Learning When to Trust a Dynamics Model for Planning in Reduced State Spaces

Dale M^{c}Conachie[1], Thomas Power[1], Peter Mitrano[1] and Dmitry Berenson[1]

***Abstract*—When the dynamics of a system are difficult to model and/or time-consuming to evaluate, such as in deformable object manipulation tasks, motion planning algorithms struggle to find feasible plans efficiently. Such problems are often reduced to state spaces where the dynamics are straightforward to model and evaluate. However, such reductions usually discard information about the system for the benefit of computational efficiency, leading to cases where the true and reduced dynamics disagree on the result of an action. This paper presents a formulation for planning in reduced state spaces that uses a classifier to bias the planner away from state-action pairs that are not reliably feasible under the true dynamics. We present a method to generate and label data to train such a classifier, as well as an application of our framework to rope manipulation, where we use a Virtual Elastic Band (VEB) approximation to the true dynamics. Our experiments with rope manipulation demonstrate that the classifier significantly improves the success rate of our RRT-based planner in several difficult scenarios which are designed to cause the VEB to produce incorrect predictions in key parts of the environment.**

***Index Terms*—Motion and Path Planning; Learning and Adaptive Systems**

I. INTRODUCTION

ROBOT motion planning algorithms have been extremely successful for systems where the dynamics can be easily specified and efficiently evaluated. However, for tasks such as manipulation of deformable objects, the dynamics are very difficult to model [1] and usually require numerical simulation to evaluate. This simulation can be time-consuming and/or inaccurate. Including such simulations inside a planner can result in plans that take hours to compute [2].

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

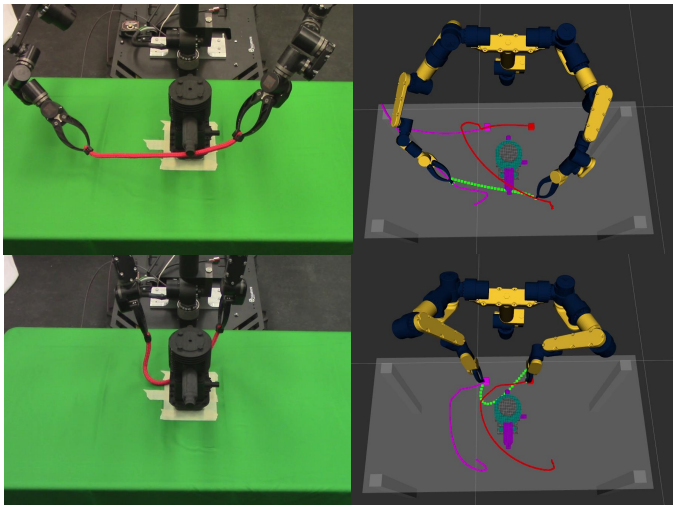

Fig. 1. Top: a plan generated without using a classifier moves the rope under a hook and gets caught. Bottom: a plan generated with a classifier avoids this mistake, and reaches the goal.

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

This paper presents both an abstract formulation of the problem of planning in a reduced state space with a classifier and how to apply this formulation to two systems. First, to illustrate our framework on a straightforward example, we consider a planar three-link arm with limited joint torque. Using a learned classifier for this system allows us to plan in configuration space (not considering dynamics) while avoiding transitions that cannot be accomplished with the limited torque. The second system focuses on rope manipulation tasks; we use a Virtual Elastic Band (VEB) [3] as the reduced dynamics model of the rope, as this has been shown to allow fast planning in difficult scenarios. However, this model assumes that there is no minimum length of the rope, which entails that the planner cannot detect cases where the slack material is caught on corners or hooks, preventing the motion plan from being completed because the caught object can

Manuscript received: September 11, 2019; Revised December 7, 2019; Accepted January 20, 2020.

This paper was recommended for publication by Editor Nancy Amato upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF grant IIS-1750489, ONR grant N000141712050, and by Toyota Research Institute (TRI). This article solely reflects the opinions of its authors and not TRI or any other Toyota entity.

[1] All authors are with the University of Michigan Robotics Institute, Ann Arbor, MI, USA. {dmcconac, tpower, pmitrano, dmitryb}@umich.edu

Digital Object Identifier (DOI): see top of this page.

overstretch (see Figure 1). Thus, we learn a classifier to bias the planner away from states where the object can be caught on an obstacle. The contributions of this paper are:

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

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

II. RELATED WORK

Robotic manipulation of deformable objects has been studied in many contexts ranging from surgery to industrial manipulation (see [4] and [5] for extensive surveys).

