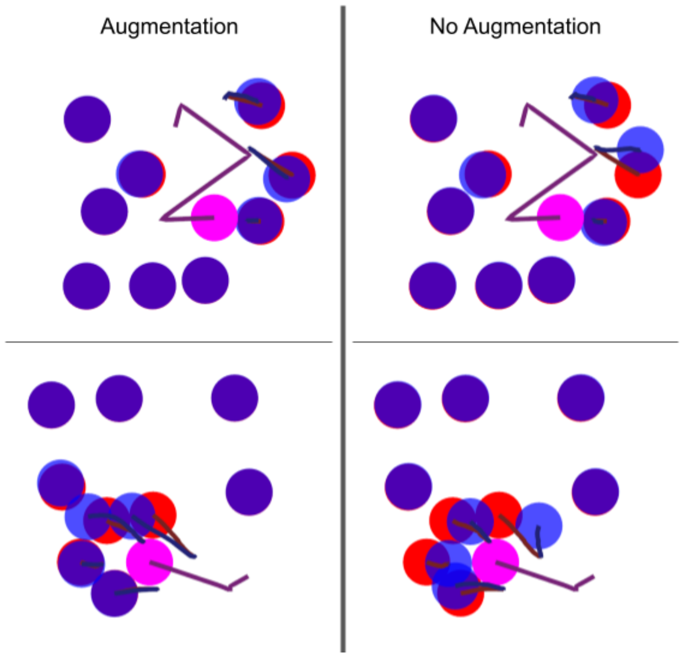

Figure 3.7: Predictions (blue) vs. ground truth (red) for planar pushing. The robot is in pink. Trajectories are visualized with lines. The left column shows predictions from a model trained with augmentation, the right column without.

($p < 0.0001$), and the VAE baseline ($p < 0.0002$). This difference in error may seem small, but note that error is averaged over objects, and most objects are stationary. Two roll-outs from with-augmentation and from without-augmentation are shown in Figure 3.7. In particular, we found that augmentation reduces "drift," where the model predicts small movements for objects that should be stationary. Finally, we note that the Gaussian noise and VAE baselines perform worse than no augmentation, suggesting that data augmentation can hurt performance if the augmentations are done poorly.

3.4.2 Bimanual Rope Manipulation

In this task, the end points of a rope are held by the robot in its grippers in a scene resembling the engine bay of a car, similar to [87], and shown in Figure 3.3. The robot has two 7-dof arms attached to a 3-dof torso with parallel-jaw grippers. The tasks the robot performs in this scene mimic putting on or taking off lifting straps from the car engine, installing fluid hoses, or cable harnesses. These tasks require moving the strap/hose/cable through narrow passages and around protrusions to various specified goal positions without getting caught. One iteration consists of planning to the goal, executing open-loop, then repeating planning and execution until a timeout or the

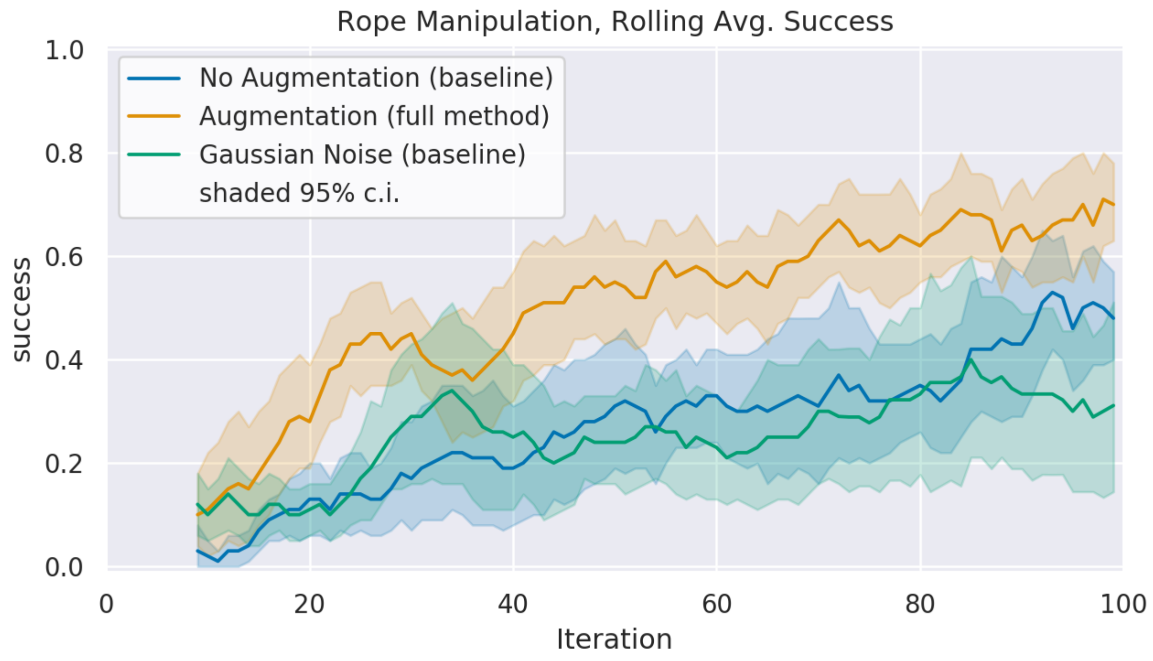

Figure 3.8: The success rate on simulated bimanual rope manipulation, using a moving window average of 10.

goal is reached. The goal is defined as a spherical region with 4.5 cm radius, and is satisfied when any of the points on the rope are inside this region.

The planner is an RRT with a learned constraint checker for edge validity (validity classifier), and more details are given in [87]. We want to learn a classifier that takes in a single transition $x = (s_t, a_t, s_{t+1}, \mathcal{E}_t)$ and predicts whether the transition is valid. Without a good constraint checker, the robot will plan trajectories that result in the rope being caught on obstacles or not reaching the goal. We apply our augmentation algorithm to the data for training this constraint checker. After an execution has completed, the newly-collected data along with all previously collected data are used to train the classifier until convergence. Example augmentations for this scenario are shown in Figure 3.2. The objective is to learn the constraint checker in as few iterations as possible, achieving a higher success rate with fewer data.

In this experiment, a total of 3038 examples were gathered (before augmentation, averaged over the 10 repetitions). Since the purpose of our augmentations is to improve performance using small datasets, it is important that this number is small. In contrast, prior work learning a similar classifier used over 100 000 examples in their datasets [87, 84]. This is similar to the number of training examples we have *after* augmentation, which is 75 950 on average. Finally, we measured the performance of our implementation and found that for the rope scenario we generate 27 augmenta-

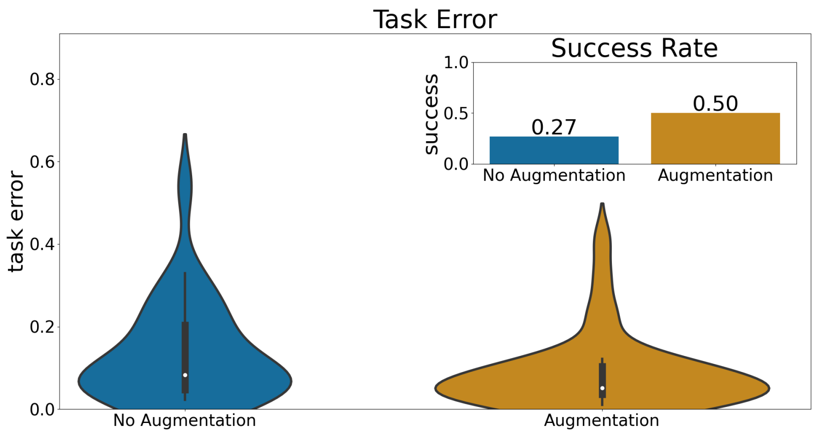

Figure 3.9: The success rate and task error distribution of bimanual rope manipulation on the real robot. Task error is the distance between the goal and the final observed state of the rope.

tions per second on average.

The primary results are shown in Figure 3.8. Over the course of 100 iterations, the success of our method using augmentation is higher than the baseline of not using augmentation, as well as the Gaussian noise baseline. We omit the VAE baseline, since it performed poorly in the planer pushing experiment. Furthermore, it is computationally prohibitive to retrain the VAE at each iteration, and fine-tuning the VAE online tends to get stuck in bad local minima. The shaded regions show the 95th percentile over 10 runs. If we analyze the success rates averaged over the final 10 iterations, we find that without augmentation the success rate is 48%, but with augmentation the success rate is 70%. The Gaussian noise baseline has a final success rate of 31%. A one-sided T-test confirms statistical significance ($p < 0.001$ for both).

3.4.3 Real Robot Results

In this section, we perform a similar experiment to the simulated bimanual rope manipulation experiment, but on real robot hardware. This demonstrates that our method is also effective on noisy sensor data. More importantly, it demonstrates how augmentation enables a robot to quickly learn a task in the real world. We use CDCPD2 [135] to track the rope state. The geometry of the car scene is approximated with primitive geometric shapes, like in the simulated car environment.

We ran the validity classifier learning procedure with a single start configuration and a single goal region, both with and without augmentation. After 30 iterations of learning, we stop and evaluate the learned classifiers several times. With augmentation, the robot successfully placed the rope under the engine 13/26 times. Without augmentation, it succeeded 7/26 times. The Gaussian noise and VAE baselines performs poorly in simulated experiments, therefore we omit them in the real robot experiments.

3.5 Limitations

First, our proposed method uses many hyper-parameters which may be difficult to tune. However, there are methods which can be used to automatically tune these parameters [8, 27].

Additionally, there are problems and applications where the proposed objective functions do not ensure validity, relevance, and diversity. In these cases, the structure of our augmentation and projection procedures can remain, while new objective functions are developed. Another limitation is that our method may not be applicable if the dataset also contains images, since the 3D transformations we apply to objects in the scene would require re-rending the altered scene. Much recent research in robotics has moved away from engineered state representations like poses with geometric information, and so there are many learning methods which operate directly on images. Extending the augmentation method to also operate on images is an open area for future research.

3.6 Conclusion

This chapter proposes a novel data augmentation method designed for trajectories of geometric state and action robot data. We introduce the idea that augmentations should be valid, relevant, and diverse, and use these to formalize data augmentation as an optimization problem. By leveraging optimization, our augmentations are not limited to simple operations like rotation and jitter. Instead, our method can find complex and precise transformations to the data that are valid, relevant, and diverse. Our results show that this method enables significantly better downstream task performance when training on small datasets. In simulated planar pushing, augmentation decreases the prediction error by 14%. In simulated bimanual rope manipulation, the success rate with augmentation is 70% compared to 47% without augmentation. We also perform the bimanual rope manipulation task in the real world, which demonstrates the effectiveness of our algorithm despite noisy sensor data. In the real world experiment, the success rate improves from 27% to 50% with the addition of augmentation.

CHAPTER IV

Focused Adaptation of Unreliable Dynamics

The previous chapter introduced a data augmentation method for manipulation. We used this method to learn reliability for dual-arm rope manipulation in the real world. The dynamics model used in these experiments was also trained on real world data, in a separate phase that was designed to ensure no contact between the rope and the environment. This extra step was time-consuming and hand-designed, and attempts to use free-space dynamics pretrained in simulation failed because the simulation and the real world rope dynamics were too different.

This chapter address this problem, by proposing a novel dynamics adaptation method. Our key insight is that a free-space data from simulation is far more similar to free-space data from the real world than it is to non-free-space data from the real world. More generally, the problem we address is to adapt a dynamics model from a source environment to a target environment, where the source and target dynamics are similar in some regions but different in others. Our proposed adaptation method focuses adaptation on data where the dynamics are similar, which we show leads to more accurate dynamics for planning and good estimates of the dynamics model's reliability. Ultimately, we use this method to simultaneously adapt a dynamics model learned in simulation and learn reliability – all online and in the real world.

4.1 Introduction

Learning dynamics models for manipulation is an increasingly popular paradigm, in part because learned models can be repeatedly improved using autonomously collected real-world data. However, fine-tuning an initial dynamics model on new data can perform poorly when the data contains complex dynamics on which the dynamics model was not initially trained. For example, suppose we want to manipulate a rope amongst clutter, and we have a dynamics model trained on free-space motions in simulation. Free-space transitions in the real world are fairly similar to the free-space in simulation, but transitions where the rope deforms on objects in the scene are very different from anything seen in simulation. We call these transitions *distracting*, because they are hard to learn from a few examples, and because they make it harder to adapt accurately to the free space dynamics. More generally, transitions from regions of dissimilar dynamics can inhibit effective transfer to regions of similar dynamics. This problem is similar to "cleaning" data in machine learning [18, 52, 15]. For dynamics learning, defining what "clean" means can be difficult, and has not been studied extensively. Instead, the dominant paradigm is simply to train on all the collected data.

However, training on all the data can fail because real world datasets for learning dynamics are often too small to learn generalized models over the entire state-action space. In our experiments, we show that simply fine-tuning on all the data can yield a model that is not accurate enough for planning. If the task can be completed while remaining in regions where dynamics are similar, then it can be worth trading accuracy in dissimilar regions for accuracy in similar regions. Our key insight is that, when we are adapting from an initial model, we can leverage the initial model to achieve significantly lower prediction error by focusing on transitions where the source and target dynamics are the most similar. The idea that transfer is easier when the source and target data are similar is well-supported in the transfer learning literature [115, 20]. Concurrent work in offline reinforcement learning has also explored how staying closed to similar data leads to better policies [126].

To implement this strategy, we propose an adaptation method which minimizes prediction error in regions where the source and target dynamics are similar. The proposed method minimizes prediction error in these regions by fine-tuning on an initially small set of data from these regions, and growing that subset over the course of training. This is done with a loss function inspired by curriculum-learning that weighs transitions according to their prediction error, assigning higher weight to low-error transitions. Under the assumption that there are paths to the goal where the source and target dynamics are similar, this adaptation method can be used to achieve high task success in the target environment.

