Hierarchical Planning for Rope Manipulation using Knot Theory and a Learned Inverse Model

Matan Sudry, Tom Jurgenson, Aviv Tamar and Erez Karpas

Technion — Israel Institute of Technology, Haifa, Israel
{matansudry, tomj}@campus.technion.ac.il, {avivt, karpase}@technion.ac.il

Abstract

This work considers planning the manipulation of deformable 1-dimensional objects such as ropes or cables, with an emphasis of planning to tie knots. We propose TWISTED: Tying With Inverse model and Search in Topological space Excluding Demos, a hierarchical planning approach which, at the high level, uses ideas from knot-theory to plan a sequence of rope configurations, while at the low level uses a neural-network inverse model to move between the configurations in the high-level plan. To train the neural network, we propose a self-supervised approach, where we learn from random movements of the rope. To focus the random movements on interesting configurations, such as knots, we propose a non-uniform sampling method tailored for this domain. In a simulation, we show that our approach can plan significantly faster and more accurately than baselines. We also show that our plans are robust to parameter changes in the physical simulation, suggesting future applications via sim2real.

1 Introduction

Deformable object manipulation is important for many applications, such as manufacturing and robotic surgery. In particular, manipulating 1-dimensional (1D) objects such as ropes, cables, and hoses, is a challenging and exciting research area that has drawn recent attention (Sundaresan et al. 2020; Van Den Berg et al. 2010; Yan et al. 2020a; Mayer et al. 2008; She et al. 2021; Schulman et al. 2016; Wu et al. 2020; Yu, Zhong, and Li 2022; Lim et al. 2022; Chi et al. 2022).

There are several challenges to 1D object manipulation. Representing the state of the object is difficult, as unlike rigid objects, the object may have infinite degrees of freedom (Yan et al. 2021; Wang et al. 2019; Wi et al. 2022). Perception of a rope-like object is complex due to self-occlusions, the similarity between different rope parts, and self-loops (Sundaresan et al. 2020; Ganapathi et al. 2021; Grannen et al. 2021; Yen-Chen et al. 2022; Yan et al. 2020b; Ma, Hsu, and Lee 2022). Planning typically requires an effective abstraction of the states and the actions, which may be difficult to define (Lu, Chu, and Cheng 2016; Osa, Sugita, and Mitsuishi 2017), and low-level control for executing a plan must handle the flexibility and deformability of the rope – all non-trivial control problems (Yan et al. 2020a; Lu, Chu, and Cheng 2017; Jin, Wang, and Tomizuka 2019). To the best of our knowledge, a system that can generally manipulate 1D objects is beyond the capabilities of current technology.

Our focus in this work is on the planning component in 1D manipulation, particularly rope manipulation and knot tying. As mentioned above, planning for rope manipulation is non-trivial, as the state space may be large or infinite, and tasks like knot-tying essentially have a 'needle in a haystack' characteristic and require exhaustive exploration to reach desired states. Accordingly, most prior studies on rope manipulation relied on *human demonstrations* in lieu of automatic search (Yan et al. 2020a; Takamatsu et al. 2006; She et al. 2021; Morita et al. 2003; Wakamatsu et al. 2004; Schulman et al. 2016; Nair et al. 2017; Teng et al. 2022).

This paper tackles the problem of rope manipulation planning without any demonstrations. Our main contribution is a hierarchical search algorithm that exploits prior knowledge about knot-tying geometry for its high-level plan, with self-supervised learning of an inverse model for executing the low-level control, which we call Tying With Inverse model and Search in Topological space Excluding Demos – TWISTED. TWISTED is trained and evaluated in a physical simulation. We demonstrate, however, that our planning results are robust to variations of physical properties such as friction. Thus, we believe that our planning approach can be integrated with real-robot perception and control for a complete 1D manipulation system in the future. We demonstrate that TWISTED can tie various types of knots and can generalize to tie knots that were not seen during the training. To the best of our knowledge, this is the first demonstration of such a capability, which cannot be obtained by previous work that required a human demonstration of the knot to tie.

Finally, while TWISTED is tailored for knot tying by building on knot theory for high-level planning, our general methodology may be useful for other tasks where a well-established theory may inform the high-level characterization of the problem, while a learning-based method is used for low-level control.

2 Background

In our work, we build on knot-theory for high-level planning. In this section, we give a brief overview of knot the-

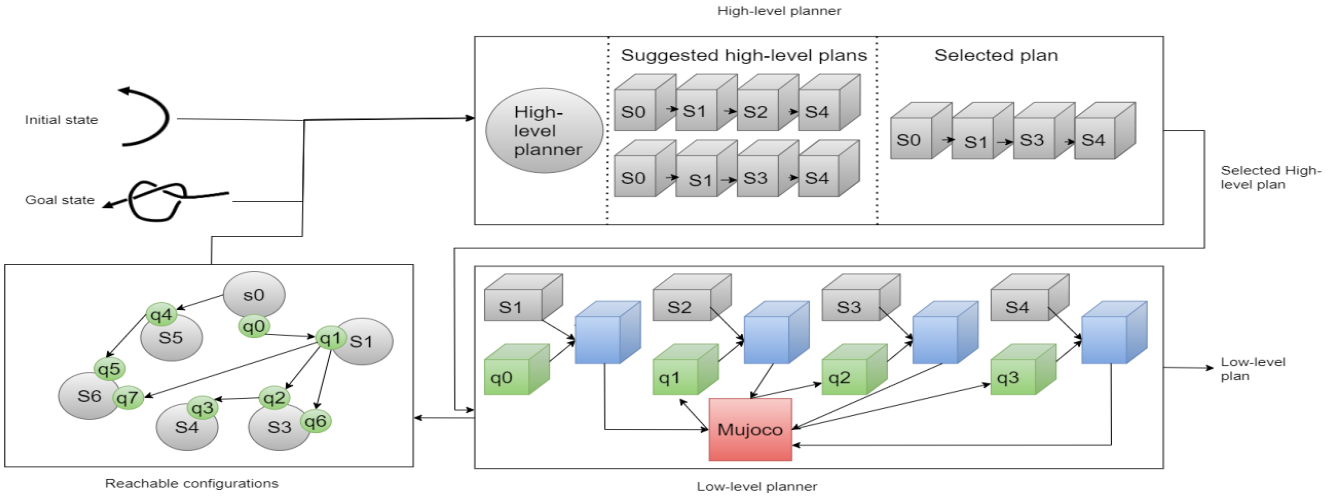

Figure 1: TWISTED: Given initial and goal topological states, we iteratively call a high-level planner to find plans to follow (top row). The plan uses an inverse model to transition between consecutive topological states (bottom right). When following a plan, new information is integrated into a graph of all known configurations, which seeds the high-level planner with initial states (bottom right). Gray boxes are high-level states, green boxes are low-level states, blue boxes represent the inverse model, and our environment in Mujoco is red.

