Motion Generation for Shaping Deformable Linear Objects with Contact Avoidance Using Differentiable Simulation*

Changjian Ying and Kimitoshi Yamazaki, *Member, IEEE*

***Abstract*— The manipulation of deformable linear objects (DLOs) by robots is challenging because of the complexity of modeling DLO dynamics. Although previous studies generally employed physical models and data-driven approaches to simulate DLO deformations, only a few studies have considered the contact of DLOs with the environment. In this study, we propose a framework integrating differentiable simulations with neural networks (NNs) to generate manipulation trajectories that avoid contact and achieve the goal shape. First, we implement a differentiable simulation to simulate the deformation and interaction of DLOs via position-based dynamics. Thereafter, we utilize the backpropagation of losses from the differentiable simulation to optimize the parameters affecting the deformation of DLOs in the simulator and explore an ideal manipulation trajectory for the task via an NN controller. The simulation and real-world experimental results reveal that the proposed method can generate valid manipulation trajectories from offline learning, which can also function well in real-world applications using the optimized parameters.**

I. Introduction

Our daily life and manufacturing processes involve interactions with various deformable linear objects (DLOs), such as ropes, cables, hoses, and surgical sutures. These objects are generally one-dimensional and flexible; they can undergo elastic deformations, such as stretching, bending, and twisting, under the action of an external force. These properties complicate the accurate simulation of the shapes of DLOs during their manipulation, thereby impeding the automation of their manipulation and control and representing a promising research topic in the robotics and automation community [1]. The DLO manipulation task can be classified into two categories [2]: one involves the explicit shape-control tasks (the goal) are aimed at controlling the shape of objects to desired geometric configurations. These tasks are often undertaken in wire harness routing. The other involves implicit shape-control tasks, which are not aimed at achieving special geometric configurations; they are aimed at achieving several high-level semantic comprehensions, e.g., knotting and unknotting ropes, as well as inserting thin strings. Here, we focused on the former. This is because the DLO-manipulation procedure (Fig. 1), e.g., the assembling of cars or aircraft, generally requires fixtures. Before inserting the DLOs into such fixtures (implicit shape control), an essential step comprises manipulating such DLOs from initial to desired shapes (explicit shape control).

*This work was supported by JST SPRING, Grant Number JPMJSP2144 (Shinshu University).

Changjian Ying is with the Graduate School of Science and Technology, Shinshu University, Nagano, 3808553 Japan. (phone: 026-269-5755; e-mail: 21hs252f@shinshu-u.ac.jp)

Kimitoshi Yamazaki is with the Faculty of Engineering, Shinshu University, Nagano, 3808553 Japan. (kyamazaki@shinshu-u.ac.jp)

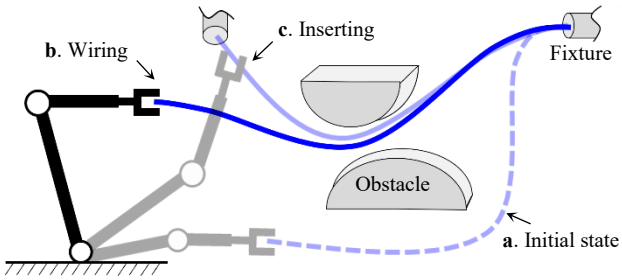

Fig.1. Schematics of a DLO mount procedure. **a** is the initial state of DLO, **b** manipulation the DLO from the initial state into a desired shape (the objective of this study and an essential aspect of the whole procedure), and **c** inserting the other end of DLO into the fixture.

The aim of this study was to construct a framework for automating the manipulation of the shapes of DLOs. Based on steps a to b (Fig. 1), we assumed a scenario in which DLO was fixed at one end, allowed to move freely at the other end, and subjected to the force of gravity by hanging naturally. We attempted to manipulate DLO into a goal shape to facilitate the next step. However, owing to the limitations of the environmental configuration of the workspace, e.g., obstacles, DLO will contact with these obstacles during the manipulation regardless of the influences of such obstacles, resulting in a final DLO shape that is significantly different from the goal, as well as risking the breakage of DLO in a worse scenario, thereby impeding the entire procedure. Therefore, the challenge becomes exploring appropriate contact-avoidance strategies for achieving the DLO goal shape.

To avoid contact and achieve the goal shape, previous studies explored various methods. Nozaki et al. [3] explored a mass-spring model (MSM)-based simulation strategy to simulate the shape of a moving cable; they formulated a control policy to avoid contact between the cable and environmental configuration by detecting the collisions from particles and obstacles. However, the utilized deformation parameters of the cables in the simulation required manual adjustments, and the control policy required modifications for different configurations. McConachie et al. [4] explored a neural network (NN)-based classifier to predict the state-action pairs of a rope and plan the transition action for ensuring it avoided nearby obstacles. However, to impart the classifier with the ability to correctly predict collision-avoidance actions, a large number of datasets must be prepared and labeled; this represents a very expensive process.

Based on the foregoing, we proposed a method, which seamlessly integrates simulation and NNs, to generate a trajectory that can avoid contacts and smoothly achieve the goal shape. We extended a position-based DLO simulation to ensure that the simulation could simultaneously simulate the interaction with the environment and be end-to-end

differentiable. By exploiting the auto-differentiation capability, the gradients of the parameters in the simulation could be readily obtained; they can be used to automatically adjust the parameters to facilitate near-real-world simulation results. Subsequently, we incorporated an NN controller in the differentiable simulation, which empowered NN to learn from the simulation via the trial-and-error approach, as well as automatically update the trajectory until the goal was achieved. The contributions of this study are as follows:

- We extended a PBD simulation to be end-to-end differentiable while simulating interactions with the environment and identified the simulation parameters via a gradient-based optimization method.
- We constructed an NN controller that can learn from the simulation to generate a manipulation trajectory, which enables DLO to avoid contact with the environment configuration and reach its goal shape.
- We confirmed the proposed method for generating manipulation trajectories by performing simulations and experiments on an actual robot.

The remainder of this paper is structured as follows: Section II presents the extant studies (literature review). In Section III, we describe the problem setting of this study, the existing issues, and the method overview. In Section IV, we formulate each part of the method in detail. In Section V, we present the validation experiments via simulation and real-world applications, after which we present the conclusions in Section VI.