Much work in deformable object manipulation relies on simulating an accurate model of the object being manipulated. The most common simulation methods use Mass-Spring models [6], [1], which are generally not accurate for large deformations [7], and Finite-Element (FEM) models [8], [9], [10]. FEM-based methods are widely used and physically well-founded, but they can be unstable when subject to contact constraints, which are especially important in this work.

Motion planning for manipulation of deformable objects is an active area of research [11] with many sampling-based planners proposed [12], [13], [14], [15], [16]. However, all the above methods either disallow contact with the environment or rely on potentially time-consuming physical simulation of the deformable object, which is often very sensitive to physical and computational parameters that may be difficult to determine. In contrast our method builds on [17], which uses reduced models for motion planning with far lower computational cost.

In terms of applying machine learning to control and planning, prior work has primarily used learned dynamics models for control [18], [19], [20], [21], [22]. Recent work [23] has also explored planning in learned reduced space, but they do not consider the error in a reduced model's prediction when planning. Visual Planning and Acting (VPA) [24] learns a latent transition model based on visual input for planning. This work also uses on a classifier to prune infeasible transitions during planning. However, despite this classifier, only 15% of generated plans were visually plausible, with only 20% of the visually plausible plans being executable. In this paper we do not focus on learning a reduction but rather on creating a framework that can be used to overcome limitations in a given model reduction.

III. GENERAL PROBLEM FORMULATION

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

The problem we address in this work is how to find a sequence of N_e commands $\{u_1^x, \ldots, u_{N_e}^x\}$ to move from a start state x_0 to a goal state. The goal set is specified by the function $\texttt{Goal}_x : \mathcal{X} \to \{0, 1\}$, which returns 1 if a state is a goal and 0 otherwise. We thus to seek to solve the following:

$$\begin{aligned} \text{find} \quad & N_e, \{u_1^x, \ldots, u_{N_e}^x\} \\ \text{s.t.} \quad & \texttt{Goal}_x(x_{N_e}) = 1 \\ & x_t = f(x_{t-1}, u_t^x, E),\ i = 1, \ldots, N_e \end{aligned} \tag{1}$$

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

To create a more tractable planning problem we define $\mathcal{B}$ to be a *reduced state space* and define a reduction function: $b = r(x, E)$. We do not assume that r is invertible. Dynamics in $\mathcal{B}$ are defined as $b_{t+1} = g(b_t, u^b, E)$ (note that u^b and u^x may be in different spaces). We treat the dynamics in this reduced state space as Markovian. $\mathcal{B}$, r, and g are user-defined. There is then an analogous goal function for reduced states $\texttt{Goal}_b : \mathcal{B} \to \{0, 1\}$. The planning problem then becomes:

$$\begin{aligned} \text{find} \quad & N_e, \{u_1^b, \ldots, u_{N_e}^b\} \\ \text{s.t.} \quad & b_0 = r(x_0, E) \\ & \texttt{Goal}_b(b_{N_e}) = 1 \\ & b_t = g(b_{t-1}, u_t^b, E),\ i = 1, \ldots, N_e \end{aligned} \tag{2}$$

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

We will solve the problem in Equation 2 using a motion planner that plans in $\mathcal{B}$. However, the plan we generate may not lead to the goal in execution because we may have lost information in r and/or g. Our approach is thus to bias our planner so that it avoids taking actions for which r and g are not reliable approximations of the behavior of the system. See Fig. 2 for an overview of our framework.

IV. LEARNING TRANSITION RELIABILITY

To bias the planner that plans in $\mathcal{B}$ we will learn a classifier that attempts to predict if a given transition $T^b =$

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

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

A. Data Generation and Labeling

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

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

$$l(t, \mathbb{T}^b, \tau^b) = \begin{cases} \texttt{Reliable} & \textbf{if } \texttt{Close}(\hat{b}_t, \tilde{b}_t) \textbf{ and} \\ & \quad \texttt{Close}(\hat{b}_{t+1}, \tilde{b}_{t+1}) \\ \texttt{Unreliable} & \text{otherwise} \end{cases} \tag{3}$$

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

B. An Illustrative Navigation Example

To clarify the above framework and learning problem formulation, we describe an example system where the various

