2024

PhysicsInformed Neural Networks to Model and Control Robots: A Theoretical and Experimental Investigation

Liu, J

https://pearl.plymouth.ac.uk/handle/10026.1/22373

Physics-informed Neural Networks to Model and Control Robots: a Theoretical and Experimental Investigation

Jingyue Liu[1,*] *Pablo Borja*[2] *Cosimo Della Santina*[1,3]

[1]Department of Cognitive Robotics, Delft University of Technology, Delft 2628 CD, The Netherlands `{J.Liu-14, C.DellaSantina}@tudelft.nl`

[2]School of Engineering, Computing and Mathematics, University of Plymouth, Plymouth PL4 8AA, United Kingdom. `pablo.borjarosales@plymouth.ac.uk`

[3]Institute of Robotics and Mechatronics German Aerospace Center (DLR) Oberpfaffenhofen 82234, Germany

[*]Corresponding author

This work concerns the application of physics-informed neural networks to the modeling and control of complex robotic systems. Achieving this goal required extending Physics-informed Neural Networks to handle non-conservative effects. We propose to combine these learned models with model-based controllers originally developed with first-principle models in mind. By combining standard and new techniques, we can achieve precise control performance while proving theoretical stability bounds. These validations include real-world experiments of motion prediction with a soft robot and of trajectory tracking with a Franka Emika Panda manipulator.

1 Introduction

Deep Learning (DL) has made significant strides across various fields, with robotics being a salient example. DL has excelled in tasks such as vision-guided navigation [1], grasp-planning [2], human-robot interaction [3], and even design [4]. Despite this, the application of DL to generate motor intelligence in physical systems remains limited. Deep Reinforcement Learning, in particular, has shown the potential to outperform traditional approaches in simulations [5–7]. However, its transfer to physical applications has been primarily hampered by the prerequisite of pre-training in a simulated environment [8–10].

The central drawback of general-purpose DL lies in its sample inefficiency, stemming from the need to distill all aspects of a task from data [11, 12]. In response to these challenges, there's a rising trend in robotics to specifically incorporate geometric priors into data-driven methods to optimize the learning efficiency [13–15]. This approach proves especially advantageous for high-level tasks that need not engage with the system's physics.

Physics-informed neural networks (PINNs) [16–18], infusing fundamental physics knowledge into their architecture and training, have found success in various fields outside robotics, from earth science to materials science [19–22]. In robotics, integration of Lagrangian or Hamiltonian mechanics with deep learning has yielded models like Lagrangian Neural Networks (LNNs) [23], and Hamiltonian Neural Networks (HNN) [24]. Several extensions have been proposed in the literature, for example, including contact models [25], or proposing graph formulations [26]. The potential of LNNs and HNNs in learning the dynamics of basic physical systems has been demonstrated in various studies [18, 27–29]. However, the exploration of these techniques in modeling intricate robotic structures, especially with real-world data, is still in its early stages. Notably, [30] applied these methods to a position-controlled robot with four degrees of freedom, which represents a relatively less complex system in comparison to contemporary manipulators.

This work deals with the experimental application of PINN to rigid and soft continuum robots [31]. Such endeavor required modifying LNN and HNN to fix three issues that prevented their application to these systems: (i) the lack of energy dissipation mechanism, (ii) the assumption that control actions are collocated on the measured configurations, (iii) the need for direct acceleration measurements, which are non-causal and require numerical differentiation. For issue (iii), we borrow a strategy proposed in [32,33], which relies on forward integrating the dynamics, while for (i) and (ii), we propose innovative solutions.

Furthermore, we exploit a central advantage of LNNs and HNNs compared to other learning techniques; the fact that the learned model has the mathematical structure that is usually assumed in robots and mechanical systems control. By forcing such a representation, we use model-based strategies originally developed for first principle models [34–36] to obtain provably stable performance with guarantees of robustness.

The use of PINNs in control has only recently started to be explored. Recent investigations [33, 37, 38] focused on combining PINNs with model predictive control (MPC), thus not exploiting the mathematical structure of the learned equations. Indeed, this strategy is part of an increasingly established trend seeking the combination of (non-PI and non-deep) learned models with MPC [39, 40]. Applications to control partial differential equations (PDEs) are discussed in [41–44], while an application to robotics is investigated in simulation in [45].

Preliminary investigations in other model-based techniques are provided in [30, 46], where, however, controllers are provided without any guarantee of stability or robustness and formulated for specific cases.

To summarize, in this work, we contribute to state of art in PINNs and robotics with the following:

1. An approach to include dissipation and allow for non-collocated control actions in Lagrangian and Hamiltonian neural networks, solving issues (i) and (ii).

2. Controllers for regulation and tracking, grounded in classic nonlinear control that exploit the mathematical structure of the learned models. For the first time, we prove the stability and robustness of these strategies.

3. Simulations and experiments on articulated and soft continuum robotic systems. To the authors' best knowledge, these are the first validation of PINN and PINN-based control applied to complex mechanical systems.

2 Preliminaries

2.1 Lagrangian and Hamiltonian Dynamics

Robots' dynamics can be represented using Lagrangian or Hamiltonian mechanics. In the former, the state is defined by the generalized coordinates $q \in \mathbb{R}^N$ and their velocities $\dot{q} \in \mathbb{R}^N$, where N represents the configuration space dimension. The Euler-Lagrange equation dictates the system's behavior $\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L(q,\dot{q})}{\partial \dot{q}}\right) - \frac{\partial L(q,\dot{q})}{\partial q} = F_{\texttt{ext}}$, where $L(q,\dot{q}) = T(q,\dot{q}) - V(q)$ with potential energy $V(q) \in \mathbb{R}$ and kinetic energy $T = \frac{1}{2}\dot{q}^\top M(q)\dot{q}$, where $M(q) \in \mathbb{R}^{N\times N}$ is the positive definite mass inertia matrix. External forces denoted as $F_{\texttt{ext}} \in \mathbb{R}^N$, include control inputs and dissipation forces.

In Hamiltonian mechanics, momenta $p \in \mathbb{R}^N$ replace the velocities, with $\dot{q} = M^{-1}(q)p$. The Hamiltonian equations $\dot{q} = \frac{\partial H(q,p)}{\partial p}$, $\quad \dot{p} = -\frac{\partial H(q,p)}{\partial q} + F_{\texttt{ext}}$, where $H(q,p) = T(q,p) + V(q)$ is the total energy. The kinetic energy in this case is defined as $T(q,p) = \frac{1}{2}p^\top M^{-1}(q)p$.

2.2 LNNs and HNNs

Lagrangian Neural Networks (LNNs) employ the principle of least action to learn a Lagrangian function $\mathcal{L}(q,\dot{q})$ from trajectory data, with the learned function generating dynamics via standard Euler-Lagrange machinery [34]. The loss function for the LNN in [23] is given by the mean squared error (MSE) between the actual accelerations $\ddot{q}$ and the ones that the learned model would expect $\hat{\ddot{q}}$

$$\mathcal{L}_{\mathrm{LNN}} = \mathrm{MSE}(\ddot{q}, \hat{\ddot{q}}). \tag{1}$$

HNNs, conversely, are designed to learn the Hamiltonian function $H(p,q)$. Once learned, this Hamiltonian function provides dynamics through Hamilton's equations. The loss function for HNN is similar an

MSE but between the predicted and actual time derivatives of generalized coordinates and momenta:

$$\mathcal{L}_{\text{HNN}} = \text{MSE}((\dot{q}, \dot{p}), (\hat{\dot{q}}, \hat{\dot{p}})). \tag{2}$$

We use fully connected neural networks with multiple layers of neurons with associated weights to learn the Lagrangian or the Hamiltonian, shown in Figure 1.

2.3 Limits of classic LNNs and HNNs

Note that both loss functions rely on measuring derivatives of the state $\ddot{q}$ and $\dot{p}$, which - by definition of state - cannot be directly measured. This issue is easily circumvented in simulation by the use of a non-causal sensor. Yet, this is not a feasible solution with physical experiments. An unrobust alternative is to estimate these values from measurements of positions and velocities numerically. This relates to issue (iii), stated in the introduction.

Moreover, existing LNNs and HNNs assume that $F_{\text{ext}} \in \mathbb{R}^N$ is directly measured. This is a reasonable hypothesis only if the system is conservative, fully actuated, and the actuation is collocated. The first characteristic is never fulfilled by real systems, while the second and the third are very restrictive outside when dealing with innovative robotic solutions as soft [31] or flexible robots [47]. Note that learning-based control is imposing itself as a central trend in these non-conventional robotic systems [48]. These considerations relate to issues (i) and (ii) stated in the introduction.
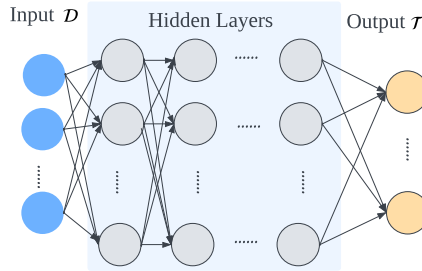

Figure 1: Fully connected network.

3 Proposed algorithms

3.1 A learnable model for non-conservative forces

In standard LNNs theory, non-conservative forces are assumed to be fully known and to be equal to actuation forces directly acting on the Lagrangian coordinates q. This is very restrictive, as already discussed in the introduction.

In this work, we include external forces given by dissipation and actuator forces, i.e., $F_{\text{ext}} = F_{\text{d}}(q, \dot{q}) + F_{\text{a}}(q)$. We propose the following model for dissipation forces

$$F_{\text{d}}(q, \dot{q}) = -D(q)\dot{q}, \tag{3}$$

where $D(q) \in \mathbb{R}^{N \times N}$ is the positive semi-definite damping matrix. Besides, we model the actuator force as

$$F_{\text{a}}(q) = A(q)u, \tag{4}$$

where $u \in \mathbb{R}^W$ is the control input signal to the system, and $A(q) \in \mathbb{R}^{N \times W}$ is an input transformation matrix. For example, A could be the transpose Jacobian associated with the point of application of an actuation force on the structure. With this model, we take into account that in complex robotic systems, actuators are, in general, not collocated on the measured configurations q. Note that, even if we accepted to impose an opportune change of coordinates, for some systems, a representation without A is not even admissible [49]. With (4), we also seemingly treat underactuated systems.

Note that [46] uses a dissipative model but considers it in a white box fashion.

Hence, we rewrite the Lagrangian dynamics as follows

$$\ddot{q} = \left(\frac{\partial^2 L(q,\dot{q})}{\partial \dot{q}^2} \right)^{-1} \left(A(q)u - \frac{\partial^2 L(q,\dot{q})}{\partial q \partial \dot{q}} \dot{q} + \frac{\partial L(q,\dot{q})}{\partial q} - D(q)\dot{q} \right), \tag{5}$$

which can be alternatively expressed as follows

$$\ddot{q} = M^{-1}(q) \left(A(q)u - C(q,\dot{q})\dot{q} - G(q) - D(q)\dot{q} \right),$$

where $C(q,\dot{q}) \in \mathbb{R}^{N \times N}$ and $G(q) \in \mathbb{R}^N$.

Similarly, the Hamiltonian takes the form

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -I & -D(q) \end{bmatrix} \begin{bmatrix} \frac{\partial H(q,\dot{q})}{\partial q} \\ \frac{\partial H(q,\dot{q})}{\partial p} \end{bmatrix} + \begin{bmatrix} 0 \\ A(q) \end{bmatrix} u. \tag{6}$$

3.2 Non-conservative non-collocated Lagrangian and Hamiltonian NNs with modified loss

Figure 2 reports the proposed network framework, which builds upon Lagrangian and Hamiltonian NNs discussed in Sec. 2.2. Our work incorporates the damping matrix network, input matrix network, and a modified loss function into the original framework. The damping matrix network is used to account for the dissipation forces in the system via (3), while the input matrix network corresponds to $A(q)$ in (4). We predict the next state by integrating (5) or (6) with the aid of the Runge-Kutta4 integrator. Clearly, different integration strategies could be used in its place.

