Efficient Real2Sim2Real of Continuum Robots Using Deep Reinforcement Learning With Koopman Operator

Guanglin Ji, Qian Gao, Yin Xiao, and Zhenglong Sun, *Senior Member, IEEE*

***Abstract*—Accurate control of continuum robots is challenging, especially in the presence of external disturbances. To address this issue, reinforcement learning (RL) has been increasingly investigated in continuum robot control due to its online policy updating capability. However, the gap in Sim2Real transfer caused by inaccurate modeling results in RL implementation a dilemma. In this article, we propose a safety-critical Real2Sim2Real online RL framework, where the Real2Sim transfer is first achieved by the identification of a continuum robot using the Koopman operator. We further improve the training efficiency by introducing an imperfect demonstration into the RL framework. The offline policy is trained in simulations and then tested on a real continuum robot platform. During tests, the tracking performance is influenced by the hysteresis effect that cannot be captured by the Koopman operator. This results in a millimeter-level tracking root mean square error (RMSE). To address this issue, we online update the policy as well as the model, and the RMSE of the online controller outperforms the offline controller by 89.16% in free space and 85.70% under external payload, respectively.**

***Index Terms*—Continuum robot, data-driven control, learning from demonstration, reinforcement learning (RL).**

I. Introduction

THE control of soft continuum robots using RL has drawn significant interest in recent years due to their naturally compliant behaviors, which are challenging to model accurately. However, conflicts still exist once mentioning why and when to implement RL instead of traditional controllers in soft continuum robots. The conflicts can be summarized from the following three aspects.

1) Inappropriate Simulator for RL Learning: In most research on RL-based control of continuum robots, simulators such as constant curvature assumptions [1], [2], finite element analysis (FEA) approaches [e.g. Adams [3] and SOFA [4]], and cosserat rod model [5] have been commonly utilized. However, these deployments are somewhat inaccurate, difficult to specify the real system, time-consuming, and lack the ability to perform real-time control evaluation. The policy trained under constant curvature assumptions is not directly applicable to real robotic control tasks as it requires retraining on the robotic platform. Moreover, the computational consumptions for FEA models and cosserat models are intensive, making their real-time implementation challenging. Failures often occur when controlling robots using an RL policy derived from an inappropriate simulator.

2) Lack of Online Learning Capability: Considering we have derived a relatively accurate model as described in Section I-1, the meaning of using RL instead of traditional controllers should be clarified. While the model performs well under ideal assumptions, it becomes less accurate when unpredictable disturbances occur, such as those caused by the interactions between the robot and its environment. These disturbances can lead to the deformation of the continuum robot body. However, conventional model-based controllers typically lack the capacity to modify the control policy in response to these changes [6], unless the external disturbances can be estimated in real time [7]. An RL controller, in contrast, can continuously update the optimal policy, even in the face of model variations. Thus, to control uncertain systems, online learning ability is critical in robotic control [8]. Online RL has been investigated in mobile robotics [8], [9], rehabilitation engineering [10], and soft robotic arm [11], where they demonstrate a promising future for complicated working conditions. However, the RL implemented in the soft robotic arm [11] demonstrating an online adaptive control effect is a PID controller tuner instead of an end-to-end RL. Since continuum robots usually work in the confined space or under external payloads, which results in the control performance being undesirable. Therefore, to control continuum

Received 11 March 2024; revised 4 August 2024 and 9 December 2024; accepted 8 January 2025. This work was supported in part by the Shenzhen Science and Technology Program under Grant RCYX20200714114736115 and Grant KJZD20230923114009018, and in part by the China Merchants Group funding of the project "Multiscale Interventional Robot System" under Grant BN00202312037. *(Corresponding author: Zhenglong Sun.)*

Guanglin Ji, Qian Gao, and Zhenglong Sun are with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen 518172, China, and also with Shenzhen Institute of Artificial Intelligence and Robotics for Society, The Chinese University of Hong Kong, Shenzhen 518129, China (e-mail: guanglinji@link.cuhk.edu.cn; qiangao1@link.cuhk.edu.cn; sunzhenglong@cuhk.edu.cn).

Yin Xiao is with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China (e-mail: xiaoyin@cuhk.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TIE.2025.3532733, provided by the authors.

Digital Object Identifier 10.1109/TIE.2025.3532733

robots with the capability of online control policy updating is critical.

3) Safety Insurance: Although the optimal policy can be derived using RL, the trial-error mechanism makes the training process unsafe. However, most research applying RL on continuum robots does not have any safety insurance [1]. Meanwhile, the working conditions of continuum robots are usually special, such as aero-engine detection [12], nuclear plant detection [13], and minimally invasive surgeries [14]. Those scenarios all require the safety-critical motions of robots. This is also the reason why RL on continuum robots cannot be easily applied in real-world applications.

The primary motivation of this article is to develop a safety-critical RL framework that resolves the conflicts mentioned above. This framework, referred to as the Real2Sim2Real framework, consists of two phases. In the Real2Sim phase, we construct a model of the continuum robot using real-world data and transfer it into the simulation to serve as the RL training environment. This enables the derivation of an optimal RL policy within the simulator. Furthermore, the Sim2Real phase involves transferring the derived policy to the real continuum robot and performing both offline and online control. In our case, it is essential to ensure that the RL policy can be updated during control tasks, allowing it to adapt to interactions with the environment.

A. Related Works

The major focus of RL is to apply the policy generated from simulators to real robots (Sim2Real transfer). Nevertheless, a significant gap still exists in Sim2Real when using RL to control robots. This gap is specifically reflected in the fact that the model of a robot used to derive the RL policy in simulation is different from the real-world model, as well as the inability to simulate the unforeseen external disturbance [15]. Sim2Real transfer has drawn great attention for years. Most of the Sim2Real transfer research was implemented on robotic arms [16], autonomous vehicles [17], unmanned aerial vehicles [18], etc. However, Sim2Real transfer for continuum robots is still an open problem.

Deploying the RL algorithm on continuum robots in simulations is considerably more straightforward than in real-world applications. In [5], a fuzzy RL-based controller was designed to control a simulated two-degree-of-freedom (DoF) continuum robot. Furthermore, a multisegment continuum robotic arm was commanded to through balls using an RL-based controller [19]. Although an accurate cosserat-rod model was used in [5], and the commanded tasks in [19] are much more difficult, they were still not tested in real world, which makes the effectiveness doubtful for real applications.

Several studies demonstrated successful deployment in both simulations and experiments using RL control policies [1], [2], [20], [21], [22]. Satheeshbabu et al. [2], [20] explored the application of deep Q network (DQN) and deep deterministic policy gradient (DDPG) for open-loop and closed-loop control of a soft continuum robotic arm using offline mode, respectively. In [21], they initially pretrained an RL control policy within simulations and ignored the challenges associated with Sim2Real transfer; instead, they retrained the RL policy during experiments. Our previous work [1] proposed to control a continuum robot using multiagent deep Q Network (MADQN) to separately control each DoF of the robot. The training process was efficient and the Sim2Real transfer performed well during experiments. Despite the efficacy of MADQN and its potential as a benchmark for decentralized control of continuum robots, the Sim2Real gap still existed, which was reflected in a short period of retraining in experiments. Furthermore, none of the above implemented RL approaches on continuum robots consider the safety standard, such as obstacle avoidance and unsafe motion behaviors during retraining on real robots.

B. Contributions

This study presents several key innovations in RL-based control for continuum robots. We propose a novel Real2Sim2Real framework designed to achieve efficient policy transfer, by parallel execution of a "simulation loop" and an "on-robot control loop." This framework enables real-time updates of the RL environment and control policy, allowing the system to adapt to continuously applied disturbances. To the best of our knowledge, this is the first attempt to apply a Real2Sim2Real approach for updating both the environment and control policies in RL-based robotic control. In this study, we employ a data-driven identified system kinematics model as the simulator within the "simulation loop," where it is updated with real-world data. The control policy is then retrained and transferred to the "on-robot control loop" to effectively handle environmental changes. To further enhance training efficiency, we propose a framework for reinforcement learning from imperfect demonstrations (RLfID), guiding the agent to sample around imperfect actions. To ensure the above framework meets safety-critical requirements, we formulate a quadratic programming (QP)-based safety-critical controller for tip obstacle avoidance. These contributions collectively enhance online policy learning, Sim2Real transfer, training efficiency, and safety in real-time applications for continuum robots.

