IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

Generalizable and Fast Surrogates: Model Predictive Control of Articulated Soft Robots using Physics-Informed Neural Networks

Tim-Lukas Habich, Aran Mohammad, Simon F. G. Ehlers, Martin Bensch, Thomas Seel, and Moritz Schappler

***Abstract*—Soft robots can revolutionize several applications with high demands on dexterity and safety. When operating these systems, real-time estimation and control require fast and accurate models. However, prediction with first-principles (FP) models is slow, and learned black-box models have poor generalizability. Physics-informed machine learning offers excellent advantages here, but it is currently limited to simple, often simulated systems without considering changes after training. We propose physics-informed neural networks (PINNs) for articulated soft robots (ASRs) with a focus on data efficiency. The amount of expensive real-world training data is reduced to a minimum — one dataset in one system domain. Two hours of data in different domains are used for a comparison against two gold-standard approaches: In contrast to a recurrent neural network, the PINN provides a high generalizability. The prediction speed of an accurate FP model is exceeded with the PINN by up to a factor of 467 at slightly reduced accuracy. This enables nonlinear model predictive control (MPC) of a pneumatic ASR. Accurate position tracking with the MPC running at 47 Hz is achieved in six dynamic experiments.**

***Index Terms*—Modeling, control, and learning for soft robots, physics-informed machine learning, model learning for control, optimization and optimal control**

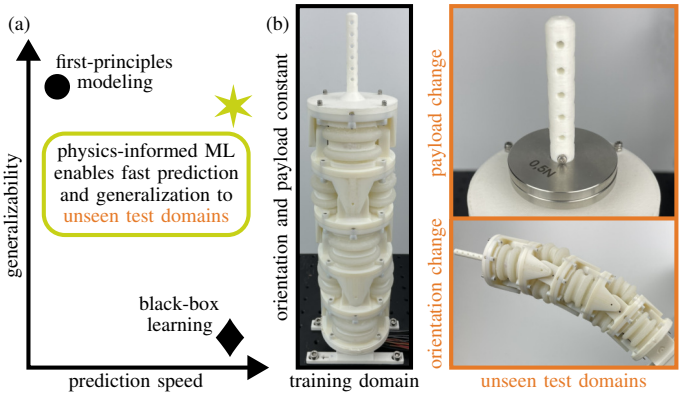


Figure 1. (a) The objective of this work is to solve the trade-off between model accuracy/generalizability and prediction speed in the field of soft robotics by using physics-informed machine learning (ML). (b) During training, data from *one* training domain is available. Trained surrogate models extrapolate for changed system dynamics in unseen test domains. We consider changing the payload and the orientation of the robot base as possible modifications during operation.

I. Introduction

Building soft robots has been an emerging research field for several years. In contrast to conventional rigid robots, they are made of significantly softer materials. The resulting compliance makes them suitable robot candidates for intrinsically safe interaction with humans, as less damage is caused in the event of a collision [1]. Modeling and controlling such rubber-like robots is challenging mainly due to complex geometries, material nonlinearities, and air compressibility in case of pneumatic actuation [2]. Therefore, handcrafted models built with conventional first-principles approaches lack accuracy. Even if the accuracy is high due to advanced modeling/identification techniques: *Prediction with first-principles models is slow* due to numerical integration with small time steps. This prevents the application of these models in real-time estimation and control.

In contrast, *learning-based modeling* only requires input-output data of the system, and the prediction with learned forward models is possible with large time steps. Therefore, high prediction speeds are possible and enable such real-time applications. However, a *huge amount of real-world data* is necessary. The recording is not only expensive and time-consuming, but there is another central problem: *Learned models only show good performance within the seen data space* (interpolation). For changing domains after training (extrapolation), there is usually poor generalization for such black-box approaches [3]. Combining the advantages of both modeling worlds in a *hybrid strategy* (and omitting the disadvantages), as sketched in Fig. 1(a), is a widespread goal in various research fields.

Within soft robotics, hybrid approaches incorporating both physical knowledge and machine learning are a highly promising line of research [4] and an unexplored area [5]. Laschi et al. [6] conclude that *traditional modeling techniques must be incorporated into existing learning strategies* for new advancements in the soft-robotics field. Recently, Falotico et al. [7] also declare the integration of physical principles into machine-learning approaches as a perspective for solving current issues in terms of accuracy and computational efficiency. They *explicitly mention PINNs*, which is the focus of our work.

