ARTIFICIAL INTELLIGENCE

Learning where to trust unreliable models in an unstructured world for deformable object manipulation

P. Mitrano[1]*, D. M[c]Conachie[1,2], D. Berenson[1]

The world outside our laboratories seldom conforms to the assumptions of our models. This is especially true for dynamics models used in control and motion planning for complex high–degree of freedom systems like deformable objects. We must develop better models, but we must also consider that, no matter how powerful our simulators or how big our datasets, our models will sometimes be wrong. What is more, estimating how wrong models are can be difficult, because methods that predict uncertainty distributions based on training data do not account for unseen scenarios. To deploy robots in unstructured environments, we must address two key questions: When should we trust a model and what do we do if the robot is in a state where the model is unreliable. We tackle these questions in the context of planning for manipulating rope-like objects in clutter. Here, we report an approach that learns a model in an unconstrained setting and then learns a classifier to predict where that model is valid, given a limited dataset of rope-constraint interactions. We also propose a way to recover from states where our model prediction is unreliable. Our method statistically significantly outperforms learning a dynamics function and trusting it everywhere. We further demonstrate the practicality of our method on real-world mock-ups of several domestic and automotive tasks.

INTRODUCTION

Control and motion planning methods that rely on models to predict the outcomes of potential actions are ubiquitous in robotics. However, 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 (*1*). What is worse, estimating how erroneous our models are can be difficult, because methods that predict uncertainty distributions based on training data (*2*, *3*) do not account for new scenarios, e.g., when new types of constraints are introduced. This is especially problematic if these constraints are not easily represented (*4*) in the model's state space. Consequently, the inability to generate meaningful predictive uncertainty distributions for new scenarios precludes the use of belief-space planning techniques (*5*–*7*).

Roboticists rarely address the unreliable model problem directly; instead, we often resort to high-frequency replanning, hoping to compensate for model errors online (*8*, *9*). However, this kind of approach assumes that the erroneous model is somehow likely to produce a useful action locally. Although 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 (i) "When should we trust a model?" and (ii) "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 relearn accurate dynamics after every change.

This paper 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 because it is heavily influenced by the rope's often nonuniform friction and stiffness properties. These properties are not only difficult to model, but 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 that 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 that 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 tasks with different environment geometry, starts, and goals 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. We find that attempting to learn these dynamics yields poor results.

Although the above methods allow us to perform useful tasks with rope despite being unable to predict dynamics on constraint 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 inaccuracies 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. Although random actions can eventually lead us to this region, we find that it is

[1]University of Michigan, Ann Arbor, MI, USA. [2]Toyota Research Institute, Cambridge, MA, USA.
*Corresponding author. Email: pmitrano@umich.edu

more efficient to use a 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 Fig. 1.

RESULTS

To rigorously evaluate our approach, we perform statistical comparisons of our method versus 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 both learning the full dynamics (FD) and 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.

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 with a method which plans with a model of the FD. We term this baseline FD, and we learn the dynamics using an approach similar to (*10*) (see the "Learning the unconstrained dynamics" section). In FD, we learn the dynamics from a dataset collected in the same way as we collect data for our classifier (see the "Phase 2 data collection" section). Our method uses two phases of data collection; the data in each phase are used differently. Therefore, to make a fair comparison, the FD 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), because anything we might consider a constraint is subsumed by the dynamics.

In addition, we remove various components of our method to demonstrate the benefits they provide. We compare with 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 with a version without recovery (classifier) and a method that takes random actions as its recovery policy (classifier + random recovery).

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 Fig. 2A. 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 underactuated 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. 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 Fig. 6A. Our complete method reached the goal 76% of the time with a mean task error of 4.48. This is better than the FD (32% success, 20.32 error) and no classifier (60% success, 8.20 error) and is significant at $P < 0.001$. Additional numerical results are shown in Table 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 that 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 with a manually engineered solution for this scenario. By inspecting the data of where the model tended to make errors, we found that, expectedly, 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 the 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 whether the predicted state was penetrating into the obstacle in place of our learned classifier [as is done in (*11*)]. Note that this is a scenario-specific method derived by using human intuition and an engineered collision checker. Although this intuition and hand-engineered solution can

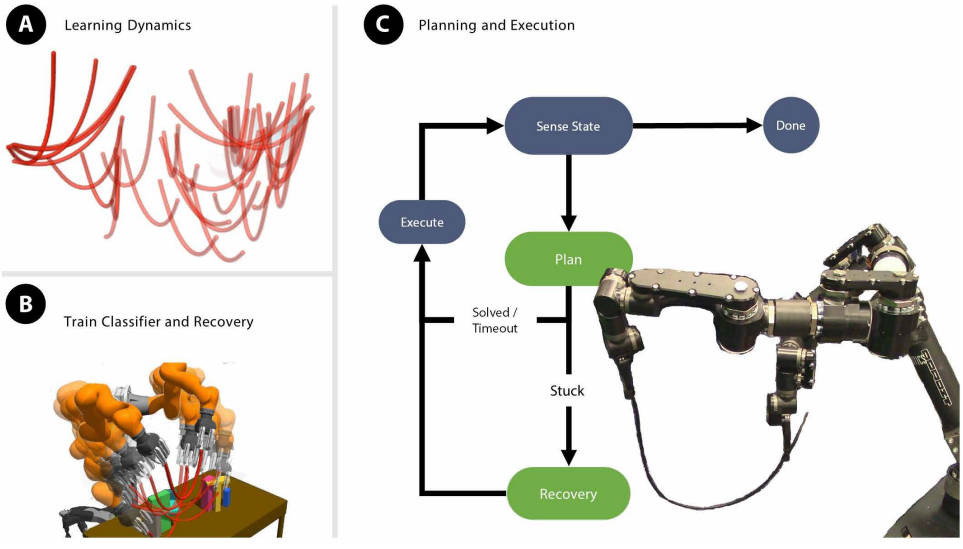

Fig. 1. Overview of the proposed method. Two data collection phases (**A** and **B**) are used to collect training data for learning dynamics, a classifier for where these dynamics are accurate, and a recovery model for what actions to take when no accurate dynamics predictions can be made. These learned components are used in an iterative process of planning, replanning, and recovery (**C**). We apply this method to a number of deformable object tasks, including a dual-arm robot manipulating an automotive hose.