II. Methods

A. Real2Sim2Real: Framework Overview

The Real2Sim2Real framework is designed to address the challenges inherent in applying RL algorithms to real-world robotic systems. Although the algorithm performance is satisfactory in simulations, the real-world model of the system and the disturbances not taken into account usually result in disaster performance in real applications. To overcome the challenges of the gap between simulations and real-world applications, we propose a Real2Sim2Real robotic learning framework for safe RL of soft continuum robots. The whole Real2Sim2Real framework is illustrated in Fig. 1, where the framework consists of two main components: Real2Sim and Sim2Real.

In the Real2Sim procedure, we utilize a space lifting technique to capture the real-world kinematics of the continuum robot. Additionally, within our framework, the Koopman operator-based model functions as the simulation environment, which helps to bridge the gap between the simulator and the

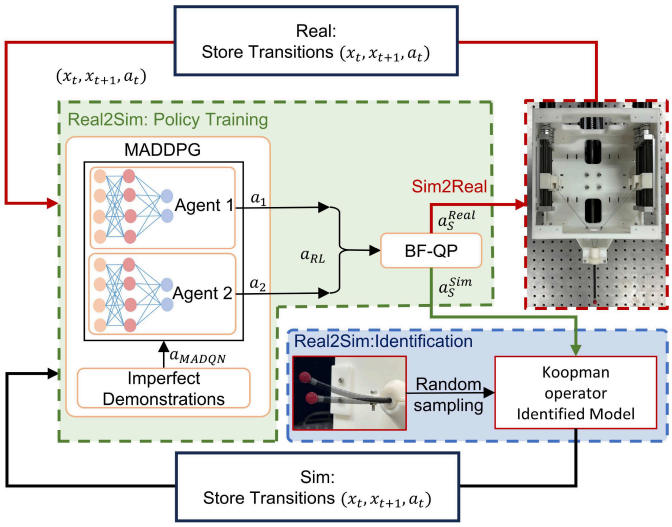

Fig. 1. Real2Sim2Real framework: the Real2sim process is divided into a system identification procedure (blue dashed box) and an RL policy learning procedure (green dashed box) and the identified system is implemented as the simulation environment. The Sim2Real procedure is illustrated using the red arrow line. The training is achieved in the simulations and then transferred into real robotic control in both offline and online mode as described in Section II-D1.

real-world model. In the simulation, we propose to learn the control policy using imperfect demonstrations for accelerating the training process.

The initial policy derived in the simulation is based on the linearized system kinematics without any disturbance. However, the optimal tracking policy can be different from the initial policy during experiments, especially when external disturbance is applied to the robot. Therefore, in the Real2Sim procedure, the control policy is required to update according to the disturbance. To update the policy, we also update the simulation environment online by using the latest data. The detailed online learning framework is introduced in Section II-D.

B. RL Environment Construction Using Real-World Data

To solve the first critical problem of accurate and efficient modeling of continuum robots as described in Section I-1, we utilize a data-driven approach to identify and model the behavior of soft continuum robots, specifically employing the Koopman operator theory [7], [23]. Our goal is to develop a differential kinematics state-space equation based on data collected from a real continuum robot.

1) Koopman Representation: Considering the state space representation of a continuum robot is given by

$$\dot{\mathbf{x}}(t) = g(\mathbf{x}(t), \mathbf{u}(t)) \tag{1}$$

where $\mathbf{x}(t) = [x(t), y(t)]^T \in X \subset \mathbb{R}^n$ is the state of the robot, $\mathbf{u}(t) \in U \subset \mathbb{R}^m$ is the control input of the continuum robot, and $g(\cdot)$ is a differentiable nonlinear state transition function. In this article, we implement the Koopman operator theory to linearize the system by lifting the system states into an infinite-dimensional function space $\mathcal{F}$. Functions in $\mathcal{F}$ are denoted as observables $f \in \mathcal{F}$, which are restricted as real-valued functions. The Koopman operator is linear and describes the evolution of f as described in [23] and [24]

$$\mathcal{K}_t f(\mathbf{x}(t), \mathbf{u}(t)) = f(g(\mathbf{x}(t), \mathbf{u}(t)), \mathbf{u}(t)) \tag{2}$$

where $\mathcal{K}_t$ is the Koopman operator. Once the system can be obtained using the Koopman operator, the system will be further transferred to the RL framework as the simulation environment during the offline policy training process. In real system experiments, the above infinite-dimensional operator is usually approximated by the finite-dimensional observable, which consists of several functions of the sampled data.

2) Observable Formulation: To approximate the Koopman operator, adequate data are required. Therefore, we randomly and continuously actuated the motors to cover most of the workspace of the continuum robot. Considering we have sampled K "snapshots": $(\mathbf{x}[k], \mathbf{u}[k], \mathbf{x}^+[k])$, where $\mathbf{x}[k]$ is the kth state measurements, $\mathbf{u}[k]$ is the kth control input, $\mathbf{x}^+[k]$ is the measurement after the state transition, and $k \in [1, K]$. To lift the sampled "snapshots" into a high-dimension space, we then define a finite-dimensional dictionary of the basis function as

$$\{d_j(\mathbf{x}, \mathbf{u})\}_{j=1}^{N+m} = \{\psi_j(\mathbf{x})\}_{j=1}^{N} \cup \{\mathbf{u}_j\}_{j=N+1}^{N+m} \tag{3}$$

where $\psi_j(\mathbf{x})$ is the state basis function, the input basis function is selected as the control input $\mathbf{u}_j$ itself, N and m represent the number of the state basis function and the input basis function, respectively. As mentioned in Section II-B1, the approximation are achieved in a subspace $\overline{\mathcal{F}} \in \mathcal{F}$ consists of N independent state basis functions ψ_j that construct the state observable

$$\boldsymbol{\psi}(\mathbf{x}) = [\psi_1(\mathbf{x}), ..., \psi_N(\mathbf{x})]^T. \tag{4}$$

Using the state observable, the input observable, and the sampled K "snapshots," we can construct two observable matrices $\boldsymbol{\Psi}_a, \boldsymbol{\Psi}_b \in \mathbb{R}^{K \times (N+m)}$

$$\boldsymbol{\Psi}_a = \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}[1])^T, \mathbf{u}[1]^T \\ ... \\ \boldsymbol{\psi}(\mathbf{x}[K])^T, \mathbf{u}[K]^T \end{bmatrix}, \boldsymbol{\Psi}_b = \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}^+[1])^T, \mathbf{u}[1]^T \\ ... \\ \boldsymbol{\psi}(\mathbf{x}^+[K])^T, \mathbf{u}[K]^T \end{bmatrix}. \tag{5}$$

So far, we have derived the state observable, the input observable, and the observable matrices. In the next part, we will introduce the Koopman operator for state-space linearization using the derived observable.

