RO57035 ROBOTICS MSC THESE

Learning dynamic models of soft manipulators via physics-inspired neural networks

Author:	Jingyue Liu
Supervisors:	Pablo Borja
	Cosimo Della Santina
Project duration:	December, 2021 – September, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE)
Delft University of Technology, Netherlands

Learning dynamic models of soft manipulators via physics-inspired neural networks

Jingyue Liu

***Abstract*—Soft robots are made of compliant materials, which increase their flexibility but also presents modeling challenges. The difficulty mainly comes from material nonlinearity, infinite degrees of freedom, uncertain parameters, and complex calculations. This project uses physics-inspired neural networks to solve the last two problems. Based on the piece-wise constant curvature approximation, this work modifies Lagrangian and Hamiltonian neural networks for dynamic modeling of soft robots. In particular, local linear damping and actuator models are added to deep Lagrangian and Hamiltonian neural networks, and a one-step integration algorithm is used in the loss calculation. This project justifies the modified network structure and uses these new neural networks to learn the dynamic model of soft manipulators.**

***Index Terms*—Soft manipulators, physics-inspired neural networks, dynamic model, Hamiltonian neural networks, Lagrangian neural networks**

I. Introduction

Rigid robots are used for various tasks in industry and everyday life. However, their rigid bodies limit their maneuverability and capacity to adapt to dynamic environments. In contrast, most creatures in nature, like octopuses, fishes, and worms, have flexible bodies that enable them to adapt well to various habitats. Inspired by that, researchers have explored the design and control of soft robots made of compliant materials [1], e.g., silicon.

As soft robotics develops rapidly, numerous challenges arise. First, the lack of mature simulation environments makes learning problematic. Additionally, soft robots are more difficult to control [2] due to the less precise model and lack of suitable sensors. To compensate for inaccurate dynamic models, model-based soft robotic controllers use high-gain Proportional-Derivative (PD) controllers, leading to non-compliant and potentially dangerous behavior [3]. But slightly improving the model can have a significant positive impact on the performance of the controller [4]. Consequently, determining the dynamic model of the soft manipulator is crucial for motion simulation, structural analysis, and control algorithm development.

A. Current modeling methods for soft robots

Model engineering, black-box model learning, and system identification are three traditional modeling techniques.

Model engineering assumes the target system is a perfectly rigid body with ideal joints interacting with ideal environments. The equations of motion can be derived from Newtonian, Lagrangian, or Hamiltonian mechanics. This method is reliable and relatively precise for rigid body systems. However, the method becomes difficult due to soft robots' compliant materials [5]. To address this issue, researchers have created simplified dynamic models of soft robots by considering various assumptions, e.g., reducing soft robots to rigid body systems [6], centering mass position [7], and abstracting robotic arms as thin rods [8]. Complex dynamics models of continuum manipulators based on finite element analysis have also been proposed, but the tedious calculation makes it impossible to implement the real-time application [9]. Furthermore, poorly understood phenomena, such as friction, wear, and fluids in robots may reduce the accuracy of the analysis of the dynamics.

Data-driven methods like machine learning and deep learning seem suitable for modeling soft robots. These methods avoid complex analysis and predominate in developing kinematic models [10][11][12] and dynamic models [13][14][15] of soft manipulators. The current learning methods for the soft robot modeling are the Gaussian process [16][17][18], Feedforward Neural Networks (FNNs) [14][19], UNet [15] and Recurrent Neural Networks (RNNs) [20][21][22]. Despite the impressive capabilities of the learning methods, the problems they bring cannot be ignored. The black-box model cannot provide valuable information about the system itself. And the learning results may be deceptive. Most studies [23][24][25] show that black-box models can make accurate predictions in the distribution of training data, but are hard to extrapolate. Highly expressive neural networks (NNs) are often overparameterized. Moreover, learning methods ignore the system's physical properties, and the resulting models often violate scientific laws.

B. Physics-inspired neural networks

System identification is the most commonly used modeling method. This approach requires knowledge of the model structure from the laws of physics, followed by parameter estimations. Physics-inspired neural networks (PINNs) can be classified as a category of system identification where the means of parameter estimation is deep learning.

PINNs solve supervised learning tasks while respecting physical laws described by general nonlinear partial differential equations (PDEs). After being proposed in [25] and [24], they have been used extensively in physics for solving PDEs and modeling complex systems, e.g., earth science, fluid mechanics, fiber optics, and materials science. The PINNs framework mainly involves three aspects: a) creating the NNs' structure according to the physics knowledge, b) using scientific knowledge as a loss function, and c) constraining the learned parameters based on physical properties such as symmetry and positivity.

Several works have built continuous-time dynamic models that combine Lagrangian or Hamiltonian mechanics with deep learning, which can be generalized as Deep Lagrangian Networks (DeLaNs) [26], Lagrangian Neural Networks (LNNs) [27], and Hamiltonian Neural Networks (HNN) [28]. These networks ensure the scientificity of the system and significantly reduce the dependence on data by introducing physical priors. These three types of networks are validated in simulations and some simple systems. However, they are hard to apply to robotics. The main reasons are: DeLaNs ignore the energy dissipation and the actuator model; LNNs do not consider external forces and only aim to prove the energy conservation of the learned system; for HNNs, direct momentum measurement is almost impractical.

C. Research objectives

Building the analytical model of soft robots requires many simplifications and complex calculations, and purely data-based models are unreliable. In comparison, PINNs might be the optimal solution. It does not need to make too many assumptions and can reduce the cost of manual calculations. Moreover, providing general laws can reduce the massive data requirements for learning and improve the transparency and reliability of the resulting models. In this project, we use the piece-wise constant curvature (PCC) method [29] to determine the configuration of the soft robot. Based on this approximation, the soft manipulator's equations of motion (EOM) can be proposed and then solved by PINNs.

The objectives of this project are:

- Enriching the DeLaNs' structure to capture more phenomena in modeling, especially the dissipation and actuator model;
- Combining the improved Lagrangian NNs and Hamiltonian NNs with the PCC method to complete the dynamic modeling of the soft manipulator;
- Comparing the models learned by the improved PINNs and FNNs, and evaluating the properties of the proposed method.

The rest of the paper is structured as follows: In Section II, we present the preliminaries. In particular, we briefly review Lagrangian and Hamiltonian mechanics in Subsection II-A, and the PCC method of the soft manipulator is presented in Subsection II-B. In Section III, we introduce the modified network structure in detail and provide the experimental setup. In Section IV, we share the modeling results in simulating one- and double-segment manipulators. In Section V, we show the results of experiments. In Section VI, we discuss our findings and some remaining questions of this project.

II. Preliminaries

A. Lagrangian and Hamiltonian Dynamics

The dynamic equations of almost all physical systems can be expressed in Lagrangian or Hamiltonian mechanics. For the Lagrangian, the state of the system is given by the generalized coordinates $\mathbf{q} \in \mathbb{R}^N$ and their velocities $\dot{\mathbf{q}} \in \mathbb{R}^N$, where N is the number of coordinates. The Lagrangian equation can be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L(\mathbf{q},\dot{\mathbf{q}})}{\partial \dot{q}_j}\right) - \frac{\partial L(\mathbf{q},\dot{\mathbf{q}})}{\partial q_j} = Q_j, \tag{1}$$

where $L(\mathbf{q},\dot{\mathbf{q}}) = K(\mathbf{q},\dot{\mathbf{q}}) - V(\mathbf{q})$. The kinetic energy can be computed by $K(\mathbf{q},\dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$, where $\mathbf{M}(\mathbf{q})$ is the mass matrix. The potential energy $V(\mathbf{q})$ includes gravitational and elastic potential energy. And Q_j is the generalized force corresponding to the j-th generalized coordinate.

Hamiltonian mechanics uses the momentum, $\mathbf{p} \in \mathbb{R}^N$, to replace the velocity $\dot{\mathbf{q}}$ in Lagrangian mechanics and its equations are

$$\frac{d\mathbf{q}}{dt} = \frac{\partial H(\mathbf{q},\mathbf{p})}{\partial \mathbf{p}}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial H(\mathbf{q},\mathbf{p})}{\partial \mathbf{q}}, \tag{2}$$

where $H(\mathbf{q},\mathbf{p}) = K(\mathbf{q},\mathbf{p}) + V(\mathbf{q})$. The kinetic energy is defined as $K(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}(\mathbf{q})^{-1}\mathbf{p}$.

In practical engineering problems, there are speed-dependent dissipative forces that reduce the total energy of the system. The Rayleigh dissipation function is denoted by

$$R(\mathbf{v}) = \frac{1}{2}\sum_{i=1}^{N}(k_x v_{i,x}^2 + k_y v_{i,y}^2 + k_z v_{i,z}^2), \tag{3}$$

where v_i is the i-th particle's velocity, k is the drag coefficient. Considering the dissipation force formula $\vec{F}_d = -\nabla_{\mathbf{v}}R(\mathbf{v})$, the generalized dissipation force can be expressed as

$$\mathbf{F}_d = \mathbf{D}(\mathbf{q})\dot{\mathbf{q}}, \tag{4}$$

where $\mathbf{D}(\mathbf{q})$ is the damping matrix. Besides, the external force can be simplified as

$$\mathbf{F}_{ext} = \mathbf{A}(\mathbf{q})\mathbf{u}, \tag{5}$$

where $\mathbf{u}$ is the input of the system. The $\mathbf{A}(\mathbf{q})$ maps $\mathbf{u}$ to the configuration coordinator.

According to the chain rule, the Lagrangian is

$$\ddot{\mathbf{q}} = \left(\frac{\partial^2 L}{\partial \dot{\mathbf{q}}^2}\right)^{-1}\left(\mathbf{A}(\mathbf{q})\mathbf{u} - \frac{\partial^2 L}{\partial \mathbf{q}\partial \dot{\mathbf{q}}} + \frac{\partial L}{\partial \mathbf{q}} - \mathbf{D}(\mathbf{q})\dot{\mathbf{q}}\right). \tag{6}$$

The Hamiltonian equation becomes

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{bmatrix} = \begin{bmatrix}0 & I\\ -I & -\mathbf{D}(\mathbf{q})\end{bmatrix}\begin{bmatrix}\frac{\partial H}{\partial \mathbf{p}}\\ \frac{\partial H}{\partial \mathbf{q}}\end{bmatrix} + \begin{bmatrix}0\\ \mathbf{A}(\mathbf{q})\mathbf{u}\end{bmatrix}. \tag{7}$$

B. Soft manipulators' dynamic models analysis

Soft robots have continuous deformation, so the number of variables in their PDEs should be infinite. Several works propose hypotheses to obtain the reduced kinematic models of soft manipulators. These works include the Pseudo Rigid method [30], the PCC method [31], and the Frenet-Serret framework [32].

Different modeling approaches ultimately go back to a typical result of constant-curvature kinematics [29]. Thus, in this project, we restrict our attention to the PCC approximation. The PCC method, shown in Fig. 1, defines three parameters for one segment, $\mathbf{q}_i = [\phi_i, \theta_i, \delta L_i]$, where ϕ_i is the plane orientation; θ_i is the curvature in that plane; and δL_i is the change of arc length. In this case, we can describe the dynamic models in Eq. (6) or (7).

The detailed information and the calculation process of the transformation matrix and Jacobian matrix using the PCC method can be found in Appendix A.

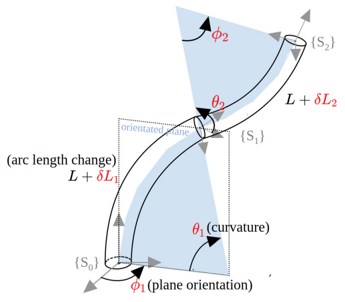


Fig. 1. PCC approach illustration: two-segment soft manipulator is shown, where S_i is the end frame, the blue parts are the orientated plane, L is the original length of each segment

III. Methodology

A. Physics-based neural networks

A classic learning approach is to train an approximation model, e.g., a neural network, $f_{NN} : \mathbf{D} \to \mathbf{T}$, where $\mathbf{D}$ is the input data and $\mathbf{T}$ is the corresponding set of labels, over numerous training samples. The model can then be used to estimate the target variable, $\hat{\mathbf{T}}$.

In this project, we want to know more detailed properties of the target system, like the mass matrix $\mathbf{M}(\mathbf{q})$, the potential forces $\mathbf{G}(\mathbf{q})$, the dissipation matrix $\mathbf{D}(\mathbf{q})$, and the input matrix $\mathbf{A}(\mathbf{q})$. For this purpose, we divide the entire network into four sub-parts.

According to physics, the mass and damping matrices are positive definite and positive semi-definite, respectively. Hence, based on the Cholesky decomposition [33], they can always be decomposed into a lower triangular matrix $\mathbf{L}$ with positive diagonal entries and its transpose, $U = \mathbf{L}\mathbf{L}^T$. The frameworks for $\mathbf{M}(\mathbf{q})$ and $\mathbf{D}(\mathbf{q})$ are shown in Figs. 2 and 3, respectively.

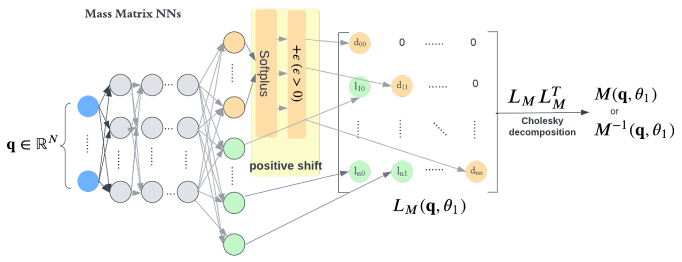


Fig. 2. Diagram for system's mass matrix or inverse mass matrix including a feed-forward neural network, a positive shift for diagonal entries and the Cholesky decomposition

The output of these two networks is a vector of dimension $(N^2+N)/2$, where the first N output values, $\begin{bmatrix}d_{00} & d_{11} & \dots & d_{NN}\end{bmatrix}$, are for the diagonal entries of the lower triangular matrix. Some activation functions, such as Softplus and ReLU, make these values non-negative.

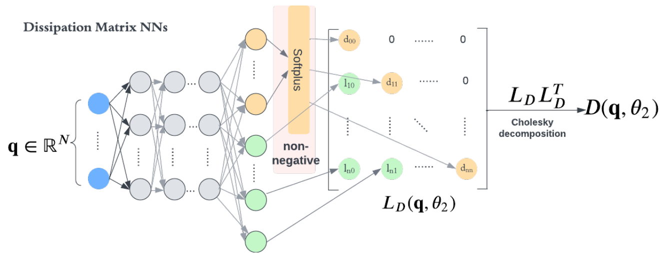

Dissipation Matrix NNs
$\mathbf{q} \in \mathbb{R}^N$
Softplus
non-negative
Cholesky decomposition
$L_D L_D^T$
$D(\mathbf{q}, \theta_2)$
$L_D(\mathbf{q}, \theta_2)$

Fig. 3. Diagram for system's dissipation matrix including a neural network

Then, a small positive shift, $+\epsilon$, can make the values positive for the mass matrix. The other $(N^2-N)/2$ number of values, $\begin{bmatrix} l_{10} & l_{20} & l_{21} \dots & l_{N(N-1)} \end{bmatrix}$, are placed in the lower left corner of the $\mathbf{L}$ matrix. Then, $\mathbf{M}(\mathbf{q})$ and $\mathbf{D}(\mathbf{q})$ can be obtained by the decomposition rule.

A simple feed-forward NN with only one output is used to calculate the value $\mathbf{V}(\mathbf{q}, \theta_3)$ representing the potential energy $\mathbf{V}(\mathbf{q})$, shown in Fig. 4.

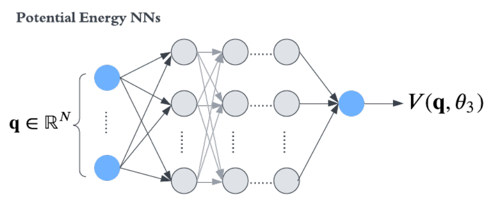


Fig. 4. Diagram for system's potential energy

The output of the input matrix network, shown in Fig. 5, is a vector with dimension MN, which then is transformed into a matrix, $A(\mathbf{q}, \theta_4) \in \mathbb{R}^{N \times M}$.

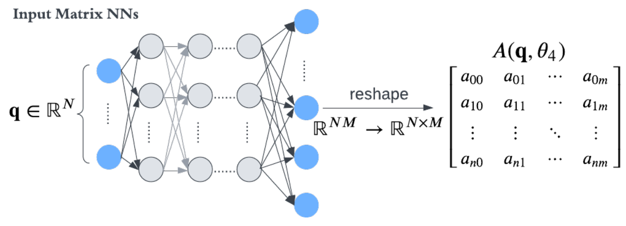

Input Matrix NNs
$\mathbf{q} \in \mathbb{R}^N$
reshape
$\mathbb{R}^{NM} \rightarrow \mathbb{R}^{N \times M}$
$A(\mathbf{q}, \theta_4)$
a_{00} a_{01} $\cdots$ a_{0m}
a_{10} a_{11} $\cdots$ a_{1m}
a_{n0} a_{n1} $\cdots$ a_{nm}

Fig. 5. Diagram for actuator model's input transformation matrix

After having these sub-networks, we can get the estimated $\hat{\mathbf{M}}(\mathbf{q})$, $\hat{\mathbf{V}}(\mathbf{q})$, $\hat{\mathbf{D}}(\mathbf{q})$, and $\hat{\mathbf{A}}(\mathbf{q})$ by feeding them the corresponding $\mathbf{q}$. Then by providing $\begin{bmatrix} \mathbf{q} & \dot{\mathbf{q}} & \mathbf{u} \end{bmatrix}^T$ for Lagrangian NNs or $\begin{bmatrix} \mathbf{q} & \mathbf{p} & \mathbf{u} \end{bmatrix}^T$ for Hamiltonian NNs, $\hat{\ddot{\mathbf{q}}}$ or $\begin{bmatrix} \hat{\dot{\mathbf{q}}} & \hat{\dot{\mathbf{p}}} \end{bmatrix}^T$ can be calculated via Eq. (6) or (7). Besides, we can use the one-step Runge-Kutta algorithm to predict the next state $\begin{bmatrix} \hat{\mathbf{q}} & \hat{\dot{\mathbf{q}}} \end{bmatrix}^T$ or $\begin{bmatrix} \hat{\mathbf{q}} & \hat{\mathbf{p}} \end{bmatrix}^T$.

If the mechanical system's state transitions are given by some dataset $\mathfrak{D} = [\mathbf{q}_k, \dot{\mathbf{q}}_k, \mathbf{u}_k, \mathbf{q}_{k+1}, \dot{\mathbf{q}}_{k+1} \mid k \in \{0, ..., N-1\}]$ or $\mathfrak{D} = [\mathbf{q}_k, \mathbf{p}_k, \mathbf{u}_k, \mathbf{q}_{k+1}, \mathbf{p}_{k+1} \mid k \in \{0, ..., N-1\}]$, the loss function is defined as

$$loss := \frac{1}{N} \sum_{k=1}^{N-1} \| \mathbf{q}_{k+1} - \hat{\mathbf{q}}_{k+1} \|^2 + \lambda \| \dot{\mathbf{q}}_{k+1} - \hat{\dot{\mathbf{q}}}_{k+1} \|^2 \tag{8}$$

for Lagrangian NNs, or

$$loss := \frac{1}{N} \sum_{k=1}^{N-1} \| \mathbf{q}_{k+1} - \hat{\mathbf{q}}_{k+1} \|^2 + \lambda \| \mathbf{p}_{k+1} - \hat{\mathbf{p}}_{k+1} \|^2 \tag{9}$$

for Hamiltonian NNs, where $\lambda \in \mathbb{R}$ is a factor that measures the weight of position and velocity in the loss function. If the acceleration is measurable, the integration process can be omitted and the loss can be directly obtained by

$$loss := \frac{1}{N} \sum_{k=1}^{N} \| \ddot{\mathbf{q}}_k - \hat{\ddot{\mathbf{q}}}_k \|^2 . \tag{10}$$

Gradient descent methods such as Stochastic Gradient Descent (SGD), Adam or RAdam are used to tune the parameters, $\theta_1, \theta_2, \theta_3$ and θ_4, to minimize the loss.

Based on the proposed analysis, a complete PINN can be constructed. Figs. 6 and 7 show PINNs for the Lagrangian and Hamiltonian, respectively.

B. Simulation and experiment design

To validate the feasibility of the proposed PINNs, we apply the modified Lagrangian and Halmitonian NNs to learn the dynamic models of some simple simulated systems. For this purpose, two tasks are created: (**Task 1**) a mass-spring-damper system; and (**Task 2**) a double pendulum system. Then, three more tasks are created to test the adaptability of the method to the soft manipulator: (**Task 3**) a one-segment planar soft manipulator with momentum data; (**Task 4**) a one-segment spatial soft manipulator with momentum and velocity data; and (**Task 5**) a two-segment spatial soft manipulator with velocity data. For the practical experiment, our experimental platform is a one-segment soft manipulator driven by four Dynamixel XH430-W350 motors and equipped with four force sensors and one IMU.