II. Related works

To achieve automated DLO shape-control tasks using robots, model-based approaches have been adopted for a long time. MSM [5] has been widely employed for DLO modeling owing to its facile implementation. Several studies have attempted DLO simulations based on the finite element model (FEM) [6-7], followed by performing shape control on the simulated DLO based on the simulation results. Additionally, the extant studies on DLO shape control adopted various approaches based on discrete elastic rods [8], Fourier-series [9], and visual servo [10] models. However, only a few of these studies considered the influence of the contact between DLOs and the environment configuration during manipulation; moreover, the parameters of these models, which determine the deformation of DLOs, must be manually tuned until the precision requirement is obtained to successfully proceed from simulation to real-world application, and this is time-consuming.

Further, previous studies often applied data-driven approaches for DLO shape control. The core of this category of approaches comprises the utilization of NNs to learn forward kinematics models of DLOs from acquired data rather than physical models, i.e., mapping from the current state of DLOs to inputting actions. Thereafter, shape control can be achieved by solving the inverse kinematics. Yu et al. [11] used the radial basis function NN to model the mapping from the current state to the current local linear deformation model. Yang et al. [12] combined the interaction network and bidirectional Long Short Term Memory network to further learn the dynamic DLO model; they used the learned model in a model predictive controller to achieve the goal shape and improved the framework for sample-efficient, online dynamics model learning through trial-and-error interactions [13]. To improve the data-acquisition efficiency, Huo et al. [14] proposed an approach for training a network from a synthetic dataset to encode key points on DLOs. However, a large amount of required data, as well as the artificial design of the data-labeling method, account for the primary limitation of these approaches. Additionally, reinforcement learning (RL) [15-16] is often applied to the manipulation of deformable objects. However, RL methods are limited by the challenge of being transferred from simulation to real-world scenarios.

Differentiable simulation has been recently applied to achieve end-to-end motion control and parameter identification of deformable objects. Liang et al. [17] constructed a differentiable cloth simulator that can be embedded as a layer in deep NNs. Hu et al. [18] proposed a differentiable, hybrid Lagrangian–Eulerian physical simulator for controlling the motion of soft robots. Chen et al. [19] used a differentiable physics engine to teach NNs how to represent high-dimensional point cloud data collected from deformable objects. Millard et al. [20] represented DLO with a tetrahedral FEM mesh and optimized the material parameters to minimize the difference between the real and simulated observations. Liu et al. [21] modeled extensible and inextensible rope-like objects based on position-based dynamics (PBD); the models solved parameter-estimation shortcomings and improved the matching of rope physics to real-world scenarios. Our approach is similar to that reported in [21], although we also added collision constraints to DLO simulation to allow it to simulate the interaction with the environment. We also embedded an NN controller in the simulation to generate the trajectory for DLO shape control.

III. Problem Statement And Framework Overview

A. Problem Statement

We assumed the existence of a three-dimensional (3D) workspace in which the environment configuration is considered obstacles. The dimensions and positions of the obstacles were known. This workspace also contained an inextensible DLO of known length. One of its ends was fixed, and the other was grasped by the movable end-effector of the robot. Here, we assumed that DLO was in a quasi-static state, which means that its shape could only be determined by the actions of the end-effector and was not affected by inertial effects. Therefore, to mitigate the effect of the gravity of DLO on its shape, we specified that the end-effector of the robot only moved in three degrees of freedom, i.e., only translates and does not rotate. Fig. 2 illustrates that our objective was to determine a proper action trajectory that can manipulate the initial shape of DLO into the goal shape when the initial and goal shapes are known.

B. Issues and approach

During DLO manipulation additional deformations other than that caused by the end motion will occur if DLO contacts with obstacles. This will produce a final DLO shape that is significantly different from the goal one. This accounts for the highest impediment to achieving the objective. Hence, it is desirable to elucidate the effect of DLO contacts with obstacles to establish a control strategy for contact avoidance. Further,

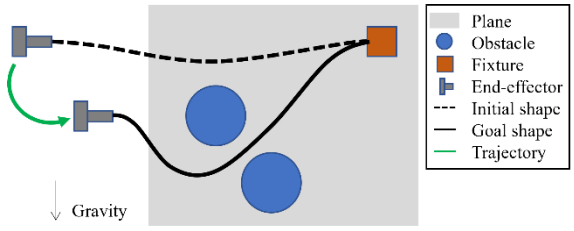

Fig. 2. Schematics of the problem setting.

the existing control strategy can still be employed instead of modifying it when the configuration of the obstacle changes, which was another critical issue requiring a solution. Here, the issues are listed, as follows:

1. Simulating DLO shape, as well as the deformation caused by its contact during the manipulation.
2. Designing a controller that can respond to environmental changes to ensure DLO manipulation with contact avoidance to achieve the goal shape.

To address Issue 1, we adopted physical simulation to simulate DLO deformation. Further, we added the collision response in this simulation and extended it to a differentiable framework. This allowed for the simulation of the effects of DLO contact with the environment, as well as the identification of the simulation parameters using the backpropagation of losses in the differentiable framework to achieve real-to-sim transfer. Regarding Issue 2, we embedded an NN controller in the DLO simulator. By adaptively optimizing the weights of NN, proper manipulation trajectories could be automatically generated for different environment layouts and goal shapes. Moreover, the generated trajectories could be transferred to real-world scenarios using the optimized parameters (sim-to-real). Fig. 3 illustrates an overview of our proposed framework, in which DLO modeling and NN mechanism are crucial. Hence, we will describe them comprehensively in the later sections.

IV. Position-Based Modeling

We utilized cosserat rods [22] based on the PBD [23] model to simulate the dynamic state of DLO and extended it to the compliant (XPBD) [24] model to reduce the time required for simulation iteration. Dissimilar to the traditional force-based model (the acceleration is computed from the combined force applied to the object based on Newton's second law of motion, after which the velocity and position of the object can be updated via time integration), PBD works immediately on the position of particles, which allows the simulator to be lightweight and facile implementation. Thus, an external manipulation of objects can be directly represented in terms of position, avoiding the overshooting issue of explicit integration in force-based models, and this is consistent with our objective (generating manipulation trajectories from the simulation). Another advantage is that the equations of this model are completely differentiable, allowing the application of the differentiable framework to readily implement its differentiability rather than adopting several extra approximation processing. Although PBD deviates from real-world behavior slightly because it does not obey the physical laws. However, from our experimental results, its deviation from reality does not have a significant impact on the achievement of the manipulation task. Here, for brevity, we

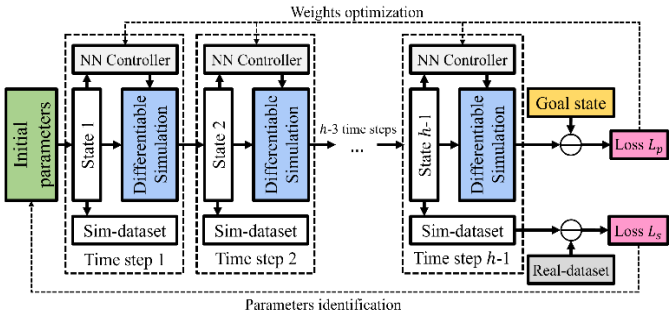

Fig. 3. Overview of the proposed framework. The solid arrows indicate the forward propagation. The whole system is end-to-end differentiable, and we optimized the weights and simulation parameters of the NN controller using the backpropagation of losses, as indicated by the dashed arrows.