3) Koopman Operator Approximation: In our problem, we define the state observable using a one-time delay basis function, namely the Hankel EDMD. Given the kth measurement of state, $\boldsymbol{\psi}$ can be represented as

$$\begin{aligned} \boldsymbol{\psi}(\mathbf{x}[k]) &= \left[\psi_1(\mathbf{x}[k]), \psi_2(\mathbf{x}[k])\right]^T \\ &= \left[x_1[k], x_2[k], x_1[k-1], x_2[k-1]\right]^T \end{aligned} \tag{6}$$

where $x_1[k] = x[k]$ and $x_2[k] = y[k]$ represent the tip position measured in the x- and y-axis, respectively. Substitute (6) into (5) and based on the Koopman operator theory [23], we can approximate the Koopman operator as

$$\mathcal{K}_t = \boldsymbol{\Psi}_a^{\dagger} \boldsymbol{\Psi}_b \tag{7}$$

where $\mathcal{K}_t \in \mathbb{R}^{(N+m)\times(N+m)}$. As clarified in [7], the transpose of $\overline{\mathcal{K}}_{T_s}$ is the most suitable transition matrix in the L^2-norm sense, and $\overline{\mathcal{K}}_{T_s}$ is given by solving the least squares problem

$$\min_{\overline{\mathcal{K}}} \sum_{k=1}^{K} \left\| \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}^+[k]) \\ \mathbf{u}[k] \end{bmatrix} - \overline{\mathcal{K}}^T \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}[k]) \\ \mathbf{u}[k] \end{bmatrix} \right\|_2^2$$
$$\Rightarrow \min_{\overline{\mathcal{K}}} \sum_{k=1}^{K} \left\| \begin{bmatrix} \mathbf{x}^+[k] \\ \mathbf{x}^+[k-1] \\ \mathbf{u}[k] \end{bmatrix} - \overline{\mathcal{K}}^T \begin{bmatrix} \mathbf{x}[k] \\ \mathbf{x}[k-1] \\ \mathbf{u}[k] \end{bmatrix} \right\|_2^2. \quad (8)$$

4) Linear Model Approximation: We look for the real model of a specific continuum robotic system, for constructing the environment of the RL controller. Now, considering the system is linearized as

$$\boldsymbol{\psi}(\mathbf{x}^+)[j] = A\boldsymbol{\psi}(\mathbf{x})[j] + B\mathbf{u}[j]$$
$$\mathbf{x}[j] = C\boldsymbol{\psi}(\mathbf{x})[j] \quad (9)$$

where A and B are the approximated transition matrix and input matrix by EDMD. Matrix C is the matrix that retrieves the original state $\mathbf{x}(t)$ from $\boldsymbol{\psi}(\mathbf{x})(t)$. Based on the Koopman operator theory, the transition matrix $A \in \mathbb{R}^{4\times4}$ and the control input matrix $B \in \mathbb{R}^{4\times m}$ satisfied the following:

$$\min_{A,B} \sum_{k=1}^{K} \left\| \boldsymbol{\psi}(\mathbf{x}^+[k]) - (A\boldsymbol{\psi}(\mathbf{x}[k]) + B\mathbf{u}[k]) \right\|_2^2$$
$$\Rightarrow \min_{A,B} \sum_{k=1}^{K} \left\| \begin{bmatrix} \mathbf{x}^+[k] \\ \mathbf{x}^+[k-1] \end{bmatrix} - A \begin{bmatrix} \mathbf{x}[k] \\ \mathbf{x}[k-1] \end{bmatrix} - B\mathbf{u}[k] \right\|_2^2. \quad (10)$$

The most suitable transition matrix and input matrix A and B are embedded in $\overline{\mathcal{K}}_{T_s}$ shown as follows:

$$\overline{\mathcal{K}}_{T_s} = \begin{bmatrix} A_{4\times4} & B_{4\times m} \\ O_{m\times4} & I_{m\times m} \end{bmatrix} \quad (11)$$

where $I_{m\times m}$ is an identity matrix and $O_{m\times4}$ is a zero matrix. The retrieved matrix $C \in \mathbb{R}^{2\times4}$ is defined as

$$C = \begin{bmatrix} I_{2\times2} & O_{2\times2} \end{bmatrix}. \quad (12)$$

C. Real2Sim: Policy Learning Using RLfID

In this section, we introduce the multiagent reinforcement learning (MARL) algorithm and obtain the initial offline policy based on the derived system (9). To improve the training efficiency, we adopt the previously proposed MADQN strategy [1] as an imperfect demonstration in this article. Such a demonstration is incorporated into the framework using a modified epsilon-greedy exploration strategy. The proposed safety-critical MARL with imperfect demonstration is shown as follows.

1) Markov Decision Process (MDP) Formulation: The RL-based discrete time control problem is usually described as a task of searching the optimal MDP sequence. In this article, the MDP is a mathematical framework for modeling continuum robot movements and the simplest one can be categorized into state space $\mathscr{S}$, action space $\mathscr{U}$, reward function $\mathscr{R}$, transition model $\mathscr{P}$, and discount factor γ. Notably, we will use i and j to index the ith timestep and j_{th} agent in the following, respectively.

a) State Space $\mathscr{S}$: The control problem is decoupled into two agents, where agent 1 controls the Yaw-DoF and agent two controls the Pitch-DoF. To be specific, each state at ith time step is described as $s^{i,1} = d^{i,1} = x^i_{\text{tar}} - x^i_{\text{tip}}$ and $s^{i,2} = d^{i,2} = y^i_{\text{tar}} - y^i_{\text{tip}}$ for agents 1 and 2, respectively. x_{tar} and y_{tar} are the target coordinates; and x_{tip} and y_{tip} are the tip coordinates derived from the state space (9).

b) Action Space $\mathscr{U}$: The action space $\mathscr{U}$ in both agents is an interval $[u_{\min}, u_{\max}]$ range from the minimum action $u_{\min}$ to the maximum action $u_{\max}$ decided by the user. The action $u^{i,1} \in \mathscr{U}$ and $u^{i,2} \in \mathscr{U}$ represent the action of agents 1 and 2 at ith time step, respectively. Four linear modules driven by the motors control the Yaw-DoF and Pitch-DoF, where $u^{i,1} = -u^{i,4}$ controls the Yaw-DoF and $u^{i,2} = -u^{i,3}$ controls the Pitch-DoF. Deployment of each motor and corresponding action index is demonstrated in Fig. 5.

c) Reward $\mathscr{R}$: The reward function is used as an evaluation of the performance of different actions. The objective of an RL problem is to maximize the accumulated reward in a MDP. In this continuum manipulator tracking control task, the reward function of agent j at time step i is given by

$$r^{i,j}_t = -k_1|d^{i,j}_t| - k_2\Delta^{i,j}_{\text{err}} \quad (13)$$

where $\Delta^{i,j}_{\text{err}} = (d^{i,j}_t - d^{i,j}_{t-1})$, and $d^{i,j}_t$ is defined in the state space $\mathscr{S}$. If $\Delta^{i,j}_{\text{err}} < 0$, the current step tracking error is smaller than the last step error, thus a positive value of $-\Delta^{i,j}_{\text{err}}$ is added to $r^{i,j}_t$, and $\Delta^{i,j}_{\text{err}} > 0$ vice versa. The reward function implies that the closer the tip is to the target, the greater the reward function is. k_1 and k_2 are coefficients selected by the user. The same reward function is implemented in agent 2 as well.

2) Continuous MARL: In this article, we employ the multiagent deep deterministic policy gradient (MADDPG) [25] as the main part of the Real2Sim2Real framework. To cooperatively control the robot, each agent associated with a motor pair is tasked with controlling a DoF and both of the agents share their states and actions during training. Additionally, to speed up the training process in simulation, the aforementioned MADQN in Section I-A is developed as an imperfect demonstration of the framework. The algorithm is described in Fig. 1.

Each agent in the MADDPG holds an actor–critic framework, where the actor is an action selector and the critic is a DQN-like Q-value generator. The critic of agent j is constructed by two neural networks (NNs), namely Q-network and target Q-network, and updated by

$$\mathscr{L}(\psi_j) = \frac{1}{N}\sum_i [(Q_j^{(\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2)}(s^1, s^2, u^1, u^2) - y^j)^2] \quad (14)$$

where N is the number of the transitions sampled from the replay buffer, $y^j = r^j + \gamma Q_j^{(\boldsymbol{\sigma}'_1,\boldsymbol{\sigma}'_2)}(s^{1'}, s^{2'}, u^{1'}, u^{2'})|_{u^{j'}=\boldsymbol{\sigma}'_j}$; r^j is the ith reward for agent j; r^j, s^j, and u^j are the abbreviation of $r^{i,j}$, $s^{i,j}$, and $u^{i,j}$; $\boldsymbol{\sigma}_j$ is the abbreviation of $\boldsymbol{\sigma}_{\psi_j}$: policy w.r.t. parameter ψ_j; $\boldsymbol{\sigma}_{j'}$ is the abbreviation of $\boldsymbol{\sigma}_{\psi'_j}$: target policy w.r.t. parameter ψ'_j; and $Q_j^{(\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2)}$ is the centralized action-value function approximated by the Q-network. The above abbreviations

are utilized in (15) as well. The deterministic policy is updated by the following policy gradient:

$$\nabla_{\psi_j} J(\boldsymbol{\sigma}_j) = \frac{1}{N}\sum_i [\nabla_{\psi_j}\boldsymbol{\sigma}_j(s^j) \nabla_{u_j} Q_j^{\boldsymbol{\sigma}}(s^1, s^2, u^1, u^2)|u^j = \boldsymbol{\sigma}_j(s^j)]. \tag{15}$$

Equation (14) makes use of a recursive representation to update the Q function for evaluating the quality of the current policy. Then, agent j updates its latest deterministic policy using (15). At each iteration step, (14) and (15) are calculated based on the sampled N transitions from the replay buffer.