The dataset $\mathfrak{D} = [\mathcal{D}_k, \mathcal{T}_k | k \in \{0, ..., k_{\text{end}}\}]$ contains information about the state transitions of the mechanical system. With this compact notation, we are not exclusively referring to a single trajectory of the system's behavior, but we aggregate data from multiple system trajectories. The input data $\mathcal{D}_k$ is composed of either $[q_k, \dot{q}_k, u_k, \Delta t]$, for Lagrangian dynamics, or $[q_k, p_k, u_k, \Delta t]$ in the case of Hamiltonian dynamics. Similarly, the corresponding label $\mathcal{T}_k$ is either $\dot{q}_{k+1}$, for the Lagrangian case, or $[q_{k+1}, p_{k+1}]$ for Hamiltonian dynamics. Here, k and $k + 1$ refer to consecutive time steps in the dataset, where k provides input data at one time step, and $k + 1$ corresponds to the label data at the subsequent time step Δt.

The values of $M(q, \theta_1)$, $V(q, \theta_2)$, $D(q, \theta_3)$, and $A(q, \theta_4)$ are estimated by four sub-networks, namely, the mass network (M-NN), potential energy network (V-NN), damping network (D-NN), and input matrix network (A-NN), as shown in Figure 2. The parameter θ_i, where $i \in \{1, 2, 3, 4\}$, represents the sub-networks' model parameter.

The kinetic energy can be calculated once the values of $\dot{q}$ or p are obtained. Then, the Lagrangian or Hamiltonian functions can be derived from the kinetic and potential energies. The derivative of the states $\ddot{q}$ or $[\dot{q}\ \dot{p}]^\top$ can be computed using (5) or (6), respectively. The predicted next state $\hat{q}$ or $[\hat{q}\ \hat{p}]^\top$ can be obtained using the Runge-Kutta4 integrator. We thus employ the following modified losses [32, 33]

$$\mathcal{L}_{\text{LNN}} = \frac{1}{\#\mathcal{D}} \sum_{k \in \mathcal{D}} \left(\| q_{k+1} - \hat{q}_{k+1} \|_2^2 + \| \dot{q}_{k+1} - \hat{\dot{q}}_{k+1} \|_2^2 \right) \tag{7}$$

for LNNs, where $\#\mathcal{D}$ is the cardinality of $\mathcal{D}$, and

$$\mathcal{L}_{\text{HNN}} = \frac{1}{\#\mathcal{D}} \sum_{k \in \mathcal{D}} \left(\| q_{k+1} - \hat{q}_{k+1} \|_2^2 + \| p_{k+1} - \hat{p}_{k+1} \|_2^2 \right) \tag{8}$$

for HNNs. Thus, compared to (1) and (2), we are calculating the MSE of a future prediction of the state—simulated via the learned dynamics—rather than of the current accelerations, which cannot be measured. Note that we also include a measure of the prediction error at the configuration level for $\mathcal{L}_{\text{HNN}}$ because the information on $\frac{\partial H(q,\dot{q})}{\partial p}$ appears disentangled from D and A (which are also learned) in the first n equations of (6).
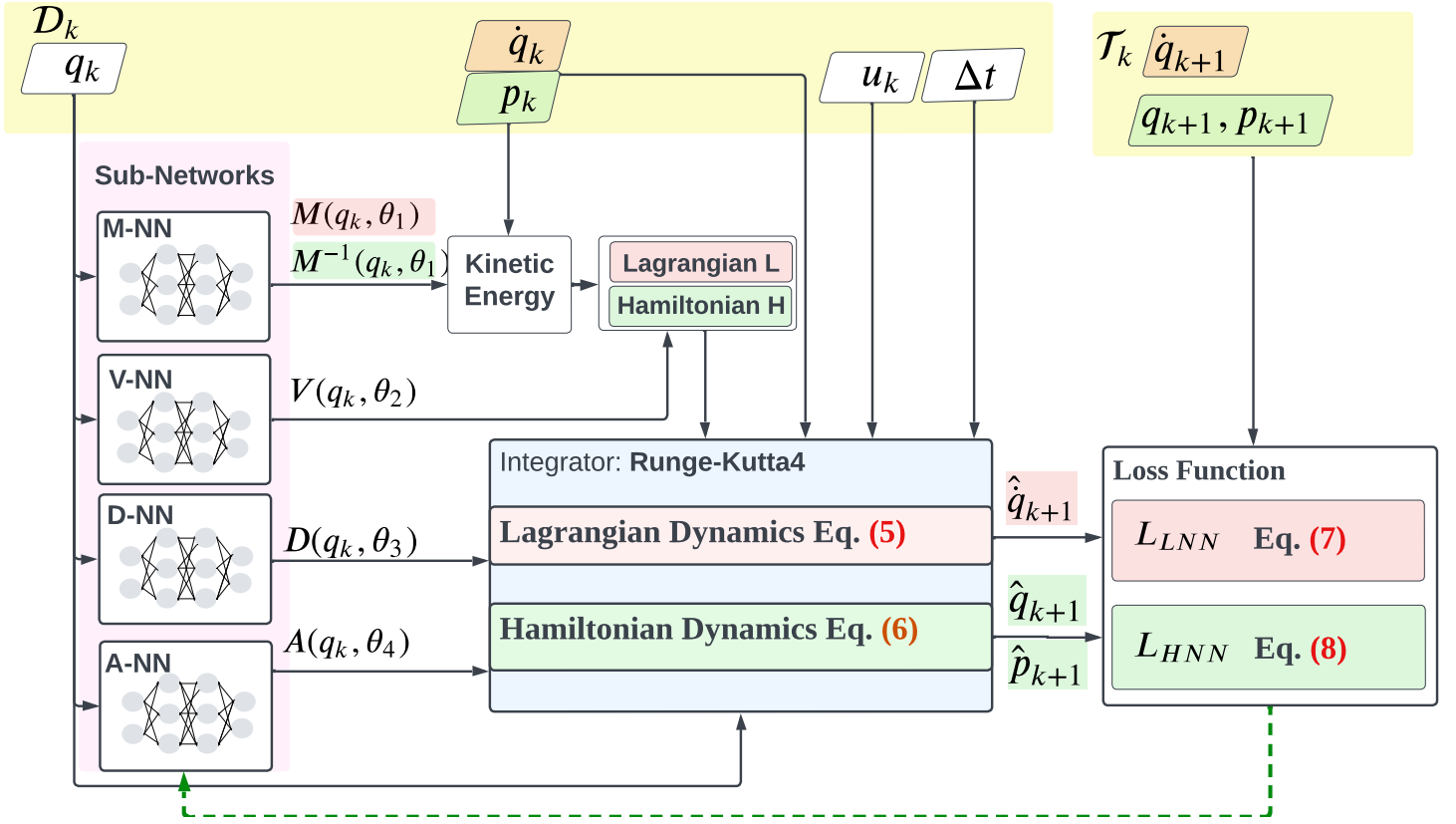
Figure 2: The overview of Lagrangian and Hamiltonian neural networks: the yellow part—i.e., $\mathcal{D}_k$ and $\mathcal{T}_k$—represents the input and label data used in the network; in red, the data and calculation process required for Lagrangian dynamics; the green parts represent the corresponding data and calculation associated with the Hamiltonian dynamics.

3.2.1 Sub-Network Structures

Constraints based on physical principles can be imposed on the parameters learned by the four sub-networks. Specifically, the mass and damping matrices must be positive definite and positive semi-definite, respectively. To this end, the network structure of the dissipation matrix can follow the prototype established for the mass matrix in [50]. This structure can be decomposed into a lower triangular matrix L_D with non-negative diagonal elements, which is then computed using the Cholesky decomposition [51] as $D = L_D L_D^\top$. The representation of $D(q)$ is illustrated in Figure 3.
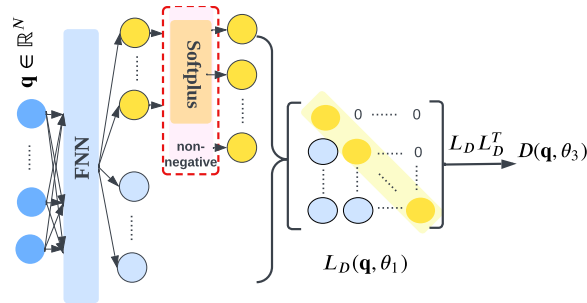

Figure 3: Diagram of the damping matrix including a feed-forward neural network, a non-negative shift for diagonal entries, and the Cholesky decomposition

The output of M-NN and D-NN is calculated as $(N^2 + N)/2$, with the first N values representing the diagonal entries of the lower triangular matrix. To ensure non-negativity, activation functions such as Softplus or ReLU are utilized as the last layer. Furthermore, the constant ϵ is introduced to guarantee that the mass matrix is positive definite. Note that ϵ is a hyperparameter that should be selected to be *small-enough* but strictly positive. The remaining $(N^2 - N)/2$ values are placed in the lower left corner of the lower triangular matrix.

The calculation of the potential energy is performed using a simple, fully connected network with a single output, which is represented as $V(q, \theta_2)$. Moreover, A-NN, depicted in Figure 4, calculates $A(q, \theta_4)$ with dimensions $\mathbb{R}^{N \times W}$.
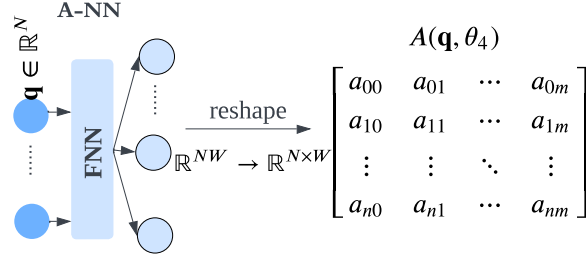

Figure 4: Diagram for actuator matrix: The fully connected network output is a vector in $\mathbb{R}^{NW}$, which is reshaped to a matrix in $\mathbb{R}^{N \times W}$. A sigmoid activation function can be applied to the matrix elements for value constraint.

3.3 PINN-based controllers

We provide in this section two provably stable controllers by combining the learned dynamics with classic model-based approaches. Before stating these results, it is important to spend a few lines remarking on the potential relationships between the outcomes obtained through proposed LNN and the ground truth, as well as their implications for controller design. Due to the inclusion of the actuator matrix and the inherent non-uniqueness of the Lagrangian, we assume that the Lagrangian $L_{\mathrm{L}}(q, \dot{q})$ learned by LNN can be represented as follows:

$$L_{\mathrm{L}}(q, \dot{q}) = aL(q, \dot{q}) + b, \tag{9}$$

where a is a non-zero constant, and b is another constant term. In this section, we highlight the components that have been learned by adding an L as a subscript to provide a clearer illustration. The LNN enables us to discover an ODE with a solution that matches that of the real ODE:

$$\underbrace{M^{-1}(q)(A(q)\tau - C(q, \dot{q})\dot{q} - G(q) - D(q)\dot{q})}_{\ddot{q}} = \underbrace{M_{\mathrm{L}}^{-1}(q)(A_{\mathrm{L}}(q)\tau - C_{\mathrm{L}}(q, \dot{q})\dot{q} - G_{\mathrm{L}}(q) - D_{\mathrm{L}}(q)\dot{q})}_{\ddot{q}_{\mathrm{L}}} \tag{10}$$

Also, by construction, $M_{\mathrm{L}}, G_{\mathrm{L}}, A_{\mathrm{L}}, D_{\mathrm{L}}$ will have all the usual properties that we expect from these terms, like M_{L} and D_{L} being symmetric and positive definite, and G_{L} being a potential force. Yet, this does not imply that $M = M_{\mathrm{L}}$, $G = G_{\mathrm{L}}$, $A = A_{\mathrm{L}}$, and $D = D_{\mathrm{L}}$. Indeed, there could exist a constant matrix P such that $PM(q), PG(q), PA(q), PD(q)$ have all the properties discussed above while simultaneously fulfilling

$$L(q, \dot{q}; PM, PG, PA, PD) = aL(q, \dot{q}; M, G, A, D) + b \tag{11}$$

So controllers must be formulated and proofs derived under the assumption of the learned terms being close to the real ones up to a multiplicative factor.