ory. The most common way to solve problems like knot-tying, with a high-dimensional and continuous state and action spaces, and long-horizon planning is to separate it into a topological representation for high-level planning and a geometric representation for low-level control, which is solved using learning (Morita et al. 2003). We represent a rope as having L links, and denote by $q \in Q$ the rope configuration[1], with $Q \subseteq \mathbf{R}^{2L+5}$. A topological state, a discrete variable that buckets similar rope configurations, is denoted by $s \in S$. To denote the relation between $q \in Q$ and it's corresponding topological state $s \in S$, we use $\texttt{Top}: Q \to S$, or $s = \texttt{Top}(q)$. We follow previous works (Yan et al. 2020a), where the discrete topological representation for S is **P-data** (see appendix Section 7.1). Finally, we define the "complexity" of a topological state $s \in S$, according to the number of crosses (link intersections, see appendix Section 7.1) it represents. The complexity of tying a rope increases as the number of crosses in the rope increase. For ease of notation, we define $\texttt{Cross}: S \to \mathbf{Z}^{0+}$ that returns the number of crosses in a topological state.

Knot theory (Reidemeister 1983) suggests *Reidemeister moves* as actions that transition the rope between topological states. In this work, we will use them as high-level actions during the search. We denote the space of Reidemeister moves as A_R and $P_R : S \times A_R \to S$ as the transition function of topological states using Reidemeister moves. Reidemeister (Reidemeister 1983) proved that between any two topological states $s, s' \in S$, there exists a sequence of actions that starts in s and ends in s', namely, $\exists a_0, \ldots, a_k \in A_R$ s.t. $S' = P_R(\ldots P_R(P_R(s, a_0), a_1) \ldots, a_k)$. The Reidemeister moves are defined as:

- The Reidemeister I (R1) moves one segment to create a new loop.
- The Reidemeister II (R2) pulls the middle segment and creates a new intersection with opposite signs.
- The Cross (C) creates a new intersection between two segments.

Examples of those actions can be visualized in Figures 3, 4 and 5

Considering the knot-tying problem as a trajectory over topological states with Reidemeister moves as actions, translates the original problem of directly manipulating rope configurations to a problem of a shorter horizon and a "lower" branching factor. This approach has been adopted in different algorithms (Yan et al. 2020a; Takamatsu et al. 2006; She et al. 2021; Morita et al. 2003; Wakamatsu et al. 2004).

Finally, it is useful to define the **topological motion primitives** action space (Yan et al. 2020a). When manipulating a rope with L links, an action is a curve $c \in C$, parameterized by the link to grab $l \in [1, L]^2$, an endpoint in (x, y) (in the workspace), and the maximal height $z_{\max}$. We denote the transition function for curves applied on configurations by $f : Q \times C \to Q$. Yan et al. (2020a) observed that the space of curves C approximates well all the possible Reidemeister moves available from a given topological state. For this reason, in this work, we follow Yan et al. (2020a), and plan using Reidemeister moves while manipulating the rope with curves.

[1]The first seven coordinates describe the global position of the middle rope link (position (x, y, z) and quaternion representation for the rotation), and the remaining $L - 1$ joints are each described by yaw and pitch values.

[2]We associate a fixed point for every link

3 Related work

Manipulating deformable objects, such as ropes, presents varying degrees of difficulty (Matas, James, and Davison 2018). Rope manipulation involves changing the shape of a rope while untying knots involves transitioning the rope from a tied configuration to a relaxed state. The difficulty in manipulating ropes arises from the limited number of actions to change the rope's topological state. Unlike knot tying, where different knots can be targeted, in rope untying, there is only one goal state: a knot-free state. Additionally, any disturbance during the knot-tying process may cause the knot to be released and the rope to return to its original state. Furthermore, the number of goals increases exponentially with the number of crosses in the rope, adding to the difficulty of knot tying compared to other manipulation tasks such as knot-untying. Next, we present related work in each of these areas.

3.1 Rope Manipulation

Current research in rope manipulation has proposed various algorithms, with a focus on either learning from human demonstrations (Van Den Berg et al. 2010; Schulman et al. 2016) or solving short-term plans through pick-and-place actions (Wu et al. 2020; Teng et al. 2022). Conversely, in our work, we tackle long-term planning, such as knot tying, without demonstrations.

Ropes dynamics are challenging to learn, compared to tasks that manipulate rigid objects. To tackle this, previous methods used self-supervised learning in order to overcome the need for human involvement in the learning process (Chi et al. 2022; Nair et al. 2017; Yan et al. 2020b). In our work, we trained a supervised neural network using self-supervised collected data without manual involvement.

Finally, some previous works attempted to learn rope manipulation using reinforcement learning (RL) methods (Lin et al. 2021; Han, Paul, and Matsubara 2017; Deng et al. 2022). However, as we show, knot-tying is a complicated task that out-of-the-box RL algorithms cannot solve (see Section 5).

3.2 Knot Un-Tying

A related but different task is un-tying ropes, where the agent needs to release a knotted rope until the rope has no crosses. Recently many works tackled the knot un-tying problem (Viswanath et al. 2021; Shivakumar et al. 2022; Viswanath et al. 2022; Grannen et al. 2021; Sundaresan et al. 2021). Some (Viswanath et al. 2021; Shivakumar et al. 2022) used graph-based algorithms, some ideas from uncertainty quantification (Shivakumar et al. 2022), but a crucial component in all these works is that they operate over a high-level abstraction both in the state and action spaces. In our work, we similarly use abstraction to make efficient use of prior knowledge about ropes, but one significant factor differentiating between tying and untying is the goal representation. In rope-untying, the task is defined as a single goal, i.e., the rope in an untangled state. However, in our case, the agent could be asked to tie many knots, which completely changes the problem.

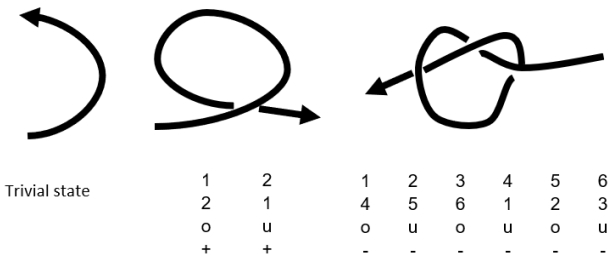

Figure 2: P-data Topological state representation: each column corresponds to an intersection along the rope of L links. Row one is ordered from 1 to L in ascending order. Row two, for each intersection, defines the other link in the intersection. The label "o"/"u" classifies the vertical arrangement at each intersection (over or under). Finally the last row identifies the "sign" – see appendix Section 7.1. E.g. in the center state representation link 1 is *over* link 2 with a "+" sign.