3) Imperfect Demonstration: In RL, the agent is required to explore the environment randomly and exploit the collected information for decision-making. The most common approach to balance exploration and exploitation is the epsilon-greedy strategy

$$\mathbf{u}^i_{\text{MARL}} = \begin{cases} \mathcal{N}(0, \delta^2), & \text{if } \epsilon > \epsilon_{\text{th}} \\ [\sigma_1(s^{i,1}), \sigma_2(s^{i,2})] & \text{else} \end{cases} \tag{16}$$

where ϵ decays during the learning process. Once $\epsilon < \epsilon\text{th}$, the algorithm stops exploring and begins following the policy generated by the proposed algorithm in the following learning and execution process. However, such a random exploration procedure is not suitable for a control process, since most of the exploration is unnecessary. To improve the learning efficiency, an imperfect demonstration is introduced to guide the sampling process. Based on the demonstration, we propose a modified epsilon-greedy strategy as

$$\mathbf{u}^i_{\text{MARL}} = \begin{cases} \mathbf{u}^i_D + \mathcal{N}(0, \delta^2) \sim \mathcal{N}(\mathbf{u}^i_D, \delta^2), & \text{if } \epsilon > \epsilon\text{th} \\ [\sigma_1(s^{i,1}), \sigma_2(s^{i,2})], & \text{else} \end{cases} \tag{17}$$

where $\mathbf{u}^i_D = \{u^{i,1}_D, u^{i,2}_D\}^T$ is the demonstrated action at ith timestep. Based on $\mathbf{u}^i_D$, we sample the actions around $\mathbf{u}^i_D$ following a Gaussian distribution $\mathcal{N}(\mathbf{u}^i_D, \delta^2)$. The standard deviation δ decides the exploration range. To derive the demonstration, we take advantage of the saved MADQN with an action space of $U_D = \{-0.2, -0.1, 0.1, 0.2\}$ in [1] to accelerate the learning process. The input $d^{i,1} = x^i_{\text{target}} - x^i_{\text{tip}}$, $d^{i,2} = y^i_{\text{target}} - y^i_{\text{tip}}$, and the output actions of the MADQN are aligned with those used in the MADDPG algorithm presented in this article. The architecture and parameters of the NNs within MADQN have been previously stored in an h5 file, as detailed in [1]. Given that the input is the tip position error, even though the two robots hold two different models, the MADQN can generate actions $\mathbf{a}_D$ with correct actuation tendencies. The RLfID algorithm is summarized in Algorithm 1.

4) Safety-Critical MARL: The last key problem addressed in Section I-3 pertains to ensuring the safety of the robot during task execution. In manipulation tasks, especially surgical procedures, it is critical to avoid obstacles at the tip such that surgical tools can reach the lesion without causing tissue damage. If an obstacle appears along the trajectory, the robotic tip needs to avoid direct contact and then resume tracking its target

Algorithm 1: Safety-Critical RLfID.

Initialization: Random exploration process $\mathcal{N}$; Replay buffer $\mathcal{D}$; $\epsilon = 1$; Agent 1 and Agent 2 for Yaw-DoF and Pitch-DoF control.

for $t = 1, 2, \ldots M$ **do**
  Initialization: state $\mathbf{s} = [s_1, s_2]$; reward $\mathbf{r} = [r_1, r_2]$.
  for $i = 1, 2, \ldots, T$ **do**
    Derive the Demonstration $\mathbf{a}^i_D$ from MADQN
    Action selection from MADDPG $\mathbf{a}^i_{MARL} = \begin{cases} \mathbf{a}^i_D + \mathcal{N}(0, \delta^2), & \text{if } \epsilon > \epsilon_{threshold} \\ [\sigma_1(s^{i,1}), \sigma_2(s^{i,2})], & else \end{cases}$
    Derive a safety-critical action $\mathbf{a}^i_s$ using Eq. 10.
    Execute $\mathbf{a}^i_s$ on the identified system 9.
    Observe reward r_1, r_2, and new states s'_1, s'_2 for Agent 1 and Agent 2.
    Store transition $(\mathbf{s}^i, \mathbf{a}^i_s, \mathbf{r}^i, \mathbf{s}^{i'})$ in $\mathcal{D}$
    Sample mini-batches from $\mathcal{D}$ for updating the NNs of Agent 1 and Agent 2.
    $\epsilon = \epsilon * \epsilon_{decay}$.
  end for
end for

trajectory after bypassing the obstacle. We deploy a safety-critical algorithm based on QP to further optimize the control signals generated by RL in the presence of obstacles. This optimization is particularly crucial when considering a group of sphere obstacles $O_b = \{O^1_b, O^2_b, ..., O^i_b\}$ intersecting with the desired trajectory of the robot, as illustrated in Fig. 4. The primary purpose of the safety-critical control is to maintain a safe distance from the tip to the obstacles greater than the radius r_o of the sphere obstacles. Furthermore, the RL policy is regarded as the reference signal in the QP process, and the optimized policy is required to closely align with the RL policy $\mathbf{u}_{\text{MARL}}$ while ensuring obstacle avoidance. The safety-critical QP problem is formulated as

$$\begin{aligned} \min_{\mathbf{u}^i_s} \quad & \|\mathbf{u}^i_s - \mathbf{u}^i_{\text{MARL}}\|^2 \\ \text{s.t. } & \boldsymbol{\psi}(\mathbf{x}^+)[i] = A\boldsymbol{\psi}(\mathbf{x})[i] + B\mathbf{u}^i_s \\ & \|\mathbf{x}^+[i+1] - x_{ob}\| \geq r_o + \tau \\ & \mathbf{x}[i] = C\boldsymbol{\psi}(\mathbf{x})[i] \end{aligned} \tag{18}$$

where $\mathbf{u}^i_s = [u^{i,1}_s, u^{i,2}_s]$ is the safety-critical action that closely align with $\mathbf{u}^i_{\text{MARL}}$ at ith control step, and τ is the tolerance of the distance to the obstacles. Notably, the initial condition of the QP is $\mathbf{u}^i_s = \mathbf{u}^i_{\text{MARL}}$ so that the optimization consumption of each step can also be minimized to accommodate the real-time control requirements.