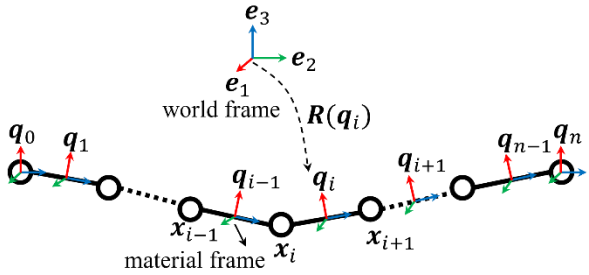

Fig. 4. Geometry of the cosserat rods represented by position $\boldsymbol{x}_i$ and quaternion $\boldsymbol{q}_i$, indicating rotation from the world frame to material frame.

only outline the results; the detail equations and derivation process are presented in [22]. We utilized the framework, as well as constraints 1) and 2) from [22] to obtain the two following extensions:

- We employed the XPBD method to solve the constraint equations.
- We added the extra constraints functions 3), 4) and 5).

DLO was discretized into n particles represented by their position and orientation (Fig. 4). The position is defined as vector $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n] \in \mathbb{R}^{3n}$ in Cartesian coordinates and the orientation is defined as a quaternion $\boldsymbol{q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_{n-1}] \in \mathbb{R}^{4(n-1)}$, which corresponds to the rotation from the standard orthogonal basis $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3) \in \mathbb{R}^3$, to the material coordinate whose origin is located in the middle of the adjacent particles. These two values are merged into $\boldsymbol{p} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n, \boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_{n-1}]$ for brevity of notation. In PBD, the corrections to the position and orientation $\Delta\boldsymbol{p}$, in each time step are computed by solving a set of constraint equations, $\boldsymbol{C}(\boldsymbol{p} + \Delta\boldsymbol{p}) = \boldsymbol{0}$. Further, the constraints can be linearized using Taylor-series expansion, as follows:

$$\boldsymbol{C}(\boldsymbol{p} + \Delta\boldsymbol{p}) \approx \boldsymbol{C}(\boldsymbol{p}) + \nabla_{\boldsymbol{p}}\boldsymbol{C}\Delta\boldsymbol{p} = \boldsymbol{0}, \tag{1}$$

where $\nabla_{\boldsymbol{p}}\boldsymbol{C}$ is the Jacobian of $\boldsymbol{C}$ w.r.t vector $\boldsymbol{p}$. This equation can be solved by restricting $\Delta\boldsymbol{p}$ to the derivative direction of the constraint function:

$$\Delta\boldsymbol{p} = \boldsymbol{M}^{-1}\left(\nabla_{\boldsymbol{p}}\boldsymbol{C}\right)^T\Delta\boldsymbol{\lambda}, \tag{2}$$

where $\boldsymbol{M}$ is the mass/inertia matrix, $diag(m_1 \cdot \boldsymbol{1}, m_2 \cdot \boldsymbol{1}, \dots, m_n \cdot \boldsymbol{1}, \boldsymbol{I}_1, \boldsymbol{I}_2, \dots, \boldsymbol{I}_{n-1})$; the change in the Lagrange multiplier, $\Delta\boldsymbol{\lambda}$, can be computed separately for each constraint, j by Gauss–Seidel solution from XPBD.

$$\Delta\lambda_j = -\left(\nabla_{\boldsymbol{p}}C_j\boldsymbol{M}^{-1}\nabla_{\boldsymbol{p}}C_j^T + \tilde{\alpha}_j\right)^{-1}\left(C_j(\boldsymbol{p}_i) + \tilde{\alpha}_j\lambda_{ij}\right). \tag{3}$$

Here λ_{ij} is the total Lagrange multiplier for constraint j at the current iteration i. $\tilde{\alpha} = \alpha/\Delta t^2$, where α and Δt denote the inverse stiffness and time step, respectively.

Furthermore, we defined several constraints to represent the deformability of DLO, as follows:

1) Stretch and shear constraint: based on the Cosserat theory, the strain measure for the shear and stretch was coupled to the difference between the tangent vector of the centerline and the normal of the cross-section. Therefore, the stretch and shear constraints $\boldsymbol{C}_s$, is formed by two adjacent particles with positions $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ and the quaternion $\boldsymbol{q}_i$, between them:

$$\boldsymbol{C}_s(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \boldsymbol{q}_i) = (\boldsymbol{x}_i - \boldsymbol{x}_{i+1})/l - \boldsymbol{R}(\boldsymbol{q}_i)\boldsymbol{e}_3, \quad (4)$$

where l is the resting length between the two particles. For simplicity, we assumed that all the l values are equal. $\boldsymbol{R}(\cdot)$ denotes the rotation matrix converted from the quaternion. Next, the derivatives w.r.t the involved positions and quaternions were derived, as follows:

$$\nabla_{\boldsymbol{x}_i}\boldsymbol{C}_s = -\nabla_{\boldsymbol{x}_{i+1}}\boldsymbol{C}_s = -1/l \cdot \boldsymbol{1}_{3\times 3}, \quad (5)$$

$$\nabla_{\boldsymbol{q}}\boldsymbol{C}_s = 2(\Im(\boldsymbol{e}_3\overline{\boldsymbol{q}}) \mid \Re(\boldsymbol{e}_3\overline{\boldsymbol{q}})\boldsymbol{1}_{3\times 3} - [\Im(\boldsymbol{e}_3\overline{\boldsymbol{q}})]^{\times}), \quad (6)$$

where $\Im(\cdot)$ and $\Re(\cdot)$ denote the imaginary and real parts of the quaternion, respectively; $\overline{\boldsymbol{q}}$ is the conjugate quaternion; and $[\cdot]^{\times}$ denotes the skew-symmetric matrix of a vector.