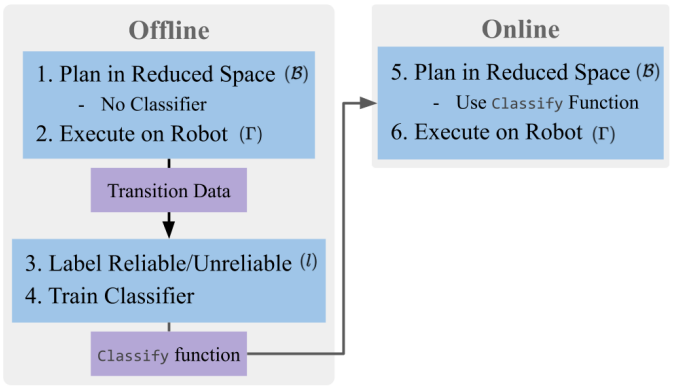

Fig. 2. An outline of our framework. First, we plan and execute many control sequences to gather a dataset of transitions. These transitions are labeled according to a function l and used to train a classifier which predicts whether a transition is reliable given the model reduction. This classifier is used to bias the planner away from unreliable transitions.

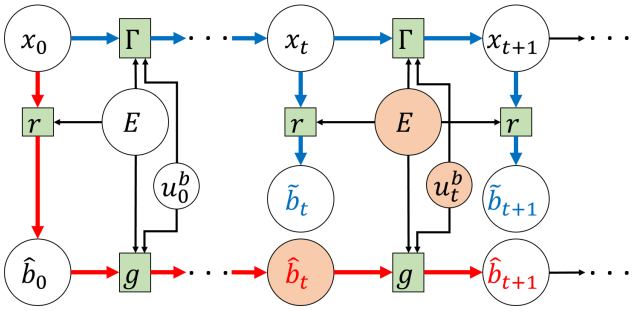

Fig. 3. Circles represent variables and boxes represent functions. Orange: variables defining the tth transition. Red path: reduced dynamics prediction; Blue path: rollout result.

functions and spaces can be easily visualized. Consider a car with state $x = [q, \dot{q}]$, where $q = [q_x, q_y, q_\theta]$ and control inputs $u^x = [u^x_a, u^x_\phi]$, which correspond to the acceleration and steering angle. $f(x_t, u^x, E)$ is the standard second-order car dynamics.

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

Let the rollout function $x_{t+1} = \Gamma(x_t, u^b, E)$ be a method that uses a controller to drive the car toward $r(x_t, E) + u^b$. Γ also checks if the car reaches the boundary of an obstacle in E and will return the state on the boundary if it does. $\texttt{Close}(\hat{b}_1, \tilde{b}_1)$ outputs 1 if the two reduced states are within a small Euclidean distance and 0 otherwise.

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

planner to transitions that are more likely to succeed when executing Γ (see Fig. 4).

C. What can be learned

While we may produce a useful classifier for planning, it is important to know that there is a fundamental limitation on what can be learned by this approach because of the loss of information that may happen in r and/or g. Consider the following scenario: Let $b_0 = r(x_0^a, E) = r(x_0^b, E) = r(x_0^c, E)$, e.g. there are multiple states where the car is at a certain position but with a different velocity. If we apply reduced dynamics prediction for some u^b, we obtain $\hat{b}_1 = g(b_0, u^b, E)$. However, we if do rollouts we obtain three resulting states: $x_1^a = \Gamma(x_0^a, u^b, E), x_1^b = \Gamma(x_0^b, u^b, E), x_1^c = \Gamma(x_0^c, u^b, E)$, and then three reduced states: $\tilde{b}_1^a = r(x_1^a, E), \tilde{b}_1^b = r(x_1^b, E), \tilde{b}_1^c = r(x_1^c, E)$. It may be the case that $\texttt{Close}(\hat{b}_1, \tilde{b}_1^k)$ does not produce the same result for $k = a, b, c$ (an example for the car system is shown in Figure 5). In terms of classification, this is a case of noisy labeling, and many methods have been devised to address this problem (e.g. SVM with slack variables).

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

D. Using the Classifier in Planning

While it is possible to query every transition considered by the planner and reject all those that are classified as `Unreliable`, such a strategy would likely be overly optimistic about what the classifier has learned. In difficult scenarios the classifier may erroneously reject a set of transitions which is necessary to reach the goal, thus causing the planner to fail. Thus we accept transitions that are classified as `Unreliable` with a small probability determined by parameters k, a manually-specified constant, and p_{acc}, the validation accuracy of the classifier (see Algorithm 1). p_{acc} is included because we wish to be more permissive about accepting transitions when the classifier performs more poorly in terms of generalization.