D. Sim2Real: Policy Transfer and Online Update

In the literature [7], [23], and [24], traditional controllers such as linear quadratic regulator (LQR) and model predictive controller (MPC) were applied to control continuum robots after deriving Koopman operator based models. However, these controllers could not update their control policy online and

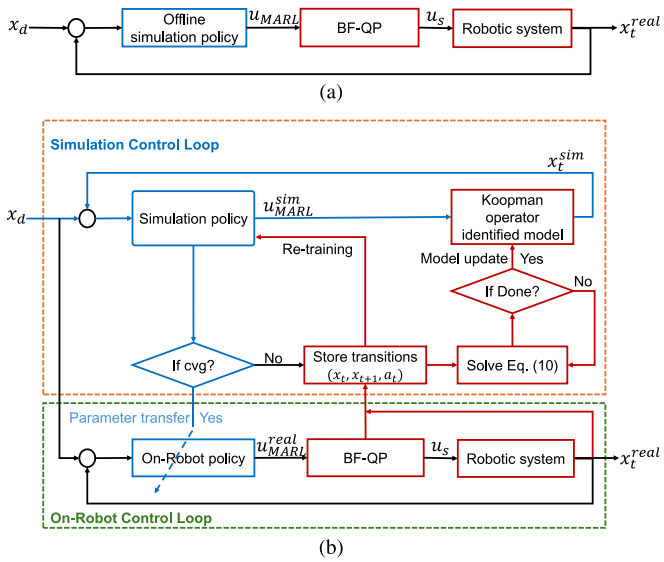

Fig. 2. Sim2Real policy transfer. (a) Offline policy transfer. (b) Online policy transfer with both model and policy update.

adapt to environmental changes. To deal with this situation, online learning is required for highly deformable continuum robots. In recent years, there has been a growing interest in exploring online inverse model updates based on regression approaches for soft continuum robots [26], [27], [28], where the current focus of online approaches in continuum robots remains primarily on model adaptation rather than both model and control policy adaptation.

RL, as a paradigm for designing the learning-based controller, enables online modification of control policies by leveraging experimental data, but can exhibit low efficiency in control policy training and Sim2Real transfer. In this section, we propose an efficient Sim2Real policy transfer framework for online policy modification. A safety-critical QP scheme is further introduced to achieve tip obstacle avoidance.

1) Sim2Real Transfer: Leveraging the system model described in (9) and the optimal RL policy outlined in (15), we initially transfer the offline trained policy to the robot without any update. To address online continuous disturbance, such as external payload, we propose an online Sim2Real transfer framework.

a) Offline transfer: We extract the trained policy from the simulator and employ closed-loop control that utilizes feedback from the tip positions. Notably, the policy does not undergo further updates during the control process. Therefore, the offline RL policy cannot adapt to environmental changes. The offline transfer framework is demonstrated in Fig. 2(a).

b) Online transfer: Different from offline transfer, the online approach involves continuous updates of the model and policy throughout its control process. As demonstrated in Fig. 2(b), two control loops parallelly formulate the online transfer framework.

In the "simulation loop," the environment (9) is continuously updated using the experimental data. Furthermore, since the environment of the "simulation loop" is changed, the simulation policy will be retrained in the simulator. It should be noted that the update process does not proceed in real time. The experimental data are stored in the online memory pool. When the tracking error is greater than the tolerance error, we recognize this situation as the external payload is applied to the robotic body, then the online memory pool will be triggered to collect the latest data. If the memory pool is filled with the latest data, the environment update process will proceed. When the solving process of (10) is finished, the simulation environment will be updated, and the retraining process of the RL will proceed.

Once the RL in the simulation is converged, the online obtained RL policy will be transferred to the "on-robot control loop" to achieve the online Sim2Real transfer. The "on-robot control loop" is the loop that applies the RL policy to the robot. If the "simulation loop" undergoes the online retraining and the policy is not converged, the "on-robot control loop" will utilize the last obtained RL policy to control the robot.

III. Results

A. Simulation Tests

The proposed framework was tested in a simulation environment first to evaluate the feasibility of its application on a continuum robot. The simulation environment was created using the Koopman operator-based identification approach as described in Section II-B1. The simulation tests were performed in Python 3.9 and the NNs were constructed using Tensorflow 2.9.1.

In the simulation, we recorded the accumulated reward of the proposed framework and compared it to the traditional algorithms as demonstrated in Fig. 3. We further tested the tracking as well as the obstacle avoidance as shown in Fig. 4. At the beginning of a tracking task, the robot sampled around the actions generated by the imperfect demonstrations $\mathbf{a}_D$, which made the tip trajectories oscillate around the target trajectories as illustrated in Fig. 4. Once the training process was finished, the robot no longer oscillated and tracked the target trajectories as expected. In addition, we observed that the proposed QP-based obstacle avoidance strategy was able to remove the tip from the area of the obstacles. It should be noted that the QP controller worked during the whole process, even when the robot was in oscillation at the very beginning.

B. Sim2Real Tests

1) Experiment Setup: The setup consisted of four Maxon motors with incremental encoders, four EPOS4 controllers, four linear modules, a stereo visual tracker, and a PC. A red color marker was attached to the tip of the manipulator. When the tip moved, the stereo visual tracker could capture the motions by color detection for real-time closed-loop control. All data were collected by MATLAB 2020b (Mathworks, USA). The proposed RL framework was trained and implemented using Python 3.9 and TensorFlow 2.9.1.

Four cables were utilized to transmit the actuation force from the motors to the continuum robot. One end of the cable was fixed at the tip of the robot, and the other end was fixed at the

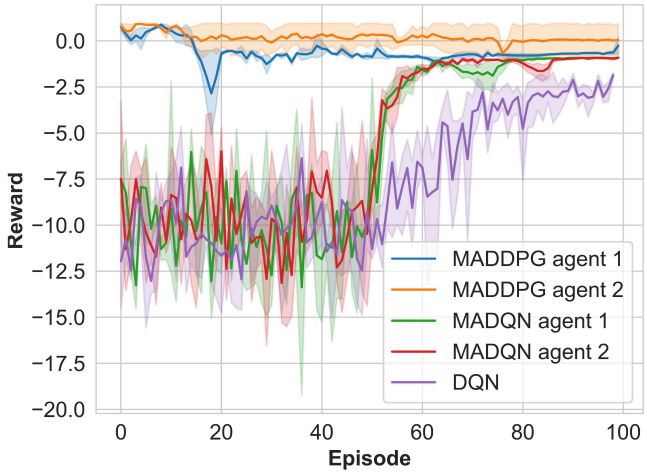

Fig. 3. Accumulated reward comparison among the proposed algorithm, MADQN, and DQN. With the imperfect demonstration, the proposed MARL algorithm can achieve a high reward from the very beginning of the training process, and the vibration disappears in a short training time.

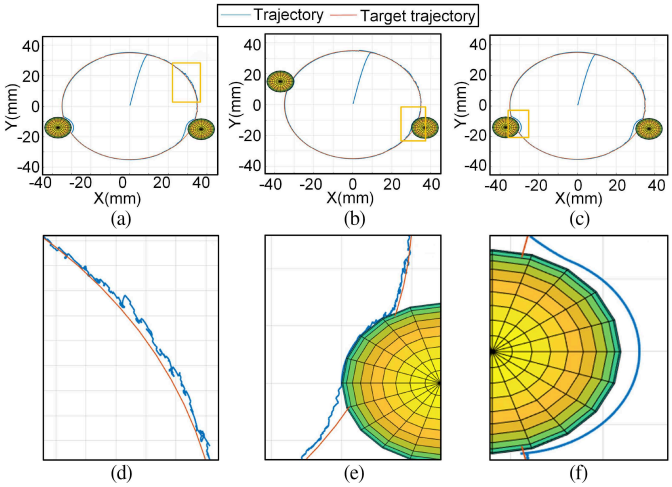

Fig. 4. Simulation results of trajectory tracking as well as obstacle avoidance. (a)–(c) Circular tracking and obstacle avoidance tasks. (d) Zoom-in view demonstrates the sampling procedure around imperfect demonstrations. (e) Zoom-in view demonstrates the obstacle avoidance scheme works even at the training stage. (f) Zoom-in view shows a smooth obstacle avoidance after the training process is finished.

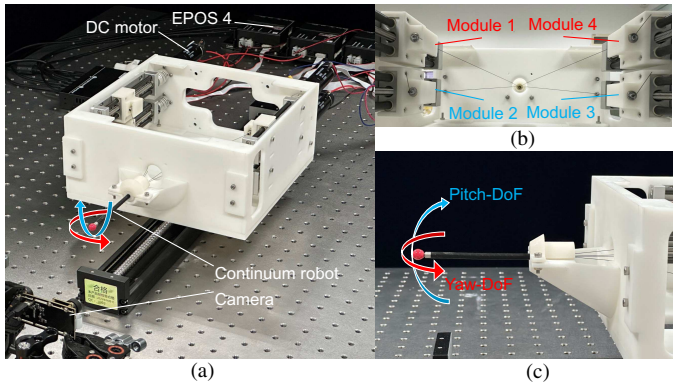

Fig. 5. System layout. (a) Overall configuration. (b) Yaw-DoF: module 1 and module 4; and pitch-DoF: module 2 and module 3. (c) Side view.