As an emerging strategy, PINNs [8] have been used in a wide variety of applications to solve ordinary differential equations (ODEs), e.g., for modeling nonlinear structural systems, and partial differential equations (PDEs) such as Navier-Stokes equations [9]. We use PINNs to provide fast and accurate

All authors are with the Leibniz University Hannover, Institute of Mechatronic Systems, 30823 Garbsen, Germany (corresponding author: Tim-Lukas Habich, e-mail: habich@imes.uni-hannover.de).

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — 433586601 and INST 187/742-1 FUGG.

surrogate models of articulated soft robots. One central requirement is the *generalization to unseen dynamics*, which were not present when recording training data. Examples are an additional mass or a changed base orientation, illustrated in Fig. 1(b).

The *four major scientific contributions* of this article are:

1) For the first time, the original form of physics-informed neural networks — introduced by Raissi et al. [8] and formulated for state-space modeling in [10], [11] — is applied to a real soft robot with multiple degrees of freedom (DoF). For this purpose, a first-principles model of our ASR is derived and identified, which is very accurate, but the forward simulation requires significant computing time. This justifies the need for a fast surrogate model with high accuracy to enable real-time applications of model-based estimation/control.
2) Two existing PINN architectures (PINC [10] and DD-PINN [11]) are extended to achieve generalizability despite system changes after training. During training, we perform a systematic hyperparameter optimization (HPO). This is often neglected in the literature due to excessive training times of several days.
3) The proposed model is compared against the gold standard of both modeling worlds, namely a hyperparameter-optimized recurrent neural network (RNN) and an identified FP model, regarding prediction speed and accuracy. For this purpose, we recorded two hours of real-world data with variable soft-robot dynamics.
4) To demonstrate one possible use case for the proposed PINN, a real-time application is realized. We choose nonlinear MPC (NMPC) of the real soft robot, which places high demands not only on prediction accuracy but also on prediction speed due to online optimization. For the first time, NMPC with PINNs is realized for multi-DoF soft robots and validated in several dynamic real-world experiments, including comparison with a baseline controller.

To enable reproducibility, we further provide the community with *open-source contributions*[1]. This includes the whole codebase for PINN training (PINC and DD-PINN), hyperparameter optimization and learning-based NMPC with PINNs for possible applications within and beyond soft robotics. The two hours of experimental data (13 real-world datasets of the articulated soft robot) are also publicly available [12].

In sum, we make *three claims*: First, our surrogate model outperforms an accurate physics-driven model in terms of prediction speed by orders of magnitude with only slightly reduced accuracy. Second, by incorporating physical knowledge during training, the PINN achieves higher prediction accuracies and generalization to out-of-distribution data compared to an RNN. Throughout this article, all models receive data from only one domain during training/identification and are tested in unseen test domains, as depicted in Fig. 1(b). Third, accurate learning-based nonlinear MPC with PINNs is enabled for different soft-robot dynamics without the need to retrain the system model or to retune the controller. This is intended to represent one practical scenario with high demands on prediction speed and accuracy. *All claims are proven experimentally.* Please refer to the supplementary video of this article to see the soft robot and dynamic movements during control.

[1]https://tlhabich.github.io/sponge/pinn_mpc

The remainder of this article is organized as follows: After an overview of related work (Sec. II), preliminaries are introduced (Sec. III). State-space modeling of the soft robot using first principles and PINNs is presented, followed by the architecture for NMPC (Sec. IV). In Sec. V, the identification and learning results are shown, and all models are compared regarding prediction speed and accuracy. Also, control experiments are presented. The article ends with conclusions, including future work directions (Sec. VI).

II. Related Work

A brief overview of the two modeling worlds, namely first-principles modeling (Sec. II-A) and black-box learning (Sec. II-B), is given. This is followed by presenting directions of hybrid learning (Sec. II-C), whose various approaches combine physical knowledge with machine learning. Further, related work on physics-informed ML for (soft) robots (Sec. II-D) are discussed.

A. First-Principles Modeling

Soft robots can be divided according to two design paradigms: soft continuum robots (SCRs) vs. ASRs [5], [13]. Depending on the robot design, a different modeling approach is recommended, which is described briefly below.