3.3.1 Regulation

The goal of the following controller is to stabilize a given configuration q_{ref}

$$u = A_{\mathrm{L}}^{-1}(q)G_{\mathrm{L}}(q) + A_{\mathrm{L}}^{-1}(q)(K_{\mathrm{P}}(q_{\mathrm{ref}} - q) - K_{\mathrm{D}}\dot{q}), \tag{12}$$

where we omit the arguments t and θ_i to ease the readability. $G_{\mathrm{L}}(q_{\mathrm{ref}})$ is the potential force which can be calculated by taking the partial derivative of the potential energy learned by the LNN; K_{P} and K_{D} are control gains.

For the sake of conciseness, we introduce the controller, and we prove its stability for the fully actuated case. However, the controller and the proof can be extended to the generic underactuated case using arguments in [36, p.50]. This will be the focus of future work.

Proposition 1. *Assume that $W = N$, with A and $A_{\mathtt{L}}$ both full rank, and the existence of a constant matrix $P \in \mathbb{R}^{N \times N}$ such that $||G_{\mathtt{L}}(q) - PG(q)|| < \delta_{\mathtt{G}}$, for some finite and positive $\delta_{\mathtt{G}}$. We assume that*

$$||A^{-1}(q)P^{-1}[A_{\mathtt{L}}(q) - PA(q)]|| < 1, \tag{13}$$

and that the gains $K_{\mathtt{P}}, K_{\mathtt{D}}$ are chosen such that

$$P^{-1}K_{\mathtt{P}} \succ 0, \text{ and, } P^{-1}K_{\mathtt{D}} \succ 0. \tag{14}$$

Then, given a maximum admitted error $\delta_{\mathtt{q}}$, the closed loop of (5) and (12) is such that

$$\lim_{t \to \infty} q(t) = q_{\mathtt{ss}} \text{ with } ||q_{\mathtt{ss}} - q_{\mathtt{ref}}|| < \delta_{\mathtt{q}}. \tag{15}$$

Remark 1. *Assumption (13) is a request on the learned matrix $A_{\mathtt{L}}(q)$ being close enough to $A(q)$ up to a multiplicative factor P, which is something we need to ensure, as discussed in Section 3.3. Indeed, if $A_{\mathtt{L}}(q) \simeq PA(q)$, then (13) is fulfilled.*

Remark 2. *Note that there always exist $K_{\mathtt{P}}$ and $K_{\mathtt{D}}$ that fulfill assumption (14). Specifically, they can be expressed as $K_{\mathtt{P}} = P\hat{K}_{\mathtt{P}}$ and $K_{\mathtt{D}} = P\hat{K}_{\mathtt{D}}$, where $\hat{K}_{\mathtt{P}}$ and $\hat{K}_{\mathtt{D}}$ denote positive definite matrices.*

Proof. Let us introduce the matrix $\Delta_{\mathtt{A}} \in \mathbb{R}^{N \times N}$ such that $A_{\mathtt{L}}(q) = PA(q) + \Delta_{\mathtt{A}}(q)$. This matrix is small enough by assumption as detailed in Remark 1. We now want to bound the difference between the inverse of $A(q)$ and $PA_{\mathtt{L}}(q)$. The goal is to write $A_{\mathtt{L}}^{-1}(q) = (PA(q))^{-1} + \Delta_{\mathtt{I}}(q)$, with $||\Delta_{\mathtt{I}}(q)|| < \delta_{\mathtt{I}}$.

Because of (13), we can use the Neumann series—see, for instance, [52, p.20]—to obtain the following

$$A_{\mathtt{L}}^{-1}(q) = (PA(q) + \Delta_{\mathtt{A}}(q))^{-1} = \sum_{k=0}^{\infty} (-(PA(q))^{-1}\Delta_{\mathtt{A}}(q))^k (PA(q))^{-1}. \tag{16}$$

Rearranging terms, we get that

$$\Delta_{\mathtt{I}}(q) = A_{\mathtt{L}}^{-1}(q) - (PA(q))^{-1} = \sum_{k=1}^{\infty} (-(PA(q))^{-1}\Delta_{\mathtt{A}}(q))^k (PA(q))^{-1}. \tag{17}$$

Therefore, we can bound the norm of $\Delta_{\mathtt{I}}(q)$ as follows

$$||\Delta_{\mathtt{I}}(q)|| \le \sum_{k=1}^{\infty} ||(PA(q)^{-1}\Delta_{\mathtt{A}}(q))||^k ||(PA(q))^{-1}|| = \frac{||(PA(q))^{-1}\Delta_{\mathtt{A}}(q)|| ||(PA(q))^{-1}||}{1 - ||(PA(q))^{-1}\Delta_{\mathtt{A}}(q)||} < \delta_{\mathtt{I}}. \tag{18}$$

Hence, the generalized forces produced by the controller $A(q)u$ are given by

$$\begin{aligned}
&A(q)A_{\mathtt{L}}^{-1}(q) \left[G_{\mathtt{L}}(q) + (K_{\mathtt{P}}(q_{\mathtt{ref}} - q) - K_{\mathtt{D}}\dot{q}) \right] \\
&= A(q)(A^{-1}(q)P^{-1} + \Delta_{\mathtt{I}}(q)) \left[(PG(q) + \Delta_{\mathtt{G}}(q)) + K_{\mathtt{P}}(q_{\mathtt{ref}} - q) - K_{\mathtt{D}}\dot{q} \right] \\
&= (P^{-1} + A(q)\Delta_{\mathtt{I}}(q)) \left[(PG(q) + \Delta_{\mathtt{G}}(q)) + K_{\mathtt{P}}(q_{\mathtt{ref}} - q) - K_{\mathtt{D}}\dot{q} \right] \\
&= G(q) + \Delta_{\mathtt{all}}(q) + \hat{K}_{\mathtt{P}}(q_{\mathtt{ref}} - q) - \hat{K}_{\mathtt{D}}\dot{q}
\end{aligned} \tag{19}$$

where $\Delta_{\mathtt{all}}(q) = P^{-1}\Delta_{\mathtt{G}}(q) + A(q)\Delta_{\mathtt{I}}(q)PG(q) + A(q)\Delta_{\mathtt{I}}(q)\Delta_{\mathtt{G}}(q) + A(q)\Delta_I K_{\mathtt{P}}(q_{\mathtt{ref}} - q) - A(q)\Delta_I K_{\mathtt{D}}\dot{q}$ is a bounded term, as sum and product of bounded terms. The gains $\hat{K}_{\mathtt{P}}$ and $\hat{K}_{\mathtt{D}}$ are positive definite matrices, resulting from the products $P^{-1}K_{\mathtt{P}}$ and $P^{-1}K_{\mathtt{D}}$, respectively, as indicated in Remark 2. Thus, the closed-loop system takes the form

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} = \Delta_{\mathtt{all}}(q) + \hat{K}_{\mathtt{P}}(q_{\mathtt{ref}} - q) - (D(q) + \hat{K}_{\mathtt{D}})\dot{q}. \tag{20}$$

To conclude, replicating the arguments provided in [53, Theorem 1] yields the result. In turn, that work was adapted from the seminal paper [54]. Alternatively, (20) can be rewritten as

$$\underbrace{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + (D(q) + \hat{K}_{\text{D}})\dot{q} + \hat{K}_{\text{P}}(q - q_{\text{ref}})}_{\text{nominal system}} = \Delta_{\text{all}}(q),$$

Thus, considering the Lyapunov candidate function

$$\mathcal{V}(q,\dot{q}) = \frac{1}{2}\dot{q}^\top M(q)\dot{q} + \frac{1}{2}(q - q_{\text{ref}})^\top \hat{K}_{\text{P}}(q - q_{\text{ref}}), \tag{21}$$

a simple stability analysis shows that the nominal system has an asymptotically stable equilibrium point at the desired configuration. Therefore, the closed-loop system can be interpreted as a perturbed system, where the perturbation is given by $\Delta_{\text{all}}(q)$. Hence, the result can be proven following arguments for perturbed systems—see, for instance, [35, Chapter 9]. □

Note that even if we provided the proof using a Lagrangian formalism, the Hamiltonian version can be derived following similar steps. Also, note that the bounds on the learned matrices are always verified for any choice of $\delta_{\text{A}}, \delta_{\text{G}}$ at the cost of training the model with a large enough training set.

We conclude with a corollary that discusses the perfect learning scenario.

Corollary 1. *Assume that $W = N$ and A is full rank. Then, the closed loop of (5) and (12) is such that*

$$\lim_{t \to \infty} q(t) = q_{\text{ref}}, \tag{22}$$

if it exists a matrix $P \in \mathbb{R}^{N \times N}$ such that $M_{\text{L}}(q) = PM(q)$, $A_{\text{L}}(q) = PA(q)$, $G_{\text{L}}(q) = PG(q)$.

Proof. Note that $\Delta_{\text{all}} = 0$ as the deltas are now all zero. So, the closed loop (20) is always the equivalent of a mechanical system, without any potential force, controlled by a PD. Note that the gains $\hat{K}_{\text{P}}$ and $\hat{K}_{\text{D}}$ are positive definitive. The proof of stability follows standard Lyapunov arguments (see, for example, [34, p.186]) by using the Lyapunov candidate given in (21). □

3.3.2 Trajectory tracking

The goal of the following controller is to track a given trajectory in configuration space $q_{\text{ref}} : \mathbb{R} \to \mathbb{R}^n$. We assume q_{ref} to be bounded with bounded derivatives. We also assume the system to be fully actuated—i.e., $W = N$, $\det(A) \neq 0$, $\det(A_{\text{L}}) \neq 0$. Under these assumptions, we extend (12) with the following controller to follow the desired trajectory

$$\begin{aligned} u = & A_{\text{L}}^{-1}(q)\left(M_{\text{L}}(q_{\text{ref}})\ddot{q}_{\text{ref}} + C_{\text{L}}(q_{\text{ref}}, \dot{q}_{\text{ref}})\dot{q}_{\text{ref}} + D_{\text{L}}(q_{\text{ref}})\dot{q}_{\text{ref}} + G_{\text{L}}(q_{\text{ref}})\right) \\ & + A_{\text{L}}^{-1}(q)\left(K_{\text{P}}(q_{\text{ref}} - q) + K_{\text{D}}(\dot{q}_{\text{ref}} - \dot{q})\right), \end{aligned} \tag{23}$$

where we omit the arguments t and θ_i to ease the readability. We highlight the components that have been learned from the ones that are not by adding an L as a subscript. We can obtain the Coriolis matrix $C_{\text{L}}(q_{\text{ref}}, \dot{q}_{\text{ref}})$ from the learned Lagrangian by taking the second partial derivative of the Lagrangian with respect to the desired joint position q_{ref} and velocity $\dot{q}_{\text{ref}}$, i.e., $\frac{\partial^2 L(q_{\text{ref}}, \dot{q}_{\text{ref}})}{\partial q_{\text{ref}} \partial \dot{q}_{\text{ref}}}$.