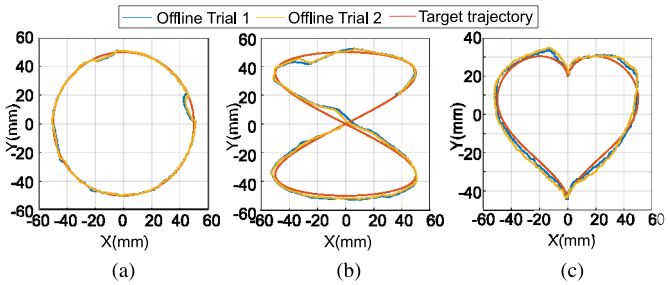

Fig. 6. Experiment results of offline tracking in the shape of: (a) “circle” figure; (b) “eight” figure; and (c) “heart” figure, respectively. Some uncertain behaviors occurred during closed-loop offline tracking.

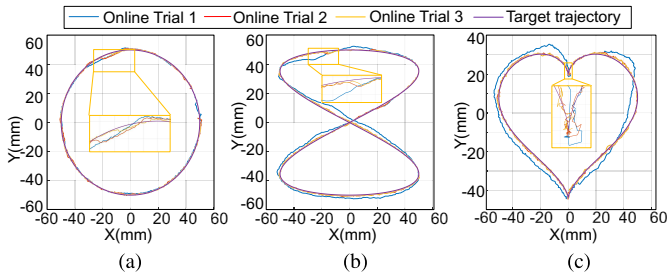

Fig. 7. Experiment results of online tracking in the shape of: (a) “circle” figure; (b) “eight” figure; and (c) “heart” figure, respectively. From (a)–(c), we can find the trajectory of online trial 3 is closer to the target trajectory than the first two trials.

linear module. When the linear module translated backward, the cable was pulled to bend the robot in a certain direction. In contrast, if the linear module moved forward, the cable was released, and the robot would recover its shape. As shown in Fig. 5, module 1 and module 4 controlled the Yaw-DoF; module 2 and module 3 controlled the Pitch-DoF.

2) Offline Sim2Real Transfer in Free Space: Using the derived RL policy in Section III-A, we first performed circular, eight-figure, and heart-figure three different tracking tests on a real continuum robot without updating the control policy. The RMSEs of the three tasks are calculated to be 4.02, 2.25, and 3.56 mm, respectively, from Fig. 6(a)–6(c). Although we utilized the Koopman operator-based identification approach to model the nonlinear behavior, however, we found that some special behaviors such as hysteresis (backlash) and dead-zone, occurred while switching moving directions, which still could not be captured by the model. Thus, we considered the tracking error significant when changing the moving directions as shown in Fig. 6, resulting in the maximum errors of 6.40, 7.11, and 6.52 mm, respectively, from Fig. 6(a)–6(c).

3) Online Sim2Real Transfer in Free Space: From the offline Sim2Real transfer tests, as shown above, we can tell that without any updating of the policy derived from the simulation, a significant tracking error still exists. To further improve the control performance, we updated the control policy during

TABLE I
ONLINE TRACKING RMSE WITHOUT PAYLOAD

Online Learning w/o Payload RMSE (mm)												
Trial	Circle				Eight				Heart			
	X	Y	Euclidean	↑	X	Y	Euclidean	↑	X	Y	Euclidean	↑
1	2.56	3.75	4.54	/	5.24	3.40	6.25	/	2.34	3.89	4.54	/
2	1.98	1.79	2.67	41.19%	0.76	0.65	1.00	84.00%	0.22	0.39	0.45	90.09%
3	0.37	0.48	0.61	87.56%	0.55	0.53	0.76	87.84%	0.21	0.29	0.36	92.07%

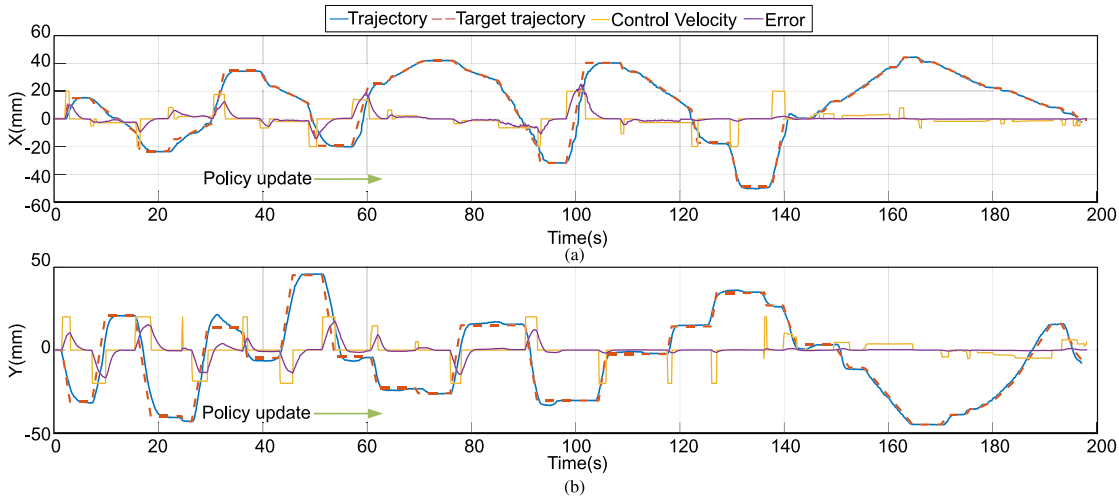

Fig. 8. Remote control test of the online control policy using a joystick. The tracking error represented by the purple line turns out to be smaller when the control period increases.

tracking so that the hysteresis behavior could be learned while tracking. The online tracking results are shown in Fig. 7, where each task was repeated three times in a single control loop. We can observe that the tracking RMSE decreased from trial 1 (offline policy) to trial 3 for each tracking task as shown in Table I, where the "↑" represents the improvement of trial 2 and trial 3 compared to trial 1. To be more straightforward, the plotted trajectories as shown in Fig. 7 demonstrate that all of the trial 3 among the three tasks are closer to the target trajectories than all of the trial 1. It is obvious that online tracking can further improve the tracking performance during tracking based on Fig. 7 and Table I.

Additionally, we tested the online transfer using a remote controller to send reference trajectories to the robot. The control policy was still using the policy derived from simulation, and it was updated in the whole control process. The tracking results are shown in Fig. 8, with the blue line and dashed red line representing the real and target trajectories along the x- and y-axis, respectively. The purple lines and yellow lines are tracking errors and remote controller control velocity. The maximum displacements along the x- and y-axis were limited from -50 to $50\,\mathrm{mm}$ for safety reasons. As shown in Fig. 8(a) and 8(b), the tracking error decreases against time, which means the updated policy further improves the tracking performance continuously.

4) Online Tracking Tests Under Payloads: In this experiment, we applied 20-g weight on the body of the continuum robot and still commended the robot to track a circular trajectory and an "eight" figure trajectory. The tracking processes were repeated three times for each figure as well. The RMSE was reported in Table II. From Fig. 9, we can tell that the start point of the tracking process was removed to $(0, -34.56)$ because of the weight. Furthermore, although an external payload was applied to the body of the robot, the derived policy still followed the target trajectories as expected, and the tracking policy can be updated using the online RL.