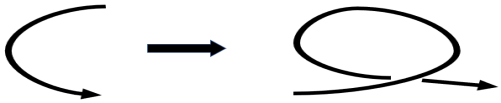

Figure 3: Topological action - Reidemeister one

3.3 Knot Tying

Knot tying has received much attention recently (Schulman et al. 2016; Sundaresan et al. 2020; Yan et al. 2020a). One of the main issues in knot tying is the size of the search space required to tie a knot. To handle this problem, some previous methods opted to use demonstrations (Yan et al. 2020a; Schulman et al. 2016), while others restricted the horizon of the task by focusing on short sequences (Sundaresan et al. 2020). In this work, we try a different approach, instead of learning from demonstrations, we try to learn directly from random actions, and as we later show, we do not restrict the horizon of the problem yet are able to achieve solutions for knots of three and even four crosses.

3.4 Task and Motion Planning

Task and Motion Planning (TAMP) decomposes complex manipulation problems into two distinct processes: high-level decision-making and motion planning (Dornhege et al. 2009; Dantam et al. 2016). TAMP is commonly used in pick and place tasks with rigid objects (Braun et al. 2022; Garrett et al. 2021). More recently, other works tried to use learning to accelerate the search (Kim et al. 2019; Paxton et al. 2017; Driess, Ha, and Toussaint 2020).

Our task is similar in many aspects but has two main differences from TAMP, (1) we use self-supervised learning to learn an inverse model that will prioritize low-level actions of other actions, and (2) our high-level planning includes states and actions from knot theory. Because of those differences, TAMP algorithms are not directly applicable to our work. For casting our problem with notations similar to TAMP, refer to appendix Section 7.2.

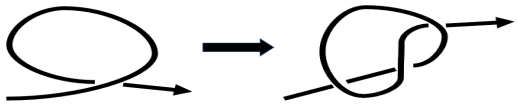

Figure 4: Topological action - Reidemeister two

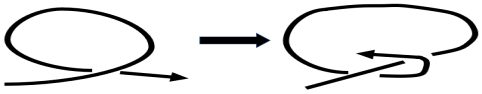

Figure 5: Topological action - Cross

4 Method

In this section we describe the components that comprise our solution – TWISTED. We start with the description of the simulated environment in Section 4.1, we follow with the description of the algorithmic components in Sections 4.2, 4.3, and 4.4, and finally describe our data collection methodology 4.5. See Figure 1 for an overview.

4.1 Simulated Environment

The environment we used to learn and test TWISTED was created by the free, open-source simulation environment of Mujoco (Todorov, Erez, and Tassa 2012). It includes the default rope of Mujoco and the end-effector moving the rope. We used a free-moving end-effector to focus on the complexity of tying knots, ignoring the additional complexity of controlling a robot manipulator[3]. It is crucial to mention that during planning, we check actions in the simulation itself, meaning that both evaluation and search use the same Mujoco environment (i.e., the search acts with a perfect world model). Our Mujoco environment is the concrete implementation of $f(q, c)$ (Section 2).

4.2 TWISTED

Our algorithm TWISTED, is best summarized as an iterative algorithm that repeats three steps: (1) starts searching from a known *reachable* configuration, (2) plans a trajectory whose states are in S, and actions in A_R in high-level, and (3) uses a low-level planner to follow subsequent states in the selected high-level plan. The iterative process of TWISTED repeats until the goal is reached (success) or a pre-specified timeout expires (terminating in failure). See Algorithm 1.

Data structures: we maintain two data structures, a tree of known reachable configurations, and a set of high-level plans.

The Known reachable configurations, is a tree T whose vertices are configurations $q \in Q$ with their corresponding topological states $\texttt{Top}(q) \in S$ and the edges are low-level actions $c \in C$. Initially, this tree contains only a root node - the rope's initial configuration q_{init} and its topological state $s_{init} = \texttt{Top}(q_{init})$.

[3]Although non-trivial, we expect that common motion planning solutions could be utilized in order to bridge the gap from a free-moving end-effector to a complete robotic manipulator.

We also maintain a list of *high level plans*, $\mathbf{P} = \{P_i = (s_0, s_1, \dots s_{l_i} = s_g)\}_i$ from *currently reachable topological states* (see Section 4.3). When a topological state s' is discovered for the first time by the low-level planner (Section 4.3), we run the high-level planner from s' and store the results into $\mathbf{P}$.

Plan selection: At the start of each iteration, we need to select a plan to execute from $\mathbf{P}$ and a configuration to start executing the plan from. One naive heuristic is to select a random configuration from the reachable configurations. However, due to the sparsity of the knot-tying problem (see 4.5), we reach exponentially fewer configurations with a higher number of crosses compared to configurations with a low number of crosses (since the number of possible configurations are exponential in the number of crosses). To avoid this less-efficient selection scheme, we seek a more sophisticated approach that promotes configurations corresponding to topological states with higher crosses. Thus our selection process is composed of three sub-procedures:

- $SelectTopologicalState()$: identifies the reachable topological states in the graph that have a high-level plan to the goal. Returns one such topological state s. Here, we use two heuristics, *random*, which returns a random topological state, and *prioritize-crosses*, which gives a probability proportional to state s with $1 + \texttt{Cross}(s)$ (preferring topological states with more crosses). The motivation is to search deeper than the *random* heuristic.
- $SelectPlan(s)$: a high-level plan (sequence of topological states) $P = s, s_1, \dots s_l = g$ is randomly selected from all the high-level plans that start in s.
- $SelectConfiguration(s)$: a configuration q is randomly selected from all configurations belonging to s.

Recall that our objective was to increase the frequency in which topological states of higher complexity are utilized as the initial state in the high-level plan. We note that even in the *random* heuristic for the $SelectTopologicalState$, we already prioritize such a selection, as sampling a random topological state induces a different distribution than sampling from all reachable configurations in T.

Plan execution: Next, in $FollowPlan$, we follow the high level plan $P = s_0, s_1, \dots s_l$, where $s_0 = s$ and $s_l = s_g$, starting in s, and incrementally try to reach $s_{i>0}$ until s_g is reached. To transition from s_i to s_{i+1}, we use the low-level planner (Section 4.3) that uses the learned inverse model (Section 4.4) to predict curves. The low-level planner applies multiple curves $\{c_j\}_j$ to the current configuration q_i. Let $q'_j = f(q_i, c_j)$, and $s'_j = \texttt{Top}(q'_j)$. If the transition for c_j reaches a configuration with more crosses, we add this information to T, even if $s'_j \neq s_{i+1}$, i.e., whenever $\texttt{Cross}(s'_j) \geq \texttt{Cross}(s_i)$.

Completeness guarantee: Finally, for completeness of the algorithm, after every iteration of TWISTED, with probability p (hyper-parameter, with a value of 0.05), we sample a random reachable configuration $q \in T$, execute $k = 100$ random actions and add them to T if the action transitions to a topological state with a greater or equal number of crosses (same conditions as in the "plan execution"

above). This ensures that given enough time, our algorithm is guaranteed to find a solution. We denote this sub-routine as $RandomExpand$

We next detail the high- and low-level planning components in TWISTED, and then expand on the inverse model.