The first contribution of this chapter is a method for adapting dynamics models to datasets which contain distracting transitions. We demonstrate the proposed method is successful in filtering out distracting data and that the resulting trained model is more accurate in the regions of state-action space where the source and tar-

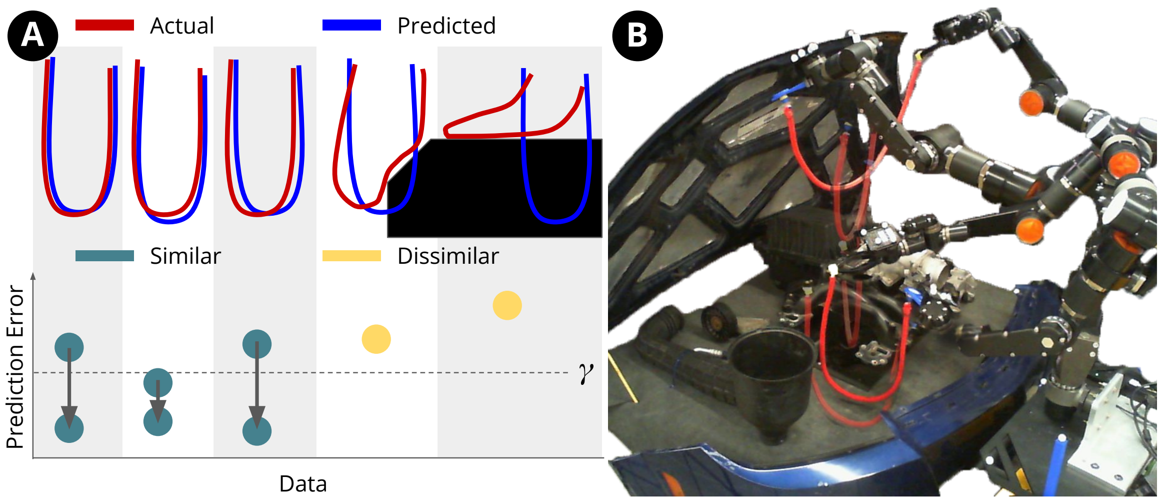

Figure 4.1: (A) An illustration of how our adaptation method focuses on regions where the source and target dynamics are similar. When focusing adaption on free-space dynamics, the prediction errors decrease for other free-space data (similar), do not decrease for collision dynamics (dissimilar). (B) A mock-up of a car engine bay. The robot must move the rope and place it under the engine without snagging it to set up for lifting the engine. We use our proposed adaptation method to improve success rate during online learning for this task.

get dynamics are similar. The second contribution is a data-efficient online-learning method that pairs our adaptation method with prior work on planning with unreliable dynamics models [87, 64]. We call our combined method for online learning FOCUS. FOCUS achieves higher success rates in the low-data regime because the adapted dynamics are more accurate, which leads to finding more reliable plans.

4.2 Problem Statement

The problem addressed in this chapter is to adapt a dynamics model trained in a source environment to data collected in a target environment, where the source and target environments have dynamics which are similar in some regions of the state-action space, but different in others. Furthermore, we consider the case where data collection is done by planning and executing paths to goals in the target environment using the learned dynamics model.

To formalize this, first consider the standard dynamics learning problem with a dataset $\mathcal{D}$ of transitions of states, actions, and next states (s, a, s'). We also assume a distance function $\text{dist}(s_1, s_2)$ that returns a scalar is given. The true dynamics are $s' = f(s, a)$, and the learned dynamics are $\hat{s}' = \hat{f}(s, a)$. We propose defining the source and target environment dynamics as similar for a transition if $\text{dist}(f_S(s, a), 0_T f(s, a)) <$

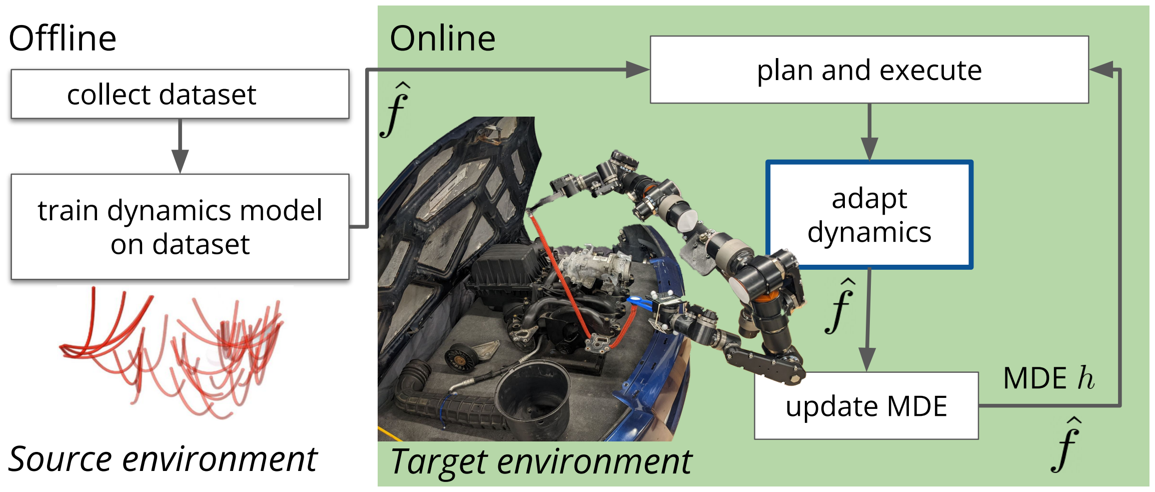

Figure 4.2: Block diagram showing the steps of our full online adaptation method. A dynamics model is initialized offline in the source environment (left), then adapted online in the target environment.

γ. The threshold γ should be small enough that it excludes distracting transitions, but large enough to include as much data from the target environment as possible. Let $\mathcal{D}_{ST}$ be the set of transitions from the target environment where this similarity condition holds.

In order to minimize the amount of data needed for adaptation to generalize, we aim to adapt the dynamics only to transitions from regions where this similarity condition holds ($\mathcal{D}_{ST}$). However, we also care about successfully completing the task, and therefore we also assume there are paths $\{s^0, a^0, \dots, a^{T-1}, s^T\}$ to the goal $s^T \in \mathcal{G}$ within the regions of similar dynamics $(s^t, a^t, s^{t+1}) \in \mathcal{D}_{ST}$.

While the ultimate objective of adaptation should be to maximize task success in the target environment, an important condition for task success is minimizing prediction error on $\mathcal{D}_{ST}$. If the goal is reachable within $\mathcal{D}_{ST}$ (as we assume), the prediction error on $\mathcal{D}_{ST}$ is small (our objective), and our motion planner is constrained to stay in $\mathcal{D}_{ST}$, then we can also expect high task success. Next, we discuss how our adaptation method minimizes error on $\mathcal{D}_{ST}$, followed by how we can achieve high task success by additionally constraining a motion planner to $\mathcal{D}_{ST}$.

4.3 Methods

4.3.1 Adapting the Dynamics

At a high level, our method minimizes prediction error on $\mathcal{D}_{ST}$ by dynamically weighing the training data $\mathcal{D}$ such that transitions that are likely to be in $\mathcal{D}_{ST}$ are given weights near 1 and transitions unlikely to be in $\mathcal{D}_{ST}$ are given weights near 0. Given a transition from our training data $(s, a, s') \in \mathcal{D}$, we cannot directly evaluate whether that transition is in $\mathcal{D}_{ST}$, since that would require knowing the true dynamics $0_T f$. However, since the initial dynamics, denoted $\hat{f}_0$, is assumed to accurately fit the source dynamics, it is likely that transitions with low prediction error under the initial dynamics $\text{dist}\big(\hat{f}_0(s, a), s'\big) < \gamma$ are in $\mathcal{D}_{ST}$. By training on transitions with initially low error, we expect the prediction error on other transitions which belong to $\mathcal{D}_{ST}$ to also decrease. This slowly brings more and more transitions to have prediction errors below γ. On the other hand, the prediction error is unlikely to decrease for transitions not in $\mathcal{D}_{ST}$, because they are dissimilar to the transitions with low initial error. Thus, at each step j of training, we assign each transition a weight as a function of the prediction error $||\hat{s}' - s'||^2$, and multiply this weight by the loss. The full loss is shown in Equation (4.1).

$$\begin{aligned}\mathcal{L}_f &= \frac{1}{T}\sum_{t=1}^{T}\Big(||\hat{s}^t - s^t||^2 w^t\Big) \\ w^t &= 1 - \sigma\big(\phi(j)(||\hat{s}^t - s^t||^2 - \gamma)\big)\end{aligned} \tag{4.1}$$

σ is the sigmoid function. When j is large, the boundary is almost hard, and transitions with error below γ have weight near 1 and transitions with error above have weight near 0. When j is small, the boundary is soft, and the weights vary less. The term $\phi(j)$ controls the rate of change of hardness. We use $\phi(j) = 0.5j$ in our experiments. We found that allowing the weighting to be soft during early training steps improves the stability in the case where few or no transitions have error below γ at the beginning of training. The parameter γ can be chosen based on either the maximum error that can be corrected by a low-level controller, or based on the distribution of error on a validation set from the source environment (e.g the 97th percentile).

4.3.2 Online Learning

In this section, we describe how the proposed adaptation method can be combined with prior work on planning with unreliable dynamics to achieve data-efficient online adaptation of dynamics models. A block diagram of the full method, which we call FOCUS, is shown in Figure 4.2. FOCUS consists of an offline phase and an online phase. In the offline phase, we train a dynamics model using data from the source

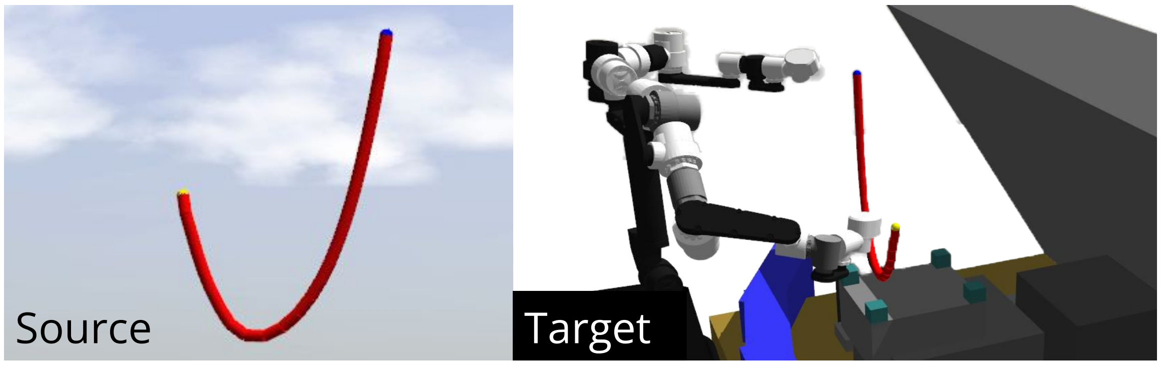

Figure 4.3: Source environment for bimanual rope manipulation (left) and simulated target environment (right) where there is a robot, obstacles, and the rope damping and stiffness are changed.

environment, which in our experiments is a simple simulation. We use random actions to collect a diverse set of data and standard techniques for training the neural network dynamics model [87].

In the online phase, we adapt the learned dynamics model to the target environment (e.g. the real world). This process alternates between (1) collecting new data in the target environment by planning and executing, (2) fine-tuning the dynamics, and (3) fine-tuning the model deviation estimator (MDE). We now explain the data collection and MDE fine-tuning steps.

4.3.2.1 Planning and Execution for Data Collection

We use a kinodynamic RRT planner where nodes are propagated using the learned dynamics model. Since the learned dynamics are adapting to only $\mathcal{D}_{ST}$ and are not accurate everywhere, we additionally constrain the planner to stay in $\mathcal{D}_{ST}$. Since $\mathcal{D}_{ST}$ is not known *a priori*, we train another neural network, called a model deviation estimator (MDE) [64, 84, 87], to predict the error of the dynamics model (more details in Section 4.3.2.2). In addition to producing more robust plans, planning with MDEs has the additional benefit of focusing data collection on $\mathcal{D}_{ST}$. By collecting more data where the source and target dynamics are close, a larger fraction of $\mathcal{D}$ is likely to be in $\mathcal{D}_{ST}$, and therefore the adaptation procedure has more data from which to learn.