While the above approach of incorporating classification can be applied to a wide range of planners, in this paper we focus on using RRT-based planners. An advantage of this approach for RRT-based planners is that we maintain the probabilistic completeness properties of the planner by guaranteeing that any transition will be accepted with a non-zero probability (although the probability is small for transitions classified as `Unreliable`).

V. APPLICATION TO A TORQUE-LIMITED PLANAR ARM

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

Algorithm 1 `CheckTransition`(T^b)

```
b' ← g(T^b.b, T^b.u^b, E)
if Classify(T^b) == Reliable then
    return b'
a ~ U[0, 1]
if a < e^{-k p_acc} then
    return b'
return ∅
```

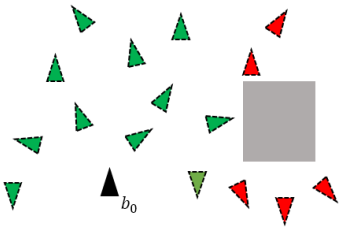

Fig. 4. Illustration of desired prediction from a classifier. Dotted triangles indicate $\hat{b}_1$s from different u_0^b inputs. Green: Classifier predicts these transitions are `Reliable`. Red: Classifier predicts these transitions are `Unreliable`. Note that the line between b_0 and $\hat{b}_1$ is collision-free for all examples shown.

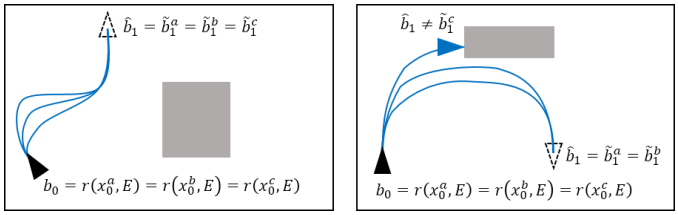

Fig. 5. Illustration of the effect of information loss on the predictability of a transition. In both scenarios states with different velocities reduce to the same b_0. Left: A case where rolling out the same u^b from different initial velocities (blue) produces the same $\tilde{b}_1$ values, since the controller is robust to initial velocity in this case. Right: A case where rolling out the same u^b with different initial velocities produces different $\tilde{b}_1$ values. At high initial velocity (c) the controller cannot turn before reaching the obstacle.

For this experiment we use the MuJoCo simulator [25] as the ground truth dynamics. Each joint is controlled using the default position servo actuator available in MuJoCo. We convert the MuJoCo simulation to a quasistatic system by waiting for the arm to settle after a configuration is commanded (this is f). These experiments do not have any obstacles, so we omit E for brevity.

A. Problem Statement

Let $x \in \mathbb{R}^3$ be the true state of the system. Let $u^x = x_{des} \in \mathbb{R}^3$ be the commanded position of the arm at each timestep. Let $f(x_t, u_t)$ be the quasistatic dynamics defined by MuJoCo. We set torque limits τ_1, τ_2, τ_3 so that $\tau_1 \ll \tau_2 = \tau_3$. This means the first joint cannot support the weight of the arm when extended horizontally. As we are not doing a model reduction, $b = r(x) = x$. Commands in both spaces are also the same: $u^b = u^x$. The dynamics in $\mathcal{B}$ are purely kinematic: $b_{t+1} = g(b_t, u_t^b) = u_t^b$. These dynamics are fast to evaluate and therefore efficient for planning, but can result in plans which do not reach the goal configuration when executed (Fig. 6). As there are no obstacles and $u^b = u^x$, the rollout function $\Gamma(x_t, u_t^b, E) = f(x_t, u_t^x)$.

The planning problem is for the arm to reach a goal end-effector position. Using the true dynamics, this corresponds to Problem (1). However, since we assume the true dynamics are not available, we seek to solve Problem (2) given the definitions of r and g above.