TABLE II
ONLINE TRACKING RMSE WITH PAYLOAD

Online Learning w/ Payload RMSE (mm)								
Trial	Circle				Eight			
	X	Y	Euclidean	↑	X	Y	Euclidean	↑
1	4.12	12.12	12.80	/	5.49	4.54	7.12	/
2	0.81	0.75	1.10	91.41%	2.69	1.84	3.26	54.21%
3	0.42	0.49	0.65	94.92%	1.25	1.06	1.64	76.97%

We then commanded the robot to track a circle and placed a 20-gram payload at around 45 s of the tracking process. When the payload was applied to the robotic body, the impulse contributed to a large deviation in the tip position. As shown in Fig. 10, the online tracking with imperfect demonstration is illustrated by the red line. With the demonstration, the tracking error can be converged in 15 s after applying the payload. However, for the online policy update procedure without using the demonstration according to the blue line, the tracking error can only be converged in 60 s after applying the payload. To further

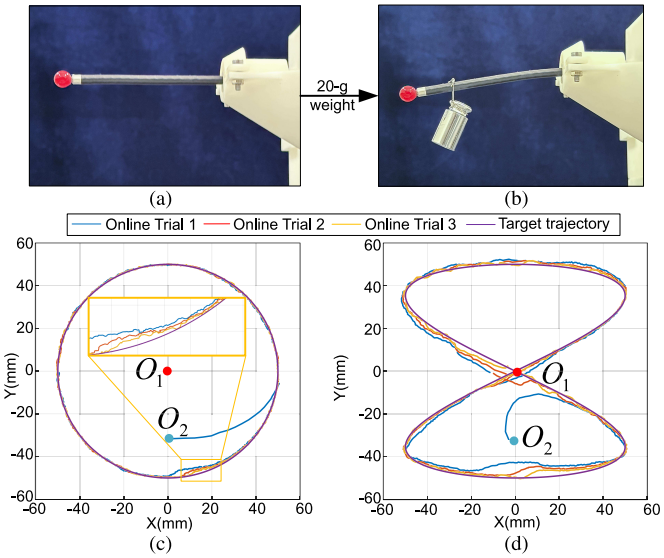

Fig. 9. Tracking under external payload. (a) and (b) 20-g weight results in the deformation of the continuum robot. (c) Online circle tracking under 20-g weight. (d) Online "eight" figure tracking under 20-g weight. Also, the start points of (c) and (d) were moved to O_2 from O_1 by the weight.

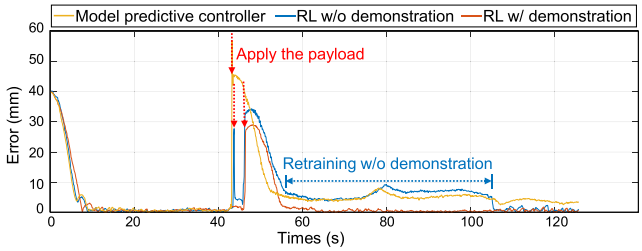

Fig. 10. Comparison test among the MPC, RL without demonstration, and the proposed RL with demonstration framework. A payload is applied to the robot in the middle of tracking tasks.

demonstrate the superiority of the proposed online policy update framework, we have compared it to the MPC in the payload tracking tests. The model utilized in the MPC was the offline obtained kinematics (9). The predictive horizon was selected as a three-step horizon. The tracking error of using the MPC is illustrated by the yellow line in Fig. 10. When the payload was not applied to the robot (0–45 s), the MPC could track the target trajectory with a submillimeter error. However, when the payload was applied to the robot, the converged tracking error of the MPC increased to around 4.67 mm.

5) Obstacle Avoidance Tests: To verify the obstacle avoidance ability of the tip, we virtually placed two sphere obstacles on the target trajectories. The safety-critical QP controller was solved throughout the whole tracking process. The tracking error and distance to obstacles are shown in Fig. 11. When $50 < t < 75$ and $150 < t < 170$ (t is the control timestep), we can tell that the robot was encountering obstacles, and the QP controller was triggered. It is obvious that the safety-critical control scheme helps the robot to avoid contact of obstacles

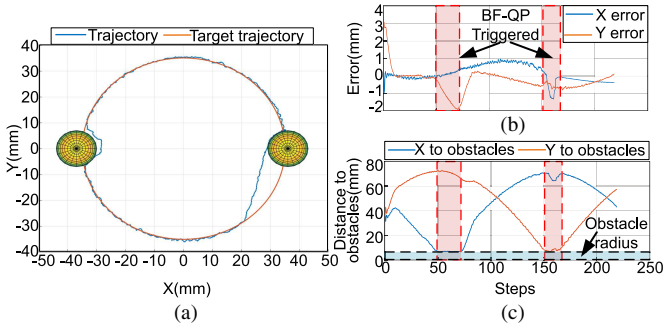

Fig. 11. Experiment results of trajectory tracking as well as obstacle avoidance. (a) Tracking results. (b) Blue line represents the tracking error along the x-axis; the red line represents the tracking error along the y-axis. (c) Blue line represents the Euclidean distance to O_b^1; the red line represents the Euclidean distance to O_b^2.

and the distance from tip to obstacles is always greater than the radius of the obstacles.

IV. Discussion and Conclusion

In this article, we presented a MARL-based Real2Sim2Real framework for decentralized control of each DoF of a mesoscale continuum robot. Motivated by noting that the lack of real-world data in simulations was a significant aspect of the Sim2Real transfer gap, we collected a set of real-world data and employed it to derive a Koopman operator-based model serving as the simulation environment. Subsequently, we implemented the real-world model in simulations, where we achieved an extremely efficient lightweight training process. This policy was then transferred to a real continuum robot platform. The online policy tracking results demonstrate the potential of online RL in the presence of external disturbances.

As recent works on RL-based control of continuum robots did not consider the implementation of online RL in continuum robots control, we achieved both offline control and online control in this work. We tested the online control performance by tracking each target trajectory for three trials. The Euclidean tracking RMSE in free space was decreased from 5.11 to 0.58 mm (89.16% improvement) on average by using continuously updated RL policy. Furthermore, we applied a 20-g weight on the robot, resulting in a large deflection during experiments. Subsequently, we conducted three trials of the tracking tests. The Euclidean RMSE was improved on average by 85.70% through online updating, ultimately resulting in submillimeter tracking accuracy. The experiment of placing the payload during the tracking process can further explain the effectiveness of the proposed imperfect demonstration. The difference between applying a payload before tracking and during tracking is that the latter setup can result in an impulse effect on the system. This will help to explain the capability of dealing with disturbances using our approach. Based on the imperfect demonstration, the duration of the online retraining process can be decreased by 75% compared to without using the demonstration. Furthermore, the compared predictive controller failed to track the target trajectory with a submillimeter tracking error

after applying the payload in the tracking process, due to the offline obtained model was not accurate anymore. Combined with the QP-based obstacle avoidance controller, the robotic tip was able to track trajectories and avoid obstacles. However, in this work, the performance of the proposed approach is limited by the two-DoF continuum robot, as we can only achieve task space tracking as well as obstacle avoidance without considering orientation. Last but not least, the training of the proposed framework remained a high accumulated reward and converged much faster than the other algorithms due to the imperfect demonstrations. Since the imperfect demonstration networks have been stored in h5 files in our previous work, the demonstration action can be directly retrieved by loading the files, with no additional computational steps required. As a result, the demonstration does not affect the real-time performance of the controller.

In the future, the work will be extended to efficient RL-based motion planning and control of multisegment continuum robots. The safety-critical framework will be enhanced for a multisegment continuum robot insertion task. This will enable the robot to perform the insertion procedure without making contact with its surrounding environment. With more controllable DoF, the robot will be able to track joint position and orientation while avoiding obstacles, ensuring that the tip can point toward the target during tracking. We are also interested in motion generations of multisegment continuum robots using the MARL framework.

REFERENCES

[1] G. Ji et al., "Towards safe control of continuum manipulator using shielded multiagent reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 7461–7468, Oct. 2021.

[2] S. Satheeshbabu, N. K. Uppalapati, G. Chowdhary, and G. Krishnan, "Open loop position control of soft continuum arm using deep reinforcement learning," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, Piscataway, NJ, USA: IEEE, 2019, pp. 5133–5139.