We use the MDE in planning as a constraint checker. If the dynamics error predicted by the MDE is below a threshold $d_{\max}$, then we add it to the planning tree. We also randomly accept transitions with high predicted dynamics error with low probability (0.01), so that the planner will occasionally return paths with exploratory actions (we call these *random-accepts*). These exploratory actions are essential for

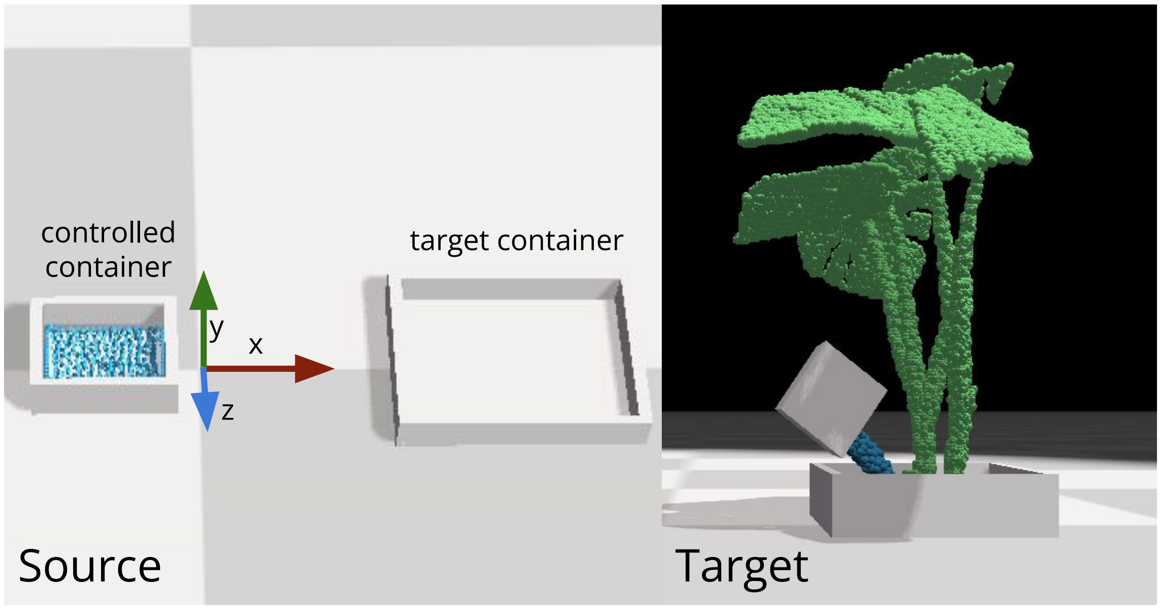

Figure 4.4: Source environment for plant watering (left) and target environment (right) where there is an additional plant, and the viscosity is tripled.

training the MDE, since they can correct over-estimation of model error from the MDE. The threshold $d_{\max}$ for allowable error is similar to γ, but may be set higher or lower to control the exploration/exploitation tradeoff.

The robot uses the planner to attempt the task, and repeatedly plans and executes open-loop until a timeout or the goal is reached. If no plan is found that reaches the goal, the plan which gets closest to the goal is executed. This repeated planning and execution is called one episode. After some fixed number of episodes (e.g. 10) we fine tune the dynamics and the MDE using all data collected so far during the online phase.

4.3.2.2 Fine-tune MDE

The MDE is used to constrain planning to regions where the dynamics model is predicted to be accurate, which has two benefits. First, it helps bias data collection to contain transitions from $\mathcal{D}_{ST}$. Second, it makes reaching the goal more likely since it avoids plans that do not match to the true dynamics. The MDE $\hat{d} = h(\mathcal{E}, s, a, \hat{s}')$ is a convolutional neural network which takes as input the environment, state, action, and next predicted state, and predicts the error of the dynamics model $\hat{d}$. We represent the environment $\mathcal{E}$ as a voxelgrid of the scene. The ground truth error used for training is the error between the true observed state and the state predicted by the learned dynamics: $d = \text{dist}(\hat{s}', s')$. The loss function is shown in Equation (4.2).

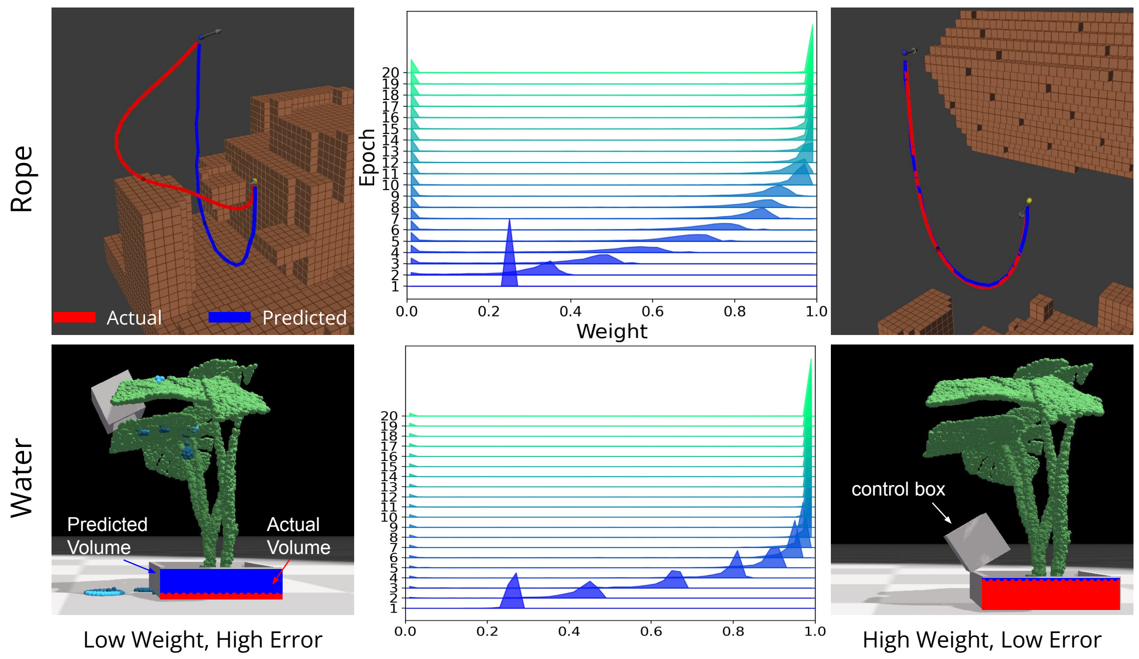

Figure 4.5: (center) Histograms showing weights assigned to the data according to Equation (4.1) during the first 20 epochs of training. A histogram is shown for each epoch, where color varies with epoch, and these histograms are staggered along the y-axis. Initially, the weights vary only slightly across the data, but the distribution becomes strongly bimodal as training progresses. Examples of transitions given weight 0 (left) and weight 1 (right) at the end of training.

Intuitively, the MDE should be easier to learn with fewer data than learning the dynamics accurately everywhere, since the MDE need only predict the magnitude of the error as opposed to the full state vector [87].

$$\mathcal{L}_h = ||\hat{d} - d||^2 e^{-k_h d} \tag{4.2}$$

k_h is a hyperparameter that reduces the need to predict high dynamics errors with high accuracy. We set $k_h = 10$.

4.4 Results

We begin by describing the two domains for our experiments: bimanual rope manipulation and plant watering. We then validate the claims that (1) our proposed adaptation method achieves lower prediction error in regions of similar dynamics, and (2) that FOCUS achieves higher success rates more quickly in the online adaptation setting compared to baselines which train on all data equally.

4.4.1 Bimanual Rope Manipulation

In this task, a 16-DOF dual arm robot is holding two ends of a rope in a scene resembling the engine bay of a car (scene shown in Figures 4.1,4.3). The task mimics putting on lifting straps on the engine, which requires moving the rope through narrow passages and around protrusions. The goal is to place the middle of the rope in a goal region defined as a sphere of radius 0.045 m. The planner outputs gripper position actions, and a local controller executes the actions while maintaining gripper orientations. The learned dynamics model predicts the state of the rope, represented as 25 points, given the initial rope state and gripper position actions.

In the rope manipulation experiments, the source simulation has no obstacles and the robot is simplified to floating kinematic grippers. We then test adaptation to two different target environments: (1) another simulation which includes the robot and obstacles and has different damping and stiffness parameters for the rope, and (2) the real world. Thus, this tests adaptation to a different rope despite the distracting transitions where the rope deforms on the robot or the obstacles. Gazebo with ODE physics is used for simulation [59]. For rope manipulation, the set $\mathcal{D}_{ST}$ would be the transitions from the target environment where the rope is in free space. We use $\gamma = 0.08$.

4.4.2 Plant Watering

The goal in this task, illustrated in Figure 4.4 is to pour at least 75% of the initial volume from a controlled container into a target container without spilling more than 5%. The source environment is a variation from the SoftGym PourWater environment [75]. The target environment has triple the viscosity, a shorter container, and a plant in the target container. Although the agent can pour from above, that causes the water to splatter, which is more difficult to predict than the free-space pours of the target environment. Additionally, the box can collide with the plant, which is dissimilar to the source dynamics where there are no obstacles. The controlled container can rotate about the z-axis The state space is the 4-DOF $[x, y, z, \theta]$ pose of the controlled container, 3-DOF $[x, y, z]$ pose of the target container, control volume, and target volume. The action space is a target pose $[x_{des}, y_{des}, \theta_{des}]$ which is followed by a proportional controller. For plant watering, the set $\mathcal{D}_{ST}$ contains free-space motions and pours, whereas collision with and pouring on the plant is not in $\mathcal{D}_{ST}$. We use $\gamma = 0.05$.

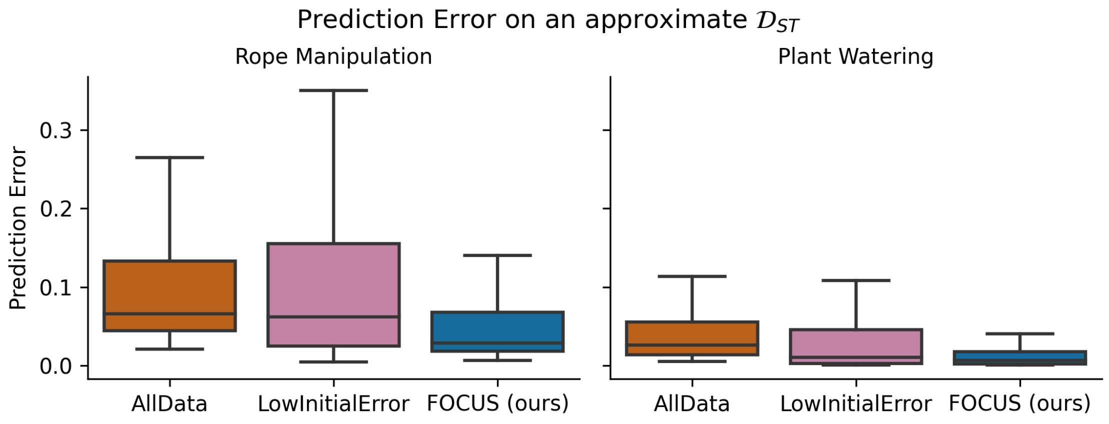

Figure 4.6: Prediction error for our method versus two baselines, evaluated on a dataset of transitions from regions where the source and target dynamics are similar.

4.4.3 Validating the Adaptation Method

We now evaluate whether the proposed adaptation method achieves lower prediction error on $\mathcal{D}_{ST}$. We start by creating validation sets which contain transitions not used for training, and which are known to be in $\mathcal{D}_{ST}$. For rope, this means transitions where the rope is in free space. For water, this means transitions which do not collide with or pour over the plant. We evaluate our method and two baselines, all starting from the same pre-trained model and adapting to the same dataset from the target environment. The baseline *AllData* fine-tunes on all transitions with equal weights. The baseline *LowInitialError* uses our weighting function, but computes the weights once using $\hat{f}_0$ and does not re-compute them throughout training. Our method uses our proposed loss function (Equation (4.1)) which re-computes the weights on each batch during training.

For bimanual rope manipulation, the dataset contains 6288 transitions and the validation set contains 792 transitions. For plant watering, the training dataset contains 854 transitions and the validation set contains 130 transitions. The results are visualized in Figure 4.6. In both experiments, the error of our method is statistically significantly lower than both baselines ($p < 0.0001$).

Figure 4.5, demonstrates the intuition behind our adaptation method. In the center, we show histograms over time of the weights assigned to the transitions in the training dataset, for water and for rope. The distribution is initially unimodal since the weighting function when $j = 0$ is soft, but as training progresses the distribution rapidly becomes bimodal, where most transitions are given a weight of 1, but some transitions are given a weight of 0. We show examples of these low and high weight

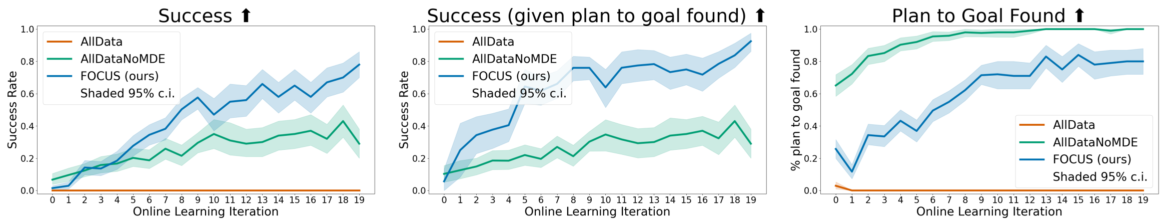

Figure 4.7: Post-learning evaluation of rope manipulation in simulation: Three metrics shown over the 20 iterations of online learning. The shaded interval is the 95% confidence interval, with the boot-strapping method used by Seaborn.

transitions on either side. For rope manipulation, we found that the number of the transitions with prediction error below γ increases from 52% at epoch 1 to 80% at epoch 20, which shows that the subset of data we train on grows. This explains why our method outperforms the *LowInitialError* baseline, since that baseline is not making use of as much of the data as our method does. The presence of transitions with 0 weight (e.g. 20% for rope at epoch 20) shows that our method is not converging to training on all examples, which does not perform well based on the *AllData* baseline.

4.4.4 Online Learning Experiments

We show that FOCUS achieves higher task success with fewer data than baselines which fine-tune the dynamics on all available data. The first baseline, called *AllDataNoMDE* does not use our proposed adaptation method and does not use MDEs when planning, which makes it a conventional online learning method. The second, called *AllData* includes MDEs in planning, but ablates our fine-tuning method. First, we evaluate in the rope manipulation domain on adaptation from one simulation to another (see Figure 4.3). We ran 20 iterations of online learning, where each iteration consists of 10 episodes, which amounts to roughly $6,000$ transitions in total during learning. We then repeated this 10 times for each method/baseline with different random seeds.

After learning, we took the models saved after each learning iteration and ran 100 episodes of evaluation per method. To maximize success rates of all methods, we use a longer timeout and do not allow random-accepts when planning. We also stop execution and replan if the error between the plan and the observed state exceeds a large threshold on model error (0.25).