B. Data Collection, Labelling, and Training

To collect training data we initialized the system from random start configurations, planning to random goal configurations using RRT-Connect [26]. In practice these are straight lines in configuration space after smoothing the path. This generated a total of 231,815 transitions to use in training and validation. A randomly selected 20% of the data is held out for validation. We define $\texttt{Close}(\hat{b}_t, \tilde{b}_t)$ based on the Euclidean distance between configurations:

$$\texttt{Close}(\hat{b}_t, \tilde{b}_t) = \|\hat{b}_t - \tilde{b}_t\| < \alpha \tag{4}$$

with $\alpha = 0.075$.

For this planar arm, our classifier is a neural network that takes b and b' as input. The network has three hidden layers with 256, 128, and 64 hidden units respectively and a single output neuron. The hidden units use a Leaky ReLu activation [27], and the output uses a sigmoid activation. With this network we achieve 99% training accuracy and 99% validation accuracy within 4 epochs.

C. Planning and Results

To test the effect of using the learned classifier, we randomly generate 100 planning queries and evaluate how well generated plans perform when executed. Each planning query consists of a random start configuration in $\mathbb{R}^3$ and goal IK solutions for a random target point in $\mathbb{R}^2$. We only consider queries for which there is at least one IK solution where the controller can maintain the configuration of the arm within $\alpha = 0.075$ of the IK solution. For planning we use the OMPL [28] implementation of RRT-Connect, setting the probability scale factor k to 1, and p_{acc} to 0.99. The resulting path is post-processed using the default `simplifyMax` options. A plan is considered successful if the distance between the final configuration and any IK solution is less than a threshold β (Fig. 6). Tests are performed on an i5-3570K @ 4.3 GHz. Planning and simplification takes approximately 2 ms without a classifier, and approximately 7 seconds with our classifier. For small β, using the classifier does not improve performance due to the steady state error inherent in the system. As β increases, we see that the planner that uses a classifier is able to successfully find paths to all queries while the baseline is unable to succeed at some queries (see attached video).

VI. APPLICATION TO ROPE MANIPULATION

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

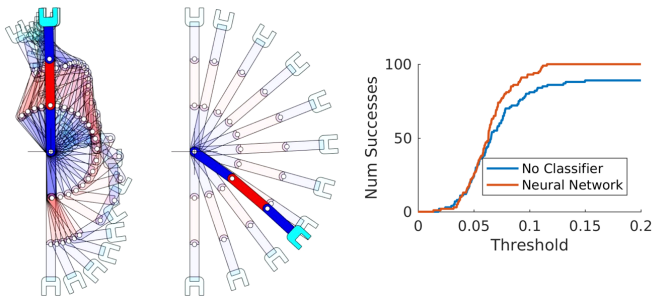

Fig. 6. Left: Plan generated using the learned classifier to go from $[\frac{-pi}{2}, 0, 0]$ to $[\frac{pi}{2}, 0, 0]$. The plan avoids transitions which move the arm toward a horizontal position and successfully completes the task. Center: Plan generated without the classifier. The plan takes the arm to the horizontal position where it fails due to the torque limit. Right: Number of successes as success threshold β varies

A. Problem Statement

Let the robot be represented by a pair of grippers with configuration $q \in SE(3)^2$. We assume that the robot configuration can be measured exactly. In this work we assume the robot to be a set of free floating grippers; in practice we can track the motion of these with inverse kinematics.

We assume that our model of the robot is purely kinematic, with no higher order dynamics. We assume that the robot has two end-effectors that are rigidly attached to the rope. The configuration of a rope is a set $\mathcal{P} \subset \mathbb{R}^3$ of $P = |\mathcal{P}|$ points. We assume that we have a method of sensing $\mathcal{P}$. The rest of the environment E is assumed to be both static, and known exactly. We assume that the robot moves slowly enough that we can treat the combined robot and rope as quasi-static. The true state of the system is then $x = [q, \mathcal{P}]$ and $u^x = \Delta q$. Let f be a joint-space controller for the robot that stops when any of the following occur: 1) the grippers contact an object; or 2) the object stretches by more than a factor λ. Due to the difficulty of simulating rope physics, we do not assume we can execute f within a motion planner.

We wish to find a sequence of N_e commands $\{u^x_1, \ldots, u^x_{N_e}\}$ to move from a start state x_0 to a goal gripper configuration q_g such that each motion is feasible (this corresponds to Problem (1)). Note that this planning problem does not require bringing the rope to a specific configuration, which can often be done using local control *after* bringing the object to a desired area (as in [17]). Because we do not have access to f, we cannot solve this problem by planning in $\mathcal{X}$ directly.