[3] W.-B. Li, W.-M. Zhang, H.-X. Zou, Z.-K. Peng, and G. Meng, "A fast rolling soft robot driven by dielectric elastomer," *IEEE/ASME Trans. Mechatron.*, vol. 23, no. 4, pp. 1630–1640, Aug. 2018.

[4] P. Schegg et al., "SofaGym: An open platform for reinforcement learning based on soft robot simulations," *Soft Robot.*, vol. 10, no. 2, pp. 410–430, 2023.

[5] M. Goharimanesh, A. Mehrkish, and F. Janabi-Sharifi, "A fuzzy reinforcement learning approach for continuum robot control," *J. Intell. Robotic Syst.*, vol. 100, pp. 809–826, 2020.

[6] H. Wang, W. Liang, B. Liang, H. Ren, Z. Du, and Y. Wu, "Robust position control of a continuum manipulator based on selective approach and Koopman operator," *IEEE Trans. Ind. Electron.*, vol. 70, no. 12, pp. 12522–12532, Dec. 2023.

[7] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Koopman-based control of a soft continuum manipulator under variable loading conditions," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 6852–6859, Oct. 2021.

[8] J. Liu, Z. Huang, X. Xu, X. Zhang, S. Sun, and D. Li, "Multi-kernel online reinforcement learning for path tracking control of intelligent vehicles," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 51, no. 11, pp. 6962–6975, Nov. 2021.

[9] N. Wang, Y. Gao, H. Zhao, and C. K. Ahn, "Reinforcement learning-based optimal tracking control of an unknown unmanned surface vehicle," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 7, pp. 3034–3045, Jul. 2021.

[10] Y. Wen, J. Si, A. Brandt, X. Gao, and H. H. Huang, "Online reinforcement learning control for the personalization of a robotic knee prosthesis," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2346–2356, Jun. 2020.

[11] J. Liu et al., "An underwater robotic system with a soft continuum manipulator for autonomous aquatic grasping," *IEEE/ASME Trans. Mechatron.*, vol. 29, no. 2, pp. 1007–1018, Apr. 2024.

[12] W. Ba, X. Dong, A. Mohammad, M. Wang, D. Axinte, and A. Norton, "Design and validation of a novel fuzzy-logic-based static feedback controller for tendon-driven continuum robots," *IEEE/ASME Trans. Mechatron.*, vol. 26, no. 6, pp. 3010–3021, Dec. 2021.

[13] X. Li, H. Yue, D. Yang, K. Sun, and H. Liu, "A large-scale inflatable robotic arm toward inspecting sensitive environments: Design and performance evaluation," *IEEE Trans. Ind. Electronics*, vol. 70, no. 12, pp. 12486–12499, Dec. 2023.

[14] Q. Gao, G. Ji, and Z. Sun, "Robust shape-distortion prediction for a wire-driven hyper-redundant manipulator with engaging rolling joints," *IEEE Trans. Med. Robot. Bionics*, vol. 5, no. 2, pp. 265–277, May 2023.

[15] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Piscataway, NJ, USA: IEEE, 2020, pp. 737–744.

[16] D. Horváth, G. Erdős, Z. Istenes, T. Horváth, and S. Földi, "Object detection using sim2real domain randomization for robotic applications," *IEEE Trans. Robot.*, vol. 39, no. 2, pp. 1225–1243, Apr. 2023.

[17] E. Candela, L. Parada, L. Marques, T.-A. Georgescu, Y. Demiris, and P. Angeloudis, "Transferring multi-agent reinforcement learning policies for autonomous driving using sim-to-real," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Piscataway, NJ, USA: IEEE, 2022, pp. 8814–8820.

[18] C. Xiao, P. Lu, and Q. He, "Flying through a narrow gap using end-to-end deep reinforcement learning augmented with curriculum learning and sim2real," *IEEE Trans. NNs Learn. Syst.*, vol. 34, no. 5, pp. 2701–2708, May 2023.

[19] R. Morimoto, M. Ikeda, R. Niiyama, and Y. Kuniyoshi, "Characterization of continuum robot arms under reinforcement learning and derived improvements," *Front. Robot. AI*, vol. 9, 2022, Art. no. 895388.

[20] S. Satheeshbabu, N. K. Uppalapati, T. Fu, and G. Krishnan, "Continuous control of a soft continuum arm using deep reinforcement learning," in *Proc. 3rd IEEE Int. Conf. Soft Robot. (RoboSoft)*, Piscataway, NJ, USA: IEEE, 2020, pp. 497–503.

[21] R. Morimoto, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi, "Model-free reinforcement learning with ensemble for a soft continuum robot arm," in *Proc. IEEE 4th Int. Conf. Soft Robot. (RoboSoft)*, Piscataway, NJ, USA: IEEE, 2021, pp. 141–148.

[22] Y. Li, X. Wang, and K.-W. Kwok, "Towards adaptive continuous control of soft robotic manipulator using reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Piscataway, NJ, USA: IEEE, 2022, pp. 7074–7081.

[23] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using Koopman operator theory," *IEEE Trans. Robot.*, vol. 37, no. 3, pp. 948–961, Jun. 2021.

[24] D. A. Haggerty et al., "Control of soft robots with inertial dynamics," *Sci. Robot.*, vol. 8, no. 81, 2023, Art. no. eadd6864.

[25] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6382–6393.

[26] K. Leeetal, "Nonparametric online learning control for soft continuum robot: An enabling technique for effective endoscopic navigation," *Soft Robot*, vol. 4, no. 4, pp. 324–337, 2017.

[27] J. D. Ho et al., "Localized online learning-based control of a soft redundant manipulator under variable loading," *Adv. Robot.*, vol. 32, no. 21, pp. 1168–1183, 2018.

[28] G. Fang et al., "Vision-based online learning kinematic control for soft robots using local Gaussian process regression," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1194–1201, Apr. 2019.

Guanglin Ji received the M.S. degree in mechanical and automation engineering from The Chinese University of Hong Kong, Hong Kong SAR, China, in 2020. He is currently working toward the Ph.D. degree in computer and information engineering with The Chinese University of Hong Kong, Shenzhen, China.

He was a Research Assistant with the Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, from 2020 to 2021. His research interests include robotics control and motion planning.

Qian Gao received the M.E. degree from Hohai University, Nanjing, China, in 2018, and the M.S. degree from the City University of Hong Kong, Hong Kong, China, in 2019, both in mechanical engineering. He is currently working toward the Ph.D. degree in computer and information engineering with The Chinese University of Hong Kong, Shenzhen, China.

He was a Visiting Scholar with the Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Hong Kong SAR, in 2019. His research interests include surgical robotics and mechatronics in medicine.

Mr. Gao serves as a reviewer of IEEE TRANSACTIONS ON MEDICAL ROBOTICS AND BIONICS as well as *International Journal of Medical Robotics* and *Computer Assisted Surgery*.

Yin Xiao received the B.E. degree in mechanical engineering from the Southern University of Science and Technology, Shenzhen, China, in 2020, and the M.E. degree from Huazhong University of Science and Technology, Wuhan, China, in 2022, both in mechanical engineering.

Currently, he is an Engineer with The Chinese University of Hong Kong, Shenzhen. His research interests include robotics design, control, and sensing.

Zhenglong Sun (Senior Member, IEEE) received the B.E. degree in biomedical engineering and the M.E. and Ph.D. degrees in mechatronic system and design from Nanyang Technological University, Singapore, in 2005, 2008, and 2014 respectively.

Currently, he is an Assistant Professor with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China. He serves as the Deputy Director of the Healthcare Robotics Center, Shenzhen Institute of Artificial Intelligence and Robotics for Society, The Chinese University of Hong Kong. He has published over 80 journal and conference papers. His research interests include medical/surgical robot, haptic sensing and feedback, and human–machine interaction.

Dr. Sun serves as a reviewer for a number of journals and conferences, such as IEEE/ASME TRANSACTIONS ON MECHATRONICS, IEEE TRANSACTIONS ON ROBOTICS, IEEE TRANSACTIONS ON MEDICAL ROBOTICS AND BIONICS, ICRA, IROS, IJCAI, and AAAI.