The results of this first experiment are summarized in Figure 4.7. The proposed method (FOCUS) shows the highest success rate when plans are found. The *AllData* method, which ablates our method for fine-tuning the dynamics, never finds paths to

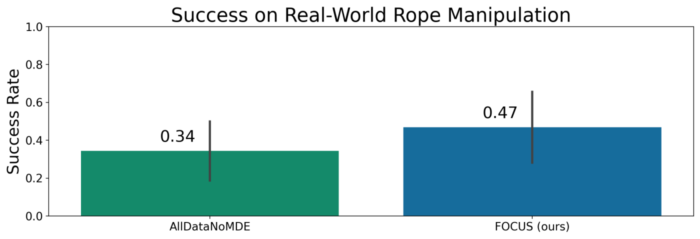

Figure 4.8: Success rate for the *AllDataNoMDE* baseline (left) versus FOCUS (right) for online adaptation to real world bimanual rope manipulation.

the goal. This is because its dynamics are not sufficiently accurate, and so the MDE constraint makes the planning problem infeasible. Accurately learning the dynamics in the *AllData* or *AllDataNoMDE* methods would involve predicting the deformation of the rope on obstacles, which is challenging given a dataset of only a few thousand transitions.

4.4.5 Real Robot Results

We performed a similar experiment to the first rope manipulation experiment, but on real robot hardware where sensor and actuator noise are substantial factors (approximately 5 cm of end-effector error). More importantly, it demonstrates how FOCUS enables a robot to quickly learn a task in the real world. We use CDCPD2 [135] to track the rope state. The geometry of the car scene is approximated with primitive geometric shapes. We use the same source simulation environment as for the simulation rope experiment, but now the target environment is the real world. The robot must learn to adapt the simulated free-space rope dynamics to the real world, despite the different real-world free-space dynamics and the fact that the rope can deform on the robots' arms or on the objects in the scene. Because perception and actuation error are higher in the real world than in simulation, we use $\gamma = 0.2$.

We ran the online learning procedure with a single start configuration and a single goal region for one random seed and compare FOCUS to the *AllDataNoMDE* baseline, since *AllDataNoMDE* performed best in simulation. After 15 iterations of learning, we freeze the models and evaluate task success 32 times. The success rates are shown in Figure 4.8. With FOCUS, the robot successfully placed the rope under the engine 15/32 times, while *AllDataNoMDE* succeeded 11/32 times. Achieving high success rates in this task is difficult due to narrow passages, perception error, and actuator error. Failure modes for FOCUS include the rope getting pulled out of the robots' hands, getting too close and catching on obstacles, and failing to find plans that reach the goal.

4.5 Conclusion

This chapter studies the problem of adapting learned dynamics models to datasets which contains transitions where the dynamics are very different from the source environment. This type of domain mismatch is common in online dynamics learning settings, where the source dynamics are learned in simulation or on a simpler task. Traditional adaptation methods can fail in this setting because trying to fit data from regions of dissimilar dynamics leads to poor predictions even in regions where the source and target dynamics are similar.

Our key insight is to instead focus adaptation on regions where the source and target dynamics are similar. We propose an adaptation method which assigns high weight to transitions with low prediction error, and dynamically re-assigns weights during the course of training. The set of low-error transitions is initially a small set, but grows as training pulls down the prediction error for other similar transitions. We combine our adaptation method with prior work on planning with unreliable dynamics to make FOCUS, a data-efficient online adaptation method. We demonstrate that FOCUS can learn a bimanual rope manipulation task in simulation and in the real world, and achieves higher task success rates than baselines which attempt to fit all the training data.

CHAPTER V

The Grasp Loop Signature

The previous chapter introduced a method for adapting dynamics when some regions of the source and target dynamics are similar and others are not. This method allowed us to adapt a dynamics model and learn reliability in the real world. This also removed the need for specialized methods for collecting real-world training data from specific regions of dynamics.

However, for all the dual-arm DOO manipulation experiments thus far, we have assumed the robot is already grasping the object, and that these grasps cannot change during the manipulation. This assumption severely limits the tasks we can do, and in some cases also makes the manipulation slow compared to an approach that allows regrasping. This chapter begins to relax this assumption and use regrasping. Although I am not the first to consider regrasping deformable objects (see Section 1.2 for related work), there has been little attention devoted to regrasping DOOs, and prior methods are not widely applicable to different robot morphologies or tasks. Using the proposed method, I enable robots to re-grasp when stuck, enabling new tasks and increasing robustness. In this chapter, we use a simulator as our dynamics model and assume it is tuned to be accurate for the given task. This assumption could be relaxed by integrating methods from previous chapters, but further experiments would be needed to test this.

5.1 Introduction

Manipulation planning for deformable one-dimensional objects (DOOs) like ropes and cables is challenging due to the high-dimensional state representation of these objects and the cost of simulating their motion. Furthermore, most tasks benefit from multiple arms to control DOO shape and avoid becoming tangled with the environment. Therefore, the planner needs to consider the DOO, the arms manipulating it,

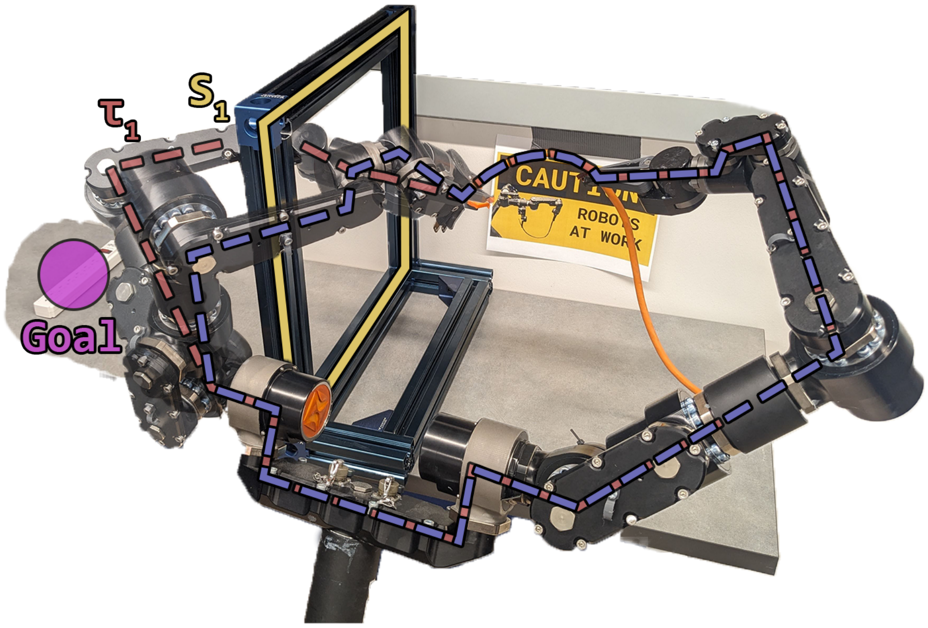

Figure 5.1: Annotated image of our real world cable threading setup. The red dashed line shows a grasp loop τ_1 that is linked with the skeleton S_1. The blue grasp loop is not linked with S_1. This distinction is captured by the proposed $\mathcal{G}_L$-signature and is used in planning.

and the environment. A task and motion planning (TAMP) approach to this problem would decompose planning into a grasp selection problem and a motion planning problem for the DOO given a specific grasp, as in [129, 95, 141]. However, the DOO planning problems are often expensive to solve. To reduce the space of grasps we need to search, we borrow the idea of a *signature* from the field of topology.

To explain what this signature represents, consider how the robot should grasp the tip of the cable in Figure 5.1. By grasping we form a loop, which we call a *grasp loop* and show as blue and red dashed lines in Figure 5.1. It is possible to grasp either around the left side or the right side of the frame, but these two grasps are categorically different in that we cannot smoothly deform from one to the other without breaking the grasp or the frame. The frame also forms a loop, called an obstacle loop. When grasping from the left (red), these two loops are linked, but when grasping from the right (blue) they are not. Our key insight is that the robot, DOO, and environment form a *graph of grasp loops* and we can use this graph to construct a signature, $\mathcal{G}_L$-signature, which captures topological information relevant for planning. To be clear, we do not address knots in the DOO. Our work is complimentary to work on tying or untying knots [127, 106, 119, 132].

The main contribution of this chapter is the $\mathcal{G}_L$-signature which compactly repre-

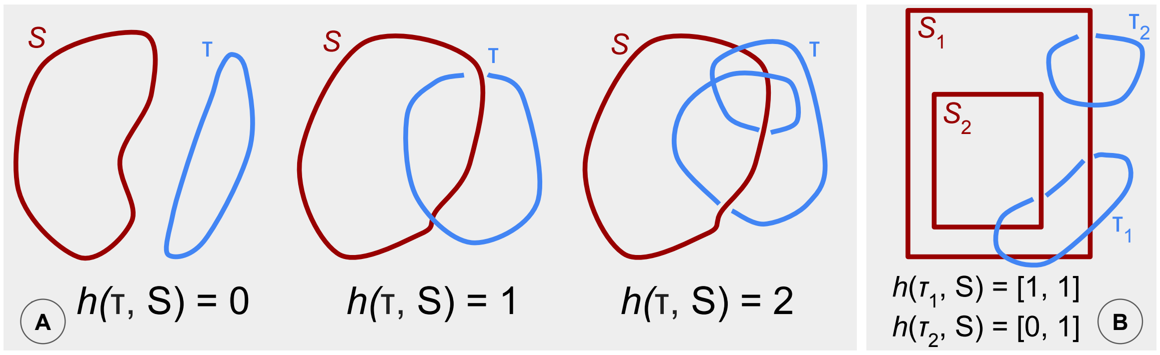

Figure 5.2: (A) Illustration of the h-signature for a loop representing the robot and DOO (blue) and a loop representing an obstacle (solid red). (B) Two examples of the h-signature for a skeleton with two obstacle loops S_1 and S_2.

sents the topology of both the object and the arms manipulating it. We claim this signature is applicable to many systems and is useful for manipulation planning. Figure 5.4 shows three examples where we demonstrate planning, and two more examples where the $\mathcal{G}_L$-signature may be useful. In simulation, we show that methods using the $\mathcal{G}_L$-signature outperform baselines and ablations which search for grasps without using topological information. Finally, we demonstrate a threading and point reaching task on a physical robot. Videos and animations can be found on our Project Website[1].

In the remainder of this chapter, we first review related work and then define the $\mathcal{G}_L$-signature. Next, we describe a method for DOO manipulation that demonstrates the utility of the $\mathcal{G}_L$-signature. Section 5.5 describes how this method can be applied in three environments, which we call Pulling, Untangling, and Threading. We conclude with a brief discussion of our real world demonstration of the Threading task, which is depicted in Figure 5.1.

5.2 Defining the $\mathcal{G}_L$-signature

5.2.1 Preliminaries

We primarily use notation that is consistent with [13]. We call a closed one-dimensional curve in 3D a *loop*. The environment is assumed to be decomposed into a *skeleton* made up of multiple *obstacle loops* $\mathbf{S} = \{S_1, \ldots, S_n\}$. Each obstacle loop is made up of line segments $S_i = \{\mathbf{s}_i^1, \ldots, \mathbf{s}_i^{n_i}\}$. An example environment and corresponding skeleton is shown in Figure 5.3. In practice, the skeleton can either

[1]https://sites.google.com/view/doo-manipulation-signature/home

be specified manually or computed automatically from a medial axis transform of a mesh or pointcloud of the environment.

[13] plans paths that are in a given homotopy class or avoid a certain homotopy class. They compare two paths by considering the homotopy class of the closed loop τ formed by joining the two paths at their shared start and end points. For a path loop τ and an obstacle loop S, [13] defines the h-signature $h(\tau, S) \in \mathbb{Z}$, which counts the number of times τ passes through S. The sign of h in this case is determined by the direction of τ. The h-signature can be extended to a list of the h-signatures with respect to each obstacle loop in the skeleton $h(\tau, \mathbf{S})) = [h(\tau, S_1), \dots, h(\tau, S_n)]$. These cases are illustrated in Figure 5.2. The equation for computing $h(\tau, S)$ is reproduced from [13]. The point $\mathbf{s}_i^{j'}$ is the point that follows $\mathbf{s}_i^j$, and r is a point on the loop τ. The integration over τ is done numerically.

$$\begin{aligned} h(\tau, S) &= \frac{1}{4\pi} \int\limits_{\tau} \sum_{j=1}^{n_i} \Phi(\mathbf{s}_i^j, \mathbf{s}_i^{j'}, r) \Delta r \\ \Phi(\mathbf{s}_i^j, \mathbf{s}_i^{j'}, r) &= \frac{1}{||d||^2} \Big(\frac{d \times p'}{||p'||} - \frac{d \times p}{||p||} \Big) \\ p = \mathbf{s}_i^j - r, \quad p' = \mathbf{s}_i^{j'}, \quad d &= \frac{(\mathbf{s}_i^{j'} - \mathbf{s}_i^j) \times (p \times p')}{||\mathbf{s}_i^{j'} - \mathbf{s}_i^j||^2} \end{aligned} \tag{5.1}$$

We take this idea but apply it to grasp loops, instead of paths. Unlike in path planning, where the direction of τ matters, we only care how or whether loops are linked. Accordingly, we assert that h is always non-negative.