Corollary 2. *The closed loop of (5) and (23) is such that, for some $\delta_{\text{q}} \geq 0$,*

$$\lim_{t \to \infty} \|q(t) - q_{\text{ref}}(t)\| < \delta_{\text{q}}, \tag{24}$$

If it exists a matrix $P \in \mathbb{R}^{N \times N}$ such that $A_{\text{L}}(q) = PA(q)$, $M_{\text{L}}(q) = PM(q)$, $C_{\text{L}}(q) = PC(q)$, $G_{\text{L}}(q) = PG(q)$, $D_{\text{L}}(q) = PD(q)$.

Proof. We can rewrite (23) by substituting the values of the learned elements in terms of P. The result is

$$A(q)u = (M(q_{\mathrm{ref}})\ddot{q}_{\mathrm{ref}} + C(q_{\mathrm{ref}}, \dot{q}_{\mathrm{ref}})\dot{q}_{\mathrm{ref}} + D(q_{\mathrm{ref}})\dot{q}_{\mathrm{ref}} + G(q_{\mathrm{ref}}))$$
$$+ P^{-1}\left(K_{\mathrm{P}}(q_{\mathrm{ref}} - q) + K_{\mathrm{D}}(\dot{q}_{\mathrm{ref}} - \dot{q}))\right). \tag{25}$$

Moreover, with the assumption (14) in Corollary 1, the closed loop is equivalent to the one discussed in [55]. Therefore, the proof follows the same steps as discussed there. □

Finally, note that we provided here only proof of stability for the perfectly learned case. Similar hypotheses and arguments to the ones in Proposition 1 would lead to similar results in the tracking case, with $||PA_{\mathrm{L}}(q) - A(q)|| < \delta_{\mathrm{A}}$, $||PM_{\mathrm{L}}(q) - M(q)|| < \delta_{\mathrm{M}}$, $||PC_{\mathrm{L}}(q) - C(q)|| < \delta_{\mathrm{C}}$, $||PG_{\mathrm{L}}(q) - G(q)|| < \delta_{\mathrm{G}}$, $||PD_{\mathrm{L}}(q) - D(q)|| < \delta_{\mathrm{G}}$, for some finite and positive $\delta_{\mathrm{A}}, \delta_{\mathrm{M}}, \delta_{\mathrm{C}}, \delta_{\mathrm{G}}, \delta_{\mathrm{D}} \in \mathbb{R}$.

4   Methods: Simulation and experiment design

To evaluate the efficacy of the proposed PINNs and PINN-based control, we apply them in three distinct tasks: (T1) Learning the dynamic model of a one-segment spatial soft manipulator, (T2) Learning the dynamic model of a two-segment spatial soft manipulator, (T3) Learning the dynamic model of the Franka Emika Panda robot. We selected (T1) and (T2) because they have a nontrivial $A(q)$, and (T3) because it has several degrees of freedom. Furthermore, we employ the learned dynamics to design and test model-based controllers for T2 and T3.

In a hardware experiment, the LNN is utilized to learn the dynamic model of the tendon-driven soft manipulator reported in [56] and the Panda robot. We show for the first time experimental closed-loop control of a robotic system (the Panda robot) with a PINN-based algorithm.

4.1   Data Generation

Training data for T1 and T2 are generated by simulating the dynamics of one-segment and two-segment soft manipulators in MATLAB. For these two cases, a random sampling strategy is employed in data generation due to the unbounded configuration space inherent to soft manipulator models in simulation. For T1, ten different initial states are combined with ten different input signals to generate data using the one-segment manipulator dynamics model. Each combination produces ten-second training data with a time step of 0.0002 seconds. For T2, we use a variable step size in Simulink to generate datasets from the mathematical model of a two-segment soft manipulator. With this approach, we create twelve different sixty-second trajectories, which are subsequently resampled at fixed frequencies of 50Hz, 100Hz, and 1000Hz. Concerning T3, the PyBullet simulation environment is used to generate training data corresponding to the Panda robot. Then, different input signals are applied to the joints to create the data of 70 different trajectories with a frequency of 1000Hz. These trajectories are thoughtfully designed to encompass a significant portion of the robot's workspace.

Regarding experimental validation, we propose the following experiments. For the tendon-driven continuum robot, we provide sinusoidal inputs with different frequencies and amplitudes to the actuators— four motors—and record the movement of the robot. An IMU records the tip orientation data with a 10Hz sampling frequency. As a result, 122 trajectories are generated, and four more are collected as the test set. For the Panda robot, we provide 70 sets of sinusoidal desired joint angles with different amplitudes and frequencies. We collect the torque, joint angle, and angular velocity data using the integrated sensors, considering a sampling frequency of 500Hz.

4.2   Baseline Model and Model Training

To provide a basis for comparison, baseline models are established for all simulations and hardware experiments. These models, which serve as a control, are constructed using a fully connected network and trained using the same datasets as the proposed models, however, with a larger amount of data and a
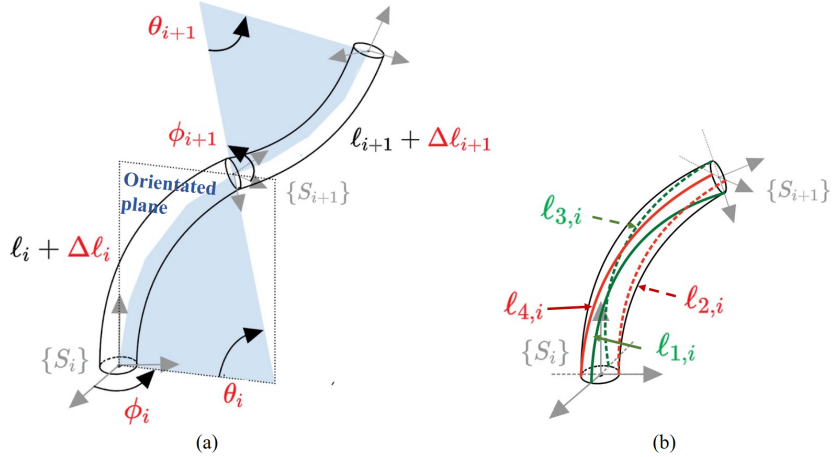
Figure 5: PCC approach illustration: (a) two-segment soft manipulator is shown, where S_i is the end frame, the blue parts are the orientated plane, ℓ_i is the original length of each segment; (b) shows the length of the four arcs whose ends connected to the frame S_i

greater number of training epochs. These baseline models aimed to demonstrate the benefits of incorporating physical knowledge into neural networks.

In this project, all the neural networks utilized are constructed using the JAX and dm-Haiku packages in Python. In particular, the JAX Autodiff system is used to calculate partial derivatives and the Hessian within the loss function. The optimization of the model parameters is carried out using AdamW in the Optax package, which inherently includes regularization terms within the optimization process, eliminating the need for additional explicit regularization terms in the loss function.

5   Simulation Results

5.1   One-segment 3D soft manipulator

To define the configuration space of the soft manipulator, we adopt the piecewise constant curvature (PCC) approximation [57] shown in Figure 5. Customarily, this approximation describes the configuration of each segment as $q_i = [\phi_i, \theta_i, \delta\ell_i]$, where ϕ_i is the plane orientation, θ_i is the curvature in that plane, and $\delta\ell_i$ is the change of arc length. In this work, the configuration-defined method reported in [58] is used to avoid the singularity problem of PCC. Hence, the configuration of each segment is given by $[\Delta_{xi}, \Delta_{yi}, \Delta_{\ell i}]$, where Δ_{xi} and Δ_{yi} are the difference of arc length.

Table 1: One-segment soft manipulator simulation detailed information

	Black-box model	Lagrangian-based learned model	Hamiltonian-based learned model
model (width × depth)	128×5	32×3, 5×3, 16×2	32×3, 5×3, 16×2
sample number	19188	8000	8000
training epoch	15000	6000	6000
traning error	$6.891e^{-5} \pm 4.63e^{-4}$	$8.418e^{-7} \pm 1.77e^{-5}$	$5.374e^{-11} \pm 7.74e^{-10}$
prediction error [m]	$7.647 \pm 10.413(5s)$	$0.171 \pm 0.272(5s)$	$0.0220 \pm 0.0210 \ (5s)$

The figure indicates that the model trained by LNNs exhibits a high degree of predictive accuracy, manifesting near-infinite prediction capabilities with over 50,000 consecutive prediction steps in this example. While some areas exhibit less precise fits, such errors do not accrue over time. These outcomes suggest that LNN-based models can effectively capture the underlying dynamics of the one-segment soft manipulator. In contrast, the black-box model converges during the training process but lacks the generalized predictive ability outside the training dataset. Its performance reveals its inability to capture and generalize the underlying dynamics. This system is also learned using HNNs by providing momen-
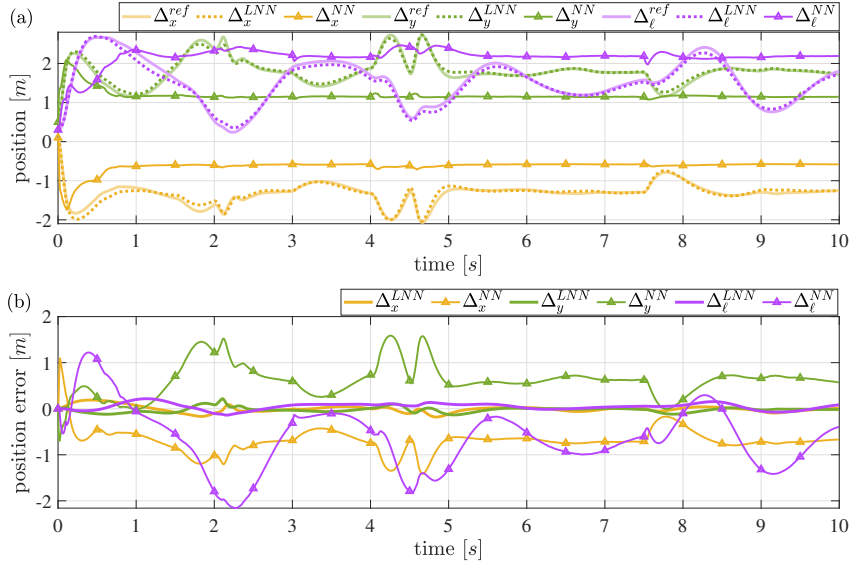
Figure 6: One-segment soft manipulator learned model comparison results: (a) depicts the predictions generated by the black-box model ($\triangle$), the Lagrangian-based learning model ($\cdots$), and the ground-truth ($-$) arising from the dynamic mathematical equations; (b) shows the prediction error of these two learned models.
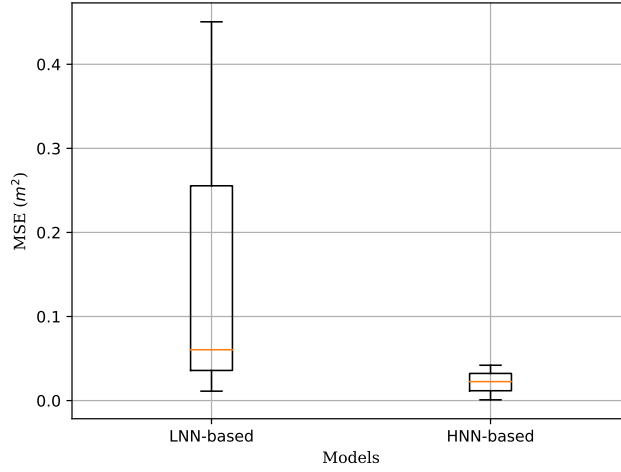

Figure 7: One-segment soft manipulator LNN-based and HNN-based learned models prediction MSE results

tum data. Hamiltonian-based neural networks yield similar quality prediction results as Lagrangian-based neural networks, as shown in Figure 8. The HNN outperforms the LNN with identical training sample size and network dimensions, primarily due to two key factors. Firstly, the nature of the optimization problem favors HNN, which benefits from a unique solution. Secondly, HNN's input data, momentum, provides a more comprehensive description of system dynamics. The detailed information regarding the one-segment soft manipulator simulation is elucidated in Table 1. The MSE shown in Table 1 and Figure 7 over a five-second duration reveals substantial performance advantages for both the Lagrangian-based and Hamiltonian-based learned models in comparison to the black-box model. Notably, the Hamiltonian-based model demonstrates a remarkable superiority, yielding a prediction error of 0.0220 ± 0.0210 for the 5-second simulation period. This underscores the model's efficacy in adeptly capturing and predicting the intricate dynamics of the system.