Algorithm 1: TWISTED algorithm

Input q_{init} Initial configuration state and s_g topological goal state

Output Low-level plan if found

1: $init : T, \mathbf{P}$ ▷ see **data structures**
2: $s_{init} = Top(q_{init})$
3: populate $\mathbf{P}$ with plans from s_{init}
4: **while** Not timeout **do**
5: $s_{selected} = SelectTopologicalState()$
6: $P_{selected} = SelectPlan(s_{selected})$
7: $q_{selected} = SelectConfiguration(s_{selected})$
8: $PlanFound = FollowPlan(q_{selected}, P_{selected})$
9: **if** $PlanFound$ **then**
10: $ReturnPlan$
11: **else**
12: $RandomExpand()$
13: **end if**
14: **end while**

4.3 Planning and Search

TWISTED is composed of two levels of planning, high-level and low-level planning, that are called as sub-procedures by the algorithm. We now describe the two planners with their states and actions.

`High-level planner`: The states in the high-level planner are the topological states, S, and the actions are Reidemeister moves A_R. The high-level planner is used as a sub-procedure to find a path from $s \in S$ (not necessarily $s_{initial}$) to $s_g \in S$. We use BFS where valid Reidemeister moves are the edges, which prunes nodes $s' \in S$ with $\texttt{Cross}(s') \geq \texttt{Cross}(s_g)$. The result of the BFS is a set (possibly empty) of paths that start in s and terminate in s_g.

`Low-level Planner`: The objective of the low-level planner is to search for a curve that transitions between two consecutive topological states in the high-level plan we currently follow. Formally, its state space is Q, and action space is C, and the objective is to move from the current topological state $s \in S$ with configuration q ($s = \texttt{Top}(q)$), to the topological state $s' \in S$. To successfully find a curve that achieves that without wasting too much computing, we utilize our inverse model (Section 4.4) and generate $K = 6$ curves $\{c_i \in C\}^K$. If any of the newly found topological states are s', we return success (if more than one action succeeds we use the first one found), and the plan execution will try to move to the next topological state in the high-level path. Otherwise, we return failure, and the iterative process of TWISTED will repeat (starting in line 4 in Algorithm 1).

4.4 Inverse model

An action generator is crucial in knot-tying as the proportion of curves that transition the rope to a given topological state could be extremely small. This makes it unlikely that a small set of randomly selected curves could be found to satisfy the required transition between topological states. Thus, we have trained an inverse model to generate action candidates that are likely to satisfy the required transition (see Figure 6).

Our inverse model is an auto-regressive model (Gregor et al. 2014) that creates the curve element-by-element. The order of prediction elements is as follows: link to pickup (categorical), the height of the curve $z_{\max}$ (continuous), destination X position (continuous), and destination Y position (continuous). Categorical outputs are modeled as a multinomial distribution, and the continuous outputs are modeled with a Normal distribution. Every such input is predicted with an *independent* sub-network, whose inputs are (1) the current configuration q, (2) the current (x, y, z) coordinates of each of the L links, (3) **next** topological state s', and (4) all the elements before the current element (e.g. z_{max} gets the link index as input).

Training: we collected data generated from *random actions* to train the inverse model (see Section4.5). The data D contains transitions (q, s, s'), s and s' are current and following topological states, and q is the current configuration. In addition, each transition is labeled by $c \in C$. Since data collection is time-consuming, we follow previous work of Yan et al. (2020a) and apply the Mirror and Reverse augmentations to our data.[4]

Inference: during inference, we follow the standard ancestral sampling scheme for auto-regressive models (Gregor et al. 2014); we predict a distribution for every element, sample from it, and feed the result to the next predicted element in the sequence.

4.5 Data collection

To train the inverse model, we must collect data that represent movements typically encountered when tying knots. The problem, however, is that without a controller that knows how to tie knots, nor human demonstrations, it is not clear how to collect such data. In particular, we find that applying random actions to the rope typically does not lead to knot-like configurations. In this section, we detail an alternative approach.

Initially, we tried to collect rollouts simply by executing random walks in our simulator. However, in doing so, we found a very low number of topological states with three crosses (only 27 states per CPU core *per hour*). We therefore designed a collection scheme that resets the environment (see below). Using our scheme, we collected 537 successful transitions per hour per CPU core. We used that to collect a data set of 1,670,000 data points.

Collection process: We maintain a set of configurations we have already seen during data collection and their respective number of crosses, i.e. $D_Q = \{(q_i, \texttt{Cross}(\texttt{Top}(q_i))\}_i$. For every data collection iteration t, we load the simulation with a configuration sampled from D_Q, q_t, take a random curve c_t, and reach a new configuration q_{t+1} with a topolog-

[4] In Yan et al. (2020a) these augmentations were applied over manual demonstrations. In our work, we apply them on randomly collected data.

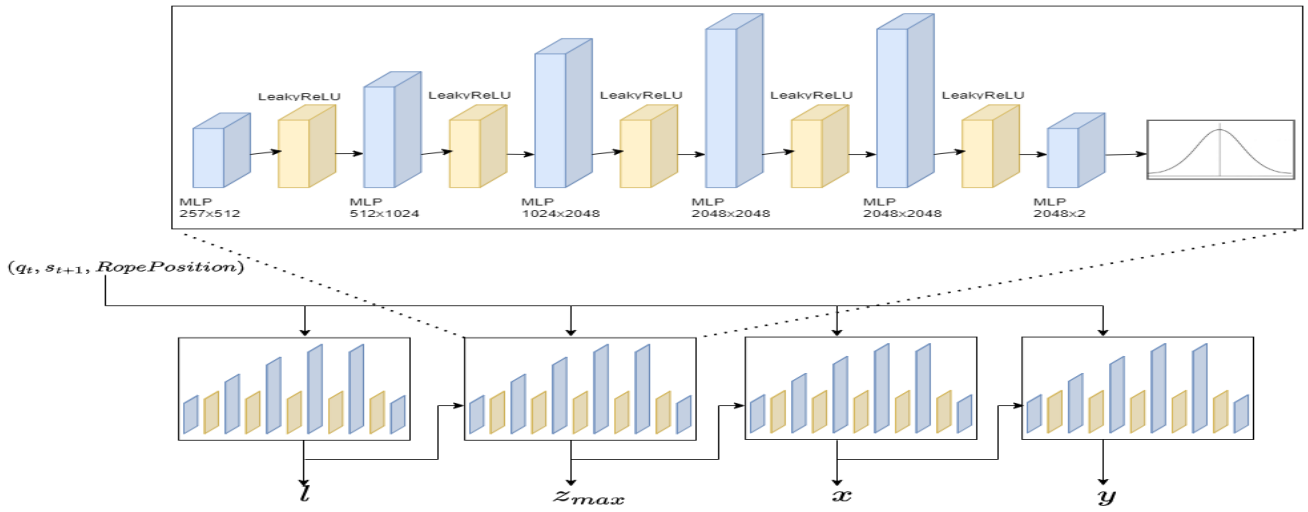

Figure 6: Inverse model - Auto-regressive Stochastic Network. The network predicts an action in an auto-regressive manner: first is predicts the link index $l \in [1, L]$, then the height of the curve z_{max}, finally it predicts the x and y coordinates of the curve. All predictions are stochastic (Multinominal for link index, and Gaussian otherwise). Besides the previous elements, the input of each element includes the current configuration q_t, the next topological state s_{t+1}, and the link positions of all the rope links. The weights of the sub-components are not shared.

ical state s_{t+1}. If the number of crosses in $Cross(s_{t+1}) > Cross(s_t)$ the transition $(q_t, s_t, c_t, q_{t+1}, s_{t+1})$ is added to D, and the configuration q_{t+1} is added to D_Q.