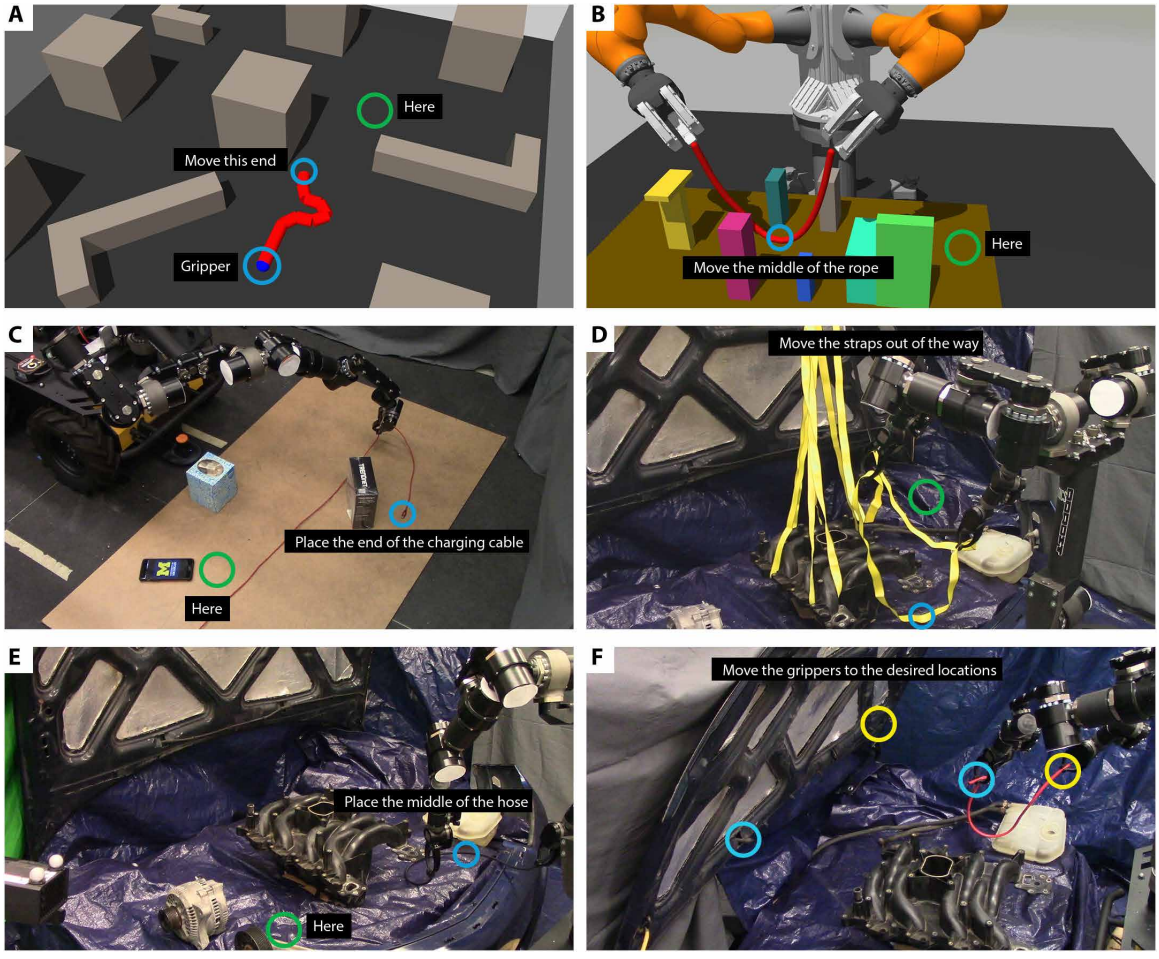

Fig. 2. Pictures of our simulation and real robot scenarios. (**A**) Simulated rope dragging. (**B**) Simulated tabletop rope manipulation. (**C**) Retrieving a charging cable. (**D**) Removing lifting straps. (**E**) Moving a hose. (**F**) Preparing to install a hose. Annotations show examples of goals for various tasks that our method can complete.

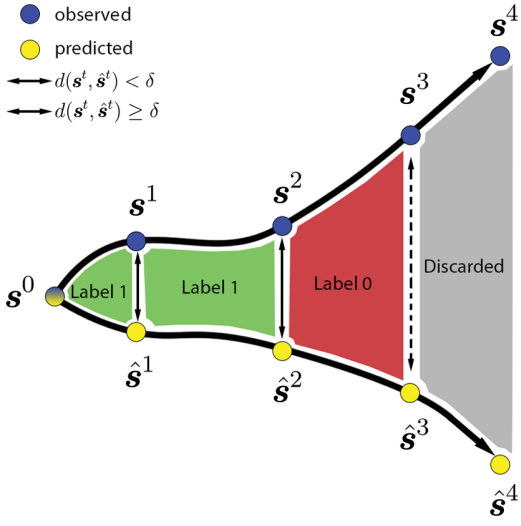

Fig. 3. Converting a trajectory into examples for training the classifier. Each colored shape represents a transition. In the first transition, the final states $\boldsymbol{s}^1$ and $\hat{\boldsymbol{s}}^1$ are close, so the transition is labeled 1 (accurate). For the third transition, the initial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

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, with $P = 0.498$ for the hypothesis that our method outperforms the collision-checking method. This is an encouraging result because it demonstrates that our method can perform on par with a scenario-specific human-engineered solution, at least in this scenario.

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 Fig. 2B. Using two arms to manipulate deformable objects allows more control of the object than one arm and introduces additional challenges in coordinating the two arms. The task we use for evaluation is to place the midpoint of the rope at a point in three dimensional space while only holding the ends. In addition to obstacles, this scenario imposes the

Table 1. Task error statistics for simulated rope dragging.