2) Bend and twist constraint: the Darboux vector $\boldsymbol{\Omega}$, was used to define the strain measure for bending and twisting. The bend and twist constraint $\boldsymbol{C}_b$, coupled with the two adjacent quaternions $\boldsymbol{q}_i$ and $\boldsymbol{q}_{i+1}$, and was used to compute the difference from the resting value:

$$\boldsymbol{C}_b(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) = \Im\left(\overline{\boldsymbol{q}}_i\boldsymbol{q}_{i+1} - \overline{\boldsymbol{q}}_i{}^0\boldsymbol{q}_{i+1}^0\right) = \boldsymbol{\Omega} - \xi\boldsymbol{\Omega}^0, \quad (7)$$

$$\xi = \begin{cases} +1 \; if \; |\boldsymbol{\Omega} - \boldsymbol{\Omega}^0|^2 < |\boldsymbol{\Omega} + \boldsymbol{\Omega}^0|^2, \\ -1 \; if \; |\boldsymbol{\Omega} - \boldsymbol{\Omega}^0|^2 > |\boldsymbol{\Omega} + \boldsymbol{\Omega}^0|^2. \end{cases} \quad (8)$$

The derivations w.r.t the involved quaternions were derived, as follows:

$$\nabla_{\mathbf{q}_i}\boldsymbol{C}_b = -\left(-\boldsymbol{q}_{i+1} \mid q_{i+1,0}\boldsymbol{1}_{3\times 3} - [\boldsymbol{q}_{i+1}]^{\times}\right), \quad (8)$$

$$\nabla_{\mathbf{q}_{i+1}}\boldsymbol{C}_b = +\left(-\boldsymbol{q}_i \mid q_{i,0}\boldsymbol{1}_{3\times 3} - [\boldsymbol{q}_i]^{\times}\right). \quad (9)$$

3) Direct distance constraint: as we considered an inextensible DLO, the distance between the adjacent particles must be kept constant during the simulation. The stretch deformation can be limited by setting the inverse stiffness to $\alpha = 0$ in (3). However, as the utilized Jacobi iterative solver in (2) contains errors, it cannot guarantee a constant distance. Therefore, we introduced an additional explicit distance constraint $\boldsymbol{C}_d$, and used the direct solution of the tridiagonal matrix algorithm proposed in [25] rather than the approximate Jacobi solution, as follows:

$$\boldsymbol{C}_d(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) = \|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\| - l, \quad (10)$$

where $\|\cdot\|$ denotes the $L2$ distance between two particles. The derivation w.r.t the positions can be effortlessly computed as

$$\nabla_{\boldsymbol{x}_i}\boldsymbol{C}_d = -\nabla_{\boldsymbol{x}_{i+1}}\boldsymbol{C}_d = \boldsymbol{x}_i - \boldsymbol{x}_{i+1}/\|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|. \quad (11)$$

4) End constraint: in this study, we considered the case in which one end of DLO was fixed, whereas the other was available for translational motion. Therefore, the bending and twisting of the part near the end particle would be smaller than that near the middle part. To achieve this difference, we defined two coordinate frames, $\boldsymbol{q}_0, \boldsymbol{q}_n$ (Fig. 4), at the two end particles representing the grasping/fixing pose. Thereafter, two extra bend-twist constraints $\boldsymbol{C}_b(\boldsymbol{q}_0, \boldsymbol{q}_1)$ and $\boldsymbol{C}_b(\boldsymbol{q}_{n-1}, \boldsymbol{q}_n)$, were added to constraint $\boldsymbol{C}_b$.

Algorithm 1 DLO detection and chain generation

Input *Image, l, n.*
extracted ← Binarize(*Image*)
thinned ← Skeletonize(*extracted*)
corners ← Corner detection(*thinned*)
thinned3d, *corners3d* ← Pixel-to-3d(*thinned*, *corners*)
start ← *corner3d*[0]
for each $\boldsymbol{y}_i \in \boldsymbol{y}$ **do**
  for each $\boldsymbol{u} \in thinned$ **do**
    $dist \leftarrow |\|\boldsymbol{u} - start\| - l|$
  end for
  $\boldsymbol{y}_i \leftarrow thinned3d[argmin(dist)]$
  $start \leftarrow \boldsymbol{y}_i$
end for
$\boldsymbol{y}_n \leftarrow$ *corner3d*[1]
return $\boldsymbol{y}$

5) Collision constraint: the collision response of DLO with the obstacles must be handled in our problem setting. Here, we did not consider the self-collision of DLO for reducing the computation time. Collision handling in PBD can also be treated as a constraint. First, we tested the ray $\boldsymbol{x}_i^t \rightarrow \boldsymbol{x}_i^{t+1}$, for each particle i, to determine if the ray penetrated the obstacle. We computed the entry point $\boldsymbol{x}_{ic}$, and surface normal $\boldsymbol{n}_{ic}$, at this position. $\boldsymbol{x}_{ic}$ was replaced by the point on the surface of the obstacle nearest to $\boldsymbol{x}_i^t$ if the ray lies completely inside the obstacle, and $\boldsymbol{n}_{ic}$ is replaced by the normal at the nearest point. Next, a unilateral constraint function $\boldsymbol{C}_c(\boldsymbol{x}_i) = (\boldsymbol{x}_i - \boldsymbol{x}_{ic}) \cdot \boldsymbol{n}_{ic}$, was defined, and an inverse stiffness $\alpha = 1$, was added to this constraint. The derivate w.r.t the position can be readily obtained by $\nabla_{\boldsymbol{x}_i}\boldsymbol{C}_c = \boldsymbol{n}_{ic}$.

V. Differentiable Framework

A. Real-to-sim stiffness identification

To obtain a real-world DLO shape that is close to the simulation shape to generate the manipulation trajectory in the simulation and transfer it to the real-world, we must identify the inverse stiffness controlling DLO deformation in the simulation from the action and state of real-world DLO. Additionally, as the DLO assumed in this study is inextensible, with movable ends performing translational motions, only its bending deformation can be considered, i.e., the inverse stiffness α_b, in the bend-twist constraint must be identified.

We proposed a greedy algorithm for perceiving a chain of real-world DLOs (Algorithm 1). The RGB-depth image containing DLO; the length l, between adjacent particles; and the number of particles are given as input. The output comprises a chain, $\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n] \in \mathbb{R}^{3n}$, with the same dimension as $\boldsymbol{x}$ in the simulation to represent the actual geometry of DLO. To determine the geometry of DLO, we

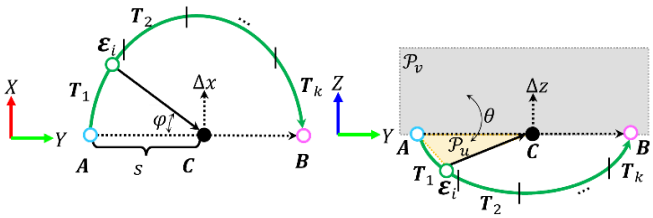

Fig. 5. Definition of the arc-like subtrajectories (green solid line). Each subtrajectory $\boldsymbol{T}_i$, is represented by discrete points $\boldsymbol{\varepsilon}_i$ (green hollow circle), which are computed from five parameters $(s, \varphi, \theta, \Delta x, \text{and } \Delta z)$, beginning from $\boldsymbol{A}$ (cyan hollow circle) and ending in $\boldsymbol{B}$ (magenta hollow circle).