5 Experiments

In the experiments, we want to answer the following items: (1) How sensitive is knot-tying planning to the action space, and is a continuous action space necessary? (2) Comparison of TWISTED with baselines (3) How sensitive is TWISTED to changes in the physical simulation? (4) How well does TWISTED generalize to unseen knots?

5.1 What makes knot-tying difficult?

One difficulty of our knot-tying problem is that it requires very accurate actions to solve. To demonstrate this, we verify that even a fine discretization of the problem leads to significantly different outcomes.

In this experiment, we measure sensitivity to discritization of curves, i.e. test if using a discretized curve reaches the same topological state as the next topological state (obtained by executing the original non-discretized curve). Formally, given a curve, $c = (i, z_{\max}, x, y) \in C$, which includes three continuous elements ($z_{\max}$, x, and y), we convert it to a discrete curve where each element is rounded. $z_{\max}$, is discritized in steps of 0.001, and x and y in steps of 0.01. We denote this discretized curve c_{dis}. Notice that the size of the discretized curve space is $21 \times 70 \times 100 \times 100 = 14,700,000$, already rather large. We measure the accuracy of the resulting topological states, namely $Top(f(q, c)) = Top(f(q, c_{dis}))$. If the accuracy is high, there is little difference in discretizing the action space, which would mean that this problem could be solved using off-the-shelf discrete planners (Chaslot et al. 2008; Finnsson and Björnsson 2008).

We ran over 600k data points of transitions from topological states of two crosses to topological states of three crosses, and got an accuracy of 82%. These results show that the problem dynamics are not smooth, and small changes in the actions can lead you to different topological states. As the space of available actions is already rather large (larger action spaces would make planning even more difficult) we conclude that discretization of the action space is not a suitable approach for our knot-tying problem.

5.2 Success Rate of Different Algorithms

In this experiment, we compare the following different algorithms, including TWISTED and its ablations, to solve the knot-tying problem.

- "low level only" - We modify TWISTED to **not use** any high-level information. Essentially, using random search over configurations with curves as actions. As there is no notion of topological states, there is no way to use the inverse model here. Instead, we sample *random curves*. It is important to notice that for this baseline, the search does not get feedback along the trajectory (in TWISTED, we do, for instance, count the number of crosses). We use this baseline to demonstrate how crucial high-level information is for knot tying.
- "Low+high level" - We modified TWISTED not to use the inverse model. Instead, we sample random actions replacing those suggested by the inverse model. Unlike the previous baseline, we do try to follow a high-level plan here. This baseline demonstrates the trade-off between intensive but more accurate action prediction (in-

verse model) vs. an approach of guessing many random actions and seeing if any suffice.

- "SAC+HER" - In this baseline, we learn a stochastic policy using the Soft Actor-Critic (SAC)(Haarnoja et al. 2018), with Hindsight Experience Replay (HER)(Andrychowicz et al. 2017), and after training we replace our inverse model with the policy. The objective of this baseline is to establish the performance of model-free RL methods and the challenging problem of knot-tying.
- "TWISTED, RND" - full TWISTED algorithm using the *random* heuristic for $SelectTopologicalState$.
- "TWISTED, CRS" - full TWISTED algorithm using the *prioritize-crosses* heuristic for $SelectTopologicalState$.

Evaluation protocol: To evaluate the performance on different difficulty levels, we split our collected data D (Section 4.5) into three levels: `easy`, `medium`, and `hard`. To classify the problems (topological goal states), we counted the frequency of each seen topological state. `Easy`, `medium` and `hard` are the 33%, 66%, and 100% percentiles appearing the most in the data. From every class of problems, we sampled ten representatives.

Results: None of the algorithms solve `medium` or `hard` in the time limit of 1800 seconds, demonstrating the hardness of the knot-tying problem. Figure 7 shows the number of solved tasks vs. the running time for `easy` problems.

First, observe that "low level only" (essentially trying random curves from random reachable configurations) is barely able to solve two out of the ten problems. This validates our earlier hypothesis in Section 4.5 that the problem is too sparse to solve without prior knowledge of the problem structure (such as our high-level search). Surprisingly, the model-free RL baseline is barely better than the random search. We observed that during training (i.e., when the policy was not used inside the low-level planner), it did manage to consistently solve all 1-cross problems, but already for 2-cross problems success rate was near zero. This suggests that knot-tying is a hard task to learn end-to-end without proper domain knowledge. We hypothesize that the main reason this baseline fails is due to the discrete nature of the topological states – in such cases algorithms cannot generalize between "similar" states because as categorical variables, there is no notion of similarity, only the relation of equality. Even a well utilized exploration method such as HER does little to mitigate this problem, because it can only reinforce patterns for goals we reach, and as we saw when acting randomly, like RL agents do at the beginning of training, there is little chance to advance to topological states with many crosses (see Section 4.5).

Finally, regarding the baselines, we see that because "low + high level" is so inferior to the full TWISTED versions, the inverse model is a valuable component of our full solution.

Comparing "TWISTED, RND" and "TWISTED, CRS", we observe that the results are not clear cut. To identify the better model, we sampled 15 additional `easy` goals to get a statically significant separation on which is the better variant of TWISTED. The "TWISTED, CRS" solved a total of $24/25$, and the "TWISTED, RND" solved only $19/25$. Under a Z-test the "TWISTED, CRS" has a statistical significance of being better than the "TWISTED, RND" (using α-level of 0.05), showing that planning deeper and utilizing prior knowledge (number of crosses) is preferable. Therefore, in our next experiments, we only use the "TWISTED, CRS" as the selected version of TWISTED.