The matrices obtained from these two physics-based learning models are shown in Table 3 and 4, where $G(q)$ represents the potential forces, i.e., $\frac{\partial V(q)}{\partial q}$. As Table 4 shows, HNNs can learn the physically meaningful matrices, while LNNs only learn one of the solutions satisfying the Euler-Lagrangian equa-
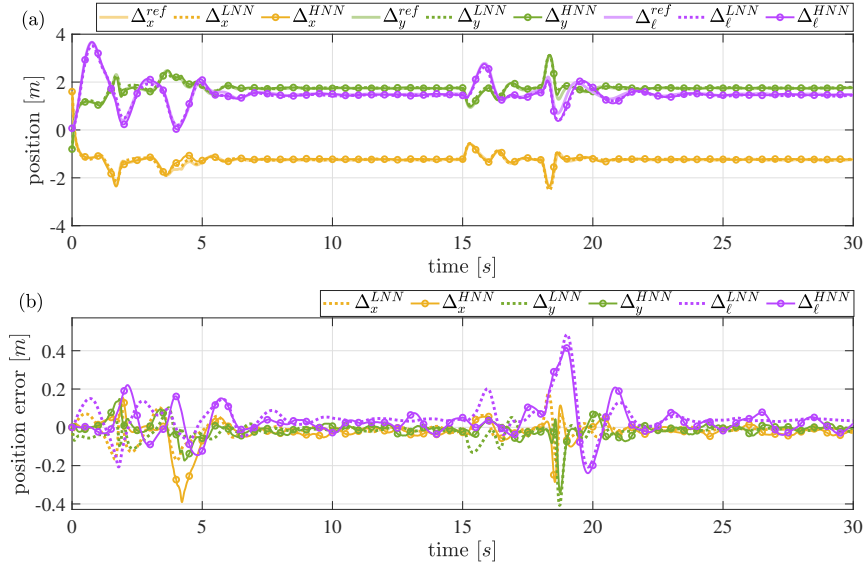
Figure 8: One-segment soft manipulator HNN and LNN comparison: (a) shows the Lagrangian-based learned model prediction results ($\cdots$), Hamiltonian-based learned model prediction results ($\circ$), and the ground-truth prediction ($-$); (b) error of the two models with the ground truth.

Table 2: Mathematical model matrices of one-segment soft manipulator

q	$M(q)$			$M^{-1}(q)$			$D(q)$			$G(q)$	$A(q)$		
1.20	$1.73e^{-3}$	$-3.12e^{-5}$	$-1.96e^{-3}$	593.09	9.35	12.47	0.1	0	0	1.29	-0.04	-1.0	0.07
-0.20	$-3.12e^{-5}$	$1.55e^{-3}$	$3.26e^{-4}$	9.35	647.61	-2.08	0	0.1	0	-0.22	0.78	0.04	-0.01
0.15	$-1.96e^{-3}$	$3.26e^{-4}$	$9.29e^{-2}$	12.47	-2.08	11.04	0	0	0.1	-1.15	0.	0.	0.77
0.80	$3.64e^{-3}$	$4.52e^{-5}$	$-1.94e^{-3}$	277.76	-2.84	5.55				0.89	0.03	-0.99	0.06
0.20	$4.52e^{-5}$	$3.47e^{-3}$	$-4.84e^{-4}$	-2.84	288.42	1.39				0.22	0.90	-0.03	0.02
0.30	$-1.94e^{-3}$	$-4.84e^{-4}$	$9.67e^{-2}$	5.55	1.39	10.46				-1.09	0.	0.	0.89

Table 3: Lagrangian-based learning model matrices of one-segment soft manipulator

q	$M_L(q)$			$D_L(q)$			$G_L(q)$	$A_L(q)$			P		
1.20	$4.23e^{-3}$	$1.20e^{-3}$	-0.03	0.16	-0.02	0.0	2.44	0.12	-1.72	-0.21	0.61	-0.02	0.03
-0.20	$1.20e^{-3}$	$5.99e^{-3}$	-0.02	-0.02	0.33	-0.01	-0.61	3.05	-0.19	-0.13	-0.02	0.28	0.01
0.15	-0.03	-0.02	0.59	0.0	-0.01	0.35	-5.25	-0.34	1.01	3.40	0.33	0.15	0.25
0.80	$6.93e^{-3}$	$1.84e^{-3}$	-0.03	0.17	-0.01	-0.0	1.62	0.19	-1.66	-0.20	0.62	-0.02	0.03
0.20	$1.84e^{-3}$	0.01	-0.02	-0.01	0.33	-0.01	0.81	2.97	-0.25	-0.13	-0.02	0.31	0.01
0.30	-0.03	-0.02	0.50	-0.0	-0.01	0.35	-4.67	-0.40	1.01	3.43	0.21	0.10	0.26

tion. Comparing the corresponding matrices in Table 2 and 3, we can find that the matrices and vectors learned by the LNNs are related to the real parameters through a transformation P. Notably, P manifests subtle variations across different states; however, in theory, P is anticipated to remain constant. The observed discrepancies are attributed to inherent learning errors within the network.

5.2 Two-segment 3D soft manipulator

The two-segment soft manipulator model is simulated in MATLAB, where the configuration space is also defined as in the one-segment case. The training and testing information for this task is presented in Table 5. In the 100Hz dataset, the Lagrangian-based learned model outperforms the black-box model with a notably lower prediction MSE of 1.690 ± 0.673 meters with less training data. Figure 9 summarizes the prediction results of the $50Hz$, $100Hz$, and $1000Hz$ learned model. From the simulations, we conclude that the higher the sampling frequency within a certain range, the more accurate the learned model is. This phenomenon is attributed to the sensitivity of the integration algorithm to step size. Employing more accurate integration algorithms or shorter time steps in future experiments is expected to enhance model precision.

Based on the learned model trained at 1000Hz, we devise a PINN-based control loop as in (12). To

Table 4: Hamiltonian-based learning model matrices of one-segment soft manipulator

q	$M_L^{-1}(q)$			$D_L(q)$			$G_L(q)$	$A_L(q)$		
1.20	600.32	16.90	15.67	$1.02e^{-1}$	$3.44e^{-3}$	$8.12e^{-5}$	1.33	−0.06	−0.94	0.05
−0.20	16.90	622.92	−1.34	$3.44e^{-3}$	$1.05e^{-1}$	$-4.39e^{-4}$	−0.18	0.83	0.02	−0.04
0.15	15.67	−1.34	11.61	$8.12e^{-5}$	$-4.39e^{-4}$	$9.91e^{-2}$	−1.15	0.0	0.01	0.78
0.80	285.01	11.08	6.65	$1.01e^{-1}$	$3.48e^{-3}$	$6.56e^{-4}$	0.93	0.03	−0.96	0.05
0.20	11.08	292.46	2.06	$3.48e^{-3}$	$1.03e^{-1}$	$-7.45e^{-5}$	0.25	0.92	−0.03	−0.02
0.30	6.65	2.06	10.59	$6.56e^{-4}$	$-7.45e^{-5}$	$9.87e^{-2}$	−1.10	−0.01	0.0	0.89

Table 5: Two-segment simulated soft manipulator training and testing detailed information

	Black-box model	Lagrangian-based learned model		
	100Hz	50Hz	100Hz	1000Hz
model(width × depth)	152×3	42×3, 5×3, 42×2	42×3, 5×3, 42×2	42×3, 5×3, 42×2
sample number	59200	45000	45000	45000
training epoch	15000	5500	5500	5500
traning error	$3.536e^{-4} \pm 1.08e^{-3}$	$5.916e^{-4} \pm 8.61e^{-3}$	$1.652e^{-4} \pm 2.12e^{-2}$	$1.822e^{-7} \pm 6.67e^{-6}$
prediction error $[m]$	$44.683 \pm 4.518(10s)$	$2.098 \pm 1.253(10s)$	$1.690 \pm 0.673(10s)$	$0.089 \pm 0.278(10s)$
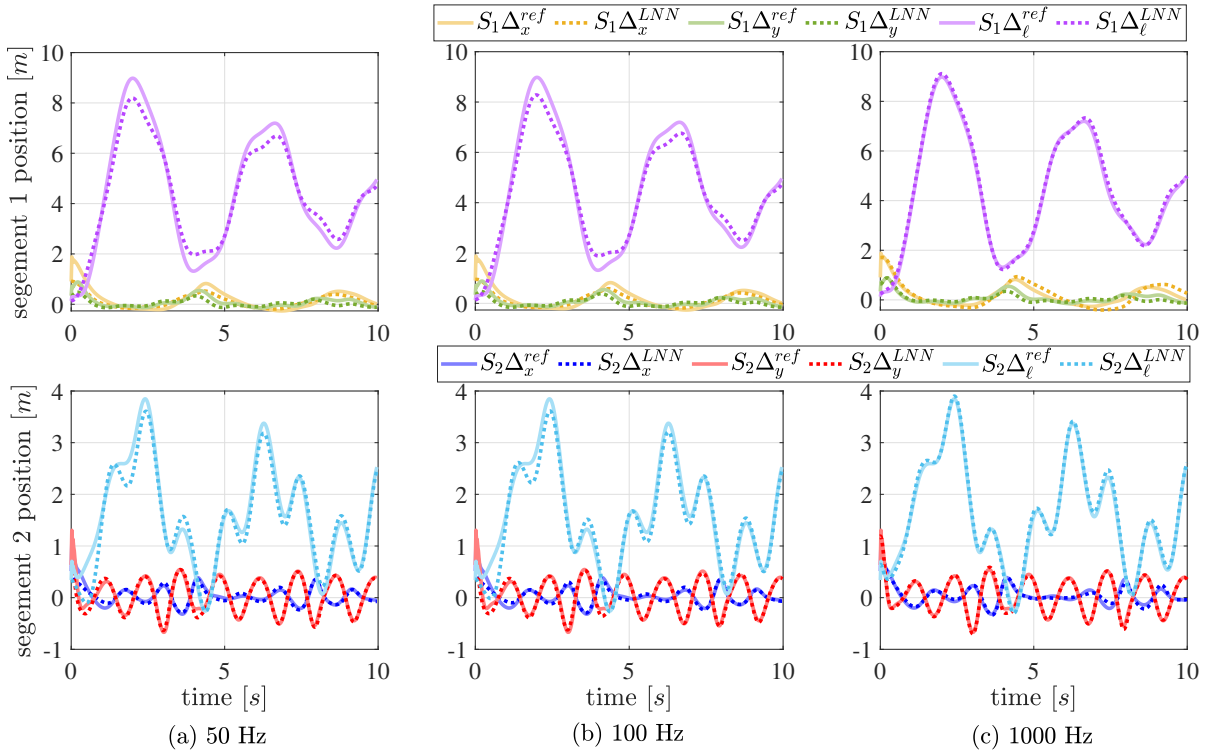
Figure 9: Two-segment soft manipulator prediction performances under different sampling frequencies

demonstrate the performance of the designed controller, we employ it to control the two-segment soft manipulator in MATLAB. The proportional gains K_P and derivative gains K_D are set to 10 and 50, respectively, for all six configurations. The alterations in the states of the two-segment manipulator under control are depicted in Figure 10, whereas the performance of the controller is demonstrated in Figure 11. Results indicate that the controller is capable of tracking a static setpoint within one second while keeping the root mean square error (RMSE) less than 0.23%, and exhibits a stable and minimal overshoot performance. These performances underscore the reliability and efficiency of the designed controller based on the learned model.