Data Generation: Mathematical models in Python and MATLAB generate training data for simulation tasks. For Tasks 1 and 2, three different initial states are randomly selected to collect 10 seconds of data. The learned models are tested with more different trajectories. For Tasks 3 and 4, ten combinations of initial states and inputs are used to generate data, each combination providing 10 seconds of data. For Task 5, we use Simulink's

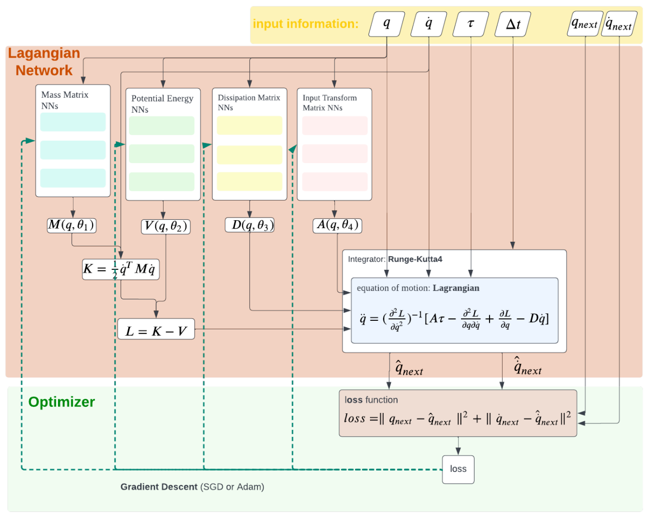


Fig. 6. The overview of Lagrangian Neural Networks

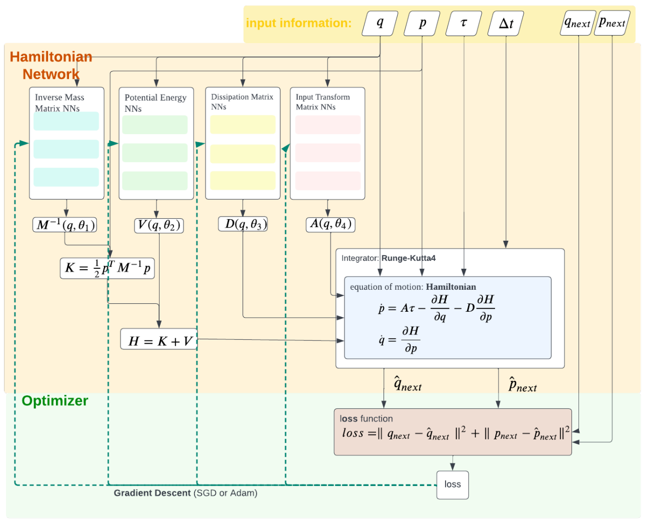


Fig. 7. The overview of Hamiltonian Neural Networks

variable-step and tiny fixed-step strategies to generate two different datasets. A variable-step approach is used to create twelve different 60-second trajectories. And they are resampled in MATLAB at fixed frequencies of $50Hz$, $100Hz$, and $1000Hz$. Among them, eight trajectories' data are used for training. The fixed-step strategy generates five trajectories, three of which are used for training. More details can be found in Table IV-B. In practical experiments, the tendon-driven continuum robot [34] generates data driven by four motors. The motor's signal is a sinusoidal signal with different frequencies and amplitudes. The tip orientation data of the robot is recorded by the IMU. A total of 128 trajectories are recorded. The last eight trajectories are used for the test.

Baseline Models: We set up baseline models for all tasks to demonstrate the value of incorporating physics knowledge into NNs. The baseline model is trained by FNNs using the same dataset but with more data.

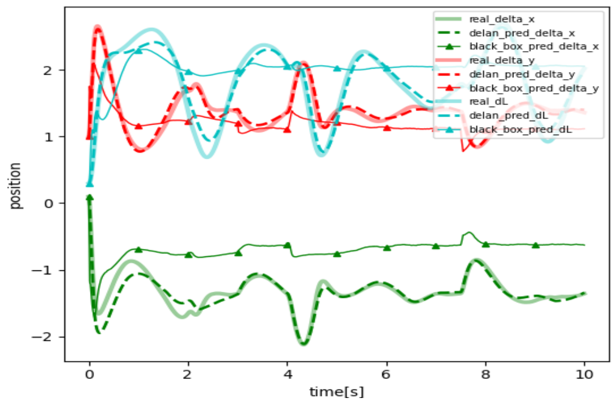
Model training: JAX and dm-Haiku packages in Python build all NNs used in this project. The JAX Autodiff system provides the computation of partial derivatives and Hessian in the loss function. The AdamW [35] in Optax package is used to optimize the model's parameters. The batch size of the optimizer varies between 500 and 1200. We record the values and variances of training loss and test loss during all training.

IV. Simulation Results

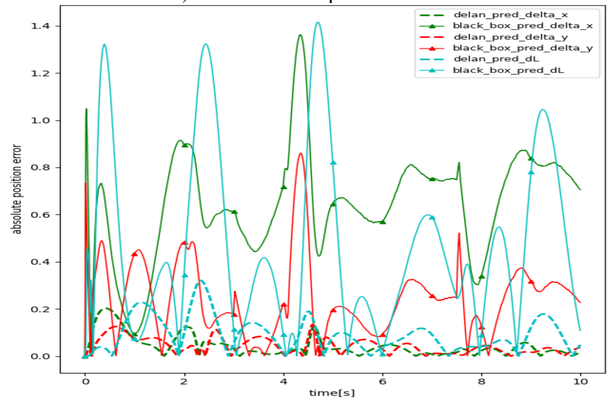
The modified Lagrangian NNs can be verified via Tasks 1, 2, and 3. The corresponding results are shown in Table IV-B and Appendix B.

A. one-segment 3D soft manipulator

In the task, the configuration-defined method in [36] is used, where θ and ϕ in the PCC method are replaced by the difference of arc length Δ_{xi} and Δ_{yi}.

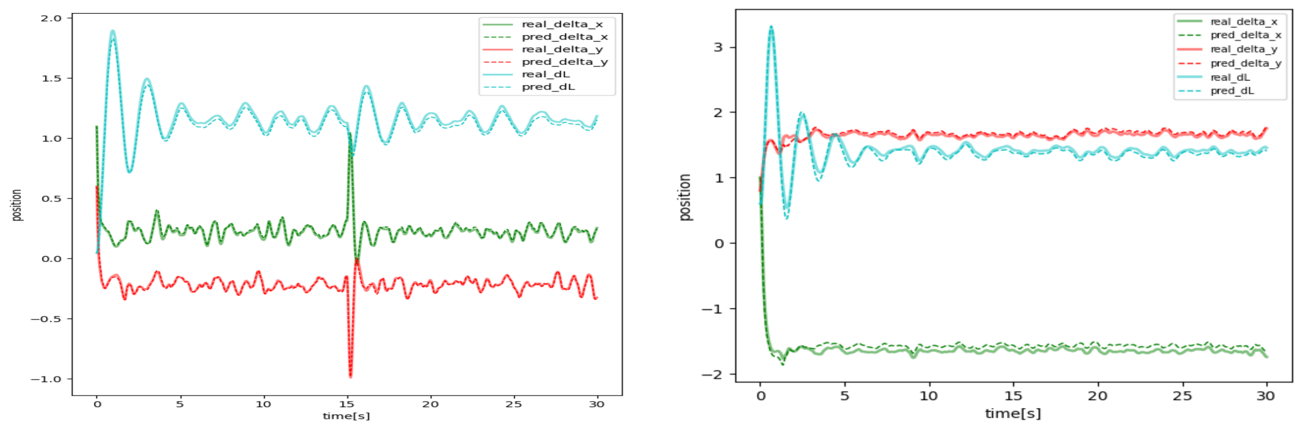


A) Prediction comparison results



B) Prediction error

Fig. 8. Task 4: Comparison of learned models: A) shows the black-box model (△) vs physics-based learning model (- -) predictions with the same initial state for 10s; B) shows the absolute error of the black-box model (△) and physics-based learning model (- -) prediction.



A) Lagrangian NNs long time prediction results

B) Hamltonian NNs long time prediction results

Fig. 9. Task 4: 30s prediction results: A) is the Lagrangian-based learned model B) is the Hamiltonian-based learned model

The black-box model is trained using 19188 samples for 15000 epochs, while the Lagrangian-based model based on Fig. 6 is trained with randomly chosen 8000 samples from the same dataset.

The prediction results of these two learned models are compared in Fig. 8. The black-box model quickly loses its predictive ability as observed in this figure. In contrast, the Lagrangian-based model is able to predict future states for any initial state and input signal.

This system is also learned with the Hamiltonian NNs in Fig. 7. Fig. 9 shows the long-term predictive ability of these two physics-based learning models.

Besides, the matrices obtained from these two physics-based learning models are shown in Tables I and II. As Table II shows, the Hamiltonian NNs can learn the physically meaningful matrices, while the Lagrangian NNs only learns one of the solutions satisfying the Lagrangian equation. A closer inspection of the matrices of Table I shows that the mass matrices and dissipation matrices obtained from the Lagrangian NNs have a transform relationship with the true value of the form, $M(q) = P(q)\hat{M}(q)$ and $D(q) = P(q)\hat{D}(q)$.

The results show that physics-based NNs exhibit better learning ability and efficiency in dynamic modeling problems than FNNs.

B. *two-segment 3D soft manipulator*

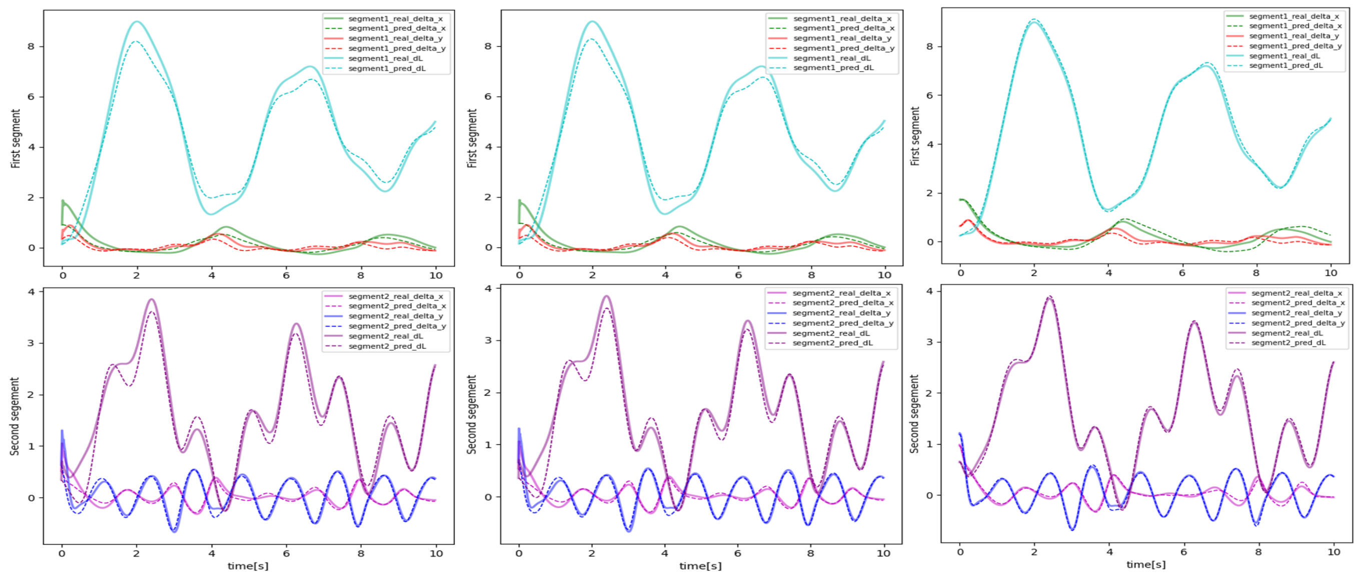
In this task, models at $50Hz$, $100Hz$ and $1000Hz$ are trained by Lagrangian NNs based on Fig. 6 for 5500 epochs with 45000 samples.

TABLE I
TASK 4: MATHEMATICAL MODEL MATRICES VS. PHYSICS-BASED LEARNING MODEL MATRICES (LAGRANGIAN LEARNING MODEL)

q	mathematical model				physics-based learning model				
	M(q)	D(q)	G(q)	A(q)	$\hat{M}(q)$	$\hat{D}(q)$	$\hat{G}(q)$	$\hat{A}(q)$	P
$\begin{bmatrix}1.2\\-0.2\\0.15\end{bmatrix}$	$\begin{bmatrix}1.728e-03 & -3.121e-05 & -1.957e-03\\-3.121e-05 & 1.546e-03 & 3.261e-04\\-1.957e-03 & 3.261e-04 & 9.286e-02\end{bmatrix}$	$\begin{bmatrix}0.1 & 0 & 0\\0 & 0.1 & 0\\0 & 0 & 0.1\end{bmatrix}$	$\begin{bmatrix}1.293\\-0.216\\-1.149\end{bmatrix}$	$\begin{bmatrix}-0.037 & -0.9934 & 0.065\\0.777 & 0.037 & -0.011\\0. & 0. & 0.771\end{bmatrix}$	$\begin{bmatrix}4.232e-3 & 1.201e-3 & -0.029\\1.201e-3 & 5.990e-3 & -0.015\\-0.029 & -0.015 & 0.586\end{bmatrix}$	$\begin{bmatrix}0.161 & -0.017 & 0.004\\-0.017 & 0.333 & -0.008\\0.004 & -0.008 & 0.347\end{bmatrix}$	$\begin{bmatrix}2.436\\-0.612\\-5.246\end{bmatrix}$	$\begin{bmatrix}0.120 & -1.715 & -0.213\\3.049 & -0.193 & -0.126\\-0.343 & 1.011 & 3.396\end{bmatrix}$	$\begin{bmatrix}0.611 & -0.020 & 0.0250\\-0.023 & 0.280 & 0.005\\.330 & 0.150 & 0.246\end{bmatrix}$
$\begin{bmatrix}0.8\\0.2\\0.3\end{bmatrix}$	$\begin{bmatrix}3.639e-03 & 4.518e-05 & -1.936e-03\\4.518e-05 & 3.470e-03 & -4.841e-04\\-1.936e-03 & -4.841e-04 & 9.666e-02\end{bmatrix}$		$\begin{bmatrix}0.894\\0.223\\-1.090\end{bmatrix}$	$\begin{bmatrix}0.026 & -0.994 & 0.064\\0.897 & -0.026 & 0.016\\0. & 0. & 0.890\end{bmatrix}$	$\begin{bmatrix}6.930e-3 & 1.839e-3 & -0.025\\1.839e-3 & 0.012 & -0.019\\-0.025 & -0.019 & 0.503\end{bmatrix}$	$\begin{bmatrix}0.166 & -0.008 & -0.002\\-0.008 & 0.327 & -0.009\\-0.002 & -0.009 & 0.346\end{bmatrix}$	$\begin{bmatrix}1.618\\0.813\\-4.670\end{bmatrix}$	$\begin{bmatrix}0.187 & -1.655 & -0.195\\2.970 & -0.254 & -0.126\\-0.401 & 1.005 & 3.425\end{bmatrix}$	$\begin{bmatrix}0.623 & -0.021 & 0.027\\-0.018 & 0.310 & 0.011\\0.205 & 0.095 & 0.261\end{bmatrix}$
$\begin{bmatrix}-1.0\\-0.6\\0.1\end{bmatrix}$	$\begin{bmatrix}1.245e-03 & 5.786e-05 & 1.406e-03\\5.786e-05 & 1.183e-03 & 8.435e-04\\1.406e-03 & 8.435e-04 & 9.342e-02\end{bmatrix}$		$\begin{bmatrix}-1.067\\-0.640\\-1.212\end{bmatrix}$	$\begin{bmatrix}0.093 & -0.944 & -0.047\\0.844 & -0.093 & -0.028\\0. & 0. & 0.788\end{bmatrix}$	$\begin{bmatrix}3.297e-03 & 4.995e-04 & -2.668e-02\\4.995e-04 & 4.186e-03 & -1.486e-02\\-2.668e-02 & -1.486e-02 & 6.665e-01\end{bmatrix}$	$\begin{bmatrix}0.1821 & -0.011 & 0.003\\-0.0107 & 0.312 & 0.014\\0.003 & 0.014 & 0.341\end{bmatrix}$	$\begin{bmatrix}-1.460\\-1.894\\-5.008\end{bmatrix}$	$\begin{bmatrix}-0.119 & -1.694 & -0.108\\3.043 & -0.097 & -0.417\\-0.997 & 0.601 & 2.470\end{bmatrix}$	$\begin{bmatrix}0.565 & 0.038 & 0.012\\0.042 & 0.309 & 0.012\\0.817 & 0.307 & 0.297\end{bmatrix}$

TABLE II
TASK 4: MATHEMATICAL MODEL MATRICES VS. PHYSICS-BASED LEARNING MODEL MATRICES (HAMILTONIAN LEARNING MODEL)

q	mathematical model				physics-based learning model			
	$M^{-1}(q)$	D(q)	G(q)	A(q)	$\hat{M}^{-1}(q)$	$\hat{D}(q)$	$\hat{G}(q)$	$\hat{A}(q)$
$\begin{bmatrix}1.2\\-0.2\\0.15\end{bmatrix}$	$\begin{bmatrix}593.089 & 9.346 & 12.465\\9.346 & 647.606 & -2.078\\12.465 & -2.078 & 11.039\end{bmatrix}$	$\begin{bmatrix}0.1 & 0 & 0\\0 & 0.1 & 0\\0 & 0 & 0.1\end{bmatrix}$	$\begin{bmatrix}1.293\\-0.216\\-1.149\end{bmatrix}$	$\begin{bmatrix}-0.037 & -0.999 & 0.065\\0.777 & 0.037 & -0.011\\0. & 0. & 0.771\end{bmatrix}$	$\begin{bmatrix}600.323 & 16.899 & 15.670\\16.899 & 622.923 & -1.336\\15.670 & -1.336 & 11.605\end{bmatrix}$	$\begin{bmatrix}1.017e-01 & 3.440e-03 & 8.115e-05\\3.440e-03 & 1.0470e-01 & -4.393e-04\\8.115e-05 & -4.393e-04 & 9.910e-02\end{bmatrix}$	$\begin{bmatrix}1.333\\-0.179\\-1.152\end{bmatrix}$	$\begin{bmatrix}-0.0576 & -0.942 & 0.049\\0.831 & 0.022 & -0.039\\-0.002 & 0.01 & 0.781\end{bmatrix}$
$\begin{bmatrix}0.8\\0.2\\0.3\end{bmatrix}$	$\begin{bmatrix}277.757 & -2.842 & 5.550\\-2.842 & 288.415 & 1.388\\5.550 & 1.388 & 10.464\end{bmatrix}$		$\begin{bmatrix}0.894\\0.223\\-1.090\end{bmatrix}$	$\begin{bmatrix}0.026 & -0.994 & 0.064\\0.897 & -0.026 & 0.016\\0. & 0. & 0.890\end{bmatrix}$	$\begin{bmatrix}285.011 & 11.078 & 6.647\\11.078 & 292.460 & 2.056\\6.647 & 2.056 & 10.586\end{bmatrix}$	$\begin{bmatrix}1.01e-01 & 3.475e-03 & 6.555e-04\\3.475e-03 & 1.034e-01 & -7.452e-05\\6.555e-04 & -7.452e-05 & 9.872e-02\end{bmatrix}$	$\begin{bmatrix}0.928\\0.254\\-1.096\end{bmatrix}$	$\begin{bmatrix}2.445e-02 & -9.552e-01 & 4.540e-02\\9.243e-01 & -2.873e-02 & -1.999e-02\\-6.097e-03 & 2.900e-03 & 8.847e-01\end{bmatrix}$
$\begin{bmatrix}-1.0\\-0.6\\0.1\end{bmatrix}$	$\begin{bmatrix}818.576 & -31.462 & -12.034\\-31.462 & 852.136 & -7.220\\-12.034 & -7.220 & 10.951\end{bmatrix}$		$\begin{bmatrix}-1.067\\-0.640\\-1.212\end{bmatrix}$	$\begin{bmatrix}0.093 & -0.944 & -0.047\\0.844 & -0.093 & -0.028\\0. & 0. & 0.788\end{bmatrix}$	$\begin{bmatrix}826.142 & 1.609 & -11.095\\1.609 & 790.088 & -9.921\\-11.095 & -9.921 & 11.208\end{bmatrix}$	$\begin{bmatrix}0.104 & -0.007 & -0.002\\-0.007 & 0.106 & -0.004\\-0.002 & -0.004 & 0.105\end{bmatrix}$	$\begin{bmatrix}-1.01\\-0.644\\-1.293\end{bmatrix}$	$\begin{bmatrix}0.053 & -0.966 & -0.065\\0.869 & -0.027 & -0.019\\-0.032 & 0.0232 & 0.872\end{bmatrix}$



A) 50Hz B) 100Hz C) 1000Hz

Fig. 10. Task 5: Two-segment soft manipulator prediction performances under different sampling frequencies

Fig.10 summarizes the prediction results of these three models. The figure shows that the higher the sampling frequency, the more accurate the learned model.

physics-based learning model are compared in Fig. 11. The figure shows that the black-box model, converges during training, but has no predictive ability.