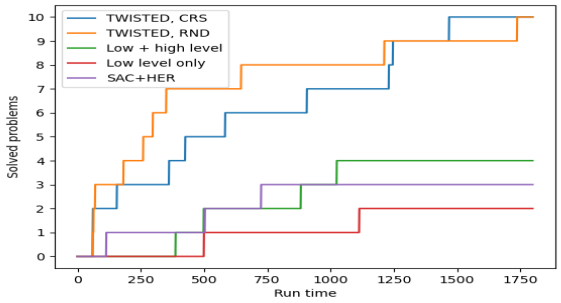

Figure 7: Success rate of different algorithms to knot tying, X axis is how much time the planner had to solve it and Y axis is success knots, total 30 instances

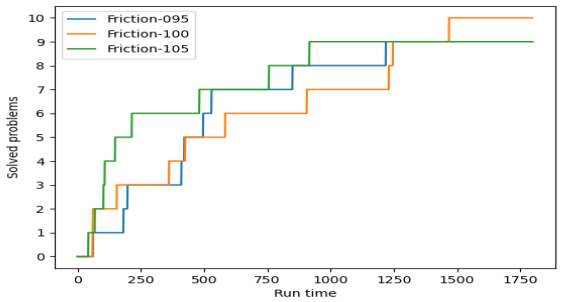

Figure 8: Success rate of different ropes with different friction to knot tying, X axis is how much time the planner had to solve it and Y axis is success knots, total 30 instances

5.3 Sensitivity Analysis

Our experiments were done in a simulation, where the planning computation and the evaluation environment are the same. Clearly, this does not apply in the real world. To motivate the usage of TWISTED in real world applications we test what happens if the model of the world in the planning computation *is mismatched with the evaluation environment*.

In our experiments, we focus on the friction coefficient. First, we validate that friction indeed significantly impacts rope tying. To measure this, we compare the next topoligical state observed when applying the same action from the same low-level state, under different friction coefficients. We evaluated over 600,000 actions, and only 82% curves had the same topological state as the original friction value.

Next, we evaluate the performance of TWISTED trained with a single friction coefficient on simulated environments with different frictions of the rope. **Variants:** 100% friction denotes the default Mujoco friction, and the one we use for

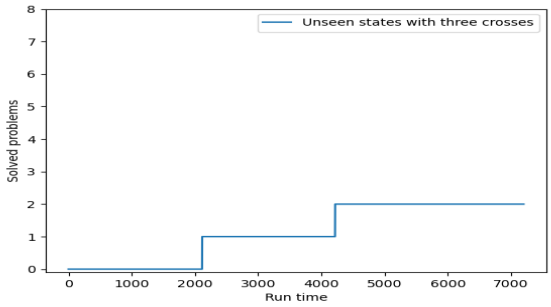

Figure 9: Success rate of unseen states with three crosses, X axis is how much time the planner had to solve it and Y axis is success knots, total eight problems instances

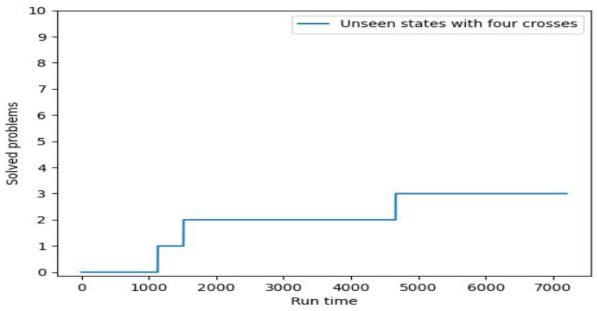

Figure 10: Success rate of unseen states with four crosses, X axis is how much time the planner had to solve it and Y axis is success knots, total ten problems instances

TWISTED. The 95% and 105% variants, denote decreasing and increasing the friction by 5%.

Results: We can see that the performance of TWISTED is well-maintained with the different friction coefficients (Figure 8). This asserts that TWISTED can handle some variations in the environment's physics such as friction (even though the resulting trajectories might be different than the original ones).

5.4 Generalization to unseen topological states

The number of available topological states for states with 3 or 4 crosses is above 500 and almost 8000 correspondingly. Naturally our data does not contains all of them because it is hard to sample topological states of higher crosses (see Section 4.5 for analysis). Thus, we require our algorithm to handle unseen topological states. We evaluate whether TWISTED can tie knots where the goal state was not represented in D. For this, we take topological states with *three* crosses not seen in D, and topological states with *four* crosses. In these out-of-distribution cases we expect the inverse model to contribute less than in well represented states, and we expect that the planning components in TWISTED will compensate for this distribution shift. For this reason, we extend search time by a factor of $4\times$.

Results are shown in figures 9 and 10. Figure 9 shows that TWISTED solved two out of eight problems with unseen three cross states. Those states are harder to reach because they were never seen in D. In figure 10 we see that TWISTED solved three out of ten problems with unseen states with four crosses. We remind that our data contain only one, two, and three crosses, and therefore these results show that TWISTED is not only memorizing the data, but can generalize to some degree to unseen goal states.

6 Limitations and Outlook

We presented TWISTED – an hierarchical planning algorithm for knot tying, that relies on knot-theory and a learned inverse model to automatically solve problems that previously required access to human demonstrations. TWISTED outperforms various baselines, including a model-free deep RL agent, and we demonstrated robustness to simulation parameters such as friction, and generalization to problems not seen during training (even to problems of greater complexity). We see this as an important step towards general 1D object manipulation.

Our work has several important limitations that need to be addressed in order to make it more practical and useful in real-world applications. First, simulation accelerates the training process but introduces a sim2real gap between the simulation and real-world performance. This gap should be tested on a real robot using real ropes. Furthermore, in this work we also simplified the problem; we control a free-moving end-effector instead of controlling a manipulator (which might make some curves unfeasible in some scenarios), and we get a perfect representation of the rope, where in reality we would need first to estimate one.

As our experiments demonstrate, TWISTED has shown better performance on more common data, but its performance decreases when trying to solve rare or unseen goals. This means that our method needs to be improved to perform well in a timely manner on all types of knots. For instance, the data collection was done randomly, but collecting data from states where the system is less capable, could ultimately provide data of higher quality and improve the performance of our learned inverse model. Another option to handle this problem is to improve online search, for instance by utilizing bandit algorithms during high-level plan selection. The benefit of bandit algorithms is that they balance exploration versus exploitation, usually making search more efficient than hand-crafted search heuristics.

Finally, to the best of our knowledge, this is the first work that manages to tie knots from random data. An exciting area for improvement would be to utilize TWISTED as a demonstrations provider to generate "valuable" data for an off-policy RL algorithm, either by distilling the planner into a policy (Silver et al. 2016) or by combining RL with imitation learning (Nair et al. 2018). This could be the missing prior knowledge that RL methods lack for knot-tying tasks (cf. Section 5).

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.;

and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Braun, C. V.; Ortiz-Haro, J.; Toussaint, M.; and Oguz, O. S. 2022. RHH-LGP: Receding Horizon And Heuristics-Based Logic-Geometric Programming For Task And Motion Planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 13761–13768. IEEE.

Chaslot, G. M. J.; Winands, M. H.; Herik, H. J. v. d.; Uiterwijk, J. W.; and Bouzy, B. 2008. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(03): 343–357.