5.2.2 Computing the $\mathcal{G}_L$-signature

The $\mathcal{G}_L$-signature is composed of the h-signatures $h(\tau, \mathbf{S})$ of grasp loops τ formed by the robot and DOO. We describe the process here and in Algorithm 4. The grasp loops are constructed based on a graphical model of the state $\mathcal{G} = (V, E)$ where vertices V are the robot base, its grippers, and attach points, and edges are paths between them. Attach points are used to represent locations on the DOO which are fixed relative to the robot (e.g plugged into the wall or rigidly mounted on the robot itself). Figure 5.3 illustrates how the graph construction step works. The robot base vertex is connected to all gripper and attach vertices because it connects to the grippers directly (via the robot geometry) and the attach points indirectly (via the environment). Edges E connect grippers/attach points to one another if they are

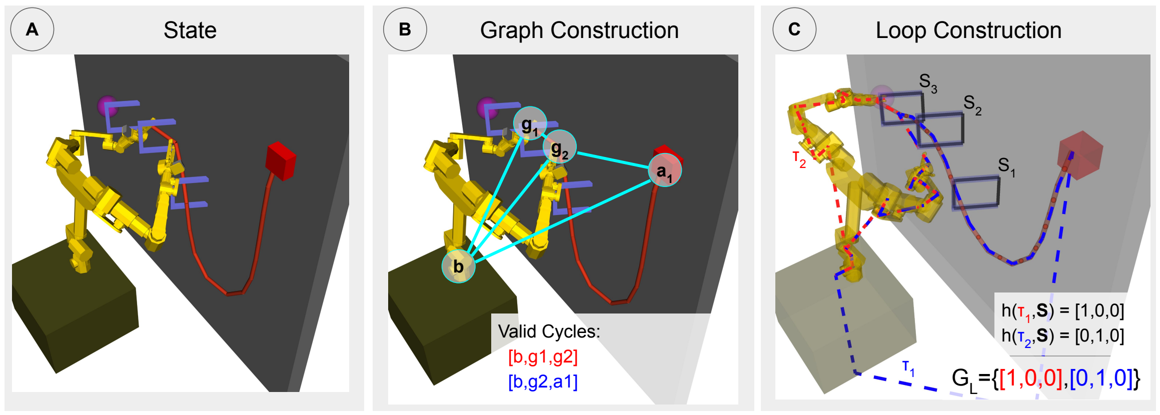

Figure 5.3: The process of constructing the $\mathcal{G}_L$-signature. (C) There are 2 grasp loops and 3 object loops, so the $\mathcal{G}_L$-signature is a set with two elements, and each element is a vector of 3 non-negative integers.

adjacent on the DOO. In Figure 5.3, the vertices (g_1, g_2) are adjacent, as are (g_2, a_1), but (g_1, a_1) are not. In Algorithm 5, the function $\text{Adj}(v_i, v_j, s)$ checks for adjacency between v_i and v_j at the given state s.

From $\mathcal{G}$, we extract all cycles $\mathcal{O}$ of exactly 3 distinct vertices which contain a gripper (`getValidCycles`), and convert each cycle to a grasp loop τ (`getLoops`). To make a grasp loop from a cycle, we concatenate the 3D paths represented by the cycles' edges. This requires a skeletonized representation of the robot geometry, which can be constructed from the kinematic tree and the origins of the links, as well as the points representing the DOO. A path between the robot base and an attach point (e.g. (b, a_1) in Figure 5.3) can be chosen arbitrarily, as long as it is the same for all states. Cycles not containing a gripper (e.g. (b, a_1, a_2)) are omitted for compactness, since attach points presumably cannot be changed by the planner.

For each grasp loop, we compute the associated h-signature $h(\tau_i, \mathbf{S})$. The $\mathcal{G}_L$-signature of the state, denoted $\mathcal{G}_L(s)$, is the *multiset* of the h-signatures of each grasp loop. In a multiset the order does not matter, but elements may repeat. The number of repetitions of an element is called its multiplicity. Two multisets are equivalent if their elements and multiplicities are equal. Preserving repetitions in the $\mathcal{G}_L$-signature allows us to represent multiple grasp loops that go through the same obstacle loop.

This may result in a grasp loop containing two grippers that has $h(\tau, \mathbf{S}) = \mathbf{0}$ (i.e. not linked $\mathbf{S}$). The red dashed grasp loop shown in Figure 5.4 B3 is an example of this. Releasing one of the grippers does not categorically change what we can do with the object, and neither would grasping with an additional gripper right next to two already grasping. Therefore, if there is a cycle with $h(\tau, \mathbf{S}) = \mathbf{0}$ containing two

Algorithm 4: Compute the $\mathcal{G}_L$-signature

```
   Input  : $V, \mathbf{S}, s$
   Output: $\mathcal{G}_L(s)$
1  $\mathcal{G} =$ addEdges$(V, s)$                    // Graph construction
2  $\mathcal{O} =$ getValidCycles$(\mathcal{G})$
3  if $|\mathcal{O}| = 0$ then
4      return $\emptyset$
5  $\tau =$ getLoops$(\mathcal{G}, \mathcal{O}, s)$
   // Remove empty gripper-gripper cycles (optional)
6  for $\tau_i \in \tau, o_i \in \mathcal{O}$ do
7      if $h(\tau_i, \boldsymbol{S}) = 0$ then
8          for $v_i, v_j \in o_i$ do
9              if $v_i, v_j$ are gripper vertices then
10                 $V = V \setminus v_j$     // choice of $v_j$ or $v_i$ is arbitrary
11                 goto 1 Graph construction
   // Compute final $\mathcal{G}_L$-signature
12 $\mathcal{G}_L(s) =$ MultiSet$\{h(\tau_i, \mathbf{S}) | \tau_i \in \tau\}$
13 return $\mathcal{G}_L(s)$
```

grippers, one of the grippers is removed from the graph and the process restarts from the graph construction step (Lines 7-16 in 4).

This graph construction assumes a fixed base, but by constructing the graph differently, we can adapt to other scenarios. For example, we might connect drones flying together in a swarm, even though there may not be a physical connection between them. Two drones grasping the DOO simultaneously would form a loop (Fig 5.4 E), allowing us to plan over the scene's topology.

Algorithm 5: addEdges

```
   Input  : $V, s$
   Output: $\mathcal{G}$
1  $E = \emptyset$
2  for $v_i \in V$ do
3      for $v_j \in V$ do
           // Skip invalid edges
4          if $v_i = v_j$ then
5              continue
6          else if $\neg Adj(v_i, v_j, s)$ then
7              continue
8          $E = E \cup (v_i, v_j)$                    // Add the edge
9  return $E$
```

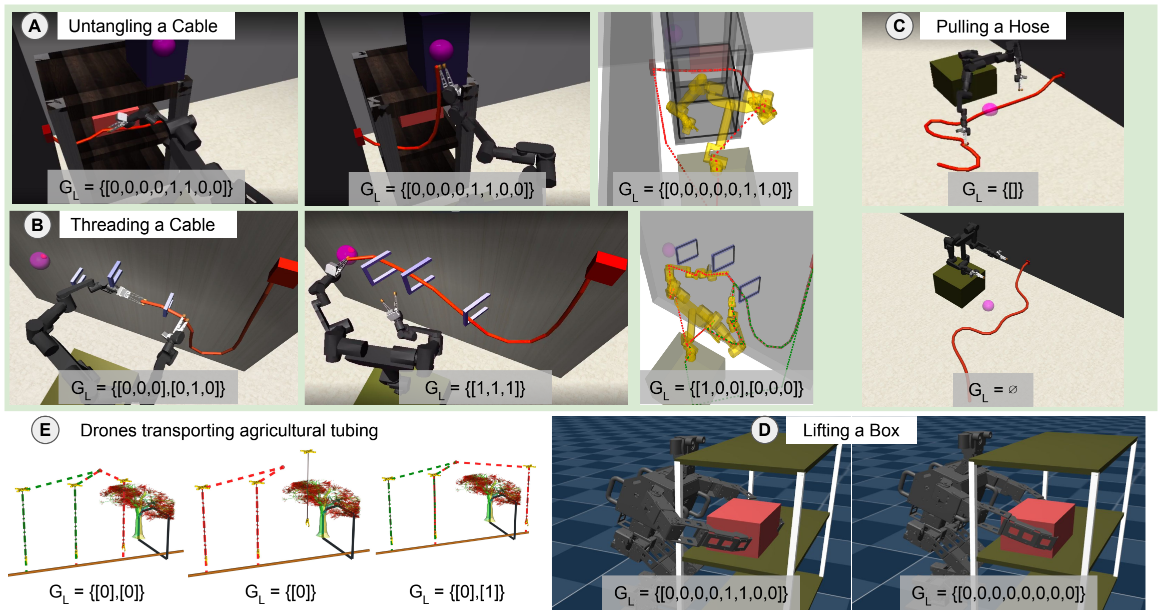

Figure 5.4: Example scenes and their $\mathcal{G}_L$ values. The Panel in green shows environments where we use the $\mathcal{G}_L$-signature in planning. Panels D and E are additional examples where the $\mathcal{G}_L$-signature may be useful.

5.2.3 Computational Complexity

The complexity of computing the $\mathcal{G}_L$-signature can be written in Big-O notation based on the number of skeletons n_s, number of line segments in the skeleton l_s, arms and/or attach points n_a, and the length of the arms and/or DOO l_a. In the base case of $n_a = 2$, the graph has 3 vertices and at most cycle of length 3. And adding another vertex adds at most one cycle, so in the worst case n_a arms/attach points create $n_a - 1$ loops. Each cycle (loop) is compared with each skeleton, and the number of comparisons scales linearly with both the number of line segments and the length of the loop, giving a total complexity of $O\big((n_a - 1)n_s l_s l_a\big)$. Our Python implementation using the NetworkX library [41] for computing the $\mathcal{G}_L$-signature for a state takes $\leq$10ms in all environments.

5.3 Illustrative Examples

Figure 5.4 shows a variety of robotic systems for which we can compute the $\mathcal{G}_L$-signature. The upper group show environments where we apply our planning methods. The lower group are examples where we believe the $\mathcal{G}_L$-signature would be useful, but do not conduct evaluations. In panel A, the second and third images show how the signature is invariant to smooth deformations, such as a change in object shape or sliding the arm along the DOO. The drones example in section C shows how we can create virtual connections between objects which are not physically connected. This allows the planner to distinguish between E1 and E3, in which the third drone is lifting the hose from different sides of the tree branch.

5.4 DOO Manipulation with $\mathcal{G}_L$-signature

5.4.1 Problem Statement

In this section, we define the DOO manipulation problem which our proposed planning method addresses. The state $s = (q, o)$ contains the robot configuration and the DOO configuration. In our experiments, the robot has two 7-dof arms attached to a 2-dof torso with parallel-jaw grippers, but the $\mathcal{G}_L$-signature can be applied to other robot morphologies. We assume we have a complete geometric model and skeleton of the environment. When manipulating with the current grasp, the action space is joint velocities $\dot{q}$. We describe points on the DOO primarily by their location $l \in [0, 1]$, where $l = 0$ is one end of the DOO and $l = 1$ is the other. Each location also corresponds to a point $p(l) \in \mathrm{R}^3$. Grasps are represented by a vector of locations $\boldsymbol{l} = [l_1, l_2]$, one for each gripper. A set of grasp locations $\boldsymbol{l}$ must also be paired with a collision-free motion of the robot to the new grasp locations, which may be reachable by many distinct joint configurations.

The goal of the manipulation is to bring a *keypoint* l_k on the DOO to a goal region with position p_{goal} and radius d_{goal}. This is a useful skill for plugging in cables, or for using tools with an attached cable or hose, and more complex tasks like cable harnessing can be described as a sequence of these point reaching goals. Additionally, one can specify a desired $\mathcal{G}_L$-signature for the goal $\mathcal{G}_{L\text{goal}}$. This type of DOO manipulation is complementary to tying or untying knots, which has been addressed in prior work [106, 119, 132].

5.4.2 DOO Point Reaching Method

Algorithm 6 describes our method for point reaching tasks. However, the cost functions can be changed and additional checks can be included to adapt the method to other tasks. Given the current grasp, we use MPPI [136] to find an action $\dot{q}$ that minimizes the goal cost C_{goal}, shown in Eq (5.2). MPPI runs until the goal is reached or progress stops. If progress stops, we plan and execute a grasp change, and resume running MPPI. This process is repeated until the goal is reached (trial success) or for $i^{\max}$ iterations (trial failure). For both MPPI and grasp planning, we model the dynamics of the robot, rope, and obstacles in MuJoCo [122].

Algorithm 6: DOO Point Reaching with the $\mathcal{G}_L$-signature

```
1  for $i < i^{max}$ do
2      $\dot{q}$ =MPPI($s, p_{\text{goal}}, C_{\text{goal}}$)
3      $s = f(\dot{q})$                          // Execute and get state
4      if $||p_{goal} - p(l_k)|| < d_{goal}$ then
5          break
6      if *trapped* then
7          $d_0 = \min(|\boldsymbol{l}_0 - l_k|)$              // Initial geodesic
8          $\boldsymbol{l}^* = PlanGrasp(s, n_x, C_{\text{grasp}}, l_k)$
9          $d^* = \min(|\boldsymbol{l}^* - l_k|)$
10         if $d^* \geq d_0$                    // Unable to grasp closer
11          then
12             Add $\mathcal{G}_L(s)$ to $\mathcal{B}$
13             $\boldsymbol{l}^* = PlanGrasp(s, n_x, l_k)$
14         ExecuteGraspChange($\boldsymbol{l}^*$)
```

The method for determining if MPPI is trapped, called trap detection, is adapted from [150]. Trap detection operates on a window of recent joint configurations $q_1, \ldots, q_T$, and computes the average one-step state difference $\bar{q} = \frac{q_T - q_1}{T}$ and keeps a running maximum of this value $\bar{q}^+$ over the trial. MPPI is considered trapped when the ratio $\frac{\bar{q}}{\bar{q}^+}$ is below a threshold (0.2-0.3 in our experiments).