Name	Dynamics	Classifier	Min (m)	Max (m)	Mean (m)	Median (m)	SD (m)
Classifier	Unconstrained	Learned	0.009	1.133	0.128	0.044	0.247
No classifier	Unconstrained	None	0.014	1.408	0.325	0.045	0.425
Full dynamics	Full	None	0.004	1.519	0.438	0.156	0.450

Table 2. Task error statistics for simulated dual-arm rope manipulation.

Name	Dynamics	Classifier	Min (m)	Max (m)	Mean (m)	Median (m)	SD (m)
Classifier learned recovery	Unconstrained	Learned	0.014	0.450	0.073	0.045	0.094
Classifier random recovery	Unconstrained	Learned	0.016	0.629	0.081	0.046	0.106
Classifier	Unconstrained	Learned	0.015	0.630	0.147	0.047	0.167
No classifier	Unconstrained	None	0.011	0.741	0.170	0.082	0.165
Full dynamics	Full	None	0.019	0.621	0.203	0.191	0.142

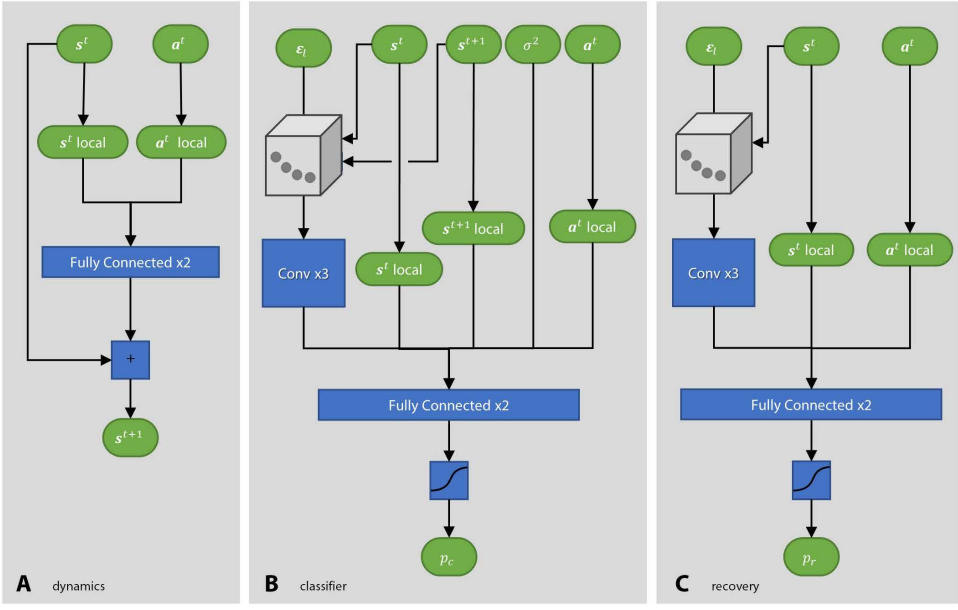

Fig. 4. Network architectures. (**A**) Dynamics, (**B**) classifier, and (**C**) recovery. Local refers to the translations used to make the states and actions invariant to the position in the world frame and is described in the "Learning the unconstrained dynamics," "Learning the classifier," and "Network architectures" sections. Green capsule shapes indicate vectors, and blue boxes indicate functions. The gray 3D box represents the 3D voxel grid representation of state (section "Network architectures," subsection "Classifier").

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. This type of manipulation, although difficult for existing methods, is a prerequisite for many practical tasks such as cable harnessing.

The task error across 100 trials is shown in Fig. 6B. Our complete method (classifier + learned recovery) reached the goal 84% of the time with a mean task error of 7.29. This is statistically significantly ($P < 0.001$) better than the FD, which reached the goal 17.7% of the time with a mean task error of 20.32, and no classifier, which reached the goal 47% of the time with a mean task error of 16.97. We also compare with our method without recovery actions and with 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. When compared with random recovery actions, which reach the goal 78% of the time, our method has a 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. In total, over all 100 trials, random recovery used 613 recovery actions, whereas our method used only 177. Additional numerical results are shown in Table 2.

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 is performed under the hood of a car and requires the robot to manipulate straps or hoses. These tasks include moving the midpoint

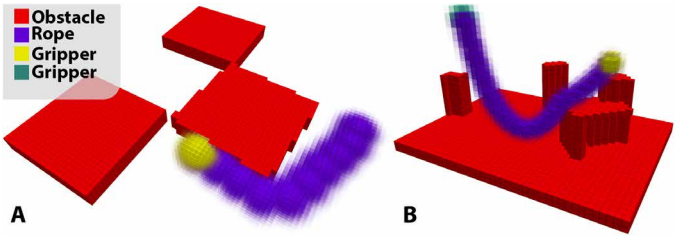

Fig. 5. Representing the state and environment in 3D voxel grids. (**A**) An example for rope dragging. (**B**) An example for dual-arm rope manipulation. Different colors are used to represent different channels in the voxel grid, and alpha is used to indicate the voxel value, with 0 being fully transparent and 1 being fully opaque.

of a hose to a specific location (Fig. 2E), positioning the ends of a wiper fluid hose for installation (Fig. 2F), and removing lifting straps from an engine (Fig. 2D).

We also perform the task of fetching a charging cable (Fig. 2C). 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 toward the phone.

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 (*12*–*14*) and methods that use goal images (*8*, *15*, *16*). More details on how we perform these tasks can be found in the "Physical robot demonstrations" section.

DISCUSSION

Our method is able to complete a variety of difficult rope manipulation tasks in clutter. The proposed learning methods are able to learn the unconstrained dynamics accurately, and by using our learned classifier, our planner successfully avoids the regions of state-action space where these unconstrained dynamics are incorrect. Using the classifier outperforms both using the FD 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.