Due to the deformable structure along a continuous backbone, SCRs have infinite DoF. Approaches such as piecewise constant curvature [14], rod models [15], or finite element method (FEM) [16] are therefore required for kinematic and dynamic modeling, to name just a few. For a structured overview, please refer to [17].

The focus of this article is on ASRs. They comprise a vertebrate-like structure with rigid links and compliant joints. The use of traditional methods for kinematics, such as the Denavit-Hartenberg notation, and multi-body dynamics, such as the Lagrange formalism, are applicable here. Lumped mass models are therefore recommended for such systems [4], which have been used in various publications for ASRs [18]–[20].

Whether ASR or SCR, measurement data from the real robot is usually used for the identification of unknown system parameters. This is somewhat similar to black-box learning but with a strong inductive bias on the model architecture derived from physics. Thus, *high generalizability and data efficiency* can be achieved with physics-based approaches. However, first-principles models can have a low accuracy due to simplifying assumptions. The *main disadvantage of accurate first-principles models for soft robots is the slow prediction speed*. This is due to the numerical integration of forward models with very fine step sizes, which can complicate model-based estimation and control.

B. Black-Box Learning

Within soft robotics, data-driven approaches for control are still in their infancy [21], with much progress having been made in recent years. In [22], *feedforward neural networks* (NNs) were first applied to feedforward control of soft extensible continuum manipulators. Many works also use feedforward NNs for MPC, e.g., of a soft actuator with one DoF [23], a soft continuum joint [24], and a six-DoF soft robot [25], [26]. In [27], Gaussian processes are trained to realize learning-based position and stiffness feedforward control of a soft actuator.

Nonlinear material behavior and friction cause hysteresis effects that can be captured using *recurrent neural networks* [28]. These are used in [29] to learn the forward dynamics of a soft manipulator with a nonlinear autoregressive exogenous model. Only an open-loop predictive control policy was implemented, which was further developed in [30] for closed-loop control. Data-driven MPC was successfully realized for a single actuator by using long short-term memory (LSTM) units [31] and for the ASR from Fig. 1(b) via gated recurrent units (GRUs) [32].

Recording real-world data requires a lot of effort, and trained networks only achieve *good accuracy within the seen data space*. In case of system changes, training must be repeated with new data due to *poor extrapolation*. In addition, convergence is more difficult for high-dimensional NNs required to model multi-DoF robots. For this reason, the prediction of the entire state vector was not possible with the chosen network architecture and the existing data in [25]. Thus, only the velocity at the next time step was learned. Data-driven control of a soft robot using the Koopman operator is realized in [33], and, more recently, a data-efficient method based on neural ODEs [34] was used to model a soft manipulator [35]. However, the authors do not use physical knowledge, which results in poor generalization.

For *different payloads*, reinforcement learning using an LSTM network as a forward-dynamics model was conducted in [36]. However, even for only $\nu{=}1$ varying model parameter (payload), different real-world datasets with variable payload conditions are necessary as training data. We do not believe that such black-box approaches are scalable for real-world applications, i.e., $\nu{>}1$: For the minimum requirement of two levels per model parameter (e.g., minimum and maximum payload), 2^ν datasets are already required. This implies that for variations of several parameters, a time-consuming and expensive data acquisition is necessary. In addition, some parameters require finer variations and, therefore, more than two levels due to poor interpolation.

C. Directions of Hybrid Learning

In summary, both first-principles and black-box approaches have problems *when generalizability and prediction speed are considered jointly*. The term generalizability describes the aim for high accuracy not only in known data domains but also for unseen system domains with changed dynamics. Hybrid approaches are promising here and apply *physical knowledge in combination with black-box learning*.

Within the pioneering work of Nguyen-Tuong and Peters for rigid robots [37], knowledge of the parametric dynamics model is incorporated into the nonparametric Gaussian process via mean or kernel function. Finite-element models have been used to analyze soft robotic segments, thus generating vast amounts of (simulated) training data for feedforward NNs [38]. External loads such as gravity or contact forces are not taken into account.