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

B. Reduction

[3] introduced the idea of a *virtual elastic band* (VEB) between the robot's end-effectors. This VEB represents the shortest path through the rope between the end-effectors. The band approximates the constraint imposed by the rope on the motion of the robot; if the end-effectors move too far apart, then the VEB will be too long, and thus the rope is stretched

beyond a task-specified maximum stretching factor. Similarly, if the VEB gets caught on an obstacle and becomes too long, then the rope is also overstretched. By considering only the geodesic between the end-effectors, we are assuming that the rest of the rope will comply to the environment, and does not need to be considered when predicting overstretch. The VEB representation allows us to use a fast prediction method when planning, but does not account for the part of the material that is slack. Denote the configuration of a VEB (i.e. a sequence of points) as v. Then $b = [q, v]$ and can be generated by $b = r(x, E)$ for the reduction function defined in Section 4.2.2 of [3]. The dynamics of a VEB, $g(b, u^b, E)$, are based on Quinlan's path deformation algorithm as presented in [29] (see Section 4.2.3 of [3]). We also augment g to return $\varnothing$ when u^b causes the object to become overstretched. b is then propagated using g and $u^b = u^x$. Since the commands are the same, our rollout function Γ, which uses u^b, is equivalent to f. To find a path in $\mathcal{B}$ we must solve Problem (2).

We use the planner described in [17] to solve this problem; this is an RRT-based planner designed for use with virtual elastic bands as part of the planning configuration space. This planner searches for a feasible path for the robot between a given start and goal configuration, while ensuring the VEB is never overstretched.

The virtual band approximation choice favors speed over model accuracy; as a consequence, there are several issues that it does not address. Specifically, environments with "hooks" can cause problems due to the approximation methods: The virtual elastic band assumes that there is no minimum length of the rope. This assumption means that the planner cannot detect cases where the slack material can get caught on corners or hooks, preventing the motion plan from being completed because the caught object can overstretch. To reduce the chances of this occurring we learn a classifier for T^b to predict if a given transition is either `Reliable` or `Unreliable` and bias the planner away from `Unreliable` transitions. We bias the planner using the `CheckTransition` function shown in Algorithm 1, which is used as an edge validity check in addition to the collision and overstretching checks described in [17].

C. Learning the Classifier

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

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

Where D computes the sum of the point-wise distances between the two virtual elastic bands, α is a constant, and `FOH` evaluates if the two bands are in the same first-order homotopy class [30]. We then apply labeling function l shown in Eq. 3 for each transition. We generate many plans using our planner (without the classifier) to produce the dataset (see Section VII-B).

For our classifier we use a neural network based on *VoxNet* [31], a network for classifying objects from a voxel grid. The network consists of two 3D convolutional layers (filter size 5, 3 and stride 2, 1 respectively) with max pooling followed by two fully-connected layers. All layers have ReLU activations except the output layer, which has a sigmoid activation.

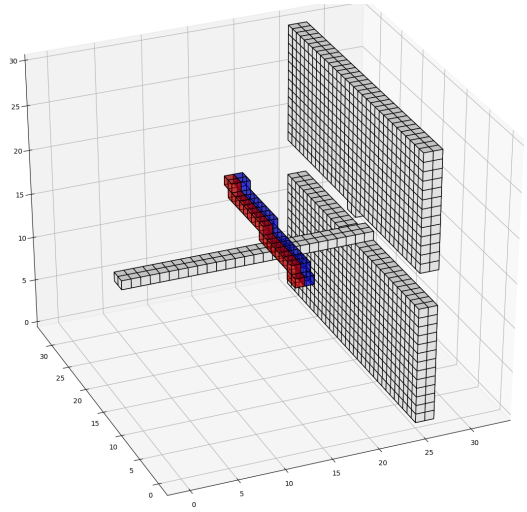

Fig. 7. Input to the VoxNet classifier is a 3-channel voxel grid, where white is the local environment, red is the pre-transition band, and blue is the post transition band. Positions outside the bounds of the environment are marked as occupied in the local environment channel.

The input for our classifier consists is a three-channel binary voxel grid, with channels $\langle E, b, b' \rangle$, where $b' = g(b, u^b, E)$. Each voxel grid is $32 \times 32 \times 32$, and is constructed by checking for occupancy at every cell center. An example of the voxelized representation is shown in Fig. 7.