Below, we describe how our proposed methods relate to existing work in learning dynamics, planning, and addressing model error. We then discuss several limitations of our method and prospects for improvement.

Related work

Our proposed methods build upon prior work in learning dynamics, planning with learned dynamics, and most specifically in techniques to address model error. For image-based prediction or for systems with high-dimensional states or complex dynamics, neural networks are a popular choice (*8*, *10*, *17*, *18*). As in our work, these models were often trained on datasets of random actions, using the mean squared prediction error as the training objective; our method for learning dynamics with neural networks is most similar to (*10*). These models were then used to create motion plans using a variety of methods such as A* (*1*), RRT (*18*), the cross-entropy method (CEM) (*19*), and gradient-based optimization (*20*). Although (*1*) and (*18*) both took steps to avoid regions where the learned

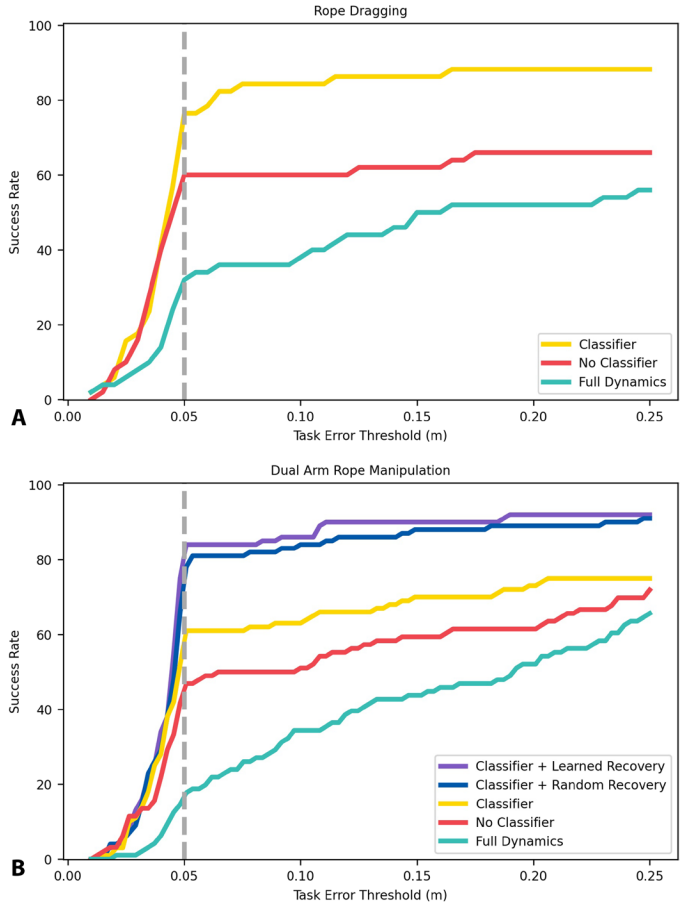

Fig. 6. Comparing success across methods. (**A**) The success rate as a function of the success threshold on task error for our simulated rope-dragging experiments. The dashed line indicates the size of the goal region used. For example, this tells us that if our goal region was 0.1, the "classifier" method would achieve about 84% success. (**B**) The success rate as a function of the success threshold on task error for our simulated dual-arm rope manipulation experiments.

model may have substantial error, neither paper focused on cases where new constraints were introduced after the model was learned. In contrast, we explicitly target this problem. A related problem was also addressed in (*21*), which made local adjustments in response to inaccurate predictions encountered in execution. This strategy is complementary to our approach of avoiding areas of predicted model inaccuracy in planning.

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 (i) compliance allows us to make mistakes, stop, and replan; (ii) dynamics are more complex in some regions than in others; and (iii) much of the state space and dynamics are irrelevant for doing useful tasks.

First, our method relies on taking actions for which we have no accurate model, which means that 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 that we can learn accurately and that is sufficiently useful. Such an assumption is particularly well suited to deformable object manipulation, where the FD 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 (i) only learning the dynamics for the subset of state-action space covered in phase 1 data collection and (ii) only learning the classifier for the relatively small subset of state-action space covered in phase 2 data collection.

Limitations

In this work, we show how to plan with unreliable models and address their inaccuracies; however, many open questions remain. For example, 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 1 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. Last, 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 that 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. Although this is a reasonable assumption, it would be interesting to incorporate our method into a framework that uses it to get as close as possible to a given goal and then switching to a feedback-based local method [e.g., (*14*, *22*, *23*)] to finalize the task [as is done in (*24*)].

In terms of recovery, we note that although our learned recovery actions markedly 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 because this is an effective strategy in the training data. However, although 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 2 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 sim-to-real methods can be used to transfer either a learned dynamics model (*25*, *26*) or a learned policy (*27*, *28*) to the real world. For instance, it would be useful to adapt online to different stiffnesses of lengths of rope without recollecting large datasets. Sim-to-real methods have been successful for a number of other robots and tasks (*29*, *30*), and incorporating these techniques into our proposed methods is an interesting direction for future work.

MATERIALS AND METHODS

In this section, we first formalize the problem being addressed. Next, we describe the main steps of our method. Each of these steps is further broken down and presented in detail. Last, we provide additional details on our simulation environments and physical robot demonstrations.

Method overview

Here, we provide a more detailed overview of our method (see Fig. 1). Our method begins with two data collection phases during which we learn the dynamics, the classifier, and recovery. In the first phase, we collect data in the absence of physical constraints (e.g., obstacles) and use these data to train a neural network dynamics model. This is our unconstrained dynamics model. In the second data collection phase, we include obstacles and other constraints. Here, we collect data to learn where our model is accurate and use this to train our classifier. Once the classifier is trained, we can again process the data collected in phase 2 to find examples of recovery actions. We then use these examples to train a method that predicts the probability that a given action will recover from a state where the model is unreliable.

Online, we perform planning with our dynamics and classifier using a kinodynamic RRT (*31*). If planning cannot proceed (i.e., we cannot find an action at the current state that is classified as yielding a reliable prediction), recovery is needed. We then use the recovery model to find actions to lead the robot back to a state where predictions are accurate and the planner can be used.