Residual/Error learning of physical models using data is one main direction of hybrid learning. This has been done for a soft continuum joint [39], a soft parallel robot [40], and soft continuum robots [41]–[44]. The main requirement for these approaches is that *the physical model must be efficiently evaluated* during real-time control or estimation, which is often not the case for accurate models of soft robots (cf. Sec. II-A). Another possible disadvantage is the need for real experimental data from all domains. This indicates, due to poor extrapolation, that a new error model must be learned if the system changes after training. Consequently, the authors of [43] collect training data in several domains (different payloads between $0\,\mathrm{g}$ and $300\,\mathrm{g}$) in order to ensure generalization within this data region (interpolation). However, we argue that trained models should extrapolate across the available training data and that data acquisition for every possible system change is not practical. In line with this, it was recently concluded that the *generalization beyond bounded training data and the handling of novel dynamical events* should be examined in future work [42]. Also, the authors of [40] declare robust control with learned models in changing environments as future work.

Alternatively to error learning, *prior knowledge for constraining NN training* results in less data required. This was realized in [45] using extreme learning machines with only one hidden layer. Due to the simple network structure, linear constraints, such as the monotonic behavior of the drives or physical restrictions of the soft robot, can be taken into account using quadratic optimization. In [46], a soft manipulator was discretized according to continuum-mechanics principles, and recurrent neural networks were trained for each node, taking applied forces and moments for every node as inputs. The authors conclude that the supervised learning used might not entirely respect underlying physics, and *trained models are not applicable to changing dynamics without retraining*.

The aforementioned problems are countered by *physics-informed machine learning* [47], also known as scientific machine learning [9]. This is done by including physical knowledge into the training process so that *trained regressors will be aware of governing physical laws*. Regarding modeling and control of dynamical systems, physics-informed ML is an alternative approach compared to classical gray-box modeling: Instead of (or in addition to) identifying parameters of white-box models consisting of simplified modeling assumptions, physics-informed ML comes from the black-box side without modeling simplifications and is equipped with additional constraints such as symmetries, causal relationships, or conservation laws [48]. Underlying ODEs/PDEs can act as learning bias during the training process, limiting the high-dimensional search space and, therefore, the necessary amount

of real-world data while still maintaining a high generalization. This is beneficial since real-world experiments are expensive, and the *training data will rarely represent all possible system conditions during inference*. Besides high generalizability despite less data, trained surrogate models are considerably faster to evaluate since they do not rely on fine spatiotemporal discretization required for numerical integration of conventional ODEs/PDEs. This significantly speeds up the solution of hard optimization problems, such as during nonlinear MPC, which is outlined in a recent overview [48] as one of the main opportunities for physics-informed ML.

D. Physics-Informed Machine Learning in (Soft) Robotics

For rigid systems, physics-inspired networks using Lagrangian and Hamiltonian mechanics were proposed [49], [50]. Mass matrix, centrifugal, Coriolis, gravitational, and frictional forces as well as (potential and kinetic) energies are learned unsupervised by minimizing the ODE residual. In [51], PINNs are employed for the nonlinear MPC of a simulated rigid robot with two degrees of freedom.

Within soft robotics, hybrid RNNs using responses from the physical model as additional inputs for state observation of one-DoF soft actuators (McKibben actuator and soft finger) are utilized in [52]. A physics-informed simulation model is trained from finite-element data and is used as *efficient-to-evaluate surrogate model* within MPC in [53]. A simulated one-DoF soft finger is considered, and the authors identify the transfer to physical prototypes and extension to multi-DoF tasks as a future research direction. Similarly, PINNs are trained as fast surrogate models of soft robotic fingers in [54], [55]. To enable fast prediction of Cosserat rod statics, PINNs are trained for one simulated tendon-driven segment of a continuum robot [56].

A real one-segment tendon-driven soft robot was modeled in [57] using Lagrangian neural networks (LNNs) and a real-world dataset. This has two drawbacks. Since the network is trained with offline-recorded data, the generalization to unseen dynamics (due to system changes not present in the training data distribution) would be poor. Also, LNNs still require numerical integration to predict future states since only system quantities (such as inertia or damping) are learned with neural networks. For this purpose, the authors [57] use Runge-Kutta integration. Similar to the forward integration of our FP model, a slow prediction speed results due to necessary fine step sizes.

The authors of [58] propose a kinematics-informed neural network for pneumatic and tendon-driven soft robots with one segment. The generalization ability is examined for motor commands, which are unseen during network training. Although this is already an important study, generalization to changed systems would be even more crucial. Learning system dynamics instead of pure kinematics would also be advantageous for optimal control, which is realized in [20] for an ASR using differential dynamic programming. Thereby, forward dynamics are simulated via inversion of the inertia matrix and they declare the realization of MPC as future work. For our robot, forward simulation via inversion of the inertia matrix and numerical integration of the *stiff ODE* is very time-consuming. Therefore, a fast and accurate surrogate model is necessary for MPC.