VII. ROPE MANIPULATION EXPERIMENTS

To characterize the planner performance with the classifier, we designed seven simulation scenarios where the rope must be moved from one side of the scene to the other, along with one physical experiment for real-world validation. All simulation experiments were conducted in the open-source Bullet simulator [32], with additional wrapper code developed at UC Berkeley [33]. The rope is modeled as a series of small capsules linked together by springs. We emphasize that our method does not have access to the model of the rope or the simulation parameters. The simulator is used as a "black box" for testing. All tests are performed using an i7-7700K 4.2 GHz CPU with 32 GB of RAM. For all experiments we set λ to 1.15, and α to 0.5.

A. Scenarios

Each scenario involves moving the rope past one (or more) hooks, while the grippers traverse a narrow slit (Fig. 8).

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

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

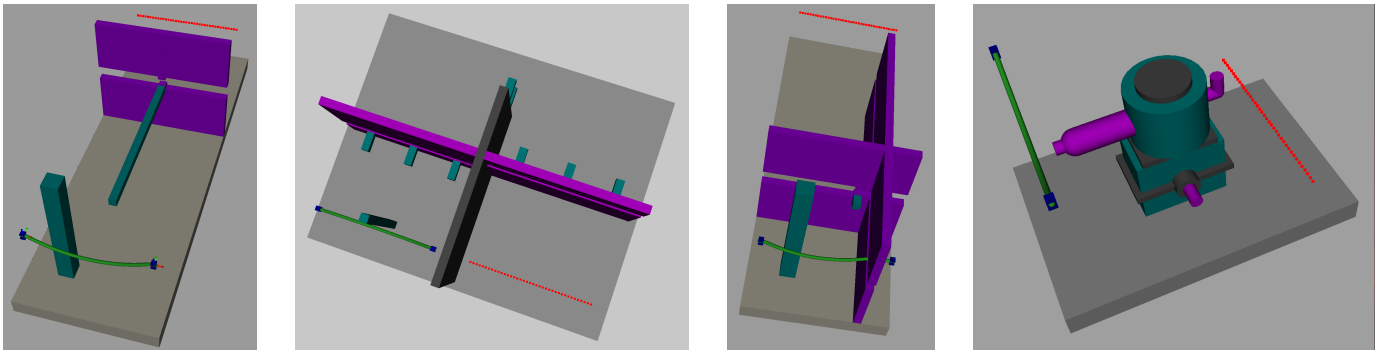

Fig. 8. The rope is shown in green, with the grippers shown in blue. The target area for the grippers is shown in red. Walls with narrow slits for the grippers are shown in purple. Hooks and other obstacles are shown in dark cyan. Left: Simple Hook; Center Left: Multi Hook; Center Right: Complex Hook; Right: Engine Assembly

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

4) Engine Assembly: In the Engine Assembly environment, we seek to move the grippers from one side of an engine model [34] to the others avoiding two hooks on the front and back of the engine. The rope has length 0.87m. The engine assembly environment is shown in Fig. 8.

5) Physical Robot: In the Physical Robot environment, we attempt the engine assembly task on a physical 16 DoF robot shown in Fig. 1 with a 3D printed model of the engine and a rope of length 0.46m.

B. Data Collection

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

C. Training the Classifier

VoxNet is trained using the Adam optimizer [35] with initial learning rate of 5×10^{-4}. We use a learning rate decay of 0.8 every 4 epochs. Since the dataset set is imbalanced, with 32% of the examples labelled as unreliable, and 68% labelled as reliable, we use a weighted random sampler to make all minibatches balanced. A randomly selected 10% of the data is held out for validation. Our minibatch size is 32, and we train for 100 epochs. We use the binary cross-entropy loss function during training. Training took approximately 16 hours on a Tesla V100-SMX2 GPU. The VoxNet classifier achieved 99% accuracy on the training set and 91% accuracy on the validation set.

D. Planning Results

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

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

TABLE I
PLANNING STATISTICS, AVERAGED OVER 30 TRIALS

VIII. DISCUSSION

We found that the use of a neural network classifier to evaluate the reliability of a model approximation can be an effective way to improve the success rate of a planner for rope manipulation. When using the classifier in the planner as a hard constraint ($k = \infty$), some starting configurations would cause the classifier to reject all transitions from that state, leading to planning failure. An interesting approach to setting k would be to treat it similar to temperature as done in T-RRT [36]. Despite training the rope manipulation classifier

on only a single rope and environment (the Simple Hook with Regular length rope), we found that the classifier was able to generalize and lead to improved planning performance with both different lengths of rope and more complex environments.