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 $f(\boldsymbol{\mathcal{E}}, \boldsymbol{s}^t, \boldsymbol{a}^t) \rightarrow \boldsymbol{s}^{t+1}$ produces the next state $\boldsymbol{s}^{t+1}$ given the environment $\boldsymbol{\mathcal{E}}$, state $\boldsymbol{s}^t$, and action $\boldsymbol{a}^t$. 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 $\boldsymbol{s}^0$ 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. Because 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 $(\boldsymbol{s}^t, \boldsymbol{a}^t)$ using a distance function in state space d and is shown in Eq. 1

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

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

$$\begin{aligned} &\text{Find} && N, \boldsymbol{a}^0, \ldots, \boldsymbol{a}^{N-1} \\ &\text{subject to} && \hat{\boldsymbol{s}}^{t+1} = h(\boldsymbol{\mathcal{E}}, \hat{\boldsymbol{s}}^t, \boldsymbol{a}^t) \quad t \in [0, N) \\ &&& d(\hat{\boldsymbol{s}}^{t+1}, \boldsymbol{s}^{t+1}) < \delta \quad t \in [0, N) \\ &&& \hat{\boldsymbol{s}}^N \in \mathcal{G} \end{aligned} \tag{2}$$

Classifier

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

$$\begin{aligned} &\texttt{INPUT} \quad \mathcal{E}, \hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1} \\ &\texttt{LABEL} \quad d(\hat{\boldsymbol{s}}^{t+1}, \boldsymbol{s}^{t+1}) < \delta \end{aligned} \tag{3}$$

Let the classifier that solves this problem be $g(\mathcal{E}, \hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1}) \rightarrow \{0,1\}$. For training this classifier, LABEL can be computed using the actual $\boldsymbol{s}^{t+1}$ recorded during data collection (see the "Phase 2 data collection" section). A diagram illustrating the inputs to the classifier and its labels is shown in Fig. 3. Last, given dynamics h and classifier g, we can approximately solve the problem in Eq. 2 using motion planning (see the "Planning with a learned model and classifier" section).

We note that the MER is conservative in the sense that we only need the final predicted $\hat{\boldsymbol{s}}^N$ and actual state $\boldsymbol{s}^N$ 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.

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 $r(\mathcal{E}, \boldsymbol{s}^t) \rightarrow \{0,1\}$, which determines whether a given state and environment pair can be escaped while enforcing the MER

$$r(\mathcal{E}, \boldsymbol{s}^t) = \begin{cases} 0 & \exists a \in \mathcal{A} \text{ for which } d\left(\hat{\boldsymbol{s}}^{t+1}, \boldsymbol{s}^{t+1}\right) < \delta \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

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

State definition

Here, we focus on rope manipulation tasks with $N_g = 1$ or $N_g = 2$ grippers; the state of each gripper is a point in $\mathbb{R}^3$. We assume that the robot end effectors are rigidly attached to the object. The configuration of the rope is a set of N_{rl} points in $\mathbb{R}^{3N_{rl}}$. 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 $N_d = 3(N_g + N_{rl})$.

Data collection for learning the unconstrained 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 $[\boldsymbol{s}^0, \boldsymbol{a}^0, \boldsymbol{s}^1, \ldots, \boldsymbol{a}^{N_t-1}, \boldsymbol{s}^{N_t}]$. In all our experiments, we use $N_t = 10$. We sample actions randomly, choosing target positions that are within 0.1 of the current gripper position(s). Put in another way, we sample actions that 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 that 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 6144 trajectories containing 10 steps, of which 1536 are reserved for validation and testing. For dual-arm rope manipulation, we collected 2048 trajectories containing 10 steps, of which 512 are reserved for validation and testing.

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 that takes in a state $\boldsymbol{s}^t$ and action $\boldsymbol{a}^t$ and predicts a change in state $\Delta\hat{\boldsymbol{s}}^{t+1}$. A diagram showing this architecture is shown in Fig. 4 and discussed in more detail in the "Network architectures" section. 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 Eq. 5

$$\begin{aligned} \text{MSE}(\boldsymbol{s}^0, \boldsymbol{a}^0, \ldots, \boldsymbol{a}^{N_t-1}) &= \frac{1}{N_t} \sum_{t=1}^{N_t-1} \|\hat{\boldsymbol{s}}^t - \boldsymbol{s}^t\|^2 \\ \hat{\boldsymbol{s}}^0 &= \boldsymbol{s}^0 \\ \hat{\boldsymbol{s}}^{t+1} &= h(\mathcal{E}, \hat{\boldsymbol{s}}^t, \boldsymbol{a}^t) \end{aligned} \tag{5}$$

Incorporating uncertainty in the learned dynamics

Prior work has shown that it can be beneficial to consider the uncertainty in the learned dynamics (*2*, *32*, *33*), even without considering constraints not seen in training. For instance, if the training data do 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) (*34*, *35*).

As is done in prior work (*2*, *3*, *36*), we use an ensemble of neural networks trained on the same phase 1 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 SDs 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 because it may be beneficial in scenarios where the unconstrained model is trained on a dataset with poor coverage of the state space.

Phase 2 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 (*37*) 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 (*38*). 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 (*36*, *39*). Those methods do not require a second data collection phase but, as a consequence, would be overly conservative because the presence of any obstacle near the rope would be considered out of distribution.

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 2048 trajectories of length 50 in phase 2, of which 512 were reserved for validation and testing. For dual-arm rope manipulation, we collected 5996 trajectories of length 20, of which 1516 were reserved for validation and testing.

In total, the combined phase 1 and 2 datasets for dragging have 163,840 transitions, and the combined phase 1 and 2 datasets for dual-arm rope manipulation have 140,400 transitions. In comparison, Li *et al.* (*40*) used 500,000 transitions to accurately learn the contact dynamics of a rope among disc-shaped obstacles in two dimensions.