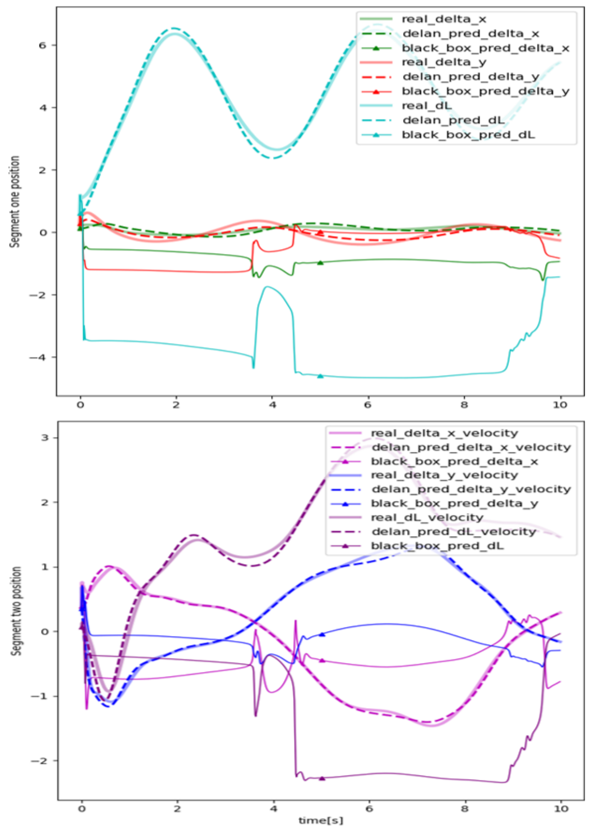


Fig. 11. Task 5: Black-box model (△) vs physics-based learning model (- -) predictions with the same initial state as the true trajectory

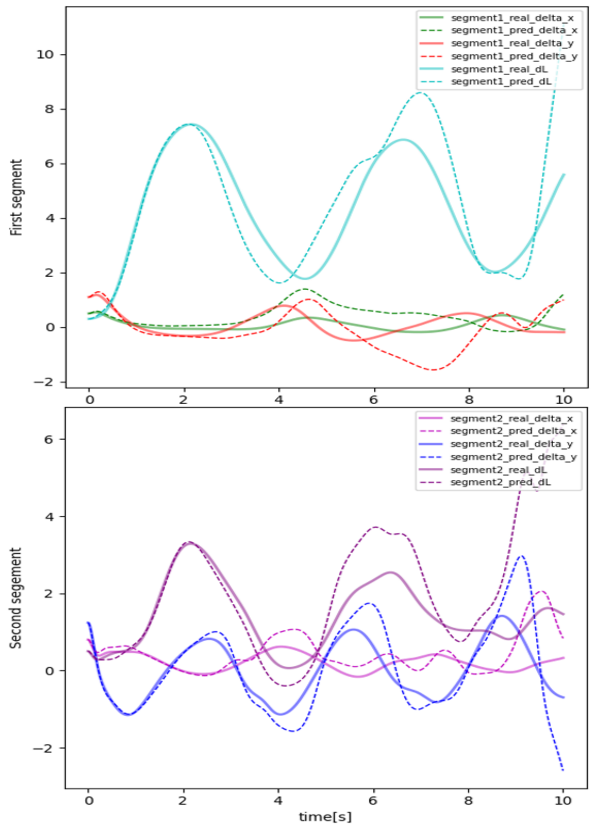


Fig. 12. Task 5: physics-based learning model (- -) trained with the fixed-step sampling frequency data

The prediction results of the black-box model and the

In addition, the results of the Lagrangian-based model

TABLE III
DETAILED INFORMATION OF THE FIVE SIMULATION TASKS

	Task	Spring-Damping System	Double Pendulum	one-segment planar soft manipulator	one-segment spatial soft manipulator
Black-Box model	timestep	0.02s	0.08s	0.02s	0.005s
	model	4 * 2	32 * 4	32 * 2	128*5
	sample number	41,599	5000	11000	19188
	training epoch	4.0 * 1e4	2.5 * 1e4	2.4 * 1e4	15000
	Train error	1.712e-04 ± 4.77e-04	3.790e-05 ± 9.65e-05	7.901e-05 ± 2.32e-04	6.891e-05 ± 4.63e-04
	Test error	1.774e-04 ± 4.39e-04	6.432e-05 ± 1.43e-04	8.121e-05 ± 5.32e-04	1.606e-04 ± 6.65e-03
	prediction error	2.001 ± inf (2.7s)	1.697 ± inf(2.7s)	0.110 ± inf(5s)	7.647 ± 10.413 (5s)
Physics-based learning model	model	(4 * 2)*2 + 4 * 2	(32 * 4)*2 + 32* 4	(128* 5)*2 + 128 *5 + 4*3	(30*3)*2 + 5*3 + 15* 2
	sample number	10000	5000	5000	8.000
	training epoch	6000	8.200	14000	5.000
	Train error	5.969e-08 ± 1.69e-07	3.277e-06 ± 9.28e-06	8.827e-07 ± 1.34e-05	8.302e-05 ± 1.48e-03
	Test error	5.172e-08 ± 1.23e-07	3.048e-06 ± 7.69e-06	4.162e-06 ± 7.38e-06	6.513e-05 ± 1.18e-03
	prediction error	1.303e-04 ± 5.324e-06 (2.7s)	2.231e-04 ± 1.332e-03(2.7s)	5.214e-06 ± 6.345e-07(5s)	0.171 ± 0.272(5s)
	Task	two-segment spatial soft manipulator			
Black-Box model	timestep	fixed time-step (0.0002s)	variable time-step (resample as 50Hz)	variable time-step (resample as 100 Hz)	variable time-step (resample as 1000 Hz)
	model	152*3		152*3	
	sample number	160000		59200	
	training epoch	1.5 * 1e4		1.5 * 1e4	
	Train error	1.758e-04 ± 2.99e-04		3.536e-04 ± 1.08e-03	
	Test error	1.865e-04 ± 3.38e-04		4.445e-04 ± 1.60e-02	
	prediction error	172.765 ± 231.762 (10s)		44.683± 4.51810s)	
Physics-based learning model	model	(42*3)*2 + 5*2 + 42*2	(42*3)*2 + 5*2 + 42*2	(42*3)*2 + 5*2 + 42*2	(42*3)*2 + 5*2 + 42*2
	sample number	42000	45000	45000	45000
	training epoch	5500	5500	5500	5500
	Train error	4.127e-09 ± 1.12e-08	5.916e-04 ± 8.61e-03	1.652e-04 ± 2.12e-02	1.822e-07 ± 6.67e-06
	Test error	2.523e-09 ± 6.03e-09	9.723e-04 ± 1.33e-02	2.748e-04 ± 5.23e-02	2.322e-07 ± 3.21e-06
	prediction error	16.715± 9.876(10s)	2.098 ± 1.253(10s)	1.690 ± 0.673(10s)	0.089 ± 0.278(10s)

trained on the fixed-step size dataset are shown in the Fig.12. From this figure we can see that the data quality issues due to the singularity of the mathematical model eventually caused the accuracy problem of the learned model.

All the results of the simulation are included in Table IV-B. And more detailed information can be found in Appendix B.

V. Experimental Results

The experiment platform is shown in Fig. 13, which is constructed based on [37] . As discussed in Subsection III-B, the manipulator is driven by four motors, and the IMU fixed on the top plate of the manipulator records the angle data of the arm tip. The recorded data are preprocessed by polynomial fitting or moving average. The data and its processing visualization results can be found in Appendix B.

A. Smoothing data model results

We implement the moving average method in MATLAB using the $movmean$ function with a window size of 20. After smoothing and resampling, the processed data is used for training by the Lagrangian NNs based on Fig. 6. Fig. 14 compares the continuous prediction ability

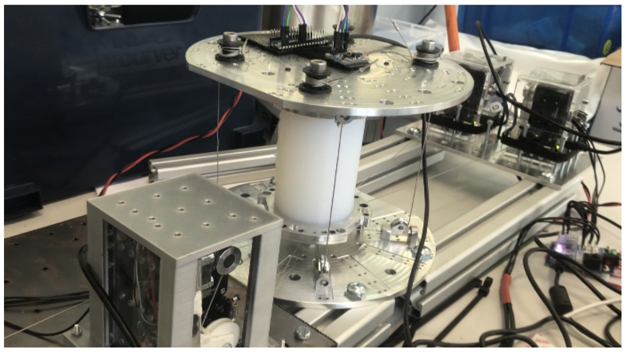


Fig. 13. Experiment platform: One-segment cable-driven soft manipulator equipped with IMU

of black-box and Lagrangian-based learning models. The physics-based model can make relatively accurate predictions within 5 seconds, which means it can predict the system's state 25 times in a row.

The long-term predictions can be realized by updating the prediction results with the real value every 3 seconds, as shown in Fig.15.

B. polynomial fitting data model results

The polynomial fitting of the data is done in MATLAB by function $polyfit$. After processing, 48200 samples

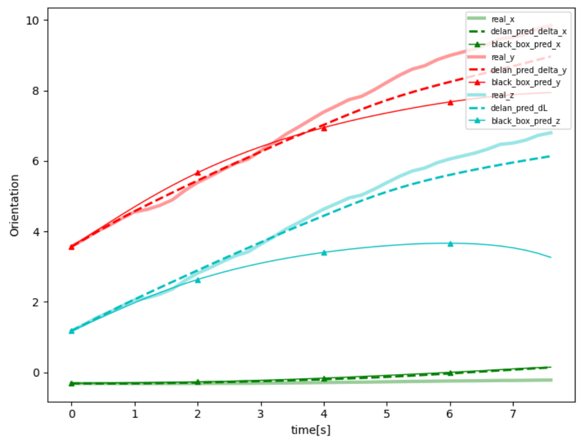


Fig. 14. The smoothing data black-box model (△) and physics-based learning model (- -) continuous prediction results for 8 seconds

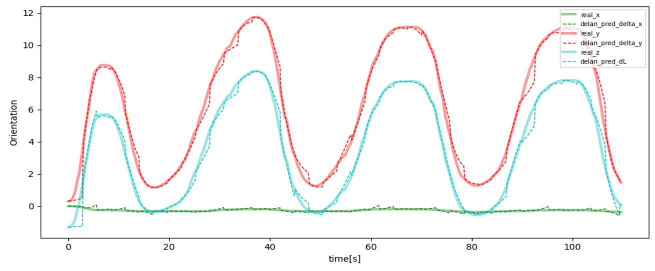


Fig. 15. Smoothing data learning model long time predictions with 3 seconds state updating

are available for training. The prediction results of the model are shown in Fig. 16.

Fig. 17 shows the prediction results of the physics-based learning model updating prediction results every 3 seconds. More detailed training and prediction results are shown in Table IV and Appendix B.

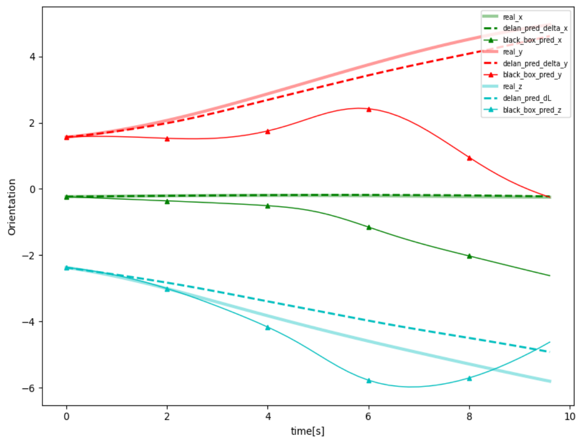


Fig. 16. Fitting data black-box model (△) and physics-based learning model (- -) continuous prediction results for 8 seconds

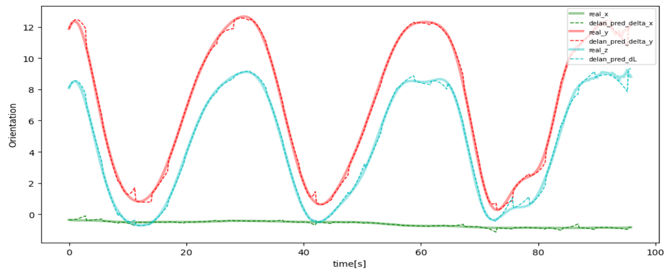


Fig. 17. Fitting data learning model long time predictions with 3 seconds state updating

TABLE IV
DETAILED INFORMATION OF THE PRACTICAL EXPERIMENT

		one-segment cable soft manipulator	
		smoothing	polynomial fitting
Black-Box model	step size	0.2s	0.2s
	model	60*3	60*3
	sample number	69426	57950
	training epoch	10000	5000
	Train error	1.985e-02 ± 1.85e-01	4.431e-03 ± 3.07e-02
	Test error	2.311e-02 ± 2.89e-01	4.350e-03 ± 4.07e-02
	prediction error	13.229± 60.762 (5s)	8.368±12.575 (5s)
Physics-based learning model	model	(21*2)*2 + 25*2 +10*2	(21*3)*2 + 18*2 +18*2
	sample number	69426	48200
	training epoch	3000	5000
	Train error	2.277e-02 ± 2.39e-01	2.758e-03 ± 2.84e-02
	Test error	2.701e-03 ± 2.92e-02	2.633e-03 ± 3.31e-02
	prediction error	2.429 ± 1.259 (5s)	6.426±36.237(5s)

VI. Discussion

A. Findings

All the simulation tasks illustrated that learning becomes more instructive and directional when guided by physical knowledge. Models trained with smaller datasets and specific data domains but with physical knowledge become more general and robust. Therefore, continuous long-term and variable step-size predictions can be easily achieved. In particular, compared to FNNs, physics-based learning methods significantly reduce data dependence, optimize predictive ability, and mend the shortcomings of the black-box nature of NNs.

The simulations also show that the accuracy of the learned model increases with decreasing sampling frequency. The sampling frequency represents the time interval between the training data and the labels. One possible explanation is that the accuracy of the Runge-Kutta integration algorithm becomes low when the step size is large. In this case, we need to increase the sampling frequency or use the acceleration information directly as training labels to avoid this problem.

We also compare the advantages and disadvantages of Lagrangian and Hamiltonian NNs in this project. The Hamiltonian NNs are low computational complexity since the computation of the Hessian matrix is avoided. And it gives the solution that satisfies the parameters of the physical system. For Lagrangian neural networks, it is surprising that as the complexity of the system increases, the relationship between the learned matrix and the actual matrix changes from simple scaling to some unknown transformation. If the system is a two-segment manipulator, we can see the similarity between these two matrices but not their exact relationship. However, a great advantage of Lagrangian NNs over Hamiltonian NNs is that velocity measurement is practical in the real world.

Based on the results of the actual experiments, we can say that the physics-based NNs can learn the dynamic model of the soft robot and make accurate predictions ten times in a row. However, compared to their simulations, the models trained on experimental data are far from satisfactory. These unpleasant results are probably related to:

- The sampling frequency of the IMU is very low. From the simulation results, we know that it is challenging to learn an accurate model even at $50Hz$. Using 5Hz data for model training definitely has a significant negative impact on the final result.
- Excessive data preprocessing methods cause the physical properties of the data to be lost. Comparing the two data processing methods, we find that polynomial fitting improves the accuracy of the FNNs models but decreases the accuracy of the physics-based learning models. We suspect that the inappropriate polynomial fitting alters the properties of the data, weakening or changing the objective physical laws of the system.
- Bad data comes from sensors' measurement error and cumulative error. The control signals are periodic. In all data plots, the angular changes in the y-axis and z-axis are periodic, but the angular changes in the x-direction are entirely random and irregular.

B. Open challenges

Based on the above findings, we can say that the preliminary results are quite promising, but there is still much room for improvement.

As for neural networks, future work can start from the following aspects:

- Solve the slow training problem of Lagrangian NNs by using the computation or estimation algorithm for the Hessian in [38].
- Obtain the ideal solution of Lagrangian NNs by providing additional labels or learning the actuator model separately.
- Design algorithms to estimate or approximate the momentum so that Hamiltonian NNs do all the learning.

In soft robot modeling, the main problem currently is accurately and efficiently capturing the robot's pose. Sensors for soft robotics are also a hot research topic. Improving soft robots' sampling efficiency and data quality is also a follow-up issue worthy of attention. With the advancement of sensor technology, the modeling of soft robots will also improve significantly.

VII. Conclusion

In this research, we mainly adopt physics-based neural networks to learn the dynamic models of soft robot manipulators from simulation to actual experiments. We improve and extend the current Lagrangian network model by adding linear damping and actuator models. We add a Runge-Kutta single-step integration method for loss calculation to overcome the lack of acceleration data. We have experimentally demonstrated the possibility and effectiveness of modeling soft robots using physics-based neural networks and provided suggestions for subsequent improvement.

VIII. Acknowledgments

The completion of this project should be attributed to Pablo's careful guidance, from literature research to experimental design. Thanks also to Cosimo for supporting my project and patiently helping me when I got stuck. Sincere thanks go to Bastian Deutschmann, the inventor of the experimental platform, and his team. In addition, thanks to Francesco and Pablo for building the experimental platform.

References

[1] M. Wehner, R. L. Truby, D. J. Fitzgerald, B. Mosadegh, G. M. Whitesides, J. A. Lewis, and R. J. Wood, "An integrated design and fabrication strategy for entirely soft, autonomous robots," *Nature*, vol. 536, no. 7617, pp. 451–455, 2016.

[2] Wilfried Sire and Guilhem Velvé Casquillas, "Soft robot: A review," https://www.elveflow.com/microfluidic-reviews/general-microfluidics/soft-robot/#_ftnref18, Last accessed on 2022-6-25.

[3] N. Ratliff, F. Meier, D. Kappler, and S. Schaal, "Doomed: Direct online optimization of modeling errors in dynamics," *Big data*, vol. 4, no. 4, pp. 253–268, 2016.

[4] A. Kazemipour, O. Fischer, Y. Toshimitsu, K. W. Wong, and R. K. Katzschmann, "Adaptive dynamic sliding mode control of soft continuum manipulators," *arXiv preprint arXiv:2109.11388*, 2021.

[5] T. George Thuruthel, E. Falotico, L. Beccai, and F. Iida, "Machine learning techniques for soft robots," *Frontiers in Robotics and AI*, vol. 8, p. 205, 2021.

[6] C. Della Santina, R. K. Katzschmann, A. Biechi, and D. Rus, "Dynamic control of soft robots interacting with the environment," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 2018, pp. 46–53.

[7] A. Kazemipour, O. Fischer, Y. Toshimitsu, K. W. Wong, and R. K. Katzschmann, "Adaptive dynamic sliding mode control of soft continuum manipulators," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 3259–3265.

[8] C. Della Santina, C. Duriez, and D. Rus, "Model based control of soft robots: A survey of the state of the art and open challenges," *arXiv preprint arXiv:2110.01358*, 2021.

[9] G. Runge and A. Raatz, "A framework for the automated design and modelling of soft robotic systems," *CIRP Annals*, vol. 66, no. 1, pp. 9–12, 2017.

[10] M. Giorelli, F. Renda, M. Calisti, A. Arienti, G. Ferri, and C. Laschi, "Learning the inverse kinetics of an octopus-like manipulator in three-dimensional space," *Bioinspiration & biomimetics*, vol. 10, no. 3, p. 035006, 2015.

[11] A. Melingui, R. Merzouki, J. B. Mbede, C. Escande, B. Daachi, and N. Benoudjit, "Qualitative approach for inverse kinematic modeling of a compact bionic handling assistant trunk," in *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014, pp. 754–761.

[12] T. G. Thuruthel, E. Falotico, M. Cianchetti, and C. Laschi, "Learning global inverse kinematics solutions for a continuum robot," in *Symposium on robot design, dynamics and control*. Springer, 2016, pp. 47–54.

[13] A. Tariverdi, V. K. Venkiteswaran, M. Richter, O. J. Elle, J. Tørresen, K. Mathiassen, S. Misra, and Ø. G. Martinsen, "A recurrent neural-network-based real-time dynamic model for soft continuum manipulators," *Frontiers in Robotics and AI*, vol. 8, p. 45, 2021.

[14] M. T. Gillespie, C. M. Best, E. C. Townsend, D. Wingate, and M. D. Killpack, "Learning nonlinear dynamic models of soft robots for model predictive control with neural networks," in *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, 2018, pp. 39–45.