The goal cost used for MPPI is shown in Equation (5.2), where the state s is used to compute the grasp locations $\boldsymbol{l}$, grasping state $\mathbb{1}_{\boldsymbol{g}}$, keypoint position $p(l_k)$, grasp positions $p(\boldsymbol{l})$, and number of contacts n_{con}.

$$\begin{aligned} C_{\text{goal}}(s, \dot{q}) = ||p(l_k) - p_{\text{goal}}|| + \alpha_1 \mathbb{1}_{\boldsymbol{g}} \cdot ||p(\boldsymbol{l}) - p_{\text{goal}}|| + \\ \alpha_2 \sqrt{n_{\text{con}}} + \alpha_3 ||\dot{q}|| \end{aligned} \tag{5.2}$$

The first term in Eq. 5.2 brings the keypoint $p(l_k)$ towards the goal p_{goal}. The second term provides a reward for moving the *gripper* towards the goal. $\mathbb{1}_{\boldsymbol{g}}$ is a binary vector indicating which grippers are grasping, and $\boldsymbol{l}$ are the current grasp locations. The dot product enforces that only grasping grippers contribute to this cost term. This term is useful when the DOO is slack and the keypoint cannot be pulled directly (See Figure 5.4 C). The third term penalizes collision between the robot and environment, based on the number of contacts n_{con} reported by the dynamics. Finally, the fourth term penalizes high joint velocities to encourage smooth motion.

The hyperparameters $\alpha_{1,2,3}$ were selected to prioritize collision avoidance first, then bringing the keypoint to the goal.

In *PlanGrasp* we sample n_x grasps ($\approx$50) and choose the best one. Grasps are sampled first by choosing a strategy for each gripper. The possible strategies are `STAY`, `GRASP`, `MOVE`, or `RELEASE`. For the `GRASP` or `MOVE` strategies, we sample a location $l \in [0, 1]$. At least one gripper must be grasping. For each candidate grasp, we simulate release and grasp dynamics using MuJoCo. Modeling grasping using friction and caging is challenging, so we instead use equality constraints between the rope and the grippers that are activated or deactivated. MoveIt [25] is used to find collision-free paths to move the grippers to the desired grasp locations. The result is a candidate state s and collision free trajectory for each candidate grasp. We choose the grasp with the lowest cost according to Eq. 5.3. With abuse of notation, we say the candidate state s, change in s, and grasp state $\mathbb{1}_{\boldsymbol{g}}$ are derived from the candidate grasp locations $\boldsymbol{l}$.

$$\begin{aligned} C_{\text{grasp}}(\boldsymbol{l}) = \mathbb{1}_{\text{feasible}} + \mathbb{1}_{\mathcal{B}}(s) + \mathbb{1}_{\mathcal{G}_L}(s, \mathcal{G}_{L\text{goal}}) + \\ \mathbb{1}_{\boldsymbol{g}} \cdot |\boldsymbol{l} - l_k| + \beta_1 \Delta s \end{aligned} \tag{5.3}$$

The first term in Eq 5.3 assigns a large penalty (e.g. 100) if no collision-free path to the grasp was found. The next two terms assign a large penalty based on the $\mathcal{G}_L$-signature of candidate state, either for matching a blocklisted signature or for not matching the goal signature. If the task has no goal signature, this term is omitted. The fourth term encourages grasping near the keypoint, based on the geodesic distance for any grasping grippers. The final term penalizes the change in robot and DOO state. This results in shorter and faster grasps and is weighted by β_1 to be the least important term. The large penalties dominate the keypoint and state-change terms.

We use a blocklist of $\mathcal{G}_L$-signature's to avoid retrying topological configurations in which we have failed to reach the goal. Blocklisting $\mathcal{G}_L$-signatures allows us to search for grasps that are different from previous states, which was an effective strategy in our experiments. However, we do not want to blocklist if the goal is reachable with a different grasp with the same signature. Therefore, we only blocklist if the planner cannot find any grasp with lower geodesic cost (4th term in Eq (5.3)) than the current grasp (Alg 6 lines 8-12). This heuristic avoids blocklisting the current $\mathcal{G}_L$-signature in the case that the current grasp cannot control the keypoint toward the goal. This is inspired by the idea of diminishing rigidity [9], which says that the control over a point on a deformable object decreases as the geodesic distance to the gripper increases. In the case of multiple grippers, the initial grasp locations $\boldsymbol{l}_0$ or new grasp locations $\boldsymbol{l}^*$ may be a list of locations, in which case we use the min when computing the geodesic distance (Alg 6 lines 8, 14).

5.5 Applications

We now describe how the above framework can be applied or adapted to DOO manipulation in three different environments, Pulling, Untangling, and Threading.

5.5.1 Pulling Environment

The Pulling environment contains a large hose attached to a wall. The scene is depicted in Figure 5.4 C. The robot is initially not grasping the hose, and the head of the hose is out of the robot's reach. The goal region, shown as a purple sphere, is near the base of the robot on the floor. This environment requires regrasping to bring the keypoint to the goal, and demonstrates the behavior of the general method in the case where there are no skeletons, and no changes in the $\mathcal{G}_L$-signature.

When applying Alg 6 in this environment, the robot initially chooses a grasp as far down the DOO as it can reach, due to the geodesic cost term in Equation 5.3. Then, the gripper pulls towards the goal due to the second term in the MPPI cost (5.2). This brings more of the DOO within reach. When the gripper reaches the goal, the cost cannot be decreased and the controller slows to a stop. At this point, trap detection triggers regrasp planning. Since the DOO is now closer, a plan is found that reaches closer to the tip ($l_k = 1$) than before. Because the grasp is closer to the tip, the current $\mathcal{G}_L$-signature is not blocklisted. This repeats until the grasp is close enough to the tip that it can be brought to the goal region. In the Pulling environment, our method succeeded in 25/25 trials, where each trial differs in the initial DOO configuration and the random seed used for sampling in planning.

5.5.2 Untangling Environment

The Untangle environment resembles a computer rack with a cable that needs to be plugged in. The scene is depicted in Figure 5.4 A. One end of the DOO is fixed to the environment (e.g. plugged in elsewhere), and the robot is initially grasping some other location on the DOO. The robot often needs to regrasp several times in order to reach the goal. Unlike in the Pulling environment, the $\mathcal{G}_L$-signature can take

Method	Success	Wall Time (m)	Sim Time (m)
$\mathcal{G}_L$-signature (ours)	22/25	12 (5)	1.4 (1.1)
Always Blocklist	22/25	14 (7)	1.3 (1.0)
No $\mathcal{G}_L$-signature	10/25	20 (10)	2.0 (1.3)
TAMP50	15/25	142 (116)	2.4 (1.7)
TAMP5	9/25	34 (22)	1.8 (1.2)

Table 5.1: Results in the Untangle environment. Times in minutes are for the completion of the task, where Sim Time does not include planning time. Standard deviations are given in parentheses.

on many different values depending on the configuration of the DOO and the grasp configuration. This demonstrates the utility of the $\mathcal{G}_L$-signature in planning when there is no goal $\mathcal{G}_L$-signature.

We evaluate Alg 6 on this task, and compare to an ablation that omits the two terms using the $\mathcal{G}_L$-signature from Eq 5.3. We call this method *No $\mathcal{G}_L$-signature*. This often results in greedy re-grasping of the keypoint. We also evaluate a version called *Always Blocklist*, where we blocklist the current $\mathcal{G}_L$-signature every time a trap is detected. Finally, we compare our proposed method to a method inspired by task and motion planning (TAMP), where H additional steps of MPPI are simulated for each candidate grasp during planning and the final goal cost is used in place of cost terms relying on the $\mathcal{G}_L$-signature. We test two versions of this method with $H = 5$ and $H = 50$. Success rates and trial times are shown in Table 5.1. Trials vary in the initial configuration of the robot, grasp location, DOO configuration, in the size of the computer rack, and in the location of the goal.

Methods using the $\mathcal{G}_L$-signature have the highest success rates and are faster than alternatives. *Always Blocklist* has an equivalent success rate as the full proposed method, but prematurely abandons grasps that would lead to reaching the goal. Our method and the *Always Blocklist* method each failed in 3 trials by trying too many unsuccessful grasps before $i^{\max}$ was reached. The *No $\mathcal{G}_L$-signature* ablation and both TAMP methods usually fail by greedily trying to grasp the keypoint. Without a very long horizon or the $\mathcal{G}_L$-signature, the planner often grasps with configurations that make reaching the goal impossible. The longer horizon used in $H = 50$ helps alleviate this issue but is insufficient in many cases while also causing a 10x increase in planning time.

Algorithm 7: DOO Threading with the $\mathcal{G}_L$-signature

```
j = 1                                              // Threading subgoal index
for $i < i^{max}$ do
    if $j < N$                                     // threading subgoals
     then
        $\dot{q}$ =MPPI($s, \mathcal{G}_{Lj}, C_{\text{goal}}$)
        $s = f(\dot{q})$                           // Execute and get state
        if *disc penetrated* then
            $\boldsymbol{l}^* = PlanGrasp(s, n_x, C_{\text{grasp}}, 1)$
            if $\mathcal{G}_L(\boldsymbol{l}^*) == \mathcal{G}_{Lj}$ then
                ExecuteGraspChange($\boldsymbol{l}^*$)
        if *trapped* then
            $\boldsymbol{l}^* = PlanGrasp(s, n_x, C_{\text{grasp}}, l - 0.05)$
            ExecuteGraspChange($\boldsymbol{l}^*$)
        if $\mathcal{G}_L(s) == \mathcal{G}_{Lj}$ then
            $j = j + 1$                            // next subgoal
    else
        $\dot{q}$ =MPPI($s, p_{\text{goal}}, C_{\text{goal}}$)
        $s = f(\dot{q})$                           // Execute and get state
        if $p_{goal} - p(l_k) < d_{goal}$ then
            break
```

5.5.3 Threading Environment

In the Threading environment, the objective is for the robot to thread the DOO through a series of fixtures in a specified order (e.g. "fixture 1, then fixture 2, then fixture 3"), after which it should bring the keypoint to a goal region. The threading is described by a series of goal signatures $\mathcal{G}_{L1}, \ldots, \mathcal{G}_{LN}$. This skill could be applied to installing cable harnesses in a car or electrical wiring in a building. One end of the DOO is fixed to the environment, and the robot is initially grasping some other location on the DOO. This environment is depicted in Figure 5.4 B.

We extend Alg 6 for this task in several ways. First, we run it iteratively, looping over each of the three threading subgoals, then finally for the point reaching subgoal. Second, when using MPPI to reach a threading subgoal, we augment Eq (5.2) with the magnetic-field cost proposed in [133]. This uses the formula $\sum_{j=1}^{n_i} \Phi(\mathbf{s}_i^j, \mathbf{s}_i^{j'}, r)$ from in Equation (5.1) for the direction of the magnetic field, but where r is the keypoint of the DOO. This causes the keypoint to follow virtual magnetic field lines through the fixture in the specified direction. Third, if a threading subgoal is reached, and the planner returns a grasp which does not match $\mathcal{G}_{L\text{goal}}$, we reject it and continue running MPPI to push the cable further through the fixture. This happens when there is no

Method	Success	Wall Time (m)	Sim Time (m)
$\mathcal{G}_L$-signature	42/50	8 (2)	1.3 (0.4)
TAMP5	21/50	17 (3)	1.3 (0.6)
Wang et al. [133]	12/50	8 (3)	1.0 (0.8)

Table 5.2: Results on the Threading task.

feasible grasp matching $\mathcal{G}_{L\text{goal}}$ due to obstacles or reachability issues. Furthermore, we also check $\mathcal{G}_{L\text{goal}}$ after executing the grasp to ensure that any deviations that occurred when executing the grasp plan do not change the $\mathcal{G}_L$-signature. To check when a threading subgoal is reached, we use the disc penetration check from [133]. The goal signatures $\mathcal{G}_{L1}, \ldots, \mathcal{G}_{LN}$ are used in the grasp planning (3rd term in Eq (5.3)), but the blocklist is not. Grasp sampling is restricted to alternating single-gripper grasps, which speeds up grasp planning. The keypoint location for grasp planning is also restricted. It is chosen to be the tip ($l_k = 1$) when a threading subgoal is reached, and further down the DOO than the current grasp when stuck ($l_k = l - 0.05$). The full Threading algorithm is shown in Alg 7, with the key differences highlighted in green. We compare our proposed method to the TAMP5 method described previously. In this environment, the TAMP method often chooses grasps that correctly thread through fixtures 1 and 2, because those grasps allow immediate progress towards the next subgoal. However, it often grasps incorrectly on fixture 3, which requires the robot to first reach further around and results in less immediate progress towards the next subgoal. We also adapted the method in [133] from a single floating gripper to our dual arm robot. As in our method, we use alternating single-gripper grasps. Instead of the more general trap detection method we use, this baseline checks the distance between the gripper and the fixture being threaded. This baseline fails similarly to the TAMP5 method, but additionally fails when MPPI is trapped but is outside the distance-to-fixture threshold. Success rates and trial times are shown in Table 5.2. Trials vary in the initial configuration of the robot, grasp location, DOO configuration, and in the positions of the fixtures. In the trials in which our method failed to complete the task, MPPI reached a joint configuration with one arm that prevented the other arm from grasping the DOO at or near the tip, as required by our method. This means the robot remained stuck until $i^{\max}$ was reached.

5.5.4 Real World Threading

We demonstrate a simplified version of the Threading task in the real world, as depicted in Figure 5.1. This shows the applicability of the proposed methods in the presence of significant calibration, perception, and dynamics modeling errors. We use CDCPD2 [135] to track the DOO and visual servoing from in-hand cameras to guide grasping. The environment geometry is specified manually, and the simulation dynamics were tuned to match the real world setup as closely as possible for the particular setup.