Chi, C.; Burchfiel, B.; Cousineau, E.; Feng, S.; and Song, S. 2022. Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects. *arXiv preprint arXiv:2203.00663*.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, 00052. Ann Arbor, MI, USA.

Deng, Y.; Xia, C.; Wang, X.; and Chen, L. 2022. Deep Reinforcement Learning Based on Local GNN for Goal-Conditioned Deformable Object Rearranging. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1131–1138. IEEE.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *2009 IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR 2009)*, 1–6. IEEE.

Driess, D.; Ha, J.-S.; and Toussaint, M. 2020. Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image. In *Proc. of Robotics: Science and Systems (R:SS)*.

Finnsson, H.; and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *Aaai*, volume 8, 259–264.

Ganapathi, A.; Sundaresan, P.; Thananjeyan, B.; Balakrishna, A.; Seita, D.; Grannen, J.; Hwang, M.; Hoque, R.; Gonzalez, J. E.; Jamali, N.; et al. 2021. Learning dense visual correspondences in simulation to smooth and fold real fabrics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 11515–11522. IEEE.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4: 265–293.

Grannen, J.; Sundaresan, P.; Thananjeyan, B.; Ichnowski, J.; Balakrishna, A.; Viswanath, V.; Laskey, M.; Gonzalez, J.; and Goldberg, K. 2021. Untangling Dense Knots by Learning Task-Relevant Keypoints. In *Conference on Robot Learning*, 782–800. PMLR.

Gregor, K.; Danihelka, I.; Mnih, A.; Blundell, C.; and Wierstra, D. 2014. Deep autoregressive networks. In *International Conference on Machine Learning*, 1242–1250. PMLR.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Han, H.; Paul, G.; and Matsubara, T. 2017. Model-based reinforcement learning approach for deformable linear object manipulation. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, 750–755. IEEE.

Jin, S.; Wang, C.; and Tomizuka, M. 2019. Robust deformation model approximation for robotic cable manipulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6586–6593. IEEE.

Kim, B.; Wang, Z.; Kaelbling, L. P.; and Lozano-Pérez, T. 2019. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 38(7): 793–812.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Lim, V.; Huang, H.; Chen, L. Y.; Wang, J.; Ichnowski, J.; Seita, D.; Laskey, M.; and Goldberg, K. 2022. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. In *2022 International Conference on Robotics and Automation (ICRA)*, 8282–8289. IEEE.

Lin, X.; Wang, Y.; Olkin, J.; and Held, D. 2021. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 432–448. PMLR.

Lu, B.; Chu, H. K.; and Cheng, L. 2016. Dynamic trajectory planning for robotic knot tying. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 180–185. IEEE.

Lu, B.; Chu, H. K.; and Cheng, L. 2017. Robotic knot tying through a spatial trajectory with a visual servoing system. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5710–5716. IEEE.

Ma, X.; Hsu, D.; and Lee, W. S. 2022. Learning latent graph dynamics for visual manipulation of deformable objects. In *2022 International Conference on Robotics and Automation (ICRA)*, 8266–8273. IEEE.

Matas, J.; James, S.; and Davison, A. J. 2018. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 734–743. PMLR.

Mayer, H.; Gomez, F.; Wierstra, D.; Nagy, I.; Knoll, A.; and Schmidhuber, J. 2008. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. *Advanced Robotics*, 22(13-14): 1521–1537.

Migimatsu, T.; and Bohg, J. 2022. Grounding predicates through actions. In *2022 International Conference on Robotics and Automation (ICRA)*, 3498–3504. IEEE.

Morita, T.; Takamatsu, J.; Ogawara, K.; Kimura, H.; and Ikeuchi, K. 2003. Knot planning from observation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 3, 3887–3892. IEEE.

Nair, A.; Chen, D.; Agrawal, P.; Isola, P.; Abbeel, P.; Malik, J.; and Levine, S. 2017. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE international conference on robotics and automation (ICRA)*, 2146–2153. IEEE.

Nair, A.; McGrew, B.; Andrychowicz, M.; Zaremba, W.; and Abbeel, P. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, 6292–6299. IEEE.

Osa, T.; Sugita, N.; and Mitsuishi, M. 2017. Online trajectory planning and force control for automation of surgical tasks. *IEEE Transactions on Automation Science and Engineering*, 15(2): 675–691.

Paxton, C.; Raman, V.; Hager, G. D.; and Kobilarov, M. 2017. Combining neural networks and tree search for task and motion planning in challenging environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6059–6066. IEEE.

Reidemeister, K. 1983. *Knot theory*. BCS Associates.

Schulman, J.; Ho, J.; Lee, C.; and Abbeel, P. 2016. Learning from demonstrations through the use of non-rigid registration. In *Robotics Research*, 339–354. Springer.

She, Y.; Wang, S.; Dong, S.; Sunil, N.; Rodriguez, A.; and Adelson, E. 2021. Cable manipulation with a tactile-reactive gripper. *The International Journal of Robotics Research*, 40(12-14): 1385–1401.

Shivakumar, K.; Viswanath, V.; Gu, A.; Avigal, Y.; Kerr, J.; Ichnowski, J.; Cheng, R.; Kollar, T.; and Goldberg, K. 2022. SGTM 2.0: Autonomously Untangling Long Cables using Interactive Perception. *arXiv preprint arXiv:2209.13706*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, 639–646. IEEE.

Sundaresan, P.; Grannen, J.; Thananjeyan, B.; Balakrishna, A.; Ichnowski, J.; Novoseller, E.; Hwang, M.; Laskey, M.; Gonzalez, J. E.; and Goldberg, K. 2021. Untangling dense non-planar knots by learning manipulation features and recovery policies. *arXiv preprint arXiv:2107.08942*.

Sundaresan, P.; Grannen, J.; Thananjeyan, B.; Balakrishna, A.; Laskey, M.; Stone, K.; Gonzalez, J. E.; and Goldberg, K. 2020. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 9411–9418. IEEE.

Takamatsu, J.; Morita, T.; Ogawara, K.; Kimura, H.; and Ikeuchi, K. 2006. Representation for knot-tying tasks. *IEEE Transactions on Robotics*, 22(1): 65–78.

Teng, Y.; Lu, H.; Li, Y.; Kamiya, T.; Nakatoh, Y.; Serikawa, S.; and Gao, P. 2022. Multidimensional Deformable Object Manipulation Based on DN-Transporter Networks. *IEEE Transactions on Intelligent Transportation Systems*.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 5026–5033. IEEE.

Van Den Berg, J.; Miller, S.; Duckworth, D.; Hu, H.; Wan, A.; Fu, X.-Y.; Goldberg, K.; and Abbeel, P. 2010. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *2010 IEEE International Conference on Robotics and Automation*, 2074–2081. IEEE.