Learning the classifier

Given the data collected in phase 2, we now describe how to construct training examples for the classifier. For this, we need both the predictions of the unconstrained dynamics and the labels of whether the MER 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 $(\boldsymbol{s}^t, \hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \boldsymbol{s}^{t+1}, \hat{\boldsymbol{s}}^{t+1})$ for each time step in each trajectory. As stated in the problem of Eq. 2, the label should be 1 if $d(\hat{\boldsymbol{s}}^{t+1}, \boldsymbol{s}^{t+1}) < \delta$ and 0 otherwise. To compute $d(\hat{\boldsymbol{s}}^{t+1}, \boldsymbol{s}^{t+1})$, we require a distance function d between the predicted state $\hat{\boldsymbol{s}}^{t+1}$ and the actual state $\boldsymbol{s}^{t+1}$. 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 the Supplementary Materials. For rope dragging, $\delta = 0.065$, and for dual-arm rope manipulation, $\delta = 0.025$. In addition, we discard all of the transitions after the first transition labeled 0 because 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 Fig. 5.

The classifier network $g(\mathcal{E}, \hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1}, \sigma^2) \rightarrow p_c \in \{0, 1\}$ takes as input the environment $\mathcal{E}$ and the transition $\hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1}$. Because 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 Fig. 4. Because this is a binary classification problem, we use binary cross-entropy loss to train it. Furthermore, because physical constraints are spatial in nature, we convert the environment and states into multichannel three-dimensional (3D) voxel grids and use convolutional neural networks (CNNs) (see details in the "Network architectures" section).

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.

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 CEM (*41*), probabilistic roadmaps (PRMs) (*42*), or trajectory optimization (*43*), we chose to use them in a kinodynamic RRT (*31*), as implemented in the Open Motion Planning Library (*44*). Kinodynamic RRT is a sampling-based tree-search algorithm and is well suited for our tasks because 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 is a criterion such as path length, which should be minimized.

In our kinodynamic RRT, we sample a single random action $\boldsymbol{a}^t$ and attempt to extend using that action. Whenever we attempt to extend from state $\hat{\boldsymbol{s}}^t$ with action $\boldsymbol{a}^t$ to the state $\hat{\boldsymbol{s}}^{t+1}$, we first check the transition $(\hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1})$ 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.

Evaluating stuck states and learning recovery actions

The formal definition of being stuck (Eq. 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. Because the classifier is trained to approximate the MER, this is a quick and effective method for determining whether recovery is needed. In addition, 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, prediction, or classification.

Given the unconstrained dynamics and the classifier, we can also use the data collected in phase 2 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 take recovery actions that 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, an action, and the environment ($\mathcal{E}_l$, $\boldsymbol{s}^t$, $\boldsymbol{a}^t$) and outputs the probability p_r that the action is recovering. Specifically, when we detect that 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 that 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 $\boldsymbol{s}^t$, $\boldsymbol{a}^t$, $\boldsymbol{s}^{t+1}$ and environment $\mathcal{E}$ in the dataset and first determines whether recovery is needed at $\boldsymbol{s}^t$. 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 $\boldsymbol{s}^t$ needs recovery. Next, we perform the same test starting at the next state $\boldsymbol{s}^{t+1}$. Here, we record the proportion of the sampled actions that were accepted, and this is used as the label for probability of recovery p_r. For example, if at $\boldsymbol{s}^t$ none of the random samples were accepted, but from $\boldsymbol{s}^{t+1}$ 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 $\boldsymbol{a}^t$ from $\boldsymbol{s}^t$ in environment $\mathcal{E}$ is likely to lead to recovery. On the other hand, if when checking $\boldsymbol{s}^{t+1}$ 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., $r(\mathcal{E}, \boldsymbol{s}^t) = 1$] 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 the "Network architectures" section.

Approaches to recovery like those from the safe exploration community (*45*–*47*) 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 that 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.

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 Fig. 4.

Dynamics model

The dynamics network is a two-layer fully connected network with 1024 hidden units in each layer and rectified linear unit (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.

Classifier

The classifier network takes in the environment $\mathcal{E}$ and a transition $\hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1}$ and outputs the probability of the MER being satisfied. Because 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 Fig. 4, a single time step is passed through a CNN encoder that maps a single state $\hat{\boldsymbol{s}}^t$ and action $\boldsymbol{a}^t$ 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. Because we use a single transition, this means that 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 $\hat{\boldsymbol{s}}^{t+1}$. Instead, we use the output for time $t + 1$ as the probability for the entire transition $\hat{\boldsymbol{s}}^t, \boldsymbol{a}^t, \hat{\boldsymbol{s}}^{t+1}$.

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 on the local origin (as defined above for dynamics). As with the local representation of states and actions, this assumes invariance to the absolute position. In addition, 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. 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 and 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 Fig. 5. We stack these representations along with the local environment to get a multichannel voxel grid.

The vector representations of state and action are the same as 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.

Recovery

Last, 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 because recovery is only used when accurate predictions cannot be made. After encoding, two fully connected layers with ReLU activation followed by a layer with sigmoid activation are used to map down to the output probability.

Full dynamics

Our FD baseline uses a different network. We use the network proposed in (*48*) 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 (*40*, *49*); however, these networks assume that a graphical model of the world is known, whereas our dynamics learning method does not.

Simulation environments

We use the Gazebo simulator with Open Dynamics Engine physics (*50*, *51*) 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 $\boldsymbol{s} = [x_g, y_g, z_g, x_1, y_1, z_1, \ldots, x_{10}, y_{10}, z_{10}]$, 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 other end of the rope (which we will call the tail) is the point that 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 $\boldsymbol{a} = [x, y, z]$, 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 s 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 $\boldsymbol{s} = [x_{g1}, y_{g1}, z_{g1}, x_{g2}, y_{g2}, z_{g2}, x_1, y_1, z_1, \ldots, x_{25}, y_{25}, z_{25}]$, which has 81 dimensions. The grippers attach to opposite ends of the rope. The actions are target positions for each gripper $\boldsymbol{a} = [x_1, y_1, z_1, x_2, y_2, z_2]$. A Jacobian-based inverse kinematic 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.