5.6 Conclusion

In this chapter, we proposed the $\mathcal{G}_L$-signature which describes the topology of closed loops formed by grasping the DOO with respect to closed loops formed by stationary objects in the environment. Our $\mathcal{G}_L$-signature builds on the h-signature proposed in prior work on topological path planning. Furthermore, we describe an algorithm for manipulating DOOs that plans grasps based on the proposed $\mathcal{G}_L$-signature. In our experiments, we find that using the $\mathcal{G}_L$-signature improves task success and reduces planning times compared to a task and motion-planning method. Finally, we use the method to thread a cable and bring it to a goal region on a real robot.

In this chapter, we used the MuJoCo simulator as our dynamics model in MPC, which required tuning the simulation to match the real world dynamics for our specific task. Future work could address this by combining the methods from Chapters II-IV with the methods in this chapter. One way to do this would be to learn the dynamics online using FOCUS, instead of using a simulator. To do this, one would need to extend the state-action representation or use multiple networks in order to handle multiple different grasps. Alternatively, we could keep the simulator dynamics but learn an MDE online using data augmentation, and integrate that by penalizing predicted deviations in grasp planning, MPC, or both. In summary, by combining the regrasping methods in this chapter with the learning and planning methods of previous chapters, the robot could plan to grasp, regrasp, and manipulate DOOs without relying entirely on accurate simulation.

CHAPTER VI

Conclusion and Outlook

In this thesis, I presented methods for learning and planning with dynamics models of deformable one-dimensional objects (DOOs). By developing data augmentation and focused adaptation methods, we achieved the data efficiency needed to learn models on real robots while they work to complete useful manipulation tasks, in fairly narrow settings. The models are not accurate everywhere, but our learning and planning methods are aware of this and account for it. Ultimately, these methods enabled the robot to perform parts of tasks like installing a hose in the hood of a car, plugging in USB or extension cables, or using a vacuum. This work advances the state of the art in robotic manipulation of DOOs, but there are several limitations and exciting directions for future work.

First, I have assumed that the state of the rope is accurately tracked at all times during the manipulation, and our experiments were set up carefully to allow for this. However, heavy occlusion, brittle calibration methods, fast motion of the rope, and tangling of the rope all make accurate tracking difficult. By requiring accurate perception, we are severely limiting the tasks the robot can do. Therefore, we need to relax our assumption of accurate perception. This could be accomplished by adding per-point uncertainty estimates to the rope state representation, or by using partial-shape or higher-level topological state information when the full shape is unknown. For example, for the task of plugging in the cable, we care primarily about where the tip of the cable is, and should not necessarily need to accurately track the entire cable. Additionally, planning and control methods may need to account for this uncertainty (for which there are many existing methods [101, 2, 10, 28]). Alternatively, one could use state representations learned directly from images or video, instead of those designed by hand [33, 40].

Related to this is our dependence on scene cameras. In all our experiments, multiple scene cameras were placed around the (stationary) robot to minimize occlusion.

This also requires calibration of these cameras with respect to the robot. This process seems ill-suited for robots in the wild, especially robots doing mobile manipulation. Even in semi-controlled environments such as warehouses, relying on scene cameras will limit the flexibility and robustness of the system. Instead, the robot should be equipped with multiple on-board cameras, and should actively move those cameras to see the objects it is manipulating or obstacles it is avoiding.

Beyond cameras and perception, we are also presently limited by the hardware of our robot arms. Notably, the robot Val used in almost all the experiments in this thesis has significant backlash, which cannot be sensed by the joint encoders. It also lacks dexterous hands, tactile sensing, or compliant controllers. In my opinion, the ideal robot for DOO manipulation would have compliant controllers, dexterous and sensorized grippers, two arms with 2-3 additional torso joints, and a mobile base.

Finally, we are limited by learning methods that are specialized to certain data types, or observation/state/action spaces. For instance, the method I proposed uses joint configurations, points representing the DOO, and voxelized environment geometry, but even within my own methods there are differences in data types. There are also methods that work well on large datasets of RGB images and end-effector pose actions [19]. While we have methods that work well in some cases, no method works well in all cases. To address this limitation, we should avoid hand-designing task-specific state and action spaces and focus on ones that can be used widely. This could make data sharing easier, reduce the effort required to shift between different data distributions, and promote the development of methods that are less specialized.

By addressing these limitations, robots will hopefully be better at tasks like installing cables, sewing sutures, or using tools with pneumatic or hydraulic hoses. More broadly, the goal is to significantly improve robotic manipulation such that we can use it to replace work currently done by humans and do new work only suitable for robots. However, it is equally important to ensure this technology benefits society. There are significant risks of exacerbating wealth inequality and job displacement [1, 4], and as scientists and engineers we must do our part to educate. We must educate others on what robots can do and how they work, but we must also educate ourselves on the impact these robots have on people and society. With this knowledge and mindset, I believe we can build a future where robotic manipulation is both capably and carefully deployed.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Acemoglu, D., and P. Restrepo (2022), Tasks, automation, and the rise in us wage inequality, *Econometrica.*

[2] Agha-mohammadi, A.-a., S. Chakravorty, and N. M. Amato (2014), FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements, *IJRR.*

[3] Arndt, K., A. Ghadirzadeh, M. Hazara, and V. Kyrki (2021), Few-shot model-based adaptation in noisy conditions, *RA-L.*

[4] Battista, A. D., S. Grayling, E. Hasselaar, T. Leopold, R. Li, M. Rayner, and S. Zahidi (2023), Future of Jobs Report, *World Economic Forum.*

[5] Bechtle, S., Y. Lin, A. Rai, L. Righetti, and F. Meier (2019), Curious iLQR: Resolving Uncertainty in Model-based RL, in *CoRL.*

[6] Bekey, G. A., and K. Y. Goldberg (1993), *Neural Networks In Robotics*, Science.

[7] Bengio, Y., J. Louradour, R. Collobert, and J. Weston (2009), Curriculum learning, *ICML.*

[8] Benton, G. W., M. Finzi, P. Izmailov, and A. G. Wilson (2020), Learning Invariances in Neural Networks from Training Data, *NeurIPS.*

[9] Berenson, D. (2013), Manipulation of deformable objects without modeling and simulating deformation, in *IROS.*

[10] Berg, J. v. d., P. Abbeel, and K. Y. Goldberg (2011), LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information, *IJRR.*

[11] Bergou, M., M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun (2008), Discrete elastic rods, *ACM Trans. Graph.*

[12] Berkenkamp, F., M. Turchetta, A. P. Schoellig, and A. Krause (2017), Safe Model-based Reinforcement Learning with Stability Guarantees, in *NeurIPS.*

[13] Bhattacharya, S., M. Likhachev, and V. Kumar (2011), Identification and representation of homotopy classes of trajectories for search-based path planning in 3d, in *RSS.*

[14] Bhattacharya, S., M. Likhachev, and V. R. Kumar (2012), Topological constraints in search-based robot path planning, *Autonomous Robots*.

[15] Bogdoll, D., M. Nitsche, and J. M. Zöllner (2022), Anomaly detection in autonomous driving: A survey, *CVPR Workshop*.

[16] Bousmalis, K., et al. (2018), Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping, in *IROS*.

[17] Brahmbhatt, S., A. Handa, J. Hays, and D. Fox (2019), ContactGrasp: Functional Multi-finger Grasp Synthesis from Contact, *IROS*.

[18] Brodley, C. E., and M. A. Friedl (1999), Identifying mislabeled training data, *Journal of Articial Intelligence Research*.

[19] Brohan, A., et al. (2023), RT-2: vision-language-action models transfer web knowledge to robotic control, *ArXiv Preprint*.

[20] Bócsi, B., L. Csató, and J. Peters (2013), Alignment-based transfer learning for robot models, in *International Joint Conference on Neural Networks (IJCNN)*.

[21] Chang, P., and T. Padir (2020), Sim2real2sim: Bridging the gap between simulation and real-world in flexible object manipulation, in *International Conference on Robotic Computing, IRC*.

[22] Chebotar, Y., A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox (2019), Closing the sim-to-real loop: Adapting simulation randomization with real world experience, in *ICRA*.

[23] Chua, K., R. Calandra, R. McAllister, and S. Levine (2018), Deep reinforcement learning in a handful of trials using probabilistic dynamics models, *NeurIPS*.

[24] Clavera, I., J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel (2018), Model-Based Reinforcement Learning via Meta-Policy Optimization, in *CoRL*.

[25] Coleman, D., I. A. Șucan, S. Chitta, and N. Correll (2014), Reducing the barrier to entry of complex robotic software: a moveit! case study, *Journal of Software Engineering for Robotics*.

[26] Courchesne, A., A. Censi, and L. Paull (2021), On assessing the usefulness of proxy domains for developing and evaluating embodied agents, in *IROS*.

[27] Cubuk, E. D., B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le (2018), Autoaugment: Learning augmentation policies from data, *ArXiv Preprint*.

[28] Deisenroth, M. P., D. Fox, and C. E. Rasmussen (2015), Gaussian Processes for Data-Efficient Learning in Robotics and Control, *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[29] Demir, S., K. Mincev, K. Kok, and N. G. Paterakis (2021), Data augmentation for time series regression: Applying transformations, autoencoders and adversarial networks to electricity price forecasting, *Applied Energy*.

[30] Eppner, C., A. Mousavian, and D. Fox (2021), Acronym: A large-scale grasp dataset based on simulation, *ICRA*.

[31] Evans, B., A. Thankaraj, and L. Pinto (2022), Context is everything: Implicit identification for dynamics adaptation, in *ICRA*.

[32] Feng, S. Y., V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy (2021), A Survey of Data Augmentation Approaches for NLP, *Association for Computational Linguistics*.

[33] Finn, C., and S. Levine (2017), Deep visual foresight for planning robot motion, *ICRA*.

[34] Fisac, J. F., A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. H. Gillula, and C. J. Tomlin (2019), A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems, *IEEE Transactions on Automatic Control*.

[35] Fu, J., S. Levine, and P. Abbeel (2016), One-shot learning of manipulation skills with online dynamics adaptation and neural network priors, in *IROS*.

[36] Gal, Y., and Z. Ghahramani (2016), Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, in *ICML*.

[37] Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014), Generative Adversarial Nets, *NeurIPS*.

[38] Grannen, J., et al. (2020), Untangling dense knots by learning task-relevant keypoints, in *CoRL*.

[39] Guzzi, J., R. O. Chavez-Garcia, M. Nava, L. M. Gambardella, and A. Giusti (2020), Path Planning With Local Motion Estimations, *RA-L*.

[40] Hafner, D., T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson (2019), Learning Latent Dynamics for Planning from Pixels, in *ICML*.

[41] Hagberg, A. A., D. A. Schult, and P. J. Swart (2008), Exploring network structure, dynamics, and function using networkx, *Python in Science Conference*.

[42] Holl, P., V. Koltun, and N. Thuerey (2020), Learning to control pdes with differentiable physics, *ArXiv Preprint*.

[43] Hoque, R., D. Seita, A. Balakrishna, A. Ganapathi, A. K. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg (2020), Visuospatial foresight for multi-step, multi-task fabric manipulation, in *RSS*.

[44] Hora, S. C. (1996), Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management, *Reliability Engineering & System Safety*.

[45] Huang, H., D. Wang, R. Walters, and R. Platt (2022), Equivariant transporter network, *RSS*.

[46] Hüllermeier, E., and W. Waegeman (2021), Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods, *Machine Learning*.

[47] Ichter, B., and M. Pavone (2019), Robot Motion Planning in Learned Latent Spaces, *RA-L*.

[48] Igarashi, T., and M. Stilman (2010), Homotopic path planning on manifolds for cabled mobile robots, *WAFR*.

[49] Iwana, B. K., and S. Uchida (2020), An Empirical Survey of Data Augmentation for Time Series Classification with Neural Networks, *PLOS One*.

[50] Jaillet, L., and T. Simeon (2008), Path deformation roadmaps: Compact graphs with useful cycles for motion planning, *IJRR*.

[51] Jia, B., Z. Hu, J. Pan, and D. Manocha (2018), Manipulating Highly Deformable Materials Using a Visual Feedback Dictionary, in *ICRA*.

[52] Jiang, G., W. Wang, Y. Qian, and J. Liang (2021), A unified sample selection framework for output noise filtering: An error-bound perspective, *JMLR*.

[53] Jonschkowski, R., and O. Brock (2015), Learning state representations with robotic priors, *Autonomous Robots*.

[54] Kaelbling, L. P., and T. Lozano-Pérez (2013), Integrated task and motion planning in belief space, *IJRR*.

[55] Kamthe, S., and M. P. Deisenroth (2018), Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control, in *AISTATS*.

[56] Kavraki, L. E., P. Svestka, J.-C. Latombe, and M. H. Overmars (1996), Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *TRO*.

[57] Knuth, C., G. Chou, N. Ozay, and D. Berenson (2021), Planning with learned dynamics: Probabilistic guarantees on safety and reachability via lipschitz constants, *RA-L*.

[58] Kobilarov, M. (2011), Cross-Entropy Randomized Motion Planning, in *RSS*.

[59] Koenig, N. P., and A. Howard (2004), Design and use paradigms for Gazebo, an open-source multi-robot simulator, in *IROS*.

[60] Koller, T., F. Berkenkamp, M. Turchetta, and A. Krause (2018), Learning-Based Model Predictive Control for Safe Exploration, in *CDC*.

[61] Koller, T., F. Berkenkamp, M. Turchetta, J. Boedecker, and A. Krause (2019), Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning, in *Workshop on Safe Autonomy*.

[62] Kroemer, O., S. Niekum, and G. Konidaris (2021), A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms, *JMLR*.

[63] Kumar, A., T. Ma, and P. Liang (2020), Understanding self-training for gradual domain adaptation, *ICML*.

[64] LaGrassa, A., and O. Kroemer (2022), Learning model preconditions for planning with multiple models, *CoRL*.