Viswanath, V.; Grannen, J.; Sundaresan, P.; Thananjeyan, B.; Balakrishna, A.; Novoseller, E.; Ichnowski, J.; Laskey, M.; Gonzalez, J. E.; and Goldberg, K. 2021. Disentangling Dense Multi-Cable Knots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3731–3738. IEEE.

Viswanath, V.; Shivakumar, K.; Kerr, J.; Thananjeyan, B.; Novoseller, E.; Ichnowski, J.; Escontrela, A.; Laskey, M.; Gonzalez, J. E.; and Goldberg, K. 2022. Autonomously Untangling Long Cables. *arXiv preprint arXiv:2207.07813*.

Wakamatsu, H.; Tsumaya, A.; Arai, E.; and Hirai, S. 2004. Planning of one-handed knotting/raveling manipulation of linear objects. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 2, 1719–1725. IEEE.

Wang, A.; Kurutach, T.; Liu, K.; Abbeel, P.; and Tamar, A. 2019. Learning Robotic Manipulation through Visual Planning and Acting. In *Robotics: science and systems*.

Wi, Y.; Florence, P.; Zeng, A.; and Fazeli, N. 2022. Virdo: Visio-tactile implicit representations of deformable objects. In *2022 International Conference on Robotics and Automation (ICRA)*, 3583–3590. IEEE.

Wu, Y.; Yan, W.; Kurutach, T.; Pinto, L.; and Abbeel, P. 2020. Learning to Manipulate Deformable Objects without Demonstrations. In *16th Robotics: Science and Systems, RSS 2020*. MIT Press Journals.

Yan, M.; Li, G.; Zhu, Y.; and Bohg, J. 2020a. Learning topological motion primitives for knot planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9457–9464. IEEE.

Yan, M.; Zhu, Y.; Jin, N.; and Bohg, J. 2020b. Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE robotics and automation letters*, 5(2): 2372–2379.

Yan, W.; Vangipuram, A.; Abbeel, P.; and Pinto, L. 2021. Learning predictive representations for deformable objects using contrastive estimation. In *Conference on Robot Learning*, 564–574. PMLR.

Yen-Chen, L.; Florence, P.; Barron, J. T.; Lin, T.-Y.; Rodriguez, A.; and Isola, P. 2022. Nerf-supervision: Learning dense object descriptors from neural radiance fields. In *2022 International Conference on Robotics and Automation (ICRA)*, 6496–6503. IEEE.

Yu, M.; Zhong, H.; and Li, X. 2022. Shape control of deformable linear objects with offline and online learning of local linear deformation models. In *2022 International Conference on Robotics and Automation (ICRA)*, 1337–1343. IEEE.

7 Appendix

7.1 P-data, an abstract representation for rope states

A common way to abstract the state space in knot tying is using the **P-data** representation (Morita et al. 2003). The P-data representation translates a rope configuration to a matrix of discrete values that depends on the number of link intersections. The P-data algorithm stages are: (1) project the 3D rope onto a 2D on the horizontal plane. (2) Select rope direction by defining the head and tail of the rope. (3) Move from head to tail and count the number of intersections along the path, starting from 1 to N. Those intersections are also called crosses. Finally, (4) each intersection gets **over/under** value based on which segment is over the other in the height dimension and also gets a **sign** plus/minus. The sign defined as

$$sign = \frac{\overrightarrow{l}_{over} \times \overrightarrow{l}_{under}}{|\overrightarrow{l}_{over} \times \overrightarrow{l}_{under}|} \cdot \overrightarrow{e_z},$$

where e_z is the unit normal of the horizontal plane, and l_{over} and l_{under} are the two strands directional vectors. Examples of P-data of projected knots in Figure 2.

7.2 TAMP-like Problem Statement

In general, the objective of rope manipulation is to move the rope's initial configuration to a desired goal using a sequence of actions. However, finding a plan built from a sequence of actions on the rope configuration can take much time. Therefore, many algorithms use abstraction to reduce the state and action spaces. The abstraction is called high-level, and it reduces the plan length thus making the problem feasible (Konidaris, Kaelbling, and Lozano-Perez 2018). In our work, we will use the topological states using p-data and Reidemeister moves as our high level states and actions. The high-level plan is translated into concrete low-level actions. High-level planning needs to generate a plan fast to guide the low-level motion planner. The high-level planner uses abstract states and actions to reduce search time and allow agents to reason over long horizons by showing only the necessary information for high-level planning (Migimatsu and Bohg 2022). The high-level plan includes a sequence of abstract states and transitions between them using abstract actions. The low-level motion planner gets a high-level plan and searches for paths through controllable joints of the agent defined as the agent's configuration space. The formal connection between high-level and low-level define as existing a transition from high-level state s to high-level state s' if there is a low-level state q in s and q' in s' with a transition between them.

High-level planning state space is formally represented as a tuple $T = \langle S, S_0, S_G, A_{abs}, T_{abs}, C_{abs} \rangle$, where:

- S is a finite and non-empty set of states,
- $S_0 \in S$ is the initial state,
- $S_G \subseteq S$ is a non-empty set of goal states,
- $A_{abs}(s) \subseteq A_{abs}$ denotes the actions applicable in each state $s \in S$,
- $T_{abs} : S \times A_{abs} \rightarrow S$ is the state transition function, such that applying action a in state s leads to state s', and
- $C(s, a)_{abs}$ is the cost of performing action a in state s

Low-level planning state space is formally represented as a tuple $T = \langle Q, Q_0, Q_G, A_{motion}, F, T_{motion}, C_{motion} \rangle$, where:

- Q is a finite and non-empty set of configuration states,
- $Q_0 \in S$ is the initial configuration state,
- $Q_G \subseteq S$ is a non-empty set of goal configuration states,
- $A_{motion}(q) \subseteq A$ denotes the actions applicable in each state $q \in Q$,
- $F : Q \rightarrow \{0, 1\}$ defines if Q is a feasible state,
- $T_{motion} : Q \times A_{motion} \rightarrow Q$ is the state transition function, such that applying action a in state q leads to state q',
- $C_{motion}(q, a)$ is the cost of performing action a in state q

The feasible configuration space is a subset of Q that satisfies the constraint: $Q_F = \{q \in Q | F(q) = 1\}$. It is important to notice that each q from low-level has only one s, and s has many q in low-level. The same is valid for actions. The goal is to generate a low-level plan that the agent can execute to solve a given task. Doing that using a high-level plan on abstract representation and low-level motion planning guided by the high-level plan to generate a plan(Srivastava et al. 2014).

The proposed algorithm for knot tying using manipulation will require high and low levels of planning to achieve the desired outcome. The initial configuration of the rope will serve as the starting point for the transition function. In contrast, the topological state of the rope will define the goal to address the challenge of infinite possible configurations for each tied rope (Takamatsu et al. 2006). This problem statement is similar to the TAMP, where multiple layers of planning are needed to define the problem.