5.3 Panda robot

Table 6 presents the training and testing results of the simulated Panda in PyBullet, while Figure 12 displays the prediction results obtained from the learned model. In comparison to the dynamics models
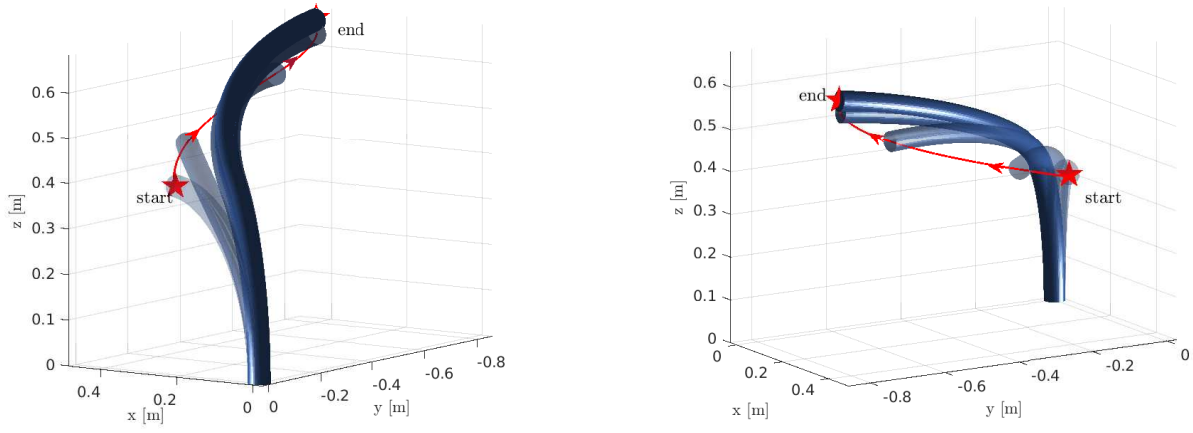
Figure 10: The sequence of movements at the times 0.0s, 0.1s, 0.3s, 0.6s, and 1.0s executed by the two-segment soft robot as a result of the implementation of the LNN-model-based controller. The red line represents the tip's position
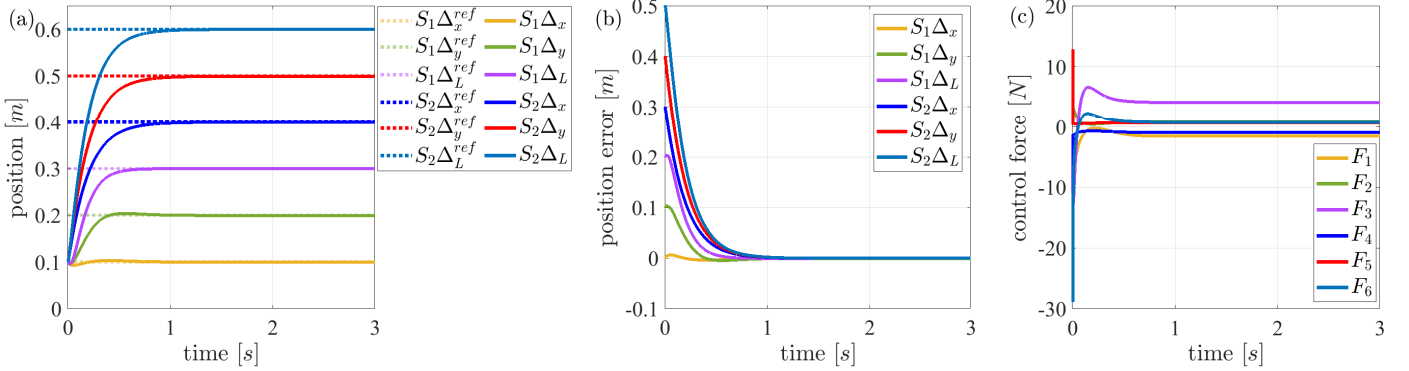

Figure 11: Two-segment soft manipulator model-based controller performance: (a) shows the evolution of the configuration variables and the desired state with dotted lines; (b) shows the error between the desired states and current states; (c) shows control effort.

formulated in MATLAB, the simulator's dynamics model is characterized by increased complexity, influenced by the inherent physical constraints in robotic systems, including restrictions on acceleration and velocity. This heightened complexity presents challenges in learning the dynamics model. Nevertheless, the LNN-based model demonstrates a smaller prediction MSE than the MSE of the black-box model. Notably, limitations emerge in long-term predictions. Consequently, in Figure 12 c, we adopted a continuous prediction approach—forecasting 50 steps consecutively and updating the model state to effectively illustrate its performance.

Table 6: Panda simulation detailed information (1000Hz)

	Black-box model	Lagrangian-based learned model
model (width × depth)	120×4	$40 \times 3, 20 \times 2$
sample number	550000	25000
training epoch	10000	10000
traning error	$1.476e^{-4} \pm 2.69e^{-3}$	$1.424e^{-4} \pm 2.90e^{-3}$
prediction error/ $[rad]$	$110.610 \pm 8.809(2\text{s})$	$8.884 \pm 6.323 \ (2\text{s})$

Based on this learned model, we build the tracking controller discussed in Sec. 3.3. The results are depicted in Figure 13, where we observe that our controller has a fast response time and can quickly adapt to changes in the reference signal. It can maintain high accuracy and low phase lag, which makes it well-suited for tracking fast-changing signals.
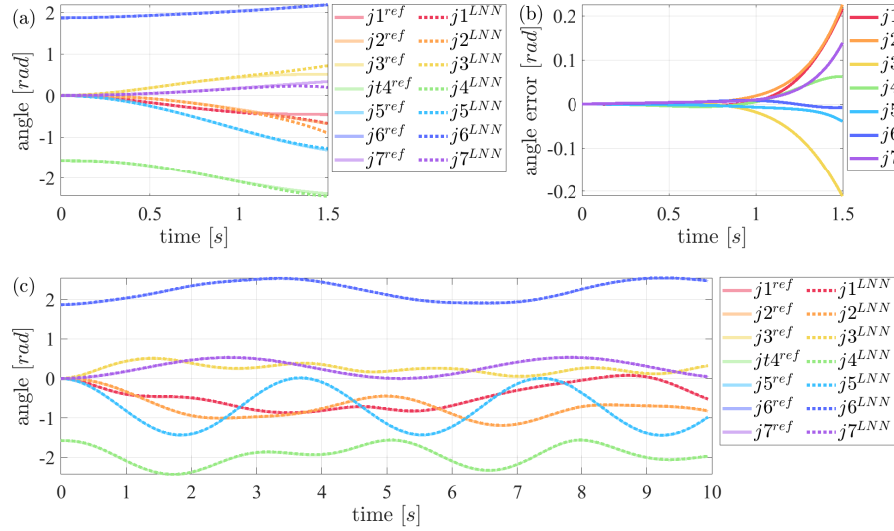
Figure 12: Franka Emika Panda learned model prediction results: (a) shows 1500 steps prediction in a row; (b) is the angle errors of the prediction concerning the ground truth; (c) shows the long prediction results with 50-step window size.
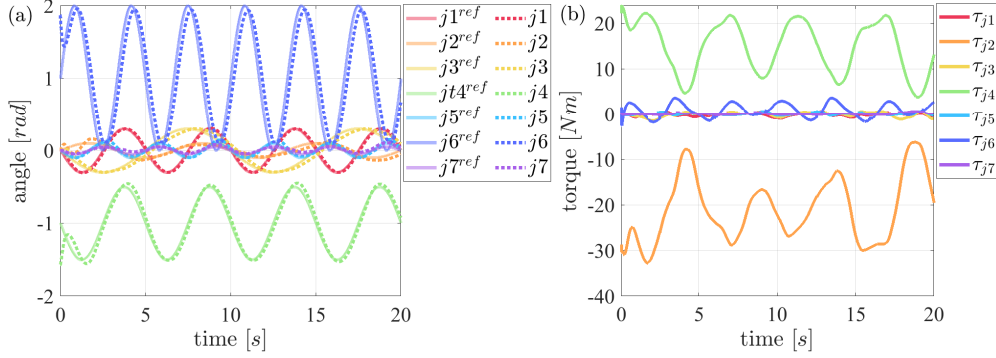

Figure 13: Performance of the model-based controller designed using the model learned by the LNNs. The desired trajectories are plotted with dotted lines.

6 Experimental Validation

6.1 One-segment tendon-driven soft manipulator – NECK

We validate the proposed approach in the platform depicted in Figure 14, which is constructed based on [56, 59]. We consider two different data preprocessing methods. (i) Moving average method: This method reduced the noise and outliers in the data, generating a more stable representation of underlying trends. However, it may overlook intricate relationships between variables, resulting in some information loss. (ii) Polynomial fitting: This method captured non-linear patterns in the data. However, it was susceptible to the influence of outliers, resulting in spurious information that may compromise the quality of the trained model.

The training and testing information is shown in Table 7.

The method of moving average is implemented in MATLAB through the utilization of the `movmean` function, with a prescribed window size of 50 points. The processed data are used for training the LNNs. In Figure 15, we compare the continuous prediction ability of black-box and Lagrangian-based learning models. The prediction performance in this figure indicates that the Lagrangian-based learning model exhibits superior predictive accuracy in this sample. Furthermore, Figure 15 (c) shows that the learning model can realize long-term predictions under the short-term update.