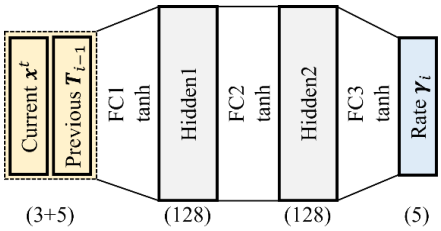

Fig. 6. Architecture of NN. The numbers in the bottom brackets indicate the number of neurons in the corresponding layers.

must first detect DLO from the environment by semantic segmentation, etc. But this is beyond the main scope of this study. Therefore, we significantly differentiated the DLO color from the background so that the part in the image can be effortlessly detected by color thresholding. Lines 2–5 represent image processing based on existing libraries. The procedure is as follows: first, the pixels of the DLO part were extracted by binarization, after which the centerline of the DLO was extracted by skeletonization. Next, both ends of DLO were determined by a Harris corner detector. Finally, the centerline and ends were converted into 3D point clouds. Lines 7–13 represent a greedy algorithm in which n particles were sampled uniformly on the point cloud of the centerline, keeping the error between the distance of adjacent particles and l under a threshold. By this method, we could achieve the rapid tracking of DLO geometry and maintain its inextensibility.

Next, we manipulated the real-world DLO, following a given trajectory while recording ϖ frames of its geometry by the proposed tracking algorithm. In the simulation, DLO is also driven by the same trajectory. The same geometry value was also recorded at the same position. The loss function L_s, involving the inverse stiffness α_b, between the simulation and real world is defined as

$$L_s(\alpha_b) = \sum_{j=1}^{\varpi}\sum_{i=1}^{n} \|\boldsymbol{x}_i^j - \boldsymbol{y}_i^j\|/n. \tag{12}$$

The gradient w.r.t α_b can be simply obtained by the backpropagation of the error in the differentiable framework. Afterward, the gradient-based optimization method, e.g., gradient descent, can be used to estimate α_b. The result of the optimization is presented in Section V.

B. Generation of the sim-to-real trajectory

This trajectory is for driving DLO from the start to the end while avoiding obstacles along a vertical plane during manipulation. Therefore, the trajectory must be a universal arc-like curve, although its exact dimensions depend on the goal shape. Thus, we defined a set of parameters to control the dimension of the trajectory and use NN to predict them. Contrary to the method of directly predicting discrete points on the trajectory, our method can prevent the issue of simulation failure due to overshooting by setting boundaries to limit the trajectory within a valid range.

1) Representation of the manipulation trajectory: a manipulation trajectory is segmented into k subtrajectories $\boldsymbol{T} = (\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_k)$ (Fig. 5). This allows for the adjustment of the trajectory parameters according to the current state of DLO to predict the next action $\boldsymbol{\varepsilon}_i$. The geometry of each subtrajectory was defined by five parameters $\boldsymbol{T}_i = (s, \varphi, \theta, \Delta x, \Delta z)$. Here, we described the process of computing the action $\boldsymbol{\varepsilon}_i$, based on these parameters. $\boldsymbol{A}$ denotes the initial position of the trajectory when the goal shape of DLO is given; the position of its movable end is regarded as the final position $\boldsymbol{B}$, of the trajectory. Thus, the center $\boldsymbol{C}$, of the subtrajectory is determined, as follows:

$$\mathfrak{D} = \boldsymbol{B} - \boldsymbol{A}, \tag{13}$$

$$\overline{\boldsymbol{C}} = s\mathfrak{D} + \boldsymbol{A}, \tag{14}$$

$$\boldsymbol{C}_x = \overline{\boldsymbol{C}}_x + \Delta x, \boldsymbol{C}_y = \overline{\boldsymbol{C}}_y, \boldsymbol{C}_z = \overline{\boldsymbol{C}}_z + \Delta z, \tag{15}$$

where $\overline{\boldsymbol{C}}$ is the point on line $\boldsymbol{AB}$, Δx and Δz are the offsets of point $\overline{\boldsymbol{C}}$ on the X- and Z-axes, respectively. Furthermore, $\boldsymbol{\varepsilon}_i$ can be obtained by

$$\boldsymbol{\mathcal{N}} = -\mathfrak{D} \times \left(\boldsymbol{e}_3 \circ rot(|\mathfrak{D}|, \theta)\right), \tag{16}$$

$$\boldsymbol{\varepsilon}_i = (\boldsymbol{\varepsilon}_{i-1} - \boldsymbol{C}) \circ rot(\boldsymbol{\mathcal{N}}, \varphi), \tag{17}$$

where $\boldsymbol{\mathcal{N}}$ denotes the normal of plane $\mathcal{P}_u$ and θ denotes the angle between planes $\mathcal{P}_u$ and $\mathcal{P}_v$ in radians; $|\cdot|$ denotes vector normalization, $\boldsymbol{\tau} \circ rot(\boldsymbol{\upsilon}, \phi)$ means rotating vector $\boldsymbol{\tau}$ by ϕ radians about the $\boldsymbol{\upsilon}$ axis. The initial value of $\boldsymbol{\varepsilon}_i$ is equal to the initial position $\boldsymbol{A}$, of the trajectory.