[15] P. Hyatt, D. Wingate, and M. D. Killpack, "Model-based control of soft actuators using learned non-linear discrete-time models," *Frontiers in Robotics and AI*, vol. 6, p. 22, 2019.

[16] A. P. Sabelhaus and C. Majidi, "Gaussian process dynamics models for soft robots with shape memory actuators," in *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2021, pp. 191–198.

[17] H. Wang, J. Chen, H. Y. Lau, and H. Ren, "Motion planning based on learning from demonstration for multiple-segment flexible soft robots actuated by electroactive polymers," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 391–398, 2016.

[18] G. Fang, X. Wang, K. Wang, K.-H. Lee, J. D. Ho, H.-C. Fu, D. K. C. Fu, and K.-W. Kwok, "Vision-based online learning kinematic control for soft robots using local gaussian process regression," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1194–1201, 2019.

[19] R. L. Truby, C. Della Santina, and D. Rus, "Distributed proprioception of 3d configuration in soft, sensorized robots via deep learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3299–3306, 2020.

[20] A. Fathi and A. Mozaffari, "Modeling a shape memory alloy actuator using an evolvable recursive black-box and hybrid heuristic algorithms inspired based on the annual migration of salmons in nature," *Applied Soft Computing*, vol. 14, pp. 229–251, 2014.

[21] B. B. Kang, D. Kim, H. Choi, U. Jeong, K. B. Kim, S. Jo, and K.-J. Cho, "Learning-based fingertip force estimation for soft wearable hand robot with tendon-sheath mechanism," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 946–953, 2020.

[22] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Learning dynamic models for open loop predictive control of soft robotic manipulators," *Bioinspiration & biomimetics*, vol. 12, no. 6, p. 066003, 2017.

[23] B. Kailkhura, B. Gallagher, S. Kim, A. Hiszpanski, and T. Han, "Reliable and explainable machine-learning methods for accelerated material discovery," *npj Computational Materials*, vol. 5, no. 1, pp. 1–9, 2019.

[24] A. Daw, A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided neural networks (pgnn): An application in lake temperature modeling," *arXiv preprint arXiv:1710.11431*, 2017.

[25] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.

[26] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," *arXiv preprint arXiv:1907.04490*, 2019.

[27] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," *arXiv preprint arXiv:2003.04630*, 2020.

[28] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[29] R. J. Webster III and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1661–1683, 2010.

[30] M. Khoshnam and R. V. Patel, "A pseudo-rigid-body 3r model for a steerable ablation catheter," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4427–4432.

[31] M. W. Hannan and I. D. Walker, "Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots," *Journal of robotic systems*, vol. 20, no. 2, pp. 45–63, 2003.

[32] N. Kuppuswamy and J.-P. Carbajal, "Learning a curvature dynamic model of an octopus-inspired soft robot arm using flexure sensors," *Procedia Computer Science*, vol. 7, pp. 294–296, 2011.

[33] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[34] B. Deutschmann, J. Reinecke, and A. Dietrich, "Open source tendon-driven continuum mechanism: A platform for research in soft robotics," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*, 2022, pp. 54–61.

[35] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[36] C. Della Santina, A. Bicchi, and D. Rus, "On an improved state parametrization for soft robots with piecewise constant curvature and its use in model based control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1001–1008, 2020.

[37] B. Deutschmann, "Tendondrivencontinuum," https://github.com/DLR-RM/TendonDrivenContinuum, 2022.

[38] H. Zhou, C. Ibrahim, W. X. Zheng, and W. Pan, "Sparse bayesian deep learning for dynamic system identification," *Automatica*, vol. 144, p. 110489, 2022.

APPENDIX

A. Analytical models of soft manipulators

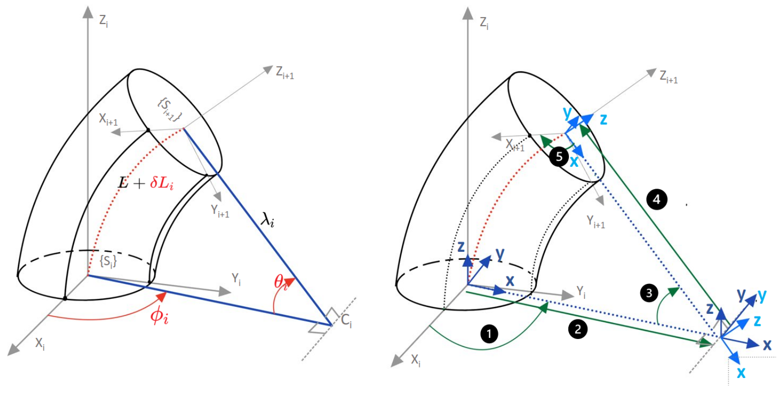


Fig. 18. PCC configuration definition and five steps D-H method

The PCC model defined three parameters for one segment, $\mathbf{q}_i = [\phi_i, \theta_i, \delta L_i]$. By using the D-H method shown in Fig.18, the transformation matrix can be calculated by three rotation matrices and two translation matrices:

$$\begin{aligned} T_{i-1}^{i}(\mathbf{q}_i) &= T_{rz}(\phi_i)T_{px}(\lambda_i)T_{ry}(\theta_i)T_{px}(-\lambda_i)T_{rz}(-\phi_i) \\ &= \begin{bmatrix} \Theta_{i-1}^{i} & \Psi_{i-1}^{i} \\ 0 & 1 \end{bmatrix}, \end{aligned} \tag{11}$$

where $\lambda_i = (L_{i0} + \delta L_i)/\theta_i$ is the radius of the curvature. Θ_{i-1}^{i} and Ψ_{i-1}^{i} are rotation matrix and translation vector respectively. $T_{rz} \in \mathbb{R}^{4\times4}$ and $T_{ry} \in \mathbb{R}^{4\times4}$ are the rotation homogeneous transformation matrices (HTM) about the z-axis and x-axis, and $T_{px} \in \mathbb{R}^{4\times4}$ is the translation HTM about y-axis, which are expressed as:

$$\begin{aligned} T_{rz}(\phi_i) &= \begin{bmatrix} cos\phi_i & -sin\phi_i & 0 & 0 \\ sin\phi_i & cos\phi_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\ T_{ry}(\theta_i) &= \begin{bmatrix} cos\theta_i & 0 & sin\theta_i & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta_i & 0 & cos\theta_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\ T_{px}(\lambda_i) &= \begin{bmatrix} 1 & 0 & 0 & \lambda_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{12}$$

The whole transformation matrix T_{i-1}^{i} is expressed by:

$$T_{i-1}^{i}(\mathbf{q}_i) = \begin{bmatrix} cos^2\phi_i(cos\theta_i - 1) + 1 & sin\phi_i cos\phi_i(cos\theta_i - 1) & sin\theta_i cos\phi_i & \lambda_i cos\phi_i(1 - cos\theta_i) \\ sin\phi_i cos\phi_i(cos\theta_i - 1) & sin^2\phi_i(cos\theta_i - 1) + 1 & sin\theta_i sin\phi_i & \lambda_i sin\phi_i(1 - cos\theta_i) \\ -sin\theta_i cos\phi_i & -sin\theta_i sin\phi_i & cos\theta_i & \lambda_i sin\theta_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{13}$$

After obtaining the one segment homogeneous transformation matrix, we can know that the forward HTM of the N-th segment relative to the base coordinator $\{S_0\}$ is:

$$^{N}_{0}T(\mathbf{q}^N) = \prod_{k=1}^{N}\{^{k}_{k-1}T(\mathbf{q}_k)T_{rz}(\frac{\pi}{a_{dof}})\} = \begin{bmatrix} \Theta_N(\mathbf{q}^N) & \Psi_N(\mathbf{q}^N) \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}, \tag{14}$$

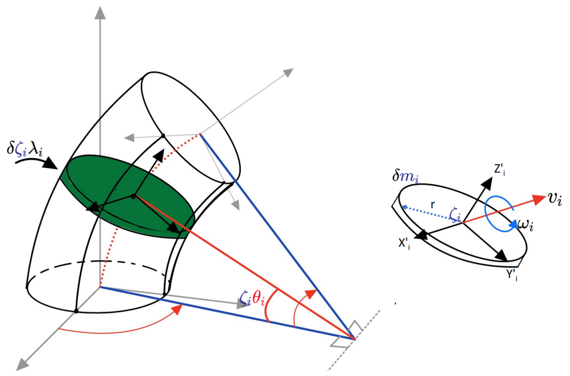


Fig. 19. Schematic diagram of slice coordinates and physical quantities

where a_{dof} is the number of actuator in each segment. The actuators' positions differ from segment to segment by a fixed angle π/a_{dof}. $\mathbf{q}^N$ is the total vector in the configuration space, which is expressed as $\mathbf{q}^N = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_N]$, where $\mathbf{q}_k = [\phi_k, \theta_k, \delta L_k]$.

If we assume that the actions of the latter segment do not affect former ones, we can use the rotation matrix Θ and translation vector Ψ to calculate the linear and angular velocity of m-segment's top coordinator relative to the base coordinator

$$v_m(\mathbf{q}^m) = J_m^{0v}(\mathbf{q}^m)\dot{\mathbf{q}}^m, \tag{15}$$

$$\omega_m(\mathbf{q}^m) = J_m^{0\omega}(\mathbf{q}^m)\dot{\mathbf{q}}^m, \tag{16}$$

where $\mathbf{q}^m$ is the first m segments' configuration space, and J_m^{0v} and $J_m^{0\omega}$ are linear and angular Jacobian in inertial coordinator. They are calculated by:

$$\begin{aligned} J_m^{0v} &= \frac{\partial \Psi_m}{\partial \mathbf{q}^m} \in \mathbb{R}^{3\times 3m}, \\ J_m^{0\omega} &= [\frac{\partial \Theta_m}{\partial \mathbf{q}^m}\Theta_m^T]^\vee \in \mathbb{R}^{3\times 3m}. \end{aligned} \tag{17}$$

We decompose one segment into countless thin plate in Fig.19 and assume that:

1) The arc shape (constant curvature) is maintained in each segment of the continuous manipulator.
2) The cross-section of each segment is symmetric about the neutral axis, with the center of gravity at the center of each sheet.
3) Segments are kinematically independent.
4) There is only a change in the length of the segment, and the radius and the position between each other remain unchanged.
5) Each segment has a constant mass, and a variable but uniform density.

The linear velocity v_i and angular velocity ω_i of the plate can be compute by Equation (15) and (16), but the i-segment's configuration θ_i should be replaced by $\zeta_i\theta_i$, where $\zeta_i \in [0, 1]$ The angular kinetic energy and translation kinetic energy of this thin plate are:

$$\delta E_{ki}^\omega = \frac{1}{2}\omega_i^T(m_i\delta\zeta_i\mathbf{I}_{xx}diag(1,1,2))\omega_i, \tag{18}$$

$$\delta E_{ki}^v = \frac{1}{2}v_i^T(m_i\delta\zeta_i\mathbf{I}_3)v_i. \tag{19}$$

Then the total kinetic energy of this i-segment is

$$\begin{aligned} K_i &= E_{ki}^\omega(\mathbf{q}^i) + E_{ki}^v(\mathbf{q}^i) \\ &= \int_{\zeta_i}(\dot{\mathbf{q}}^T J_i^{\omega T}(m_i\delta\zeta_i\mathbf{I}_{xx}diag(1,1,2))J_i^\omega\dot{\mathbf{q}}) + \int_{\zeta_i}(\dot{\mathbf{q}}^T J_i^{vT}(m_i\delta\zeta_i\mathbf{I}_3)J_i^v\dot{\mathbf{q}}). \end{aligned} \tag{20}$$

The potential energy of this thin plate is defined as

$$\delta P_i^g(\zeta_i, \mathbf{q}^i) = (m_i \delta\zeta)\Psi_i^T(\zeta\theta_i)\mathbf{g}, \tag{21}$$

where $\mathbf{g} = [0, 0, g]^T$ is the gravity vector of $\{S_0\}$, and $\Psi_i^T(\zeta\theta_i)$ is the translation vector which θ_i is replaced by $\zeta\theta_i$. The total gravitational energy of this i-segment is then determined as

$$P_i^g(\mathbf{q}^i) = \int_{\zeta_i} \delta P_i^g = m_i(\int_{\zeta_i} \Psi_i^T)g. \tag{22}$$

The elastic energy of any continuum section is due to the axial elastic deformation of variable length actuators (e.g.PMAs) during operation. The elastic energy of the i-th continuum segment is given by

$$P_i^e(\mathbf{q}^i) = \frac{1}{2}\mathbf{q}^{iT}\mathbf{K_i^e}\mathbf{q}^i, \tag{23}$$

where $\mathbf{K_i^e} = Diag\{K_{i1}, K_{i2}, K_{i3}\}$ and K_{ij} is the elastic stiffness coefficients of actuators. Then, the total potentia energy is calculated as

$$V_i = P_i^g(\mathbf{q}^i) + P_i^e(\mathbf{q}^i). \tag{24}$$

Afterwards, we can use Lagrangian mechanics to deduce it equations of motion.

B. Simulations

1. Task 1: mass-spring-damper system In this task, we use the Lagrangian NNs and FNNs to learned an unactuated spring damping system, shown in Fig. 20.

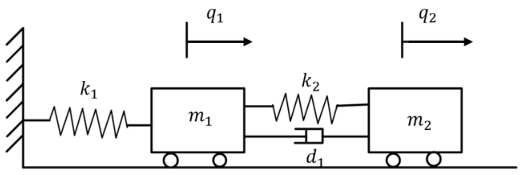


Fig. 20. Unactuated mass-spring-damper system example

The dynamics are given by:

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}\begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 + k_2 \end{bmatrix}\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} + \begin{bmatrix} b & -b \\ -b & b \end{bmatrix}\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = 0, \tag{25}$$

where m_1 and m_2 are 1.0 and 1.8, k_1 and k_2 are 2.1 and 4.6, b is 0.3. In this example, we compare the prediction ability of a naive black-box and our physics-based learning model. The training and testing loss of the black-box model and the physics-based learning model is shown in Figs.21 and 22.

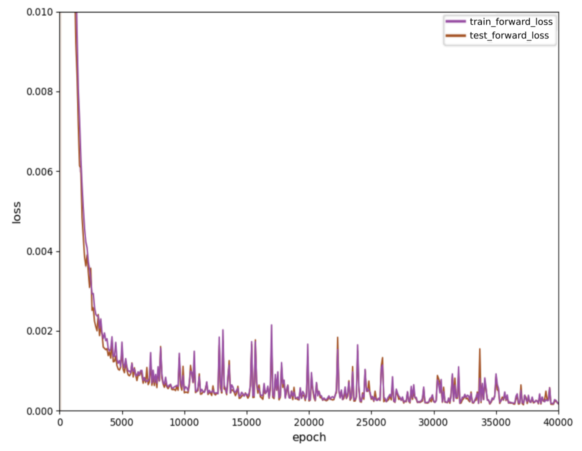


Fig. 21. Naive black box model training curve

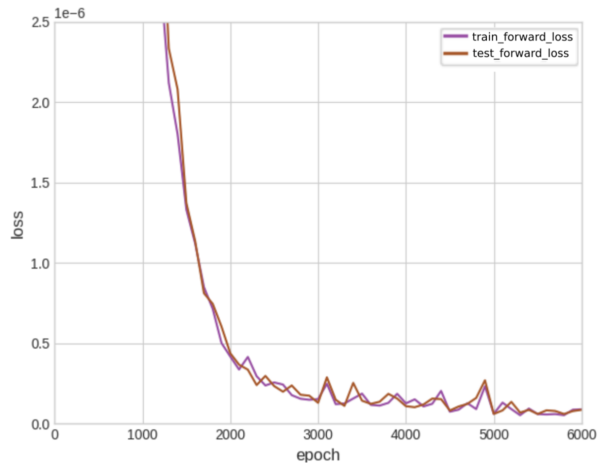


Fig. 22. Physics-based learning model training curve

A random initial states is provided to the two models, and the prediction results and ground truth are shown in Fig.23.

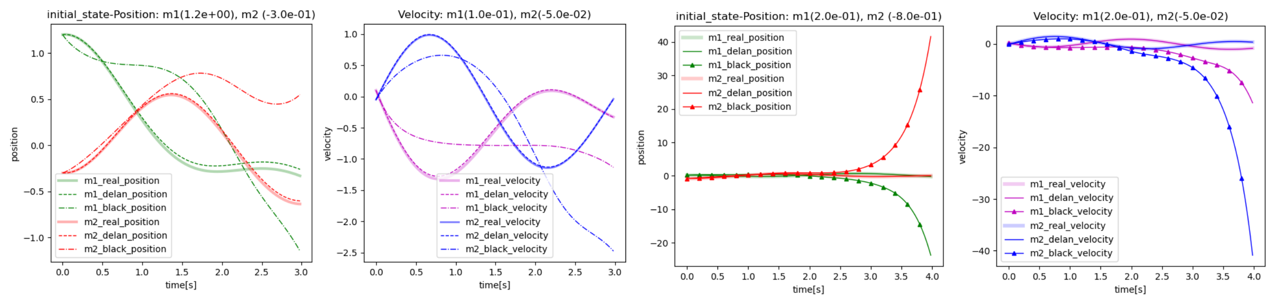


Fig. 23. Task 1: Black-box learning model and physics-based learning model prediction results comparison

Furthermore, physics-based learning models provide us with more information about the system. In this task, although the system is linear, we train it as a non-linear system. The ground truth and learned matrices are provided in Table V.

TABLE V
TASK 1: MATHEMATICAL MODEL MATRICES VS. PHYSICS-BASED LEARNING MODEL MATRICES

	Ground Truth	$q_1 = \begin{bmatrix}0.5 & 0.5\end{bmatrix}$	$q_2 = \begin{bmatrix}0.0 & 0.0\end{bmatrix}$	$q_3 = \begin{bmatrix}0.0 & -0.8\end{bmatrix}$	*Estimated* α
M	$\begin{bmatrix}1.0 & 0\\ 0 & 1.8\end{bmatrix}$	$\begin{bmatrix}0.301 & -0.002\\ -0.002 & 0.543\end{bmatrix}$	$\begin{bmatrix}0.299 & -0.002\\ -0.002 & 0.539\end{bmatrix}$	$\begin{bmatrix}0.299 & -0.0029\\ -0.0029 & 0.535\end{bmatrix}$	0.302
G	$\begin{bmatrix}2.1q_1 - 2.1q_2\\ -2.1q_1 + 6.7q_2\end{bmatrix}$	$\begin{bmatrix}-0.004\\ 0.692\end{bmatrix}$	$\begin{bmatrix}-0.005\\ 0.005\end{bmatrix}$	$\begin{bmatrix}0.506\\ -1.596\end{bmatrix}$	
D	$\begin{bmatrix}0.3 & -0.3\\ -0.3 & 0.3\end{bmatrix}$	$\begin{bmatrix}0.0905 & -0.093\\ -0.093 & 0.095\end{bmatrix}$	$\begin{bmatrix}0.089 & -0.091\\ -0.091 & 0.0919\end{bmatrix}$	$\begin{bmatrix}0.093 & -0.091\\ -0.091 & 0.089\end{bmatrix}$	

In Table V, if the ground truth is timed by the estimated α, we will get a matrix that is really close to the learned one. In addition, since the physical parameters themselves are learned, the step size of the simulation is not limited to be consistent with the learned data. Fig.24 shows the prediction with a time step of 0.02s, and Fig.25 shows the prediction with a time step of 0.1s.

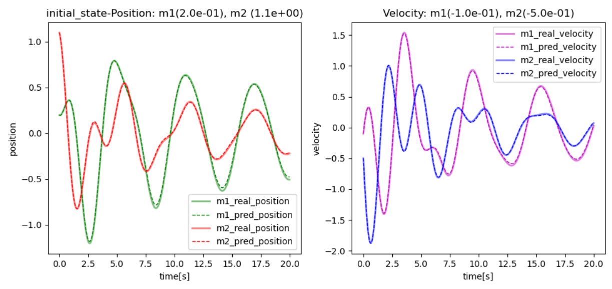


Fig. 24. Task 1: Physics-based learning model prediction results with time-step size (0.02s) as the training data

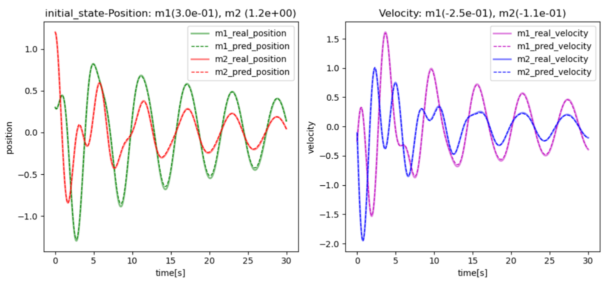


Fig. 25. Task 1: Physics-based learning model prediction results with time-step size (0.1s)

2. Task 2: double pendulum In this task, we use the physics-based learning network to learn a simple nonlinear systems. A mathematical model of the simplified double pendulum is established as shown in the Fig.26.