The polynomial fitting of the data is done in MATLAB using the function `polyfit`. The prediction results of the model are shown in Figure 16. The learned model exhibits a decent performance when the
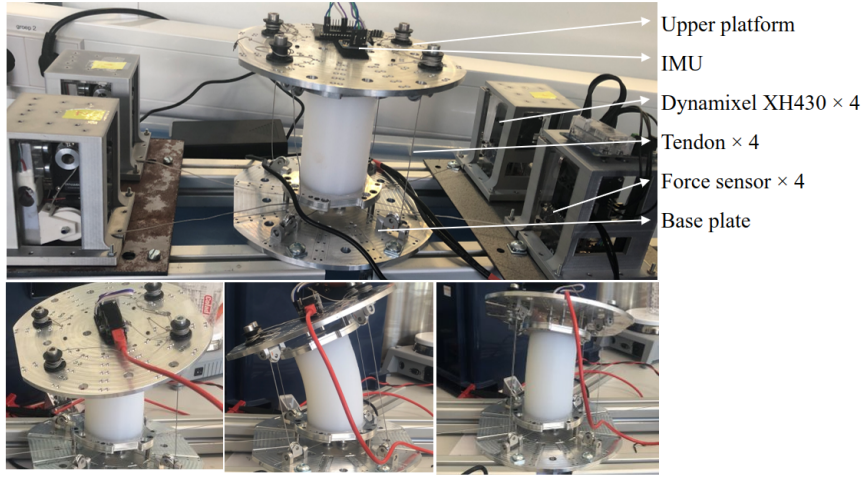
Figure 14: Experiment platform: One-segment tendon-driven soft manipulator equipped with IMU

Table 7: The tendon-driven soft robot – NECK training and testing information

		Black-box model	Lagrangian-based learned model
smoothing	model	60×3	$21 \times 2, 25 \times 2, 10 \times 2$
	sample number	69426	69426
	training epoch	10000	3000
	traning error	$1.985e^{-2} \pm 1.85e^{-1}$	$2.277e^{-2} \pm 2.39e^{-1}$
	prediction error [°]	13.229 ± 60.762 (5s)	2.429 ± 1.259 (5s)
fitting	model	60×3	$21 \times 2, 25 \times 2, 10 \times 2$
	sample number	57950	48200
	training epoch	5000	5000
	training error	$4.431e^{-3} \pm 3.07e^{-2}$	$2.758e^{-3} \pm 2.84e^{-2}$
	prediction error[°]	8.368 ± 12.575 (5s)	6.426 ± 36.237 (5s)
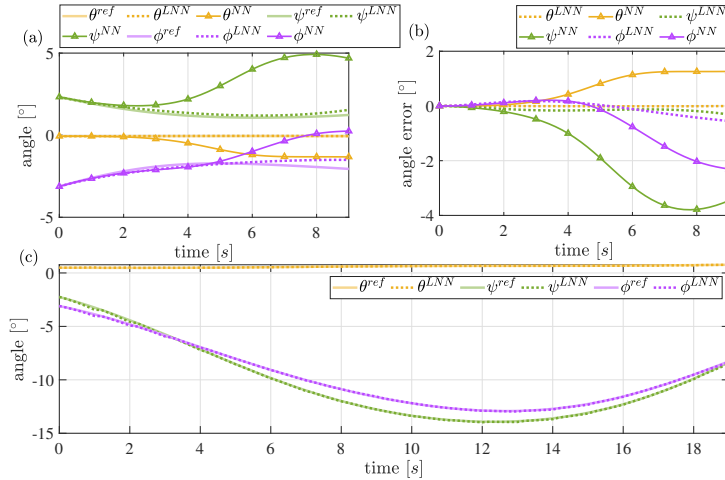
Figure 15: The smoothing data black-box model ($\triangle$) and physics-based learning model (- -) continuous prediction results: (a) and (b) show prediction 43 prediction steps in a row; (c) depicts the prediction results with 5-step window size.

window size is reduced, as shown in Figure 16(c). In contrast to the previous model, this model exhibits significant prediction errors shown in Table 7. This can be caused by the significant noise in the sensors and misinformation caused by the approximation used to fit the data.

6.2 Rigid Robot – Franka Emika Panda

The collected data are processed through a Butterworth filter in MATLAB to reduce noise. Further details are provided in Table 8. In the experiment, we observe small joint acceleration, which results in
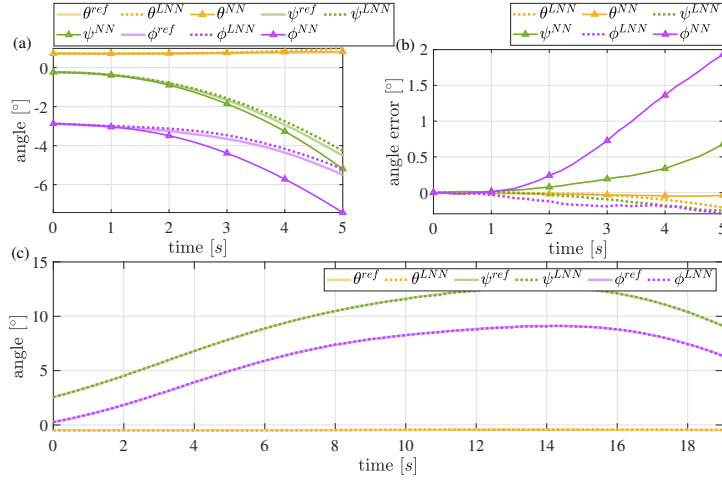
Figure 16: The fitting data black-box model ($\triangle$) and physics-based learning model ($\cdots$) continuous prediction results: (a) and (b) show 25 prediction steps in a row; (c) shows the prediction results with 5-step window size.

minimal velocity change. To prevent the network from focusing solely on learning a large mass matrix and neglecting other important factors, we utilize a scaling sigmoid function. This function ensures that the elements in the mass matrix are scaled within a specific range. For this particular case, we have set the scaling factor to 3.50.

Table 8: Panda experiment detailed information (500Hz)

	Black-box model	Lagrangian-based learned model
model (width $\times$ depth)	120×5	$40\times3, 20\times2$
sample number	550000	25000
training epoch	10000	3000
traning error	$1.371e^{-5} \pm 2.03e^{-5}$	$1.68e^{-7} \pm 6.64e^{-6}$
prediction error[rad]	182.495 ± 64.645 (2s)	2.681 ± 1.383 (2s)

Figure 17 illustrates the predictive performance of our physics-based model, where Figure 17 (b) depicts the continuous prediction error within 2 seconds or 1000 prediction steps and (c) shows that updating the model's input with real-time state data can help us make a long prediction.

A controller based on the equation presented in (23) is proposed for the actual robot. The proportional gain matrix, $K_\mathtt{P}$, is set to a diagonal matrix with entries 600, 600, 600, 600, 250, 150, and 50, respectively. The derivative gain matrix, $K_\mathtt{D}$, is set to a diagonal matrix with entries 30, 30, 30, 30, 10, 10, and 5, respectively. Figure 18 illustrates a series of photographs depicting the periodic movement used to track a sinusoidal trajectory within a time frame of 10 seconds. The whole tracking performance is shown in Figure 19.

Furthermore, we have presented the trajectory of the end-effector, which is a helical motion shown in Figure 20, and its resultant control effect has been visually demonstrated in Figure 19.

In these Figures, we can observe that the designed controller has satisfactory performance, as evidenced by its ability to track a desired trajectory. The tracking error, while present in some joints, remains within acceptable bounds and does not significantly impair the overall performance of the controller in practical applications. An examination of the controller's performance reveals that, while generally effective, its performance exhibits some degree of variability across different joints. The overall performance of the controller remains within acceptable levels and suggests its potential for effective use in real-world applications.
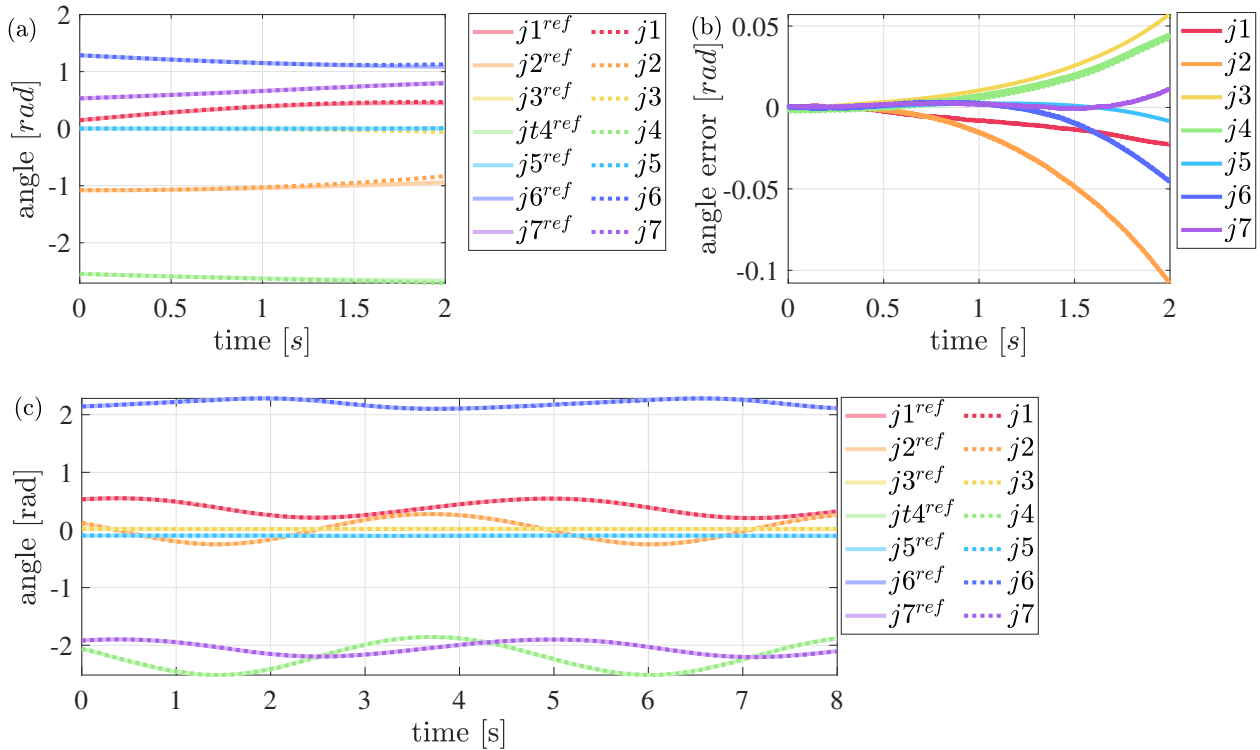
17

Figure 17: Panda physics-based learning model prediction results: (a) and (b) show prediction of about 800 steps in a row; (c) depicts the prediction results with a 5-step window size.

Figure 18: Photo sequence of one periodic movement resulting from the application of the LNN-model-based controller tracking trajectory

7 Conclusions

This paper presented an approach to consider damping and the interaction between robots and actuators in PINNs—specifically, LNNs and HNNs—, improving the applicability of these neural networks for learning dynamic models. Moreover, we used the Runge-Kutta4 method to avoid acceleration measurements, which are often unavailable. The modified PINNs proved suitable for learning the dynamic model of rigid and soft manipulators. For the latter, we considered the PCC approximation to obtain a simplified model of the system.