2) Neural network: NN is constructed with three fully connected layers (Fig. 6). The input layer is the current state, $\boldsymbol{x}^t$, of DLO and previous parameters $\boldsymbol{T}_{i-1}$, of the subtrajectory. This layer is followed by two hidden layers with tanh activation functions. The output layer is the rate vector $\boldsymbol{\gamma}_i$, of the parameters of the current subtrajectory. As the value of the tanh function must be $[-1,1]$, the trajectory parameters can be limited to a valid range after multiplying them by a max vector $\boldsymbol{\Lambda}$, as follows:

$$\boldsymbol{T}_i = \boldsymbol{\gamma}_i \cdot \boldsymbol{\Lambda}. \tag{18}$$

To implement the backpropagation of the loss, thus updating NN via gradient-based optimization, to predict the appropriate trajectory parameters, a loss function L_p, must be manually designed. In the application of differentiable simulation, the simplest L_p can be the difference between the final state of the object and its goal state, e.g., $MSE(\boldsymbol{x}^h, \boldsymbol{x}^*)$, where MSE is the mean-squared error h denotes the number of simulation time steps, and $\boldsymbol{x}^*$ is the goal particles of DLO. However, ignoring the collision of the DLO with the obstacle will result in local optimization without achieving the task. Therefore, based on the direction of loss descent (Fig. 7), we added the difference between the particle at the position of the collision and its corresponding goal particle into L_p, as follows:

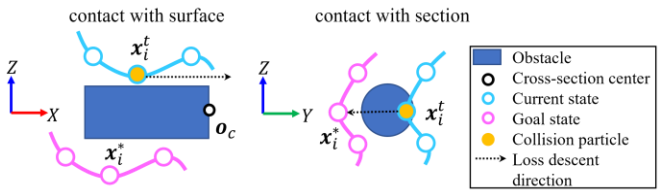

Fig. 7. Illustration of the direction of the loss-descent function when collisions occur. The arrows (dotted line) indicate the direction that must be moved to avoid collisions.

$$L_p = \omega_1 \sum_{\Gamma} L_c + \omega_2 MSE(\boldsymbol{x}^h, \boldsymbol{x}^*), \tag{19}$$

with

$$L_c = \begin{cases} \left|\boldsymbol{x}_{i,x}^t - \boldsymbol{o}_{c,x}\right| \ if\ \boldsymbol{x}_i^t\ c.w.surface, \\ \left|\boldsymbol{x}_{i,x}^t - \boldsymbol{x}_{i,x}^*\right| + \left|\boldsymbol{x}_{i,y}^t - \boldsymbol{x}_{i,y}^*\right| \ if\ \boldsymbol{x}_i^t\ c.w.section, \end{cases} \tag{20}$$

where ω_1 and ω_2 are the weight factor to trade-off among different loss terms, Γ denotes the number of collisions in the total time steps. $\boldsymbol{o}_c$ is the center of the cross-section of the obstacle, $|\cdot|$ denotes the absolute value and $c.w.$ means contact with. In the next section, we confirmed that the task can be achieved using L_p involving contact effects, whereas the one without considering contact effect cannot.

In summary, the process of determining an appropriate trajectory from the differentiable simulation via the trial-and-error approach based on the backpropagation of the loss is shown in Algorithm 2. The first line represents the initialization. In addition to assigning initial values to some parameters in the simulation, must allocate initial values to the weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$ of the fully connected layers in NN, for which we adopt the Xavier initial values. Next, we updated the parameters of NN via the backpropagation of the loss (Lines 7–9, where η denotes the learning rate) and predicted the trajectory parameters (Line 4). The output is a list of actions $(\boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2 \dots, \boldsymbol{\varepsilon}_h)$ for all the time steps computed by the trajectory parameters, i.e., a discrete trajectory curve.

VI. Experiment

We implemented DLO simulation, as described in Section IV using a differentiable framework—*Taichi* [26]. Thereafter, we constructed the same configuration in the real world as in the simulation. The actual DLO was a fiber rope (length = 445 mm, diameter = 8 mm, and color = white). One end of the DLO was fixed to a vertical resin plate, and the other was grasped by a CR5 robot. The background color was black. The RGB-depth images of the rope were captured by a calibrated Azure Kinect camera, and the image was processed using OpenCV [27], following Algorithm 1. The rope was discretized into 30 particles and 29 segments, i.e., $n = 30$ and $l = 445/29 \approx 15$ mm. The time step Δt of the simulation was 0.01 sec.

Algorithm 2 Differentiable framework for learning trajectory

1 **Input** $\boldsymbol{x}^1$, $\boldsymbol{x}^*$, $\boldsymbol{w}^1$, $\boldsymbol{b}^1$, $\boldsymbol{\Lambda}$, h
2 **while** $iter < iterations$ **do**
3 **for each** $\boldsymbol{x}^t$, $\boldsymbol{T}_i$ **do**
4 $\boldsymbol{T}_i \leftarrow NN(\boldsymbol{x}^t, \boldsymbol{T}_{i-1}, \boldsymbol{w}^i, \boldsymbol{b}^i)$
5 compute $\boldsymbol{\varepsilon}_t$ using (17)
6 $\boldsymbol{x}^{t+1} \leftarrow DiffSim(\boldsymbol{\varepsilon}_t, \boldsymbol{x}^t)$
7 compute L_p using (19)
8 compute $\nabla_{\boldsymbol{w}} L_p$, $\nabla_{\boldsymbol{b}} L_p$ using auto-differentiation
9 $\boldsymbol{w}^{i+1} \leftarrow -\eta \nabla_{\boldsymbol{w}} L_p$, $\boldsymbol{b}^{i+1} \leftarrow -\eta \nabla_{\boldsymbol{b}} L_p$
10 **end for**
11 $iter + 1$
12 **end while**
13 **return** $(\boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2 \dots, \boldsymbol{\varepsilon}_h)$

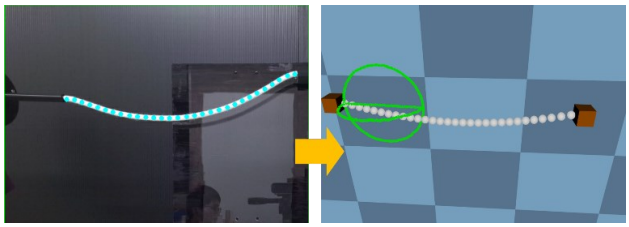

Fig. 8. Real-world appearance of the rope (left) and simulation (right). The cyan circles represent the perceived shape, $\boldsymbol{y}$, of the rope using the tracking algorithm. Both sides are moved along the given trajectory (green line) and 40 frames were recorded simultaneously.

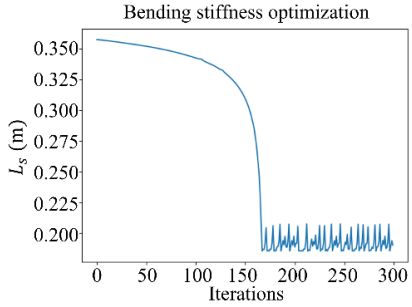

Fig. 9. Graph of the loss value versus number of iterations.