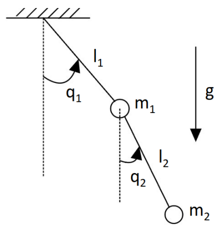


Fig. 26. Simplified double pendulum system example

underlying dynamics of this simplified double pendulum is given by

$$\begin{bmatrix} (m_1+m_2)l_1^2 & m_2l_1l_2cos(q_1-q_2) \\ m_2l_1l_2cos(q_1-q_2) & m_2l_2^2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -2m_2l_1l_2sin(q_1-q_2)\dot{q}_2 & m_2l_1l_2sin(q_1-q_2)\dot{q}_2 \\ -m_2l_1l_2sin(q_1-q_2)\dot{q}_1 & 2m_2l_2sin(q_1-q_2)\dot{q}_1 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \begin{bmatrix} -(m_1+m_2)gl_1sinq_1 \\ -m_2gl_2sinq_2 \end{bmatrix} = \mathbf{0}, \tag{26}$$

where, in our case, m_1 and m_2 is 1.0; l_1 and l_2 is 1.0; and g is 9.8. Both models are trained on the same dataset with 5000 samples. The black-box model is trained for 30000 epochs, while the physics-based learning model is trained for only 8200 epochs. The loss curves training black box model and physics-based learning model of task two is shown in Figs.27 and 28.

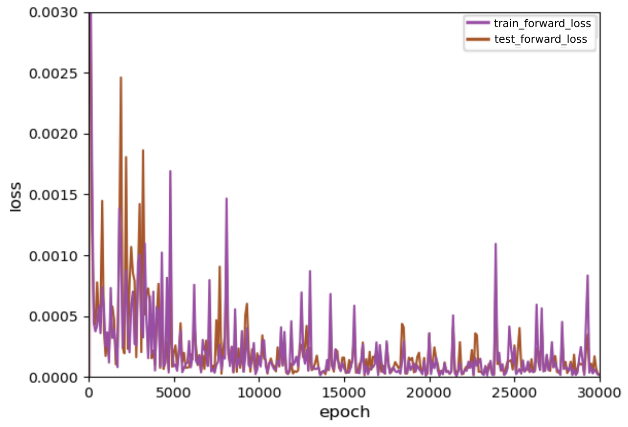


Fig. 27. Black-box model training curve

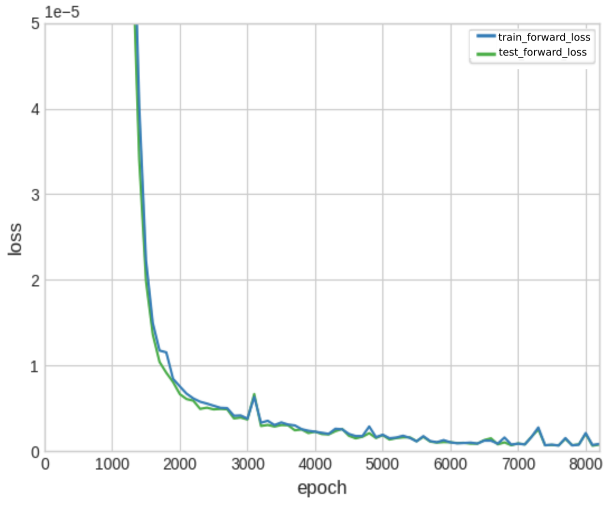


Fig. 28. Physics-based learning model training curve

The predictive ability of physics-based learning model and the traditional FNNs model for this system are compared in Fig.29.

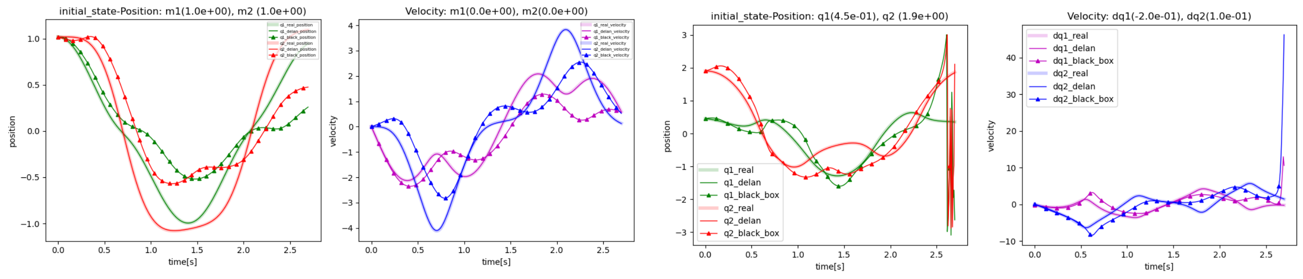


Fig. 29. Task 2 : Black-box learning model and physics-based learning model prediction results comparison

Here we give several random configuration states, and compute the mass matrices and potential forces for mathematical models and physics-based learning models of these states.

TABLE VI
TASK 2: MATHEMATICAL MODEL MATRICES VS. PHYSICS-BASED LEARNING MODEL MATRICES

state (q)	Mathematical model		Physics-based learning model			α
	M(q)	G(q)	M(q)	G(q)	D(q)	
$\begin{bmatrix}\pi/5\\ \pi/9\end{bmatrix}$	$\begin{bmatrix}2. & 0.961\\ 0.961 & 1.\end{bmatrix}$	$\begin{bmatrix}11.521\\ 3.352\end{bmatrix}$	$\begin{bmatrix}0.644 & 0.309\\ 0.309 & 0.322\end{bmatrix}$	$\begin{bmatrix}3.711\\ 1.085\end{bmatrix}$	$\begin{bmatrix}0.001 & 0.001\\ 0.001 & 0.002\end{bmatrix}$	0.323
$\begin{bmatrix}2.123\\ 0.01\end{bmatrix}$	$\begin{bmatrix}2. & -0.516\\ -0.516 & 1.\end{bmatrix}$	$\begin{bmatrix}16.687\\ 0.098\end{bmatrix}$	$\begin{bmatrix}0.672 & -0.188\\ -0.188 & 0.344\end{bmatrix}$	$\begin{bmatrix}5.845\\ -0.021\end{bmatrix}$	$\begin{bmatrix}0.001 & 0.001\\ 0.001 & 0.002\end{bmatrix}$	0.321
$\begin{bmatrix}-0.6\\ -2.3\end{bmatrix}$	$\begin{bmatrix}2. & -0.129\\ -0.129 & 1.\end{bmatrix}$	$\begin{bmatrix}-11.067\\ -7.308\end{bmatrix}$	$\begin{bmatrix}0.635 & -0.040\\ -0.040 & 0.326\end{bmatrix}$	$\begin{bmatrix}-3.395\\ -2.21\end{bmatrix}$	$\begin{bmatrix}0.001 & 0.001\\ 0.001 & 0.002\end{bmatrix}$	0.312
$\begin{bmatrix}1.9\\ -0.35\end{bmatrix}$	$\begin{bmatrix}2. & -0.628\\ -0.628 & 1.\end{bmatrix}$	$\begin{bmatrix}18.548\\ 3.360\end{bmatrix}$	$\begin{bmatrix}0.679 & -0.239\\ -0.239 & 0.357\end{bmatrix}$	$\begin{bmatrix}6.205\\ -0.999\end{bmatrix}$	$\begin{bmatrix}0.001 & 0.001\\ 0.001 & 0.002\end{bmatrix}$	0.345
$\begin{bmatrix}\pi/3.3\\ 0.28\end{bmatrix}$	$\begin{bmatrix}2. & 0.783\\ 0.783 & 1.\end{bmatrix}$	$\begin{bmatrix}15.966\\ 2.708\end{bmatrix}$	$\begin{bmatrix}0.645 & 0.252\\ 0.252 & 0.321\end{bmatrix}$	$\begin{bmatrix}5.130\\ 0.881\end{bmatrix}$	$\begin{bmatrix}0.001 & 0.001\\ 0.001 & 0.002\end{bmatrix}$	0.323

Fig.30 is the prediction results with a time step as 0.08s, and Fig.31 shows the prediction results with a time step as 0.15s.

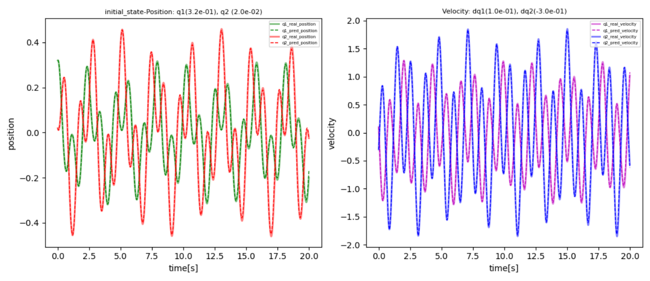


Fig. 30. Task 2: Physics-based learning model prediction results with time-step size (0.08s) as the training data

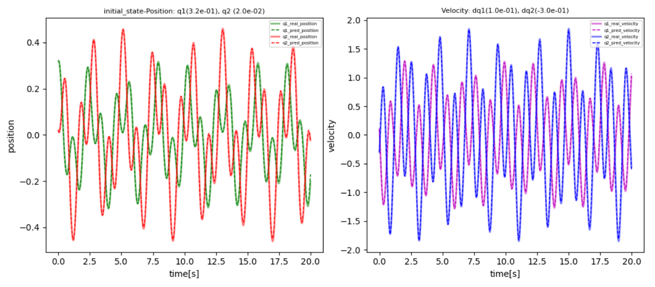


Fig. 31. Task 2: Physics-based learning model prediction results with time-step size (0.15s)

3. Task 3: one-segment planar soft manipulator The task is to validate a physics-based learning model based on Hamiltonian mechanics shown in Fig.7, and its implementation in a slightly more difficult system. On the basis of the PCC model, the cylinder manipulator is further simplified into a thin rod.

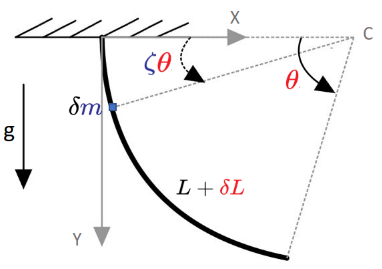


Fig. 32. Simplified one-segment planar soft manipulator system

Its configuration space is defined as $\mathbf{q} = \begin{bmatrix}\delta L & \theta\end{bmatrix}^T$. The translation vector is

$$\Psi = \begin{bmatrix}(L+\delta L)(1-cos(\zeta\theta))/\theta \\ (L+\delta L)sin(\zeta\theta)/\theta\end{bmatrix}. \tag{27}$$

$$H = \int_0^1 \frac{1}{2}\dot{\mathbf{q}}^T(\frac{\partial\Psi}{\partial\mathbf{q}})^T md\zeta\frac{\partial\Psi}{\partial\mathbf{q}}\dot{\mathbf{q}} + (\int_0^1 -gmd\zeta(L+\delta L)sin(\zeta\theta)/\theta + \int_0^1(\frac{1}{2}k_2\delta L^2 + \frac{1}{2}k_1\theta^2)) \tag{28}$$

In this example, we set the original length of the manipulator L to 1.0, the mass m to 1.0, the damping coefficient d to 0.4, and the spring coefficient k_1 and k_2 to 10.0. After obtaining a dataset of 11000 samples, we use 5000 of them to train our physics-based learning model for 18000 epochs, and use all 11000 to train a black-box learning model for 24000 epochs. The loss plots for training black box model and physics-based learning model are shown in Fig.34.

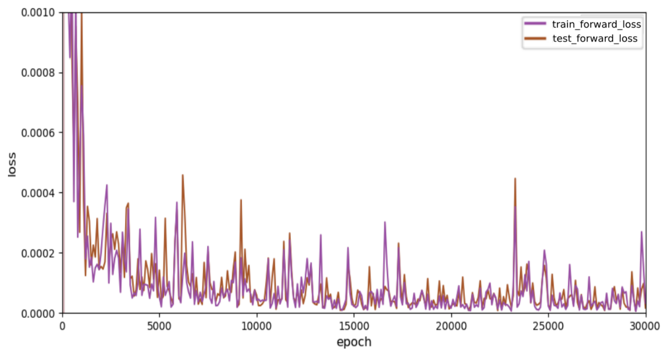


Fig. 33. Naive black box model training curve

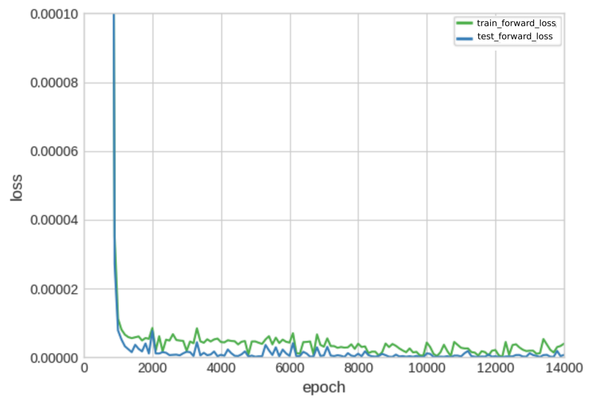


Fig. 34. Physics-based learning model training curve

The 2.7 second simulation results of black-box model and physics-based learning model can be seen in Fig.35.

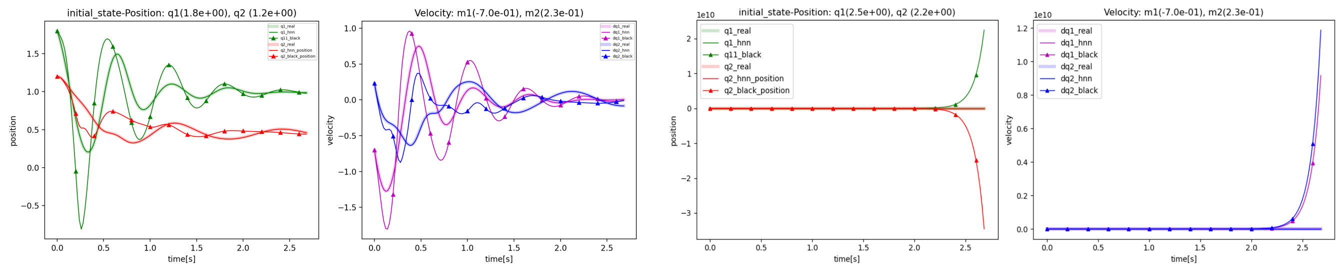


Fig. 35. Task 3 : Black-box learning model and physics-based learning model prediction results comparison

The comparison results of the mass matrix, potential force, damping matrix and input matrix of the mathematical model and learning model are shown in Table VII. Since this task uses a Hamiltonian NNs, the learned matrices can be directly compared with the theoretical value.

TABLE VII
TASK 3: MATHEMATICAL MODEL MATRICES VS. PHYSICS-BASED LEARNING MODEL MATRICES

q	mathmatical model				physics-based learning model			
	$M^{-1}(q)$	D(q)	G(q)	A(q)	$M^{-1}(q)$	D(q)	G(q)	A(q)
[1.5, 1]	$\begin{bmatrix}5.677 & 0.856\\0.856 & 3.487\end{bmatrix}$	$\begin{bmatrix}0.400 & 0.0\\0.0 & 0.400\end{bmatrix}$	$\begin{bmatrix}17.106\\5.948\end{bmatrix}$	$\begin{bmatrix}1.000\\0.0\end{bmatrix}$	$\begin{bmatrix}5.738 & 0.868\\0.868 & 3.489\end{bmatrix}$	$\begin{bmatrix}4.011e-01 & -1.830e-04\\-1.830e-04 & 4.000e-01\end{bmatrix}$	$\begin{bmatrix}17.138\\5.955\end{bmatrix}$	$\begin{bmatrix}1.000e+00\\3.410e-04\end{bmatrix}$
[1.2, 0.2]	$\begin{bmatrix}15.065 & 1.088\\1.088 & 3.303\end{bmatrix}$		$\begin{bmatrix}13.069\\-2.344\end{bmatrix}$		$\begin{bmatrix}15.102 & 1.100\\1.100 & 3.303\end{bmatrix}$	$\begin{bmatrix}0.401 & -0.001\\-0.001 & 0.400\end{bmatrix}$	$\begin{bmatrix}13.090\\-2.356\end{bmatrix}$	$\begin{bmatrix}1.002\\-0.001\end{bmatrix}$
[0.3, 0.3]	$\begin{bmatrix}11.895 & 0.232\\0.232 & 3.018\end{bmatrix}$		$\begin{bmatrix}3.317\\-1.868\end{bmatrix}$		$\begin{bmatrix}11.906 & 0.243\\0.243 & 3.021\end{bmatrix}$	$\begin{bmatrix}0.400 & 0.001\\0.001 & 0.400\end{bmatrix}$	$\begin{bmatrix}3.319\\-1.862\end{bmatrix}$	$\begin{bmatrix}1.001\\-4.537e-04\end{bmatrix}$
[0.8, 0.9]	$\begin{bmatrix}5.744 & 0.437\\0.437 & 3.131\end{bmatrix}$		$\begin{bmatrix}9.191\\4.351\end{bmatrix}$		$\begin{bmatrix}5.721 & 0.438\\0.438 & 3.127\end{bmatrix}$	$\begin{bmatrix}0.401 & 0.001\\0.001 & 0.401\end{bmatrix}$	$\begin{bmatrix}9.186\\4.358\end{bmatrix}$	$\begin{bmatrix}1.000\\-1.302e-03\end{bmatrix}$
[0.6, 0.2]	$\begin{bmatrix}14.174 & 0.511\\0.511 & 3.073\end{bmatrix}$		$\begin{bmatrix}6.575\\-2.760\end{bmatrix}$		$\begin{bmatrix}14.177 & 0.511\\0.511 & 3.076\end{bmatrix}$	$\begin{bmatrix}0.400 & 0.001\\0.001 & 0.400\end{bmatrix}$	$\begin{bmatrix}6.577\\-2.763\end{bmatrix}$	$\begin{bmatrix}1.001\\-1.078e-03\end{bmatrix}$
[0.5, 0.5]	$\begin{bmatrix}9.015 & 0.338\\0.338 & 3.050\end{bmatrix}$		$\begin{bmatrix}5.603\\0.196\end{bmatrix}$		$\begin{bmatrix}9.020 & 0.348\\0.348 & 3.055\end{bmatrix}$	$\begin{bmatrix}0.400 & 0.001\\0.001 & 0.400\end{bmatrix}$	$\begin{bmatrix}5.606\\0.207\end{bmatrix}$	$\begin{bmatrix}1.001\\2.314e-04\end{bmatrix}$
[0.1, 0.1]	$\begin{bmatrix}16.538 & 0.091\\0.091 & 3.002\end{bmatrix}$		$\begin{bmatrix}1.090\\-3.901\end{bmatrix}$		$\begin{bmatrix}16.574 & 0.085\\0.085 & 3.004\end{bmatrix}$	$\begin{bmatrix}0.401 & 0.000\\0.000 & 0.401\end{bmatrix}$	$\begin{bmatrix}1.096\\-3.903\end{bmatrix}$	$\begin{bmatrix}1.001\\2.186e-03\end{bmatrix}$

And the different time-step size prediction results are shown in Fig.37.