The modified PINN approach exploits the knowledge of the underlying physics of the system, which results in a largely improved accuracy in the learned models compared with the baseline models, which were trained using a fully connected network. The results show that PINNs exhibit a more instructive and directional learning process because of the prior knowledge embedded into the approach. Notably, physics-based learning models trained with fewer data are more general and robust than the traditional black-box ones. Therefore, continuous long-term and variable step-size predictions can be achieved. Furthermore, the learned model enables decent anticipatory control, where a naive PD can be integrated for a good performance, as illustrated in the experiments performed with the Panda robot.
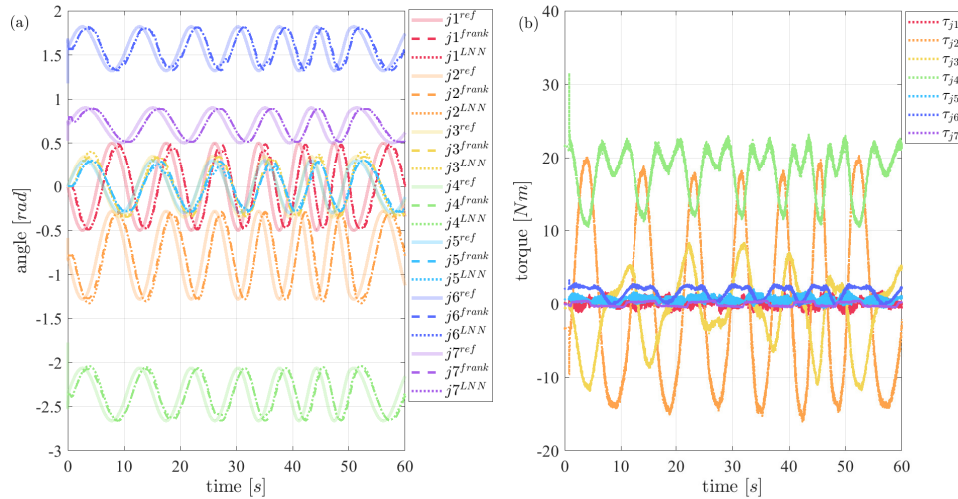
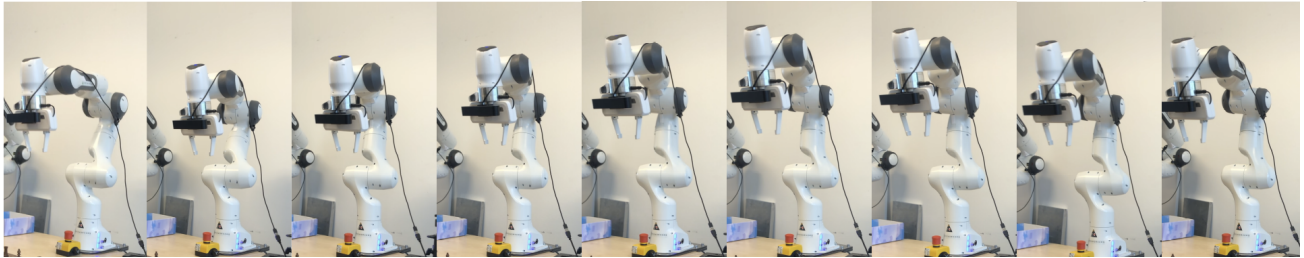Figure 19: Performance of the model-based controller that is designed using the learned model.

Figure 20: Photo sequence of helical motion of the end-effector by using LNN-model-based controller

References

[1] A. I. Chen, M. L. Balter, T. J. Maguire, M. L. Yarmush, *Nature Machine Intelligence* **2020**, *2*, 2 104.

[2] J. Ichnowski, Y. Avigal, V. Satish, K. Goldberg, *Science Robotics* **2020**, *5*, 48 eabd7710.

[3] D. Mukherjee, K. Gupta, L. H. Chang, H. Najjaran, *Robotics and Computer-Integrated Manufacturing* **2022**, *73* 102231.

[4] F. Stella, C. Della Santina, J. Hughes, *Nature Machine Intelligence* **2023**.

[5] L. Buşoniu, T. De Bruin, D. Tolić, J. Kober, I. Palunko, *Annual Reviews in Control* **2018**, *46* 8.

[6] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, et al., *Nature* **2022**, *602*, 7896 223.

[7] N. Rudin, D. Hoeller, P. Reist, M. Hutter, In *Conference on Robot Learning*. PMLR, **2022** 91–100.

[8] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al., *arXiv preprint arXiv:1910.07113* **2019**.

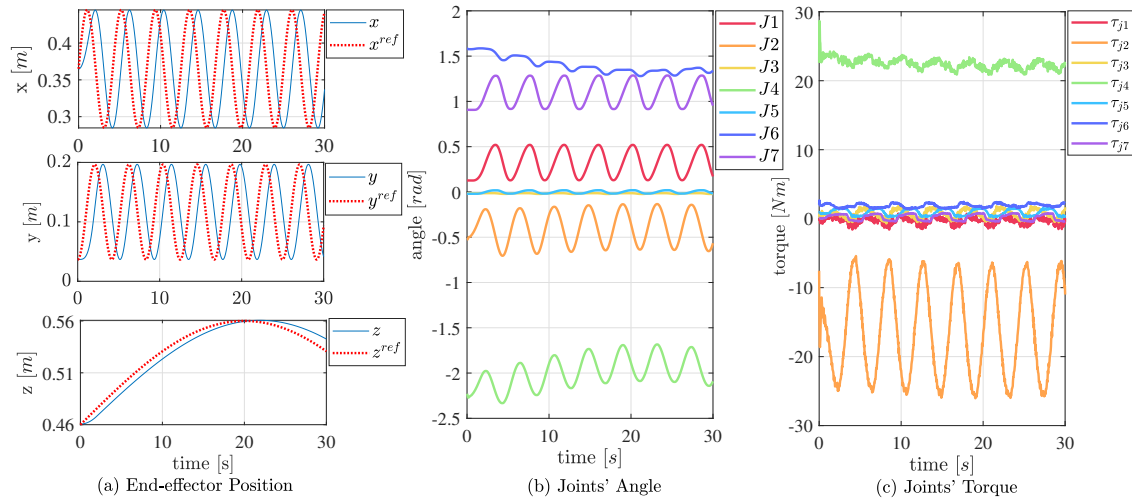(a) End-effector Position   (b) Joints' Angle   (c) Joints' Torque

Figure 21: Performance of the model-based controller that is designed using the learned model: (a) shows the desired end-effector trajectory; (b) shows the corresponding joints' angle and the control results; (c) is the controller's input torques for such motion.

[9] W. Zhao, J. P. Queralta, T. Westerlund, In *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, **2020** 737–744.

[10] P. Kulkarni, J. Kober, R. Babuška, C. Della Santina, *Advanced Intelligent Systems* **2022**, *4*, 1 2100095.

[11] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al., *The International journal of robotics research* **2018**, *37*, 4-5 405.

[12] G. Antonelli, S. Chiaverini, P. Di Lillo, *Nonlinear Dynamics* **2023**, *111*, 7 6487.

[13] H. Beik-Mohammadi, S. Hauberg, G. Arvanitidis, G. Neumann, L. Rozo, *arXiv preprint arXiv:2106.04315* **2021**.

[14] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, V. Sitzmann, In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, **2022** 6394–6400.

[15] J. Urain, N. Funk, G. Chalvatzaki, J. Peters, *arXiv preprint arXiv:2209.03855* **2022**.

[16] A. Daw, A. Karpatne, W. Watkins, J. Read, V. Kumar, *arXiv preprint arXiv:1710.11431* **2017**.

[17] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, *Nature Reviews Physics* **2021**, *3*, 6 422.

[18] F. Djeumou, C. Neary, E. Goubault, S. Putot, U. Topcu, In *Learning for Dynamics and Control Conference*. PMLR, **2022** 263–277.

[19] M. Chen, R. Lupoiu, C. Mao, D.-H. Huang, J. Jiang, P. Lalanne, J. Fan **2021**.

[20] B. Huang, J. Wang, *IEEE Transactions on Power Systems* **2022**, *38*, 1 572.

[21] Z. Mao, A. D. Jagtap, G. E. Karniadakis, *Computer Methods in Applied Mechanics and Engineering* **2020**, *360* 112789.

[22] S. A. Niaki, E. Haghighat, T. Campbell, A. Poursartip, R. Vaziri, *Computer Methods in Applied Mechanics and Engineering* **2021**, *384* 113959.

[23] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, S. Ho, *arXiv preprint arXiv:2003.04630* **2020**.

[24] S. Greydanus, M. Dzamba, J. Yosinski, *Advances in neural information processing systems* **2019**, *32*.

[25] Y. D. Zhong, B. Dey, A. Chakraborty, *Advances in Neural Information Processing Systems* **2021**, *34* 21910.

[26] R. Bhattoo, S. Ranu, N. A. Krishnan, *Machine Learning: Science and Technology* **2023**, *4*, 1 015003.

[27] M. A. Roehrl, T. A. Runkler, V. Brandtstetter, M. Tokic, S. Obermayer, *IFAC-PapersOnLine* **2020**, *53*, 2 9195.

[28] Y. D. Zhong, B. Dey, A. Chakraborty, In *Learning for dynamics and control*. PMLR, **2021** 1218–1229.

[29] R. Bhattoo, S. Ranu, N. Krishnan, *Advances in Neural Information Processing Systems* **2022**, *35* 29789.

[30] M. Lutter, J. Peters, *The International Journal of Robotics Research* **2023**, *42*, 3 83.

[31] C. Della Santina, M. G. Catalano, A. Bicchi, M. Ang, O. Khatib, B. Siciliano, *Encyclopedia of Robotics* **2020**, *489*.

[32] J. K. Gupta, K. Menda, Z. Manchester, M. J. Kochenderfer, *arXiv preprint arXiv:1902.08705* **2019**.

[33] J. K. Gupta, K. Menda, Z. Manchester, M. Kochenderfer, In *Learning for Dynamics and Control*. PMLR, **2020** 328–337.

[34] R. M. Murray, Z. Li, S. S. Sastry, S. S. Sastry, *A mathematical introduction to robotic manipulation*, CRC press, **1994**.

[35] H. K. Khalil **2015**.

[36] C. Della Santina, C. Duriez, D. Rus, *IEEE Control Systems Magazine* **2023**, *43*, 3 30.

[37] Y. Zheng, C. Hu, X. Wang, Z. Wu, *Journal of Process Control* **2023**, *128* 103005.

[38] S. Sanyal, K. Roy, *arXiv preprint arXiv:2209.09025* **2022**.

[39] L. Hewing, J. Kabzan, M. N. Zeilinger, *IEEE Transactions on Control Systems Technology* **2019**, *28*, 6 2736.

[40] I. Mitsioni, P. Tajvar, D. Kragic, J. Tumova, C. Pek, *IEEE Transactions on Robotics* **2023**.

[41] S. S.-E. Plaza, R. Reyes-Baez, B. Jayawardhana, In *Learning for Dynamics and Control Conference*. PMLR, **2022** 520–531.

[42] S. Sánchez-Escalonilla, R. Reyes-Báez, B. Jayawardhana, In *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, **2022** 2463–2468.

[43] F. Arnold, R. King, *Engineering Applications of Artificial Intelligence* **2021**, *101* 104195.

[44] S. Mowlavi, S. Nabi, *Journal of Computational Physics* **2023**, *473* 111731.

[45] J. Nicodemus, J. Kneifl, J. Fehr, B. Unger, *IFAC-PapersOnLine* **2022**, *55*, 20 331.

[46] M. Lutter, K. Listmann, J. Peters, In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, **2019** 7718–7725.

[47] C. Della Santina, *Encyclopedia of Robotics* **2021**, *20*.

[48] C. Laschi, T. G. Thuruthel, F. Lida, R. Merzouki, E. Falotico, *IEEE Control Systems Magazine* **2023**, *43*, 3 100.

[49] P. Pustina, C. Della Santina, F. Boyer, A. De Luca, F. Renda, *arXiv preprint arXiv:2306.07258* **2023**.

[50] M. Lutter, C. Ritter, J. Peters, *arXiv preprint arXiv:1907.04490* **2019**.

[51] L. N. Trefethen, D. Bau, *Numerical linear algebra*, volume 181, Siam, **2022**.

[52] K. B. Petersen, M. S. Pedersen, *The Matrix Cookbook*, Technical University of Denmark, **2008**.

[53] M. Montagna, P. Pustina, A. De Luca, In *I-RIM Conference.* **2023** .

[54] P. Tomei, *IEEE Transactions on automatic control* **1991**, *36*, 10 1208.

[55] R. Kelly, R. Salgado, *IEEE Transactions on Robotics and Automation* **1994**, *10*, 4 566.

[56] B. Deutschmann, J. Reinecke, A. Dietrich, In *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft).* **2022** 54–61.

[57] M. W. Hannan, I. D. Walker, *Journal of robotic systems* **2003**, *20*, 2 45.

[58] C. Della Santina, A. Bicchi, D. Rus, *IEEE Robotics and Automation Letters* **2020**, *5*, 2 1001.

[59] B. Deutschmann, Tendondrivencontinuum, `https://github.com/DLR-RM/TendonDrivenContinuum`, **2022**.