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 the "Learning the unconstrained dynamics," "Learning the classifier," and "Evaluating stuck states and learning recovery actions" sections.

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 in training and 0.0097 in testing. These numbers are small in comparison with the length of the rope (0.5 m) and the size of the environment (2 m by 2 m). For a visual demonstration of this level of error, please see our video. 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 and 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 FD 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 FD baseline.

We report learning metrics on the phase 2 dataset (see the "Learning the classifier" section 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 (*38*), 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 dual-arm rope manipulation, the training loss is 0.139, and the testing loss is 0.146.

Physical robot demonstrations

To demonstrate the practicality of our method, we designed real-world mock-ups 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 (*52*), much less in cluttered environments where the object is partially occluded. Online perception of the object is not within the scope of this paper. 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 future work, we will incorporate online perception into our execution pipeline when the appropriate perception algorithms are available.

In both scenes, we reuse the unconstrained dynamics models learned in simulation directly. For rope dragging, we also reuse 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 tested on in the real world. For more information, see the video in the Supplementary Materials.

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 s. During these 180 s, 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 Fig. 1 and the Supplementary Materials for more details).

At the end of the trial, the final state is used to determine the task error, and we report the 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.

SUPPLEMENTARY MATERIALS

robotics.sciencemag.org/cgi/content/full/6/54/eabd8170/DC1
Text
Fig. S1. Dragging classifier: Examples of contact.
Fig. S2. Classifier threshold sensitivity.
Movie S1. Video overview of our method and physical robot demonstrations.