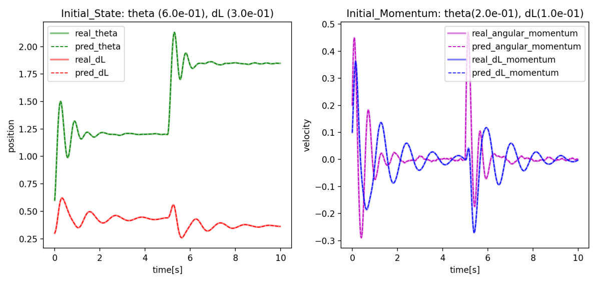


Fig. 36. Task 3: Physics-based learning model prediction results with time-step size (0.02s) as the training data

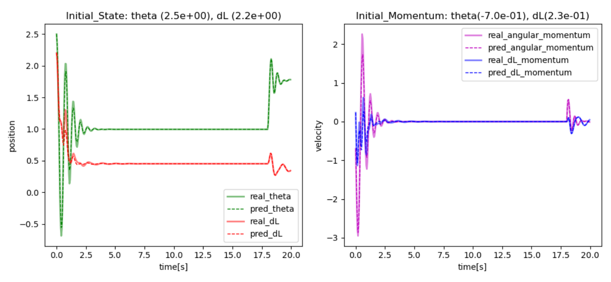


Fig. 37. Task 3: Physics-based learning model prediction results with time-step size (0.1s)

4. Task 4: one-segment spatial soft manipulator The loss plots for training black-box model and Lagrangian-based and Hamiltonian-based learning model are shown below:

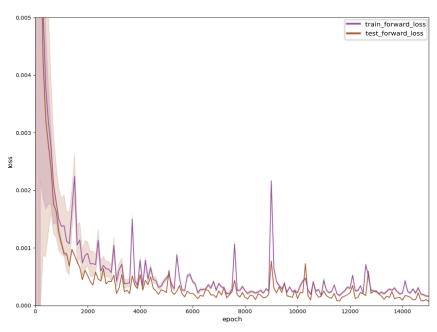


Fig. 38. Black-box model training curve

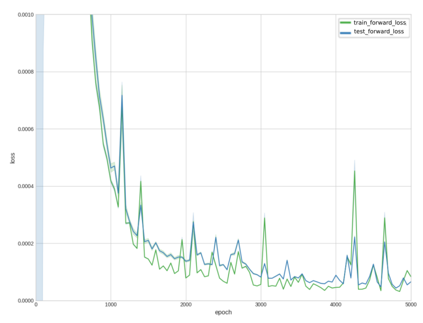


Fig. 39. Lagrangian NNs training curve

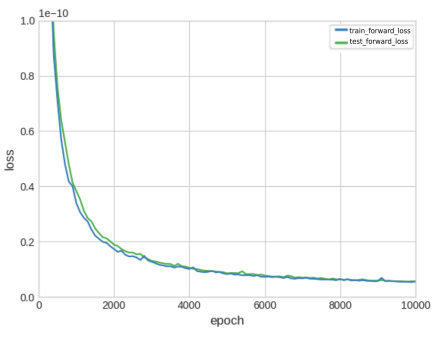


Fig. 40. Hamiltonian NNs training curve

More Lagrangian-based learning model prediction results are showing in Fig.41.

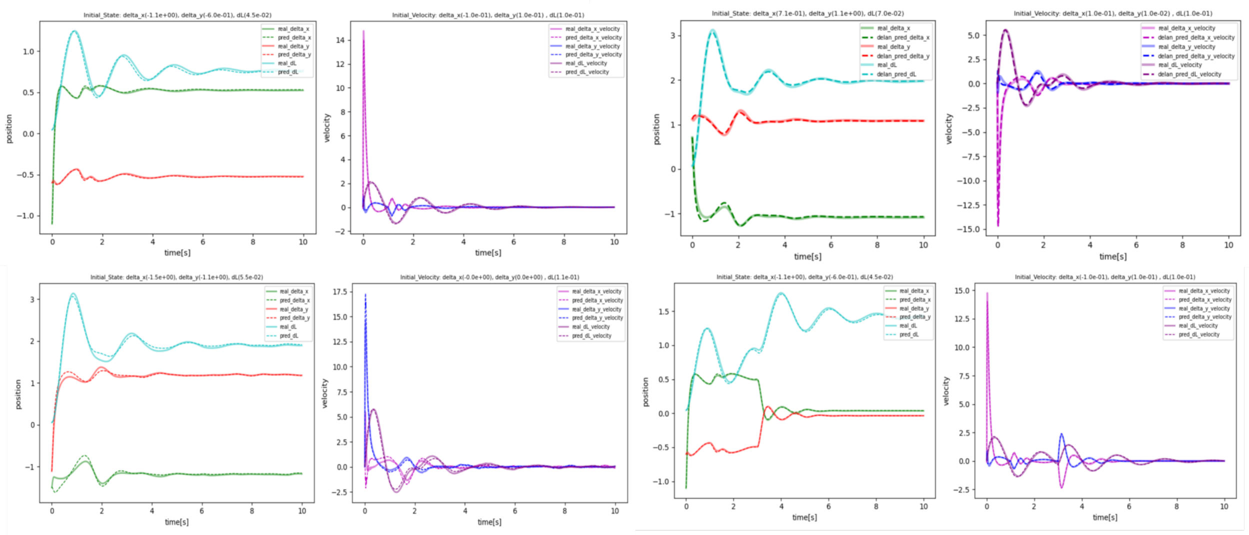


Fig. 41. Task 4: The prediction results of Lagrangian-based learning model

More learned matrices information are provided in Table VIII

TABLE VIII
TASK 4 APPENDIX: PHYSICS-BASED LEARNING MODEL MATRICES (LAGRANGIAN LEARNING MODEL)

	q	M(q)	D(q)	G(q)	A(q)
mathematical model	$\begin{bmatrix}1.0 & 0.2 & 0.15\end{bmatrix}$	$\begin{bmatrix}1.758e-03 & 2.691e-05 & -1.665e-03\\ 2.691e-05 & 1.629e-03 & -3.331e-04\\ -1.665e-03 & -3.331e-04 & 9.493e-02\end{bmatrix}$	$\begin{bmatrix}0.1 & 0 & 0\\ 0 & 0.1 & 0\\ 0 & 0 & 0.1\end{bmatrix}$	$\begin{bmatrix}1.080\\ 0.216\\ -1.148\end{bmatrix}$	$\begin{bmatrix}-0.032 & -0.994 & 0.055\\ 0.842 & -0.032 & -0.011\\ 0. & 0. & 0.835\end{bmatrix}$
	$\begin{bmatrix}0.1 & 0.1 & 0.0\end{bmatrix}$	$\begin{bmatrix}5.991e-04 & 4.756e-07 & -9.992e-05\\ 4.756e-07 & 5.991e-04 & -9.992e-05\\ -9.992e-05 & -9.992e-05 & 9.991e-02\end{bmatrix}$		$\begin{bmatrix}0.105\\ 0.105\\ -1.469\end{bmatrix}$	$\begin{bmatrix}0.002 & -0.998 & 0.003\\ 0.998 & -0.002 & 0.003\\ 0. & 0. & 0.997\end{bmatrix}$
	$\begin{bmatrix}-0.5 & -0.1 & 0.1\end{bmatrix}$	$\begin{bmatrix}1.335e-03 & 5.252e-06 & 7.409e-04\\ 5.252e-06 & 1.310e-03 & 1.482e-04\\ 7.409e-04 & 1.482e-04 & 9.871e-02\end{bmatrix}$		$\begin{bmatrix}-0.536\\ -0.107\\ -1.340\end{bmatrix}$	$\begin{bmatrix}0.008 & -0.998 & -0.025\\ 0.959 & -0.008 & -0.005\\ 0. & 0. & 0.957\end{bmatrix}$
	$\begin{bmatrix}-1.5 & 1.0 & 0.25\end{bmatrix}$	$\begin{bmatrix}2.483e-3 & -2.812e-4 & 2.887e-3\\ -2.812e-4 & 2.248e-3 & -1.925e-3\\ 2.887e-3 & -1.925e-3 & 8.496e-2\end{bmatrix}$		$\begin{bmatrix}-1.633\\ 1.088\\ -0.864\end{bmatrix}$	$\begin{bmatrix}-0.212 & -0.858 & -0.0956\\ 0.6814 & 0.212 & 0.064\\ 0. & 0. & 0.540\end{bmatrix}$
physics-based learning model	$\begin{bmatrix}1.0 & 0.2 & 0.15\end{bmatrix}$	$\begin{bmatrix}4.082e-3 & 1.300e-3 & -0.027\\ 1.300e-3 & 5.915e-3 & -0.018\\ -0.027 & -0.018 & 0.580\end{bmatrix}$	$\begin{bmatrix}0.163 & -0.014 & 0.004\\ -0.014 & 0.330 & -0.006\\ 0.004 & -0.006 & 0.346\end{bmatrix}$	$\begin{bmatrix}2.008\\ 0.782\\ -5.233\end{bmatrix}$	$\begin{bmatrix}0.187 & -1.695 & -0.206\\ 3.004 & -0.268 & -0.112\\ -0.420 & 1.027 & 3.459\end{bmatrix}$
	$\begin{bmatrix}0.1 & 0.1 & 0.0\end{bmatrix}$	$\begin{bmatrix}2.164e-3 & 7.384e-4 & -2.700e-2\\ 7.384e-4 & 2.479e-3 & -1.658e-2\\ -2.700e-2 & -1.658e-2 & 7.623e-1\end{bmatrix}$	$\begin{bmatrix}0.179 & 0.011 & 0.011\\ 0.011 & 0.307 & 0.010\\ 0.011 & 0.010 & 0.338\end{bmatrix}$	$\begin{bmatrix}0.408\\ 0.474\\ -5.18\end{bmatrix}$	$\begin{bmatrix}0.185 & -1.835 & -0.162\\ 3.094 & -0.067 & -0.057\\ -0.675 & 0.842 & 3.123\end{bmatrix}$
	$\begin{bmatrix}-0.5 & -0.1 & 0.1\end{bmatrix}$	$\begin{bmatrix}3.365e-03 & 7.760e-04 & -2.565e-02\\ 7.760e-04 & 4.455e-03 & -1.683e-02\\ --2.565e-02 & -1.683e-02 & 6.403e-01\end{bmatrix}$	$\begin{bmatrix}0.184 & 0.007 & -0.002\\ 0.007 & 0.304 & 0.008\\ -0.002 & 0.008 & 0.339\end{bmatrix}$	$\begin{bmatrix}-0.646\\ -0.203\\ -5.167\end{bmatrix}$	$\begin{bmatrix}0.083 & -1.782 & --0.145\\ 3.047 & -0.062 & -0.159\\ -0.816 & 0.704 & 2.824\end{bmatrix}$
	$\begin{bmatrix}-1.5 & 1.0 & 0.25\end{bmatrix}$	$\begin{bmatrix}-0.212 & -0.858 & -0.096\\ 0.681 & 0.212 & 0.064\\ 0. & 0. & 0.540\end{bmatrix}$	$\begin{bmatrix}0.180 & 0.011 & 0.003\\ 0.011 & 0.261 & 0.001\\ 0.003 & 0.001 & 0.306\end{bmatrix}$	$\begin{bmatrix}-2.022\\ 3.091\\ -4.283\end{bmatrix}$	$\begin{bmatrix}0.048 & -1.551 & -0.19\\ 2.856 & -0.181 & 0.130\\ -1.349 & 0.182 & 2.535\end{bmatrix}$

More Hamiltonian-based learning model prediction results are in Fig.42.

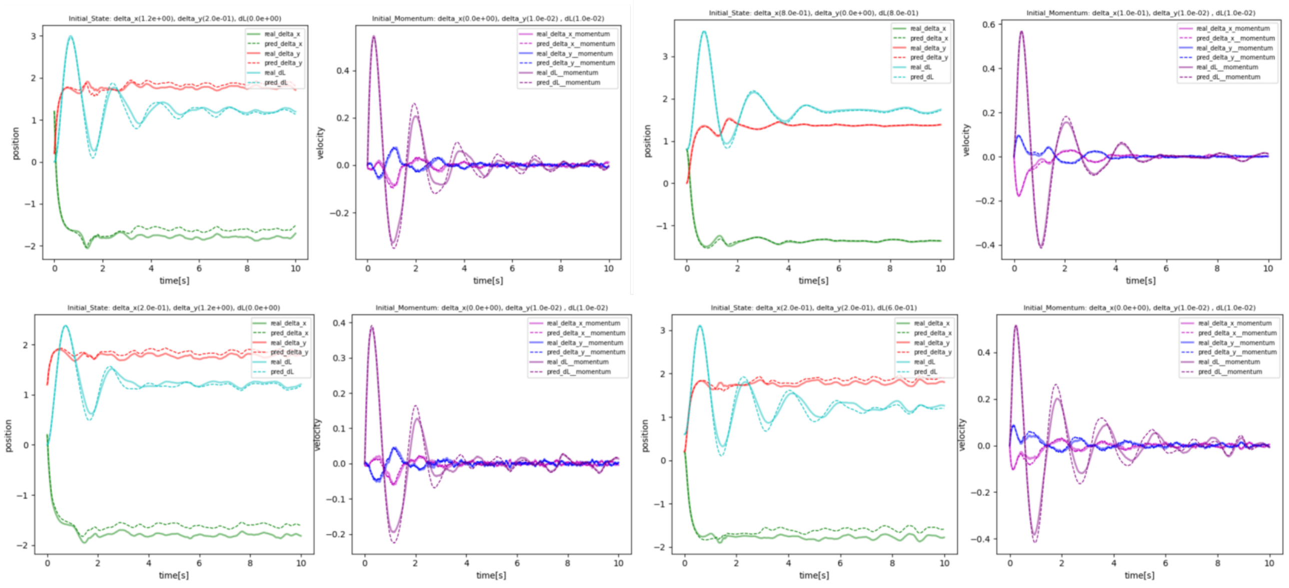


Fig. 42. Task 4: The prediction results of Hamiltonian-based learning model

More learned matrices information of Hamiltonian NNs are also provided in Table IX

TABLE IX
TASK 4 APPENDIX: PHYSICS-BASED LEARNING MODEL MATRICES (HAMILTONIAN LEARNING MODEL)

	q	$M-1(q)$	D(q)	G(q)	A(q)
mathematical model	$\begin{bmatrix}1.0 & 0.2 & 0.15\end{bmatrix}$	$\begin{bmatrix}578.506 & -7.488 & 10.123\\ -7.488 & 614.447 & 2.025\\ 10.123 & 2.025 & 10.719\end{bmatrix}$	$\begin{bmatrix}0.1 & 0 & 0\\ 0 & 0.1 & 0\\ 0 & 0 & 0.1\end{bmatrix}$	$\begin{bmatrix}1.080\\ 0.216\\ -1.198\end{bmatrix}$	$\begin{bmatrix}0.032 & -0.994 & 0.055\\ 0.842 & -0.032 & 0.011\\ 0. & 0. & 0.835\end{bmatrix}$
	$\begin{bmatrix}0.1 & 0.1 & 0.0\end{bmatrix}$	$\begin{bmatrix}1.669e3 & -1.0469 & 1.669\\ -1.0469 & 1.669e3 & 1.669\\ 1.669 & 1.669 & 1.001e1\end{bmatrix}$		$\begin{bmatrix}0.105\\ 0.105\\ -1.469\end{bmatrix}$	$\begin{bmatrix}0.002 & -0.998 & 0.003\\ 0.998 & -0.002 & 0.003\\ 0. & 0. & 0.997\end{bmatrix}$
	$\begin{bmatrix}-0.5 & -0.1 & 0.1\end{bmatrix}$	$\begin{bmatrix}752.028 & -2.377 & -5.641\\ -2.377 & 763.436 & -1.128\\ -5.641 & -1.128 & 10.174\end{bmatrix}$		$\begin{bmatrix}-0.5361578\\ -0.107\\ -1.340\end{bmatrix}$	$\begin{bmatrix}0.008 & -0.998 & -0.025\\ 0.959 & -0.008 & -0.005\\ 0. & 0. & 0.957\end{bmatrix}$
	$\begin{bmatrix}-1.5 & 1.0 & 0.25\end{bmatrix}$	$\begin{bmatrix}423.139 & 41.416 & -13.442\\ 41.416 & 457.653 & 8.961\\ -13.442 & 8.961 & 12.431\end{bmatrix}$		$\begin{bmatrix}-1.633\\ 1.088\\ -0.864\end{bmatrix}$	$\begin{bmatrix}-0.212 & -0.858 & -0.0956\\ 0.6814 & 0.212 & 0.064\\ 0. & 0. & 0.540\end{bmatrix}$
physics-based learning model	$\begin{bmatrix}1.0 & 0.2 & 0.15\end{bmatrix}$	$\begin{bmatrix}587.968 & 18.122 & 13.207\\ 18.122 & 606.240 & 3.171\\ 13.208 & 3.171 & 11.156\end{bmatrix}$	$\begin{bmatrix}1.012e-01 & 4.457e-03 & 8.152e-04\\ 4.457e-03 & 1.0390e-01 & -9.149e-05\\ 8.152e-04 & -9.149e-05 & 9.875e-02\end{bmatrix}$	$\begin{bmatrix}1.128\\ 0.258\\ -1.206\end{bmatrix}$	$\begin{bmatrix}0.003 & -0.938 & 0.041\\ 0.887 & -0.037 & -0.025\\ -0.006 & 0.007 & 0.836\end{bmatrix}$
	$\begin{bmatrix}0.1 & 0.1 & 0.0\end{bmatrix}$	$\begin{bmatrix}1.570e3 & -2.301e1 & 3.543\\ -2.301e1 & 1.554e3 & 1.531\\ 3.543 & 1.531 & 1.023e1\end{bmatrix}$	$\begin{bmatrix}0.102 & 0.003 & 8.515e-04\\ 0.003 & 0.104 & -0.001\\ 8.515e-04 & -0.001 & 0.100\end{bmatrix}$	$\begin{bmatrix}0.105\\ 0.116\\ -1.453\end{bmatrix}$	$\begin{bmatrix}2.653e-03 & -0.987 & -0.001\\ 0.997 & 0.003 & 0.009\\ -0.010 & 0.011 & 0.986\end{bmatrix}$
	$\begin{bmatrix}-0.5 & -0.1 & 0.1\end{bmatrix}$	$\begin{bmatrix}773.202 & 6.858 & -5.651\\ 6.858 & 750.952 & -2.085\\ -5.651 & -2.085 & 10.256\end{bmatrix}$	$\begin{bmatrix}1.015e-01 & 2.024e-05 & 1.091e-03\\ 2.024e-05 & 1.016e-01 & -5.076e-04\\ 1.091e-03 & -5.076e-04 & 1.011e-01\end{bmatrix}$	$\begin{bmatrix}-0.538\\ -0.111\\ -1.337\end{bmatrix}$	$\begin{bmatrix}-0.002 & -0.9989 & -0.035\\ 0.964 & 0.009 & 0.007\\ -0.017 & 0.014 & 0.977\end{bmatrix}$
	$\begin{bmatrix}-1.5 & 1.0 & 0.25\end{bmatrix}$	$\begin{bmatrix}420.579 & 18.688 & -13.034\\ 18.688 & 405.393 & 8.132\\ -13.034 & 8.132 & 12.472\end{bmatrix}$	$\begin{bmatrix}0.102 & -0.004 & 0.001\\ -0.004 & 0.103 & -0.004\\ 0.001 & -0.004 & 0.103\end{bmatrix}$	$\begin{bmatrix}-1.548\\ 1.113\\ -0.883\end{bmatrix}$	$\begin{bmatrix}-0.260 & -0.758 & -0.112\\ 0.747 & 0.150 & 0.071\\ -0.008 & -0.005 & 0.577\end{bmatrix}$

5. Task 5: two-segment spatial soft manipulator The loss plots for fixed time-step data, 50Hz resampled data, 100Hz resampled data and 1000Hz resampled data training loss are shown in the followings:

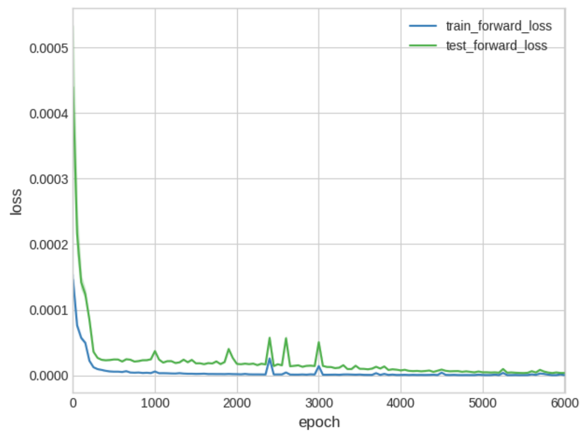


Fig. 43. Fixed time-step data training curve

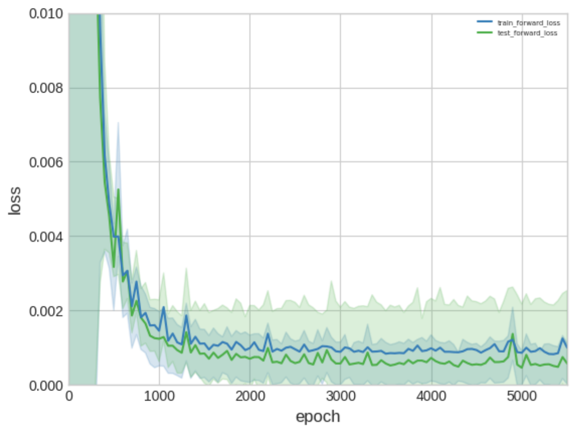


Fig. 44. 50Hz resampled data training curve

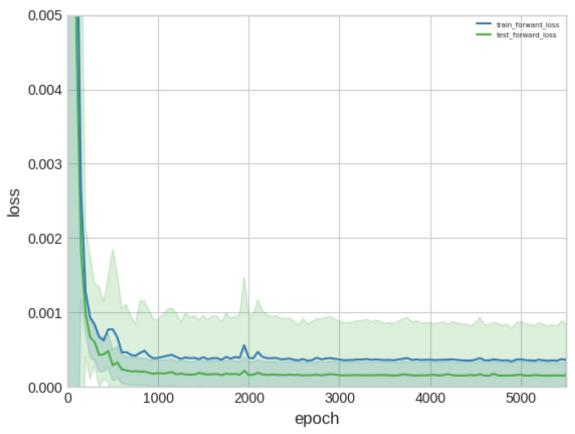


Fig. 45. 100Hz resampled data training curve

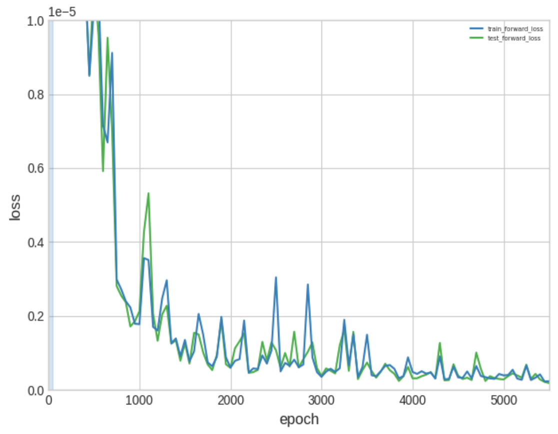


Fig. 46. 1000Hz resampled data training curve

The prediction ability of the fixed step-size learned model is proved Fig.47.