[65] LaGrassa, A., S. Lee, and O. Kroemer (2020), Learning skills to patch plans based on inaccurate models, in *IROS*.

[66] Lakshminarayanan, B., A. Pritzel, and C. Blundell (2017), Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles, in *NeurIPS*.

[67] Lamiraux, F., and L. E. Kavraki (2001), Planning Paths for Elastic Objects under Manipulation Constraints, *IJRR*.

[68] Langsfeld, J. D., K. N. Kaipa, and S. K. Gupta (2018), Selection of trajectory parameters for dynamic pouring tasks based on exploitation-driven updates of local metamodels, *Robotica*.

[69] Laskin, M., K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas (2020), Reinforcement Learning with Augmented Data, *NeurIPS*.

[70] LaValle, S. M., and J. J. J. Kuffner (2001), Randomized Kinodynamic Planning, *IJRR*.

[71] Lee, J., J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter (2020), Learning quadrupedal locomotion over challenging terrain, *Science Robotics*.

[72] Li, Y., J. Wu, J. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake (2019), Propagation Networks for Model-Based Control Under Partial Observation, *ICRA*.

[73] Li, Z., and A. B. Farimani (2022), Graph neural network-accelerated lagrangian fluid simulation, *Computers & Graphics*.

[74] Lim, V., H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg (2022), Planar robot casting with real2sim2real self-supervised learning, *Workshop on Representing and Manipulating Deformable Objects, ICRA*.

[75] Lin, X., Y. Wang, J. Olkin, and D. Held (2020), Softgym: Benchmarking deep reinforcement learning for deformable object manipulation, in *CoRL*.

[76] Lin, X., Y. Wang, Z. Huang, and D. Held (2021), Learning visible connectivity dynamics for cloth smoothing, in *CoRL*.

[77] Lowrey, K., S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov (2018), Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system, in *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*.

[78] Ma, E. (2019), NLP Augmentation.

[79] Mahler, J., J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg (2017), Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics, in *RSS*.

[80] Mahler, J., et al. (2016), Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards, in *ICRA*.

[81] Matas, J., S. James, and A. J. Davison (2018), Sim-to-Real Reinforcement Learning for Deformable Object Manipulation, in *CoRL*.

[82] McConachie, D., and D. Berenson (2016), Bandit-based model selection for deformable object manipulation, in *WAFR*.

[83] McConachie, D., A. Dobson, M. Ruan, and D. Berenson (2020), Manipulating deformable objects by interleaving prediction, planning, and control, *IJRR*.

[84] McConachie, D., T. Power, P. Mitrano, and D. Berenson (2020), Learning When to Trust a Dynamics Model for Planning in Reduced State Spaces, *RA-L*.

[85] Miller, A. T., and P. K. Allen (2004), Graspit! A versatile simulator for robotic grasping, *IEEE Robotics Autom. Mag.*

[86] Mitrano, P., and D. Berenson (2022), Data augmentation for manipulation, *RSS*.

[87] Mitrano, P., D. M^{c}Conachie, and D. Berenson (2021), Learning Where to Trust Unreliable Models in an Unstructured World for Deformable Object Manipulation, *Science Robotics*.

[88] Mitrano, P., A. LaGrassa, O. Kroemer, and D. Berenson (2023), Focused adaptation of dynamics models for deformable object manipulation, *ICRA*.

[89] Mousavian, A., C. Eppner, and D. Fox (2019), 6-dof graspnet: Variational grasp generation for object manipulation, in *ICCV*.

[90] Mrowca, D., C. Zhuang, E. Wang, N. Haber, F.-F. Li, J. Tenenbaum, and D. L. Yamins (2018), Flexible neural representation for physics prediction, in *NeurIPS*.

[91] Murthy, J. K., et al. (2021), gradsim: Differentiable simulation for system identification and visuomotor control, in *ICLR*.

[92] Nagabandi, A., G. Kahn, R. S. Fearing, and S. Levine (2018), Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning, in *ICRA*.

[93] Nagabandi, A., G. Yang, T. Asmar, R. Pandya, G. Kahn, S. Levine, and R. S. Fearing (2018), Learning Image-Conditioned Dynamics Models for Control of Underactuated Legged Millirobots, in *IROS*.

[94] Nagabandi, A., I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn (2019), Learning to adapt in dynamic, real-world environments through meta-reinforcement learning, in *ICLR*.

[95] Nair, A., D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine (2017), Combining self-supervised learning and imitation for vision-based rope manipulation, in *ICRA*.

[96] Navarro-Alarcón, D., Y.-H. Liu, J. G. Romero, and P. Li (2013), Model-Free Visually Servoed Deformation Control of Elastic Objects by Robot Manipulators, *TRO*.

[97] Ohno, H. (2020), Auto-encoder-based generative models for data augmentation on regression problems, *Soft Computing*.

[98] OpenAI, et al. (2019), Solving Rubik's Cube with a Robot Hand, *ArXiv Preprint*.

[99] Peng, X. B., M. Andrychowicz, W. Zaremba, and P. Abbeel (2018), Sim-to-Real Transfer of Robotic Control with Dynamics Randomization, in *ICRA*.

[100] Peng, X. B., M. Andrychowicz, W. Zaremba, and P. Abbeel (2018), Sim-to-real transfer of robotic control with dynamics randomization, *ICRA*.

[101] Platt, R. J., L. P. Tedrake, Russ an Kaelbling, and T. Lozano-Pérez (2010), Belief space planning assuming maximum likelihood observations, in *RSS*.

[102] Portelas, R., C. Colas, L. Weng, K. Hofmann, and P. Oudeyer (2020), Automatic curriculum learning for deep RL: A short survey, *IJCAI*.

[103] Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2006), Maximum margin planning, in *ICML*.

[104] Rodriguez, D., and S. Behnke (2018), Transferring category-based functional grasping skills by latent space non-rigid registration, *RA-L*.

[105] Ruan, M., D. McConachie, and D. Berenson (2018), Accounting for directional rigidity and constraints in control for manipulation of deformable objects without physical simulation, in *IROS*.

[106] Saha, M., and P. Isto (2007), Manipulation planning for deformable linear objects, *IEEE Trans. Robotics*.

[107] Sánchez, D., W. Wan, and K. Harada (2019), Tethered tool manipulation planning with cable maneuvering, *RA-L*.

[108] Sastry, S. S., and A. Isidori (1989), Adaptive control of linearizable systems, *IEEE Transactions on Automatic Control*.

[109] Schneider, J. G. (1996), Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning, in *NeurIPS*.

[110] Shorten, C., and T. M. Khoshgoftaar (2019), A survey on Image Data Augmentation for Deep Learning, *Journal of Big Data*.

[111] Simard, P., D. Steinkraus, and J. Platt (2003), Best practices for convolutional neural networks applied to visual document analysis, *International Conference on Document Analysis and Recognition*.

[112] Siméon, T., J. Laumond, J. Cortés, and A. Sahbani (2004), Manipulation planning with probabilistic roadmaps, *IJRR*.

[113] Smith, R. (2005), Open Dynamics Engine, *Tech. rep.*, University of Auckland.

[114] Smolentsev, L., A. Krupa, and F. Chaumette (2023), Shape visual servoing of a tether cable from parabolic features, *ICRA*.

[115] Sorocky, M. J., S. Zhou, and A. P. Schoellig (2020), Experience selection using dynamics similarity for efficient multi-source transfer learning between robots, in *ICRA*.

[116] Srinivas, A., A. Jabri, P. Abbeel, S. Levine, and C. Finn (2018), Universal Planning Networks: Learning Generalizable Representations for Visuomotor Control, in *ICML*.

[117] Sucan, I. A., M. Moll, and L. E. Kavraki (2012), The open motion planning library, *IEEE Robotics Autom. Mag.*

[118] sundaresan, P., J. Grannen, B. Thananjeyan, A. Balakrishna, M. Laskey, K. Stone, J. E. Gonzalez, and K. Goldberg (2020), Learning rope manipulation policies using dense object descriptors trained on synthetic depth data, in *ICRA*.

[119] Sundaresan, P., et al. (2021), Untangling dense non-planar knots by learning manipulation features and recovery policies, in *RSS*.

[120] Tekden, A. E., A. Erdem, E. Erdem, M. Imre, M. Y. Seker, and E. Ugur (2020), Belief Regulated Dual Propagation Nets for Learning Action Effects on Groups of Articulated Objects, *ICRA*.

[121] Terzopoulos, D., J. C. Platt, A. H. Barr, and K. W. Fleischer (1987), Elastically deformable models, in *SIGGRAPH*.

[122] Todorov, E., T. Erez, and Y. Tassa (2012), Mujoco: A physics engine for model-based control, in *IROS*.

[123] Torrey, L., and J. Shavlik (2010), Transfer learning, in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global.

[124] Tran, T., T. Pham, G. Carneiro, L. J. Palmer, and I. D. Reid (2017), A Bayesian Data Augmentation Approach for Learning Deep Models, *NeurIPS*.

[125] Vemula, A., Y. Oza, J. A. Bagnell, and M. Likhachev (2020), Planning and Execution using Inaccurate Models with Provable Guarantees, in *RSS*.

[126] Vuong, Q., A. Kumar, S. Levine, and Y. Chebotar (2022), Dasco: Dual-generator adversarial support constrained offline reinforcement learning, *NeurIPS*.

[127] Wakamatsu, H., A. Tsumaya, E. Arai, and S. Hirai (2005), Manipulation planning for knotting/unknotting and tightly tying of deformable linear objects, *ICRA*.

[128] Wakamatsu, H., A. Tsumaya, E. Arai, and S. Hirai (2006), Manipulation planning for unraveling linear objects, *ICRA*.

[129] Waltersson, G. A., R. Laezza, and Y. Karayiannidis (2022), Planning and control for cable-routing with dual-arm robot, *ICRA*.

[130] Wang, A., T. Kurutach, P. Abbeel, and A. Tamar (2019), Learning Robotic Manipulation through Visual Planning and Acting, in *RSS*.

[131] Wang, C., Y. Zhang, X. Zhang, Z. Wu, X. Zhu, S. Jin, T. Tang, and M. Tomizuka (2022), Offline-online learning of deformation model for cable manipulation with graph neural networks, *RA-L*.

[132] Wang, W., and D. Balkcom (2018), Knot grasping, folding, and re-grasping, *IJRR*.

[133] Wang, W., D. Berenson, and D. Balkcom (2015), An online method for tight-tolerance insertion tasks for string and rope, *ICRA*.

[134] Wang, Y., S. Chaudhuri, and L. E. Kavraki (2018), Bounded Policy Synthesis for POMDPs with Safe-Reachability Objectives, in *Conference on Autonomous Agents and MultiAgent Systems*.

[135] Wang, Y., D. McConachie, and D. Berenson (2021), Tracking Partially-Occluded Deformable Objects while Enforcing Geometric Constraints, *ICRA*.

[136] Williams, G., P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou (2016), Aggressive driving with model predictive path integral control, *ICRA*.

[137] Wood, J. (2021), Robot workers are being hired at record rates in US companies - here's why, *World Economic Forum*.

[138] Wu, Y., W. Yan, T. Kurutach, L. Pinto, and P. Abbeel (2019), Learning to manipulate deformable objects without demonstrations, *ArXiv Preprint*.

[139] Wu, Y., W. Yan, T. Kurutach, L. Pinto, and P. Abbeel (2020), Learning to Manipulate Deformable Objects without Demonstrations, in *RSS*.

[140] Yan, M., Y. Zhu, N. Jin, and J. Bohg (2020), Self-Supervised Learning of State Estimation for Manipulating Deformable Linear Objects, *RA-L*.

[141] Yan, W., A. Vangipuram, P. Abbeel, and L. Pinto (2020), Learning predictive representations for deformable objects using contrastive estimation, in *CoRL*.

[142] Yu, K., M. Bauzá, N. Fazeli, and A. Rodriguez (2016), More than a Million Ways to Be Pushed: A High-Fidelity Experimental Data Set of Planar Pushing, *IROS*.

[143] Yu, M., K. Lv, H. Zhong, S. Song, and X. Li (2022), Global model learning for large deformation control of elastic deformable linear objects: An efficient and adaptive approach, *TRO*.

[144] Yu, M., H. Zhong, and X. Li (2022), Shape control of deformable linear objects with offline and online learning of local linear deformation models, in *ICRA*.

[145] Zaremba, W. (2021), OpenAI Disbands Robotics, *Interview on YouTube*.

[146] Zhan, R., X. Liu, D. F. Wong, and L. S. Chao (2021), Meta-curriculum learning for domain adaptation in neural machine translation, in *AAAI*.

[147] Zhang, F., and Y. Demiris (2022), Learning garment manipulation policies toward robot-assisted dressing, *Science Robotics*.

[148] Zhang, M., S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine (2019), SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning, in *ICML*.

[149] Zhang, Y., P. David, and B. Gong (2017), Curriculum domain adaptation for semantic segmentation of urban scenes, in *ICCV*.

[150] Zhong, S., Z. Zhang, N. Fazeli, and D. Berenson (2021), TAMPC: A controller for escaping traps in novel environments, *RA-L*.

[151] Zhong, S., Z. Zhang, N. Fazeli, and D. Berenson (2021), TAMPC: A Controller for Escaping Traps in Novel Environments, *RA-L*.

[152] Zhou, K., and J. C. Doyle (1998), Essentials of Robust Control, *Prentice Hall*.

[153] Zhu, X., D. Wang, O. Biza, G. Su, R. Walters, and R. Platt (2022), Sample efficient grasp learning using equivariant models, *RSS*.

[154] Ziyan, G., A. Elibol, and N. Y. Chong (2021), Planar Pushing of Unknown Objects Using a Large-Scale Simulation Dataset and Few-Shot Learning, *International Conference on Automation Science and Engineering (CASE)*.