REFERENCES AND NOTES

1. A. Wang, T. Kurutach, K. Liu, P. Abbeel, A. Tamar, Learning robotic manipulation through visual planning and acting, in *Robotics Science and Systems*, Freiburg im Breisgau, Germany (2019).
2. K. Chua, R. Calandra, R. McAllister, S. Levine, Deep reinforcement learning in a handful of trials using probabilistic dynamics models, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018).
3. B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017).
4. S. Kamthe, M. Deisenroth, Data-efficient reinforcement learning with probabilistic model predictive control. *Int. Conf. Artif. Intell. Stat.* **84**, 1701–1710 (2018).
5. Y. Wang, S. Chaudhuri, L. E. Kavraki, Bounded policy synthesis for POMDPs with safe-reachability objectives, in *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent* Systems, Stockholm, Sweden (2018).
6. A.-a. Agha-mohammadi, S. Chakravorty, N. M. Amato, Firm: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *Int. J. Robot. Res.* **33**, 266–304 (2013).
7. J. V. D. Berg, P. Abbeel, K. Goldberg, Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information, in *Robotics Science and Systems*, June 2010.
8. C. Finn, S. Levine, Deep visual foresight for planning robot motion, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 29 May to 3 June 2017.
9. T. Koller, F. Berkenkamp, M. Turchetta, J. Boedecker, A. Krause, Learning-based model predictive control for safe exploration and reinforcement learning, in *Robotics Science and Systems Workshop on Safe Autonomy* (2019).
10. A. Nagabandi, G. Kahn, R. S. Fearing, S. Levine, Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, in *International Conference on Robotics and Automation* (2018).
11. F. Lamiraux, L. E. Kavraki, Planning paths for elastic objects under manipulation constraints. *Int. J. Robot. Res.* **20**, 188–208 (2001).
12. J. Matas, S. James, A. J. Davison, Sim-to-real reinforcement learning for deformable object manipulation, in *Conference on Robot Learning*, Zürich, Switzerland (2018).
13. P. Sundaresan, J. Grannen, B. Thananjeyan, A. Balakrishna, M. Laskey, K. Stone, J. E. Gonzalez, K. Goldberg, Learning rope manipulation policies using dense object descriptors trained on synthetic depth data, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 31 May to 31 August 2020.
14. Y. Wu, W. Yan, T. Kurutach, L. Pinto, P. Abbeel, Learning to manipulate deformable objects without demonstrations, in *Robotics Science and Systems (Robotics Science and Systems)* (2020).
15. A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, S. Levine, Combining self-supervised learning and imitation for vision-based rope manipulation, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 29 May to 3 June 2017, pp. 2146–2153.
16. M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, S. Levine, SOLAR: Deep structured representations for model-based reinforcement learning, in *International Conference on Machine Learning* (2019).
17. G. A. Bekey, K. Y. Goldberg, *Neural Networks in Robotics* (Springer, 1993).
18. B. Ichter, M. Pavone, Robot motion planning in learned latent spaces. *IEEE Robot. Autom. Lett.* **4**, 2407–2414 (2019).
19. D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, H. David, L. Honglak, D. James, Learning latent dynamics for planning from pixels. *Int. Conf. Mach. Learn.* **97**, 2555–2565 (2019).
20. A. Srinivas, A. Jabri, P. Abbeel, S. Levine, C. Finn, Universal planning networks. *Int. Conf. Mach. Learn.* **80**, 4732–4741 (2018).
21. A. Vemula, Y. Oza, A. J.Bagnell, M. Likhachev, Planning and execution using inaccurate models with provable guarantees, in *Robotics Science and Systems* (2020).
22. D. Navarro-Alarcon, Y. Liu, J.G. Romero, P. Li, Visually servoed deformation control by robot manipulators, in *Proceedings of IEEE International Conference on Robotics and Automation*, 6 to 10 May 2013.
23. B. Jia, Z. Hu, J. Pan, D. Manocha, Manipulating highly deformable materials using a visual feedback dictionary, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 21 to 25 May 2018.
24. D. McConachie, A. Dobson, M. Ruan, D. Berenson, Manipulating deformable objects by interleaving prediction, planning, and control. *Int. J. Robot. Res.* **39**, 957–982 (2020).
25. J. Fu, S. Levine, P. Abbeel, One-shot learning of manipulation skills with online dynamics adaptation and neural network priors, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9 to 14 October 2016.
26. I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, P. Abbeel, Model-based reinforcement learning via meta-policy optimization, in *Conference on Robot Learning* (2018).
27. K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, V. Vanhoucke, Using simulation and domain adaptation to improve efficiency of deep robotic grasping, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 21 to 25 May 2018.
28. X. B. Peng, M. Andrychowicz, W. Zaremba, P. Abbeel, Sim-to-real transfer of robotic control with dynamics randomization, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 21 to 25 May 2018.
29. J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning quadrupedal locomotion over challenging terrain. *Sci. Robot.* **5**, eabc5986 (2020).
30. OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, L. Zhang, Solving rubik's cube with a robot hand. arXiv:1910.07113 (2019).
31. S. M. LaValle, J. J. Kuffner Jr., Randomized kinodynamic planning. *Int. J. Robot. Res.* **20**, 378–400 (2001).
32. J. G. Schneider, Exploiting model uncertainty estimates for safe dynamic control learning, in *Proceedings of the 9th International Conference on Neural Information Processing Systems* (MIT Press, Cambridge, MA, 1996).
33. S. Bechtle, Y. Lin, A. Rai, L. Righetti, F. Meier, Curious ilqr: Resolving uncertainty in model-based RL. *Proc. Conf. Robot Learn.* **100**, 162–171 (2019).
34. E. Hullermeier, W. Waegeman, Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Mach. Learn.* **110**, 457–506 (2021).
35. S. C. Hora, Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management. *Reliab. Eng. Syst. Saf.* **54**, 217–223 (1996).
36. Y. Gal, Z. Ghahramani, Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *Int. Conf. Mach. Learn.* **48**, 1050–1059 (2016).
37. J. Guzzi, O. R. Chavez-Garcia, M. Nava, L. M. Gambardella, A. Giusti, Path planning with local motion estimations. *IEEE Robot. Autom. Lett.* **5**, 2586–2593 (2020).
38. D. McConachie, T. Power, P. Mitrano, D. Berenson, Learning when to trust a dynamics model for planning in reduced state spaces. *IEEE Robot. Autom. Lett.* **5**, 3540–3547 (2020).
39. K. Zhou, J. C. Doyle, *Essentials of Robust Control* (Prentice Hall, 1998), vol. 104.
40. Y. Li, J. Wu, J. Zhu, J. B. Tenenbaum, A. Torralba, R. Tedrake, Propagation networks for model-based control under partial observation, in *Proceedings of the International Conference on Robotics and Automation*, 20 to 24 May 2019, pp. 1205–1211.
41. M. Kobilarov, Cross-entropy randomized motion planning. *Rotob. Sci. Syst.* **31**, 855–871 (2011).
42. L. E. Kavraki, P. Svestka, J. C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **12**, 566–580 (1996).
43. N. D. Ratliff, A. J. Bagnell, M. A. Zinkevich, Maximum margin planning, in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, USA (2006).
44. I. A. Şucan, M. Moll, L. E. Kavraki, The open motion planning library. *IEEE Robot. Autom. Mag.* **19**, 72–82 (2012).
45. F. Berkenkamp, M. Turchetta, A. P. Schoellig, A. Krause, Safe model-based reinforcement learning with stability guarantees, in *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS)*, 4 to 9 December 2017.
46. T. Koller, F. Berkenkamp, M. Turchetta, A. Krause, Learning-based model predictive control for safe exploration, in *Proceedings of the 2018 IEEE Conference on Decision and Control*, Miami, FL, USA (2018), pp. 6059–6066.
47. J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, C. J. Tomlin, A general safety framework for learning-based control in uncertain robotic systems. *IEEE Trans. Autom. Control* **64**, 2737–2752 (2019).
48. A. Nagabandi, G. Yang, T. Asmar, R. Pandya, G. Kahn, S. Levine, R. S. Fearing, Learning image-conditioned dynamics models for control of under-actuated legged millirobots, in *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Spain (2017).
49. D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, D. L. K. Yamins, Flexible neural representation for physics prediction, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Montréal, Canada (2018).
50. N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan (2004).
51. R. Smith, *Open Dynamics Engine* (2005); www.ode.org/.

52. M. Yan, Y. Zhu, N. Jin, J. Bohg, Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robot. Autom. Lett.* **5**, 2372–2379 (2020).

Funding: This work was supported by NSF Grant IIS-1750489 (to D.B.) and ONR grant N000141712050 (to P.M., D.M., and D.B.) and by Toyota Research Institute (TRI). This article solely reflects the opinions of its authors and not of TRI or any other Toyota entity (P.M., D.M., and D.B.). **Author contributions:** Conceptualization: P.M., D.M., and D.B. Data curation: P.M. Formal analysis: P.M. Funding acquisition: D.B. Investigation: P.M. and D.M. Methodology: P.M., D.M., and D.B. Project administration: D.B. Resources: D.B. Software: P.M. and D.M. Supervision: D.B. Validation: P.M. and D.B. Visualization: P.M. and D.M. Writing—original draft: P.M. Writing—review and editing: P.M., D.M., and D.B. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All data and code needed to evaluate the conclusions in the paper can be found at https://sites.google.com/view/unreliable-deform-manipulation.

Submitted 2 August 2020
Accepted 26 April 2021
Published 19 May 2021
10.1126/scirobotics.abd8170

Citation: P. Mitrano, D. M[c]Conachie, D. Berenson, Learning where to trust unreliable models in an unstructured world for deformable object manipulation. *Sci. Robot.* **6**, eabd8170 (2021).

Science Robotics

Learning where to trust unreliable models in an unstructured world for deformable object manipulation

P. Mitrano, D. McConachie, and D. Berenson

Sci. Robot. **6** (54), eabd8170. DOI: 10.1126/scirobotics.abd8170

Science Robotics (ISSN 2470-9476) is published by the American Association for the Advancement of Science. 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.