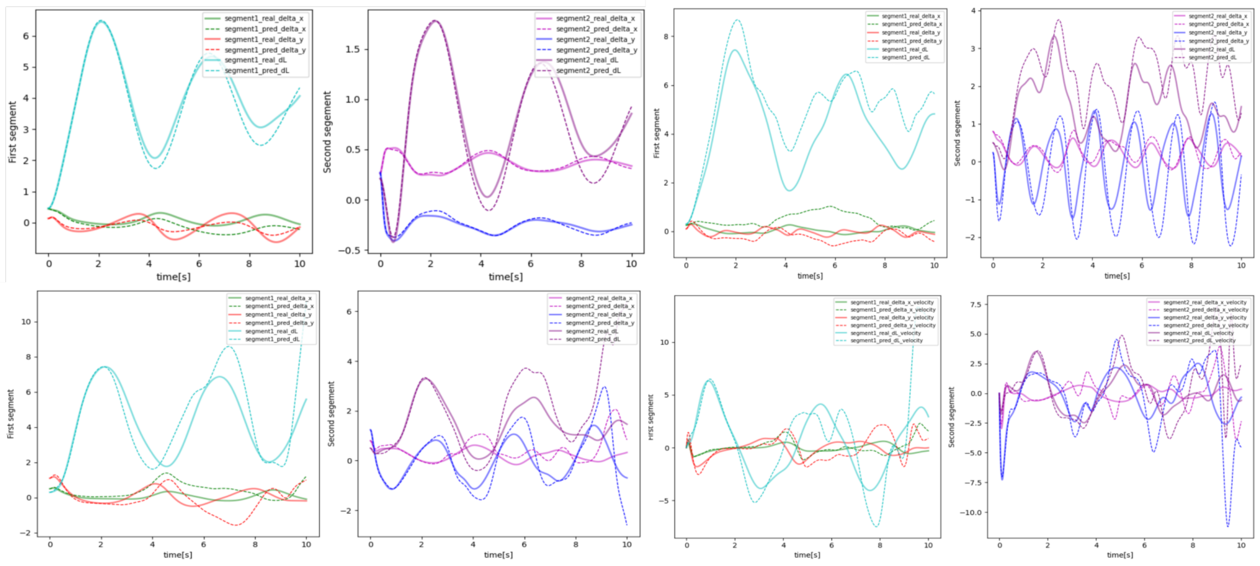


Fig. 47. Task 5: Prediction results trained on the fixed time-step size (0.0002s) data

The prediction ability of the 50Hz learned model is proved by Fig.48

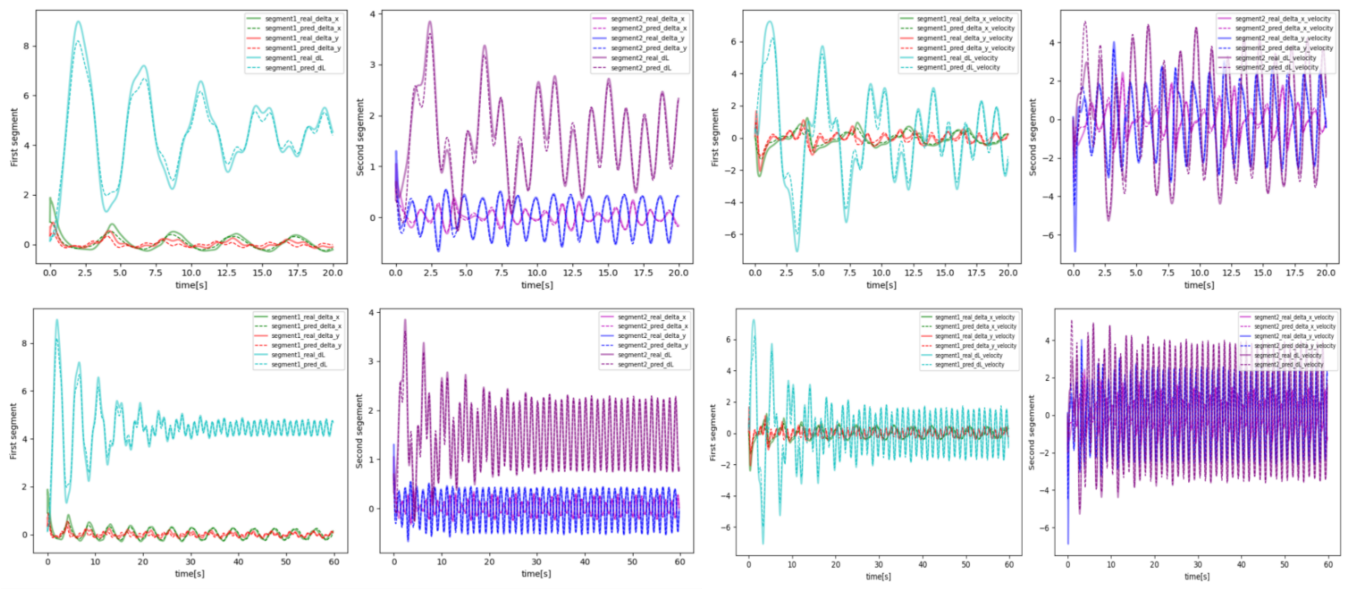


Fig. 48. Task 5: Prediction results trained on the 50Hz resampled data

The prediction ability of the 100Hz learned model is proved by Fig.49:

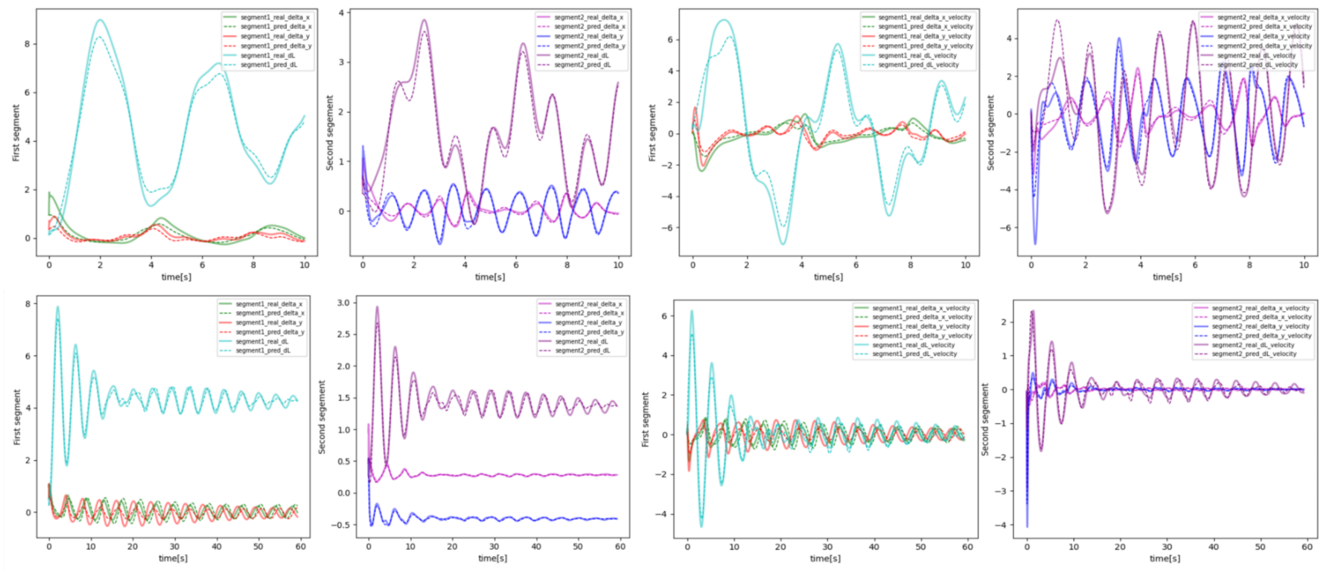


Fig. 49. Task 5: Prediction results trained on the 100Hz resampled data

The prediction ability of the 1000Hz learned model is proved by Fig.50:

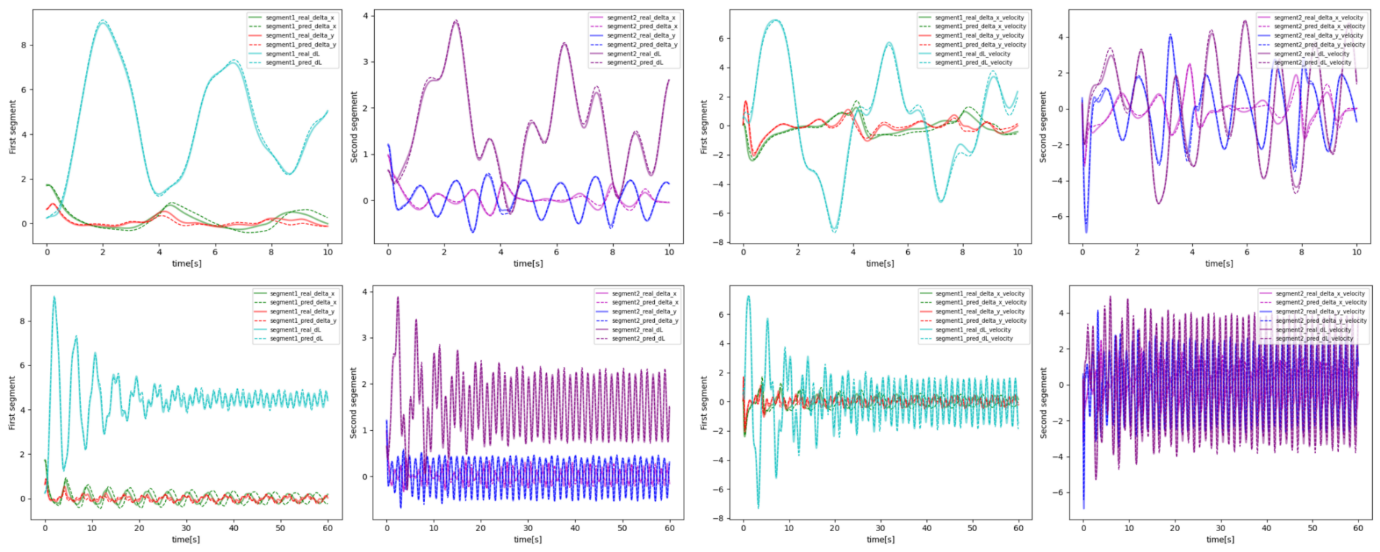


Fig. 50. Task 5: Prediction results trained on the 1000Hz resampled data

Experiment

Here we provide a few examples of unprocessed data in Fig.51.

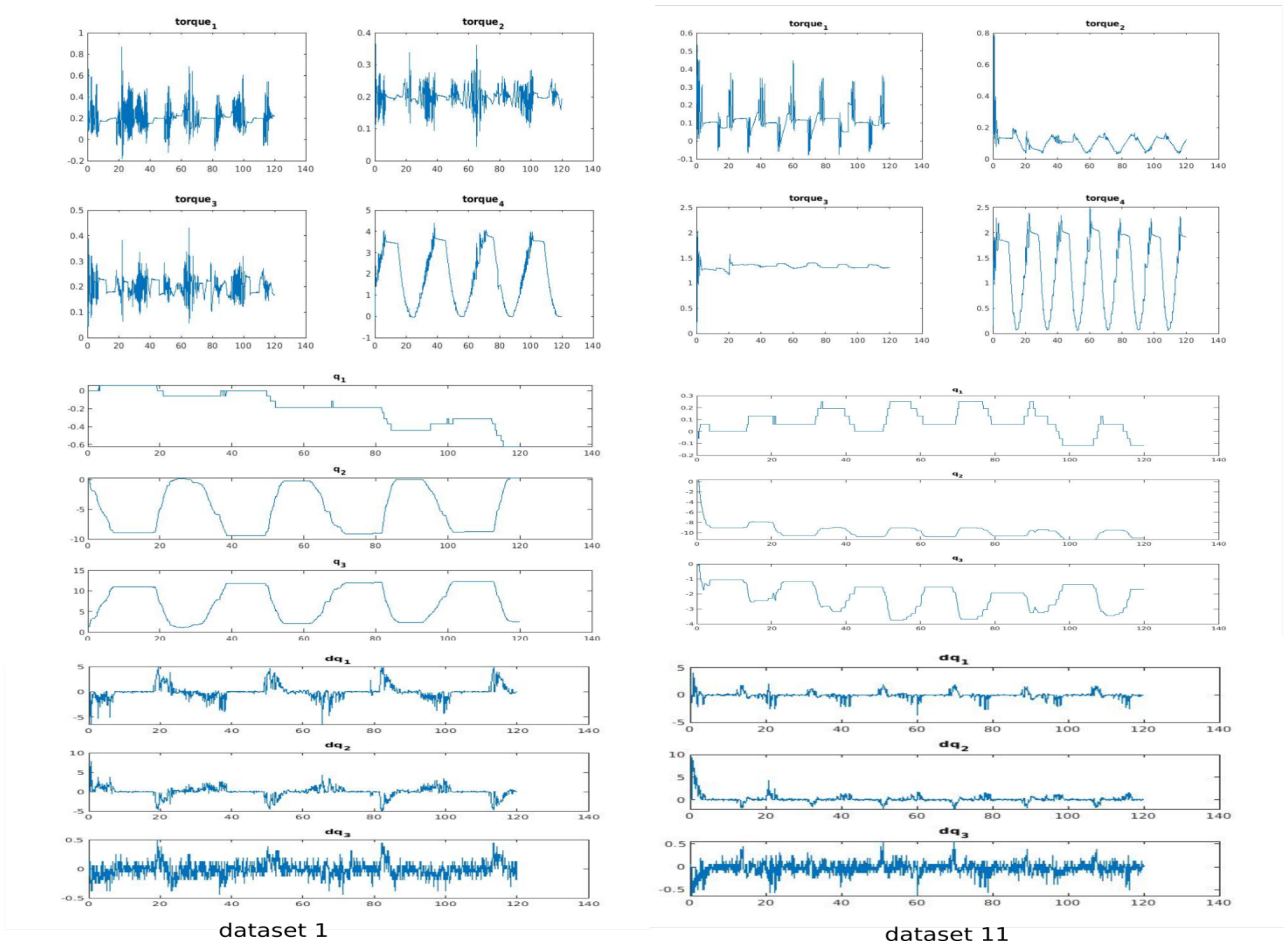


Fig. 51. Examples of raw data collected by the force sensors and IMU

1) smoothing data experiment results: The preprocessing performance is shown in Fig.52.

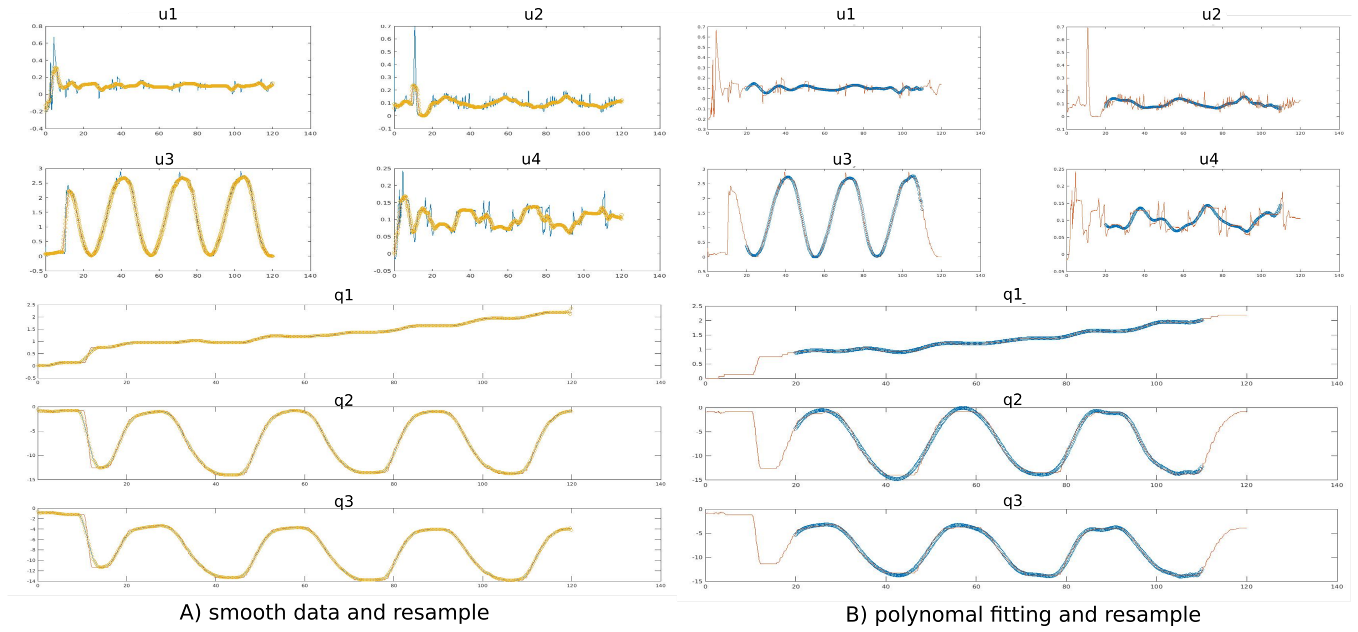


Fig. 52. Process data method: A) shows the 50 windows length smooth data method results; B) shows the polynomial fitting method results

Compared with the data processed by polynomial fitting and the simulation results, the smoothing data model has a slow convergence speed under this algorithm, and the convergence range is also small. The training curves of the smoothing data models are in Figs.53 and 54.

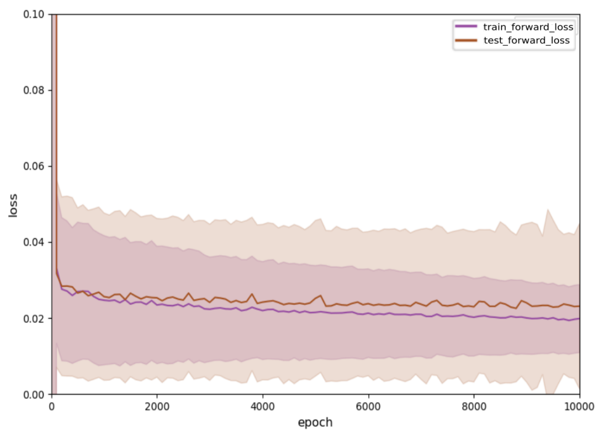


Fig. 53. Naive black box model training curve

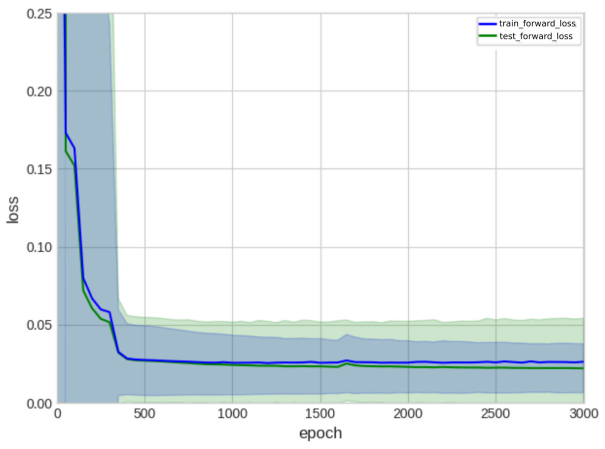


Fig. 54. Physics-based learning model training curve

More no updating prediction results of the physics-based learning model are shown in Fig.55.