A. Real-to-sim stiffness identification

To obtain the geometries of the rope relative to the different actions from the real-world experiment and simulation, first, we formulated an arc trajectory in the real world and discretized it into 400 waypoints. Next, the end of the robot was controlled to move sequentially through these waypoints. Concurrently, all the chain points $(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots \boldsymbol{y}_{30})$ of the rope were recorded at 10 intervals using our proposed tracking algorithm (Fig. 8). Thus, we obtained 40 frames $(\mathfrak{w} = 40)$ of the geometries of the rope. Second, in the simulation, we subjected the same 400 waypoints to the action of the rope at each time step. Additionally, 40 frames of the chain points $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_{30})$ of the rope were recorded at the same interval as that in the real-world experiment.

Afterward, we used the backpropagation of the loss to obtain the derivative of the loss w.r.t the inverse stiffness of the bend constraint α_b, and optimized α_b^* based on the gradient-descent method to minimize the difference between the losses in the simulation and real-world experiment. Thus, we set several optimization parameters: the initial value of $\alpha_b^0 = 0.1$, learning rate $\eta = 1.0^{-4}$, and iterations = 300. Fig. 9 shows that the loss was finally converged to ~0.2 m, while the optimal $\alpha_b^* = 0.0089$ at this moment.

B. Generation and validation of the manipulation trajectory

We confirmed that using the proposed NN controller and a loss function, considering the contact, can generate an offline manipulation trajectory exhibiting contact avoidance to achieve the goal shape in the simulation and confirmed that the execution of the real-world trajectory achieves the task using

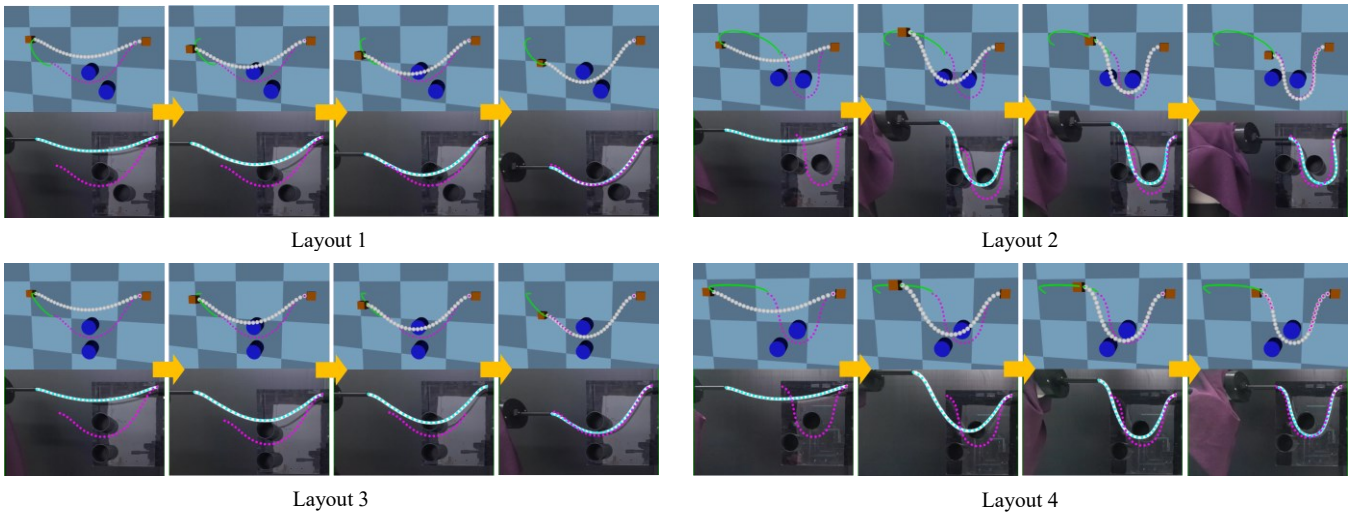

Fig. 10. Results of the motion along the generated trajectories (green line) in the simulation (top) and real-world (bottom) under four different layouts. The magenta circles indicate the goal shape, the orange arrow indicates the manipulation from the initial shape to the final shape.

TABLE I Hyperparameters

Parameters	Meaning	Value
Iterations	Number of iterations	200
η	Learning rate	0.001
ω_1	Weight factor of *contact* term	0.1
ω_2	Weight factor of *MES* term	1.0

the optimal stiffness from the previous section. The experimental settings were as follows: two cylinders (diameter = 60 mm and height = 60 mm) were fixed to the same resin plate as the rope, and the distance between the centers of both cylinders was 100 mm. We used AR markers to predetermine the positions of the cylinders and removed them before moving the rope. The goal shape of the rope, which was partially between the gaps of the cylinders, was generated in the simulation. The total number of time steps for the simulation was 300 ($h = 300$). The trajectory was segmented into 10 subtrajectories ($k = 10$), and the max vector of the parameters of the subtrajectories was $\mathbf{\Lambda} = (0.008, 1.5, 0.5, 0.05, 0.05)$. TABLE I presents the values of the other learning-related hyperparameters.

We experimented with four different sets of layouts. In the simulation (Fig. 10) NN could generate an arc-like trajectory by learning iteratively from the simulation when the goal shape of the rope is given Thus, driving the movable end of the rope along this trajectory can achieve contact avoidance and achieve the goal shape. This demonstrates that the manipulation trajectory of DLO can be effectively generated by a framework that integrates NNs with the simulation. Furthermore, even if the environment configuration changes, it is not necessary to redesign or modify the controller or adjust any parameters as the method exhibits generalizability. Conversely, the simulation results can be extensively reproduced by executing the trajectory that was transferred from the simulation to the real world. This demonstrates that the simulation-generated trajectories could also apply to the real world. Further, this confirms that the parameter identified by the backpropagation of loss from differentiable simulations is valid in the real world. However, when the rope was subjected to considerable deformation, the gap between the simulation and real-world results increased significantly (see

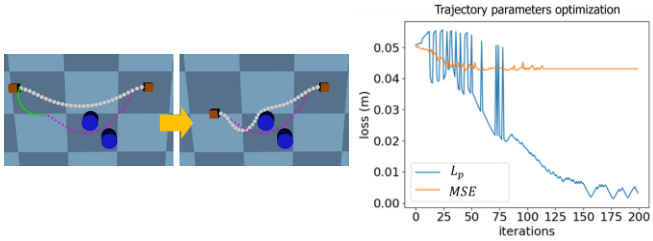

Fig. 11. Result of the simulation using the pure MES (left) and comparison between the learning curves using MSE and L_p (right).

Fig. 10, Layout 2). This is mainly because an actual rope produces a certain plastic deformation degree, whereas our simulation only considered elastic deformation.