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

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

One interesting point is that increased classification accuracy does not necessarily lead to better planning performance. We experimented with different classifiers during development, and although VoxNet had the best classification accuracy on the validation set and usually the best performance, other classifiers performed equivalently or better for planning in some environments. It would therefore be desirable to find a way to use planner success to train the classifier, and we plan to investigate this in future work.

IX. CONCLUSION

This paper proposed a novel formulation of planning in reduced state spaces that directly addresses the challenge of generating plans that are feasible in the true state space while planning in a reduced space with approximate dynamics. We addressed this model mismatch problem by introducing the use of a classifier into the planning process. This classifier is trained to classify state transitions in the reduced state space as reliable or unreliable. Our experiments on both a torque-limited 3-link planar arm and rope manipulation tasks show that 1) we can learn a classifier from training data with high validation accuracy; and 2) using this classifier to bias the planner away from unreliable transitions improves success rate in all tested scenarios.

REFERENCES

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

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

[3] D. M^{c}Conachie, M. Ruan, and D. Berenson, "Interleaving planning and control for deformable object manipulation," in *ISRR*, 2017.

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

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

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

[7] B. Maris, D. Botturi, and P. Fiorini, "Trajectory planning with task constraints in densely filled environments," in *IROS*, 2010, pp. 2333–2338.

[8] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," *SIGGRAPH*, pp. 49–54, 2002.

[9] G. Irving, J. Teran, and R. Fedkiw, "Invertible finite elements for robust simulation of large deformation," *SIGGRAPH*, pp. 131–140, 2004.

[10] P. Kaufmann, S. Martin, M. Botsch, and M. Gross, "Flexible simulation of deformable models using discontinuous Galerkin FEM," in *SIGGRAPH*, 2008.

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

[12] O. Burchan Bayazit, Jyh-Ming Lien, and N. Amato, "Probabilistic roadmap motion planning for deformable objects," in *ICRA*, vol. 2, 2002, pp. 2126–2133.

[13] R. Gayle, M. Lin, and D. Manocha, "Constraint-Based Motion Planning of Deformable Robots," in *ICRA*, 2005, pp. 1046–1053.

[14] M. Moll and L. E. Kavraki, "Path Planning for Deformable Linear Objects," *TRO*, vol. 22, no. 4, pp. 625–636, 2006.

[15] M. Saha, P. Isto, and J.-C. Latombe, "Motion planning for robotic manipulation of deformable linear objects," in *ISER*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 23–32.

[16] O. Roussel, A. Borum, M. Taïx, and T. Bretl, "Manipulation planning with contacts for an extensible elastic rod by sampling on the submanifold of static equilibrium configurations," in *ICRA*, May 2015, pp. 3116–3121.

[17] D. M^{c}Conachie, A. Dobson, M. Ruan, and D. Berenson, "Manipulating Deformable Objects by Interleaving Prediction, Planning, and Control," *IJRR*, Accepted. [Online]. Available: https://sites.google.com/umich.edu/dmcconachie

[18] B. Jia, Z. Hu, J. Pan, and D. Manocha, "Manipulating highly deformable materials using a visual feedback dictionary," in *ICRA*, 2018.

[19] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *ICRA*, 2017.

[20] E. Banijamali, R. Shu, M. Ghavamzadeh, H. Bui, and A. Ghodsi, "Robust Locally-Linear Controllable Embedding," *AISTATS*, oct 2017.

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

[22] G. Sutanto, N. Ratliff, B. Sundaralingam, Y. Chebotar, Z. Su, A. Handa, and D. Fox, "Learning Latent Space Dynamics for Tactile Servoing," *ICRA*, 2019.

[23] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *RA-L*, vol. 4, no. 3, pp. 2407–2414, 2019.

[24] A. Wang, T. Kurutach, K. Liu, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," in *RSS*, 2019.

[25] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*. IEEE, 2012, pp. 5026–5033.

[26] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[27] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, 2013.

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

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

[30] L. Jaillet and T. Siméon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *IJRR*, vol. 27, no. 11-12, pp. 1175–1188, 2008.

[31] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *IROS*, 2015, pp. 922–928.

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

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

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

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[36] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based rrt for path planning in continuous cost spaces," in *IROS*, 2008, pp. 2145–2150.