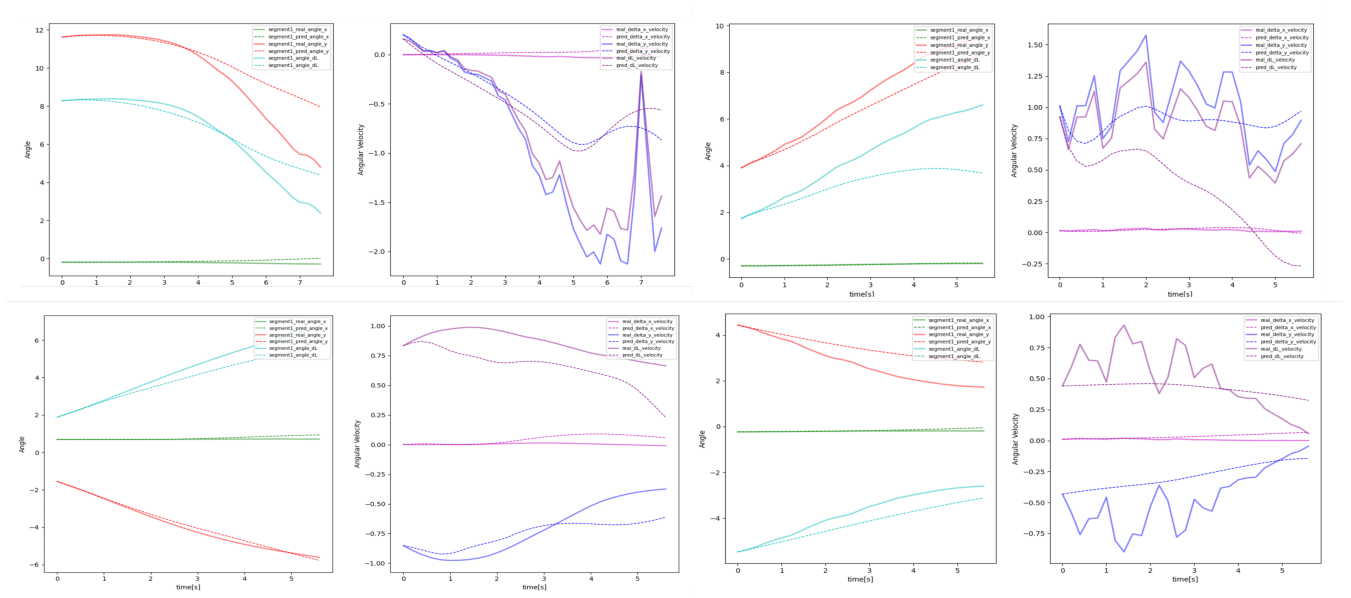


Fig. 55. 6 seconds prediction results of the physics-based learning model without resetting

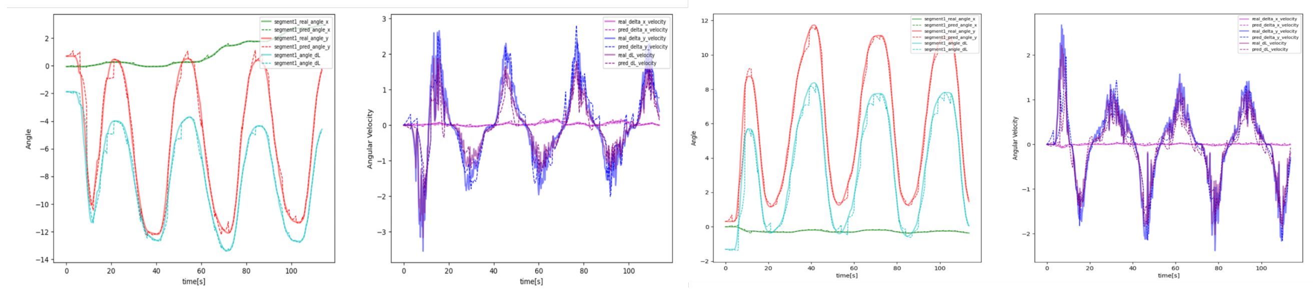


Fig. 56. Prediction results of the physics-based learning model with 4 seconds resetting

Besides, we provided some comparison results with the simple FNNs in Fig.57.

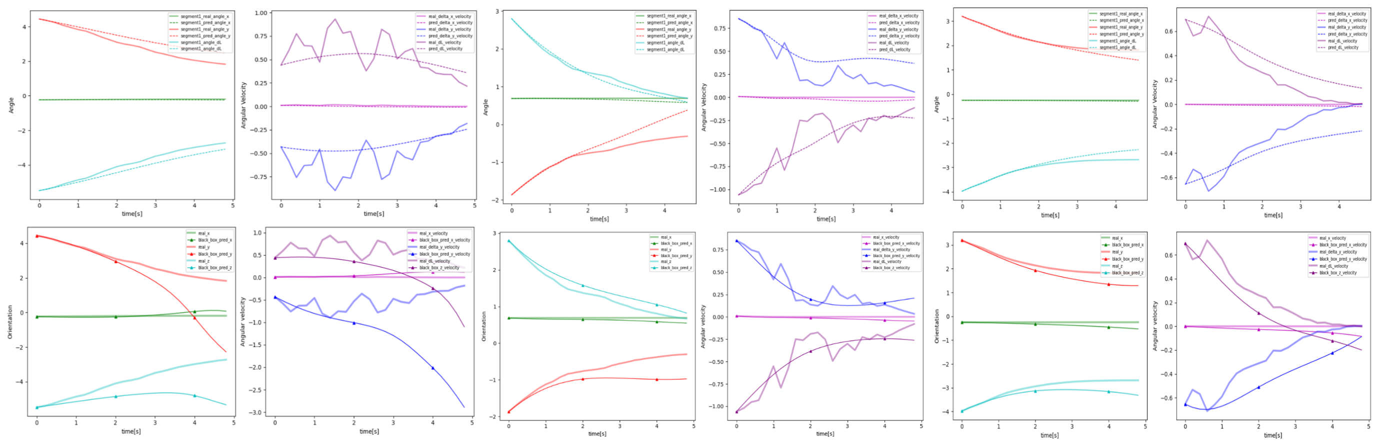


Fig. 57. Black-box model (△) vs physics-based learning model(- -) predictions with the same initial state as the smoothing data

2) polynomial fitting data experiment results: The training curves of the fitting data model are shown in Figs.58 and 59

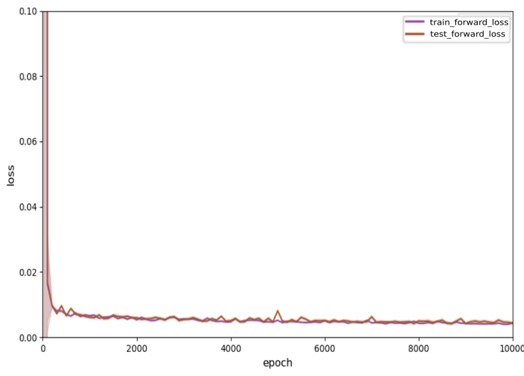


Fig. 58. Naive black box model training curve

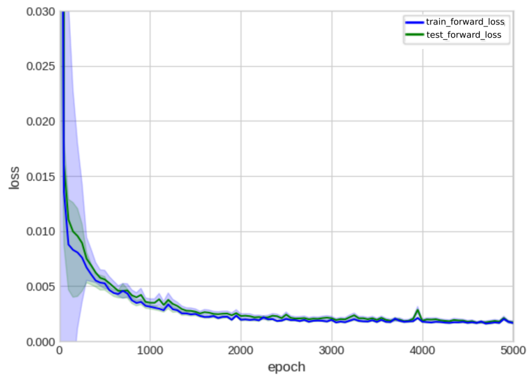


Fig. 59. Physics-based learning model training curve

More testing results are provided in Figs.60, 61 and 62.

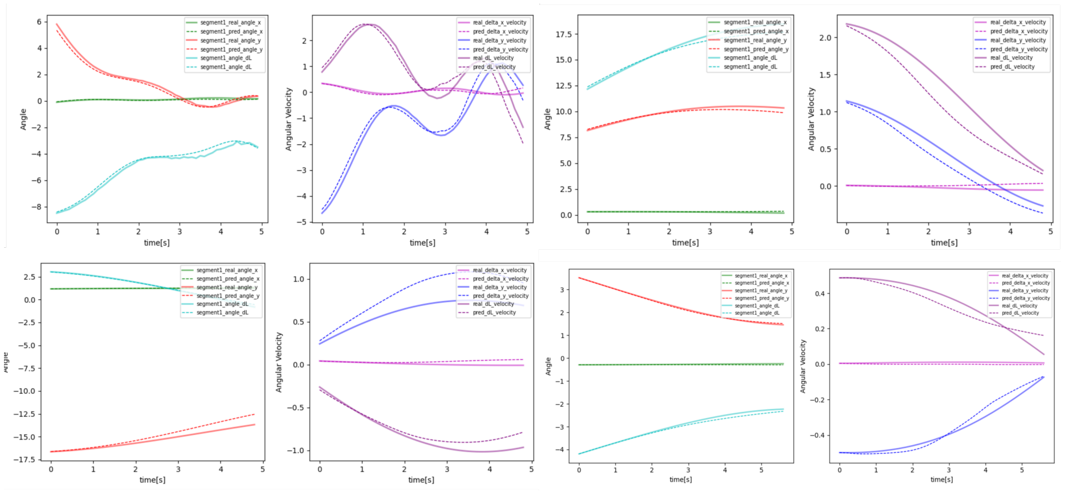


Fig. 60. 5 seconds prediction results of the physics-based learning model without resetting

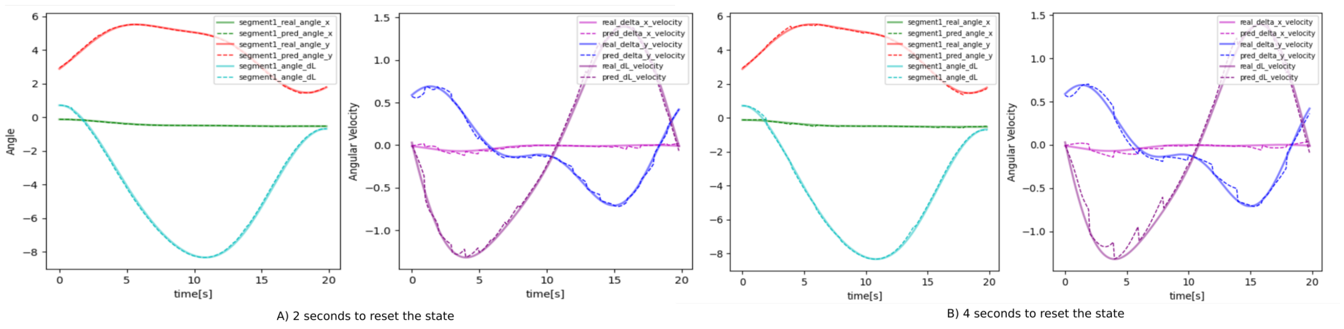


Fig. 61. 20 seconds prediction results of the physics-based learning model with A) 10-steps update state and B) 20-steps update state

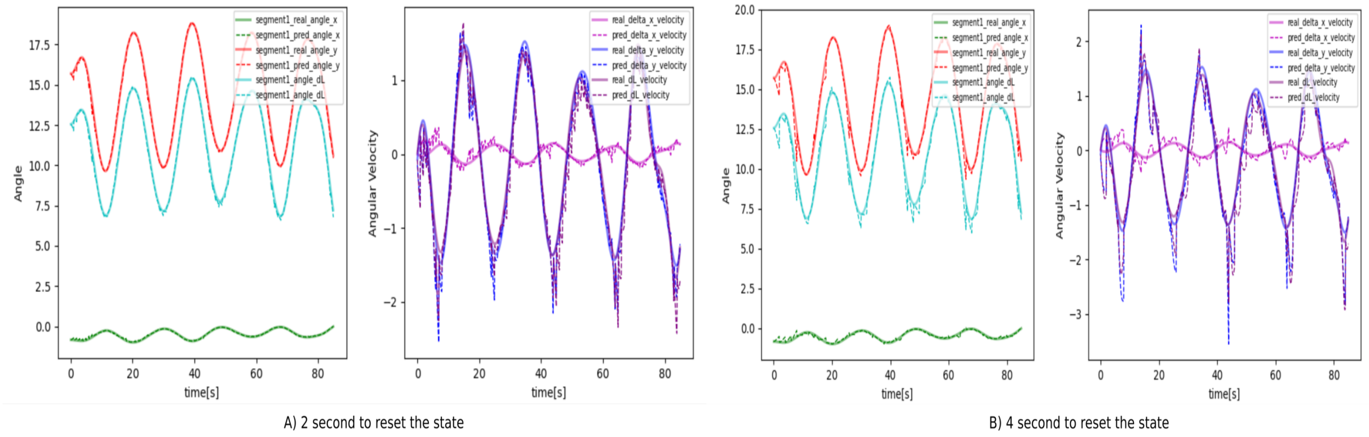


Fig. 62. 80 seconds prediction results of the physics-based learning model with A) 10-steps update state and B) 20-steps update state

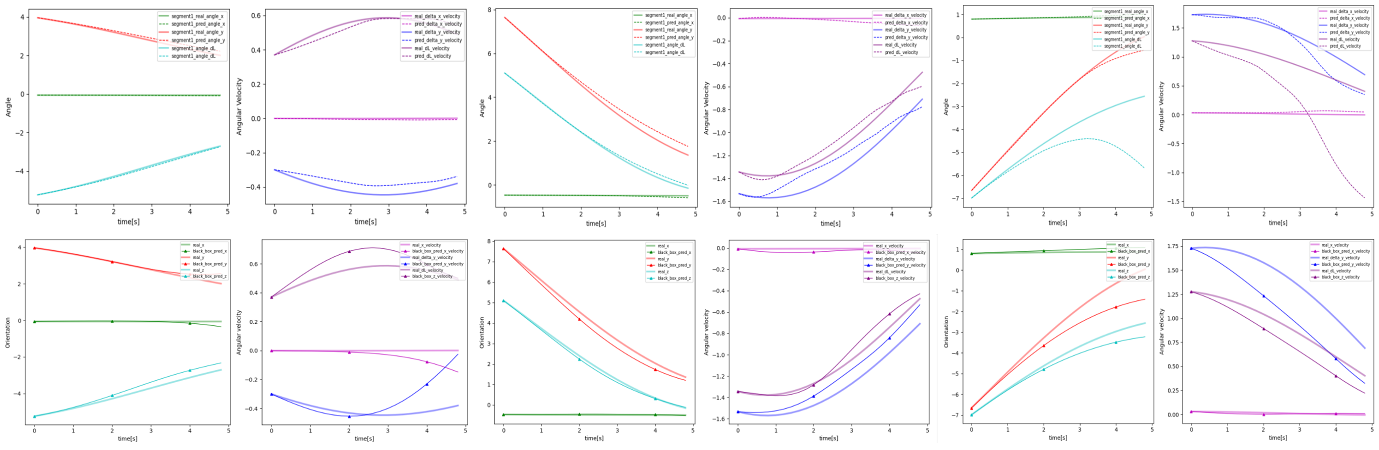


Fig. 63. Black-box model (△) vs physics-based learning model(−) predictions with the same initial state as the fitting data

C. JAX introduction

In machine learning (ML), TensorFlow and PyTorch are the two most packages, but some new forces should not be underestimated in addition to these two frameworks. JAX is one of them, which promises to make ML programming more intuitive, structured, and concise. We will briefly compare the features of Tensorflow, PyTorch and JAX, and then introduce some common JAX methods and the methods used in this project in detail.

TABLE X
COMPARISON OF TENSORFLOW, PYTORCH, AND JAX FRAMEWORKS

	TensorFlow	PyTorch	JAX
Developed by	Google	Facebook	Google
Flexible	No	Yes	Yes
Target Audience	Researchers, Developers	Researchers, Developers	Researchers
Development Stage	Mature	Mature	Developing
Low/High-level API	High level	Both	Both

TensorFlow: The availability of high-level API-Keras makes model layer definition, loss function, and model creation very easy, but this high-level interface of Keras has certain disadvantages. It gives researchers less freedom in working with models; Tensorflow provides TensorBoard, which is effectively a Tensorflow visualization toolkit.

PyTorch: PyTorch includes low-level APIs that allow for more and more control over machine learning models. We can inspect and modify the output of the model's forward and backward passes during training. PyTorch also allows users to extend the code to easily add new loss functions and user-defined layers. PyTorch's Autograd module implements back-propagation derivatives in deep learning algorithms. Autograd can automatically provide differentiation for all operations on the Tensor class, which simplifies the complex process of manually calculating derivatives.

JAX: JAX can perform composable transformations of Python+NumPy programs: vmap or pmap, JIT to GPU/TPU, etc. The most important aspect of JAX compared to PyTorch is how gradients are calculated. In Torch the graph is created during the forward pass and the gradients are computed during the backward pass, on the other hand, in JAX the computation is represented as a function.

Below, we introduce the shining points of JAX in detail.
1) Numpy on CPU, GPU and TPU: JAX provides an API similar to NumPy and is mainly used for array manipulation programs written to perform transformations. Some people even think that JAX can be regarded as Numpy v2, which not only speeds up Numpy but also provides an automatic derivation $(grad())$ function for Numpy so that we can implement a machine learning framework only with JAX. This means that everywhere you use numpy, you can use jax.numpy instead, and the usage is the same. Some simple examples are provided below.

```
import jax.numpy as jnp

def predict(params, inputs):
    for W, b in params:
        outputs = jnp.dot(inputs, W) + b
        inputs = jnp.tanh(outputs)
    return outputs

def loss_fn(params, inputs, targets):
    preds = predict(params, inputs)
    return np.sum((preds - targets)**2)
```

2) Autograd from mathematical view: Using jax can easily calculate gradient, Jacobian and Hessian from a formula perspective.

TABLE XI
JAX AUTODIFF API

Computation	Method
Gradient	jax.grad()
Evaluate a function and its gradient	jax.value_and_grad()
Jacobian	jax.jacfwd() / jax.jacrev()
Hessian	jax.hessian()

```
import jax

loss_grad_fn = jax.grad(loss_fn)
loss_hessian_fn = jax.hessian(loss_fn)
```

Hessian calculation with JAX is compilated via XLA to efficient GPU, CPU, or TPU code. The official documentation proves that the speedup is about 500 times compared to normal NumPy code. Here is a simple example to test the efficiency of the autodiff tool using JAX.

```
import jax.numpy as jnp
```

```
from jax import grad, jit , vmap, jacfwd, jacrev
import jax
import numpy as np
import timeit


@jit
def gauss2d(x0, y0, amp, sigma, rho, diff , a):
    """
    Sample model: Gaussian on a 2d plane
    """
    dx=a [...,0]- x0
    dy= a [...,1]- y0
    r=jnp.hypot(dx, dy)
    return amp*jnp.exp(-1.0/ (2*sigma**2) * (r**2 +
                                              rho*(dx*dy)+
                                              diff *(dx**2-dy**2)))

def mkobs(p, n, s):
    "A simulated observation"
    aa=np.moveaxis(np.mgrid[-2:2:n*1j, -2:2:n*1j], 0, -1)
    aa=np.array(aa, dtype="float64")
    m=gauss2d(*p, aa)
    return aa, m + np.random.normal(size=m.shape, scale=s)


@jit
def cauchy(x, g, x0):
    return 1.0/(jnp.pi * g) * g **2/((x-x0)**2+g**2)

@jit
def cauchyll(o, m, g):
    return -1 * jnp.log(cauchy(m, g, o)).sum()

def makell(o, a, g):
    def ll (p):
        m=gauss2d(*p, a)
        return cauchyll(o, m, g)
    return jit ( ll )

def hessian(f):
    return jit (jacfwd(jacrev( f )))

P=jnp.array ([0.,0., 1.0, 0.5, 0., 0.], dtype="float64" )

a, o = mkobs( P, 3000, 0.5)
ff =makell(o, a, 0.5)
hf=hessian(ff)
ndhf=jax.hessian(ff)

# JIT warmup call. Smallish effect if number is large in timeit
```

```
hf(P).block_until_ready()

print("Time with JAX:", timeit.timeit("hf(P).block_until_ready()", number=10,
        globals=globals()))
print("Time with finite diff:", timeit.timeit("ndhf(P)", number=10, globals=globals()))
```

Running this Intel CPU (no GPU) I get following timings for 10 runs using timeit:
- Time with JAX and automatic differentiation: 16s
- Time with JAX function valuation and finite-difference differentiation (with Numdifftools) : 891s
- Time with plain numpy and numerical differentiation (with Numdifftools): 9900s
Runing this with GPU:
- Tine with JAX and and automatic differentiation: 0.128s

3) Just-in-time compilation (JIT): JAX's $jax.jit()$ transformation will perform just-in-time (JIT) compilation of a JAX Python function for efficient execution in XLA.

```
loss_grad_fn = jax.jit(jax.grad(loss_fn))
```

4) Vectorization: The vmap function in JAX creates a function which maps fun over argument axes.

```
perex_grads = jax.jit(jax.vmap(loss_grad_fn, in_axes=(None, 0, 0)))
```