Additionally, we also tested the loss function by ignoring the contact-related term in Layout 1, i.e., pure $MSE(\boldsymbol{x}^h, \boldsymbol{x}^*)$. Fig. 11 shows that the loss value finally converged to the moment when the rope contacted the cylinder and could not decrease further. This is because, under pure MSE, the contact-avoiding trajectory rather increased the loss value such that NN did not update in the direction away from the cylinders. However, under L_p, the loss value was decreased to less than 0.01 m, consequently achieving the goal shape. This comparison demonstrates the superiority of our proposed loss function, which considers the effect of contact.

VII. Conclusion

Here, we introduced a method, which integrated simulations and NNs, to achieve the shape-control task of DLOs while considering the effects of contacts. First, we extended the simulation based on the PBD model that could simulate DLO deformation when manipulated and in contact with the environment, as well as are end-to-end differentiable. Thereafter, the backpropagation of the loss from the real-world to the simulation was applied to identify the stiffness parameter in the simulation. Finally, we used an NN controller to predict the manipulation trajectory from a simulation that could achieve the task. The experiments demonstrated that the backpropagation of the loss could ensure the automatic identification of the parameters in the simulation. Concurrently, the offline-generated trajectory from the simulation is available in the real-world scenario, further

confirming the effectiveness of this method. Future studies will include the online generation of manipulation trajectories with feedback to cope with more complex environment configurations and goal shapes.

REFERENCES

[1] H. Yin, A. Varava, D. Kragic, "Modeling, learning, perception, and control methods for deformable object manipulation," Sci. Robot. 6, eabd8803, 2021.

[2] R. Laezza,, R. Gieselmann, F. T. Pokorny and Y. Karayiannidis. "ReForm: A Robot Learning Sandbox for Deformable Linear Object Manipulation," in 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 4717-4723, 2021.

[3] K. Nozaki, C. Ying, Y. Matsuura, and K. Yamazaki, "Manipulation Planning for Wiring Connector-Attached Cables Considering Linear Object's Deformability," Int. J. Automation Technol., Vol.17 No.4, pp. 399-409, 2023.

[4] D. McConachie, T. Power, P. Mitrano and D. Berenson, "Learning When to Trust a Dynamics Model for Planning in Reduced State Spaces," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 3540-3547, 2020.

[5] N. Lv, J. Liu, X. Ding, J. Liu, H. Lin and J. Ma., "Physically based real-time interactive assembly simulation of cable harness," Journal of Manufacturing Systems, vol. 43, no. 3, pp. 385-399, 2017.

[6] S. Duenser, J. M. Bern, R. Poranne, and S. Coros, "Interactive robotic manipulation of elastic objects," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3476–3481, 2018.

[7] A. Koessler, N. Roca Filella, B. Bouzgarrou, L. Lequievre, and J.-A. Corrales Ramon, "An efficient approach to closed-loop shape control of deformable objects using finite element models," in 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021.

[8] N. Lv, J. Liu and Y. Jia, "Dynamic Modeling and Control of Deformable Linear Objects for Single-Arm and Dual-Arm Robot Manipulations," IEEE Transactions on Robotics, vol. 38, no. 4, pp. 2341-2353, 2022.

[9] J. Zhu, B. Navarro, P. Fraisse, A. Crosnier and A. Cherubini, "Dual-arm robotic manipulation of flexible cables," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 479-484, 2018.

[10] J. Zhu, B. Navarro, R. Passama, P. Fraisse, A. Crosnier and A. Cherubini, "Robotic Manipulation Planning for Shaping Deformable Linear Objects with Environmental Contacts," in IEEE Robotics and Automation Letters, vol. 5, no. 1, pp. 16-23, 2020.

[11] M. Yu, H. Zhong and X. Li, "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), pp. 1337-1343, 2022.

[12] Y. Yang, J. A. Stork and T. Stoyanov, "Online Model Learning for Shape Control of Deformable Linear Objects," in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4056-4062, 2022.

[13] Y. Yang, J. A. Stork and T. Stoyanov, "Learning to Propagate Interaction Effects for Modeling Deformable Linear Objects Dynamics," in 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 1950-1957, 2021.

[14] S. Huo et al., "Keypoint-Based Planar Bimanual Shaping of Deformable Linear Objects Under Environmental Constraints With Hierarchical Action Framework," in IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 5222-5229, 2022.

[15] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," in 4th Conference on Robot Learning (CoRL), 2020.

[16] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to Manipulate Deformable Objects without Demonstrations," in Robotics: Science and Systems (RSS), 2020.

[17] J. Liang, M. C. Lin, and V. Koltun, "Differentiable cloth simulation for inverse problems," Advances in Neural Information Processing Systems (NeurIPS), 2019.

[18] Y. Hu et al., "ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics," in 2019 International Conference on Robotics and Automation (ICRA), pp. 6265-6271, 2019.

[19] S. Chen, Y. Liu, S. W. Yao, J. Li, T. Fan and J. Pan, "DiffSRL: Learning Dynamical State Representation for Deformable Object Manipulation with Differentiable Simulation," in IEEE Robotics and Automation Letters, vol. 7, no. 4, pp. 9533-9540, 2022.

[20] D. Millard, J. A. Preiss, J. Barbic and G. S. Sukhatme, "Direct System Identification of Deformable Objects using Differentiable Finite Element Dynamics," in 3rd Workshop on Robotics Manipulation of Deformable Objects: Challenges in Perception, Planning and Control for Soft Interaction (ROMADO-SI), 2022.

[21] F. Liu, E. Su, J. Lu, M. Li and M. C. Yip, "Robotic Manipulation of Deformable Rope-Like Objects Using Differentiable Compliant Position-Based Dynamics," in IEEE Robotics and Automation Letters, vol. 8, no. 7, pp. 3964-3971, 2023.

[22] T. Kugelstadt, and E. Schömer, "Position and orientation based Cosserat rods", Symposium on Computer Animation, 2016.

[23] M. Müller, B. Heidelberger, M. Hennix and J. Ratcliff, "Position based dynamics," Journal of Visual Communication and Image Representation, vol. 18, no. 2, pp. 109-118, 2007.

[24] M. Macklin, M. Müller, and N. Chentanez, "XPBD: Position-based simulation of compliant constrained dynamics," in Proceedings of the 9th International Conference on Motion in Games, pp. 49–54, 2016.

[25] L. Xu and Q. Liu, "Real-time inextensible surgical thread simulation," International Journal of Computer Assisted Radiology and Surgery, vol. 13, no. 7, pp. 1019–1035, 2018.

[26] Y. Hu, L. Anderson, T. M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," The International Conference on Learning Representations (ICLR), 2020.

[27] OpenCV. (2015). Open Source Computer Vision Library.