In [59], experimental data is extended with simulated finite-element data of *out-of-distribution* (unseen) loading/deformation cases to train a PINN of a single-bellows actuator. The main motivation here is to build a fast surrogate of a computationally expensive finite-element model. Although the NN is 435 times faster compared to the FEM, one simulation still takes $>20\,\mathrm{s}$ for the parallel actuator consisting of five bellows. This computation time further increases for soft robots with more bellows making it unsuitable for real-time applications. Also, only statics are modeled to predict deformations.

Most similar to our proposed approach, the authors of [60] formulate a physics-informed LSTM network with a *variable system property as model input*. This allows the training of the model for unseen system properties, which are not present when recording training data. However, it is only applied to a simple one-DoF mass-spring-damper system. Control is also not considered.

To summarize, related soft-robotics work only considers simple (often simulated) actuators and not PINNs for state-space models [10], [11], which are specifically designed for real-time MPC. To the best of our knowledge, no published work considers PINNs for fast MPC of — neither real nor multi-DoF — soft robotic systems. Dynamics changes after training are also predominantly neglected.

III. Preliminaries

This section covers the basics of RNNs (Sec. III-A) and PINNs (Sec. III-B). The method used for optimizing their hyperparameters (Sec. III-C) is also briefly described.

A. Recurrent Neural Networks

In contrast to utilizing physical knowledge, forward dynamics can be trained with black-box approaches on existing training data. Recurrent neural networks are the gold standard for time-series learning and can approximate dynamical system behavior by incorporating information from previous time steps. They can represent state-space models [61] and are promising for learning long-term dependencies, such as hysteresis. The predicted states

$$[\boldsymbol{h}(T_\mathrm{s}), \hat{\boldsymbol{x}}(T_\mathrm{s})] = \boldsymbol{f}_\mathrm{RNN}(\boldsymbol{h}_0, \boldsymbol{x}_0, \boldsymbol{u}_0) \tag{1}$$

for one step with sample time T_s can be computed given hidden state $\boldsymbol{h}_0{=}\boldsymbol{h}(0)$, initial state $\boldsymbol{x}_0{=}\boldsymbol{x}(0)$, and input $\boldsymbol{u}_0{=}\boldsymbol{u}(0)$ at time[2] $t{=}0$. Note that we use the current state $\boldsymbol{x}_0$ as network input due to the MPC context of this work.

Gated recurrent units [62] are well suited for time-series learning due to the ability to prevent vanishing and exploding gradients during backpropagation. Such network architecture outperformed LSTM networks for the used ASR [32]. We, therefore, use GRUs as the reference method for a purely data-driven approach in this work.

[2]For better comparison, the continuous notation with time t is used throughout the article for all models, although the implementation of RNNs is discrete.

The network is trained batch-wise via the common backpropagation-through-time (BPTT) algorithm [63], which is briefly described in the following. During simulation with known initial values $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$, the hidden state is initialized to $\boldsymbol{h}_0{=}\boldsymbol{0}$ and then recursively passed for each time step in the same way as the predicted state $\hat{\boldsymbol{x}}(T_\mathrm{s})$ for a given input trajectory $\boldsymbol{u}(t)\forall t$. This behavior is imitated during training on a given dataset, such that the network predictions are compared to the ground-truth data. Since the RNN outputs are recursively passed as inputs for the next time step, backpropagation must be done through the whole time series, which requires extensive computational effort and memory. This makes it often unfeasible in practice for larger datasets [3]. Multi-step truncation enables a feasible approximation of the gradient [3], whereby BPTT is performed only for subsets (batches) of the whole training set. We perform such truncated BPTT for each batch of n_b datapoints and detach the gradients of $\hat{\boldsymbol{x}}$ and $\boldsymbol{h}$ at the beginning of each new batch. For each batch, the Adam optimizer adjusts the weights depending on the error between the predictions and the ground-truth data.

B. Physics-Informed Neural Networks

Purely black-box models require a large amount of experimental data, which is very expensive (e.g., costs for staff and maintenance of the system) to acquire in the real world. Usually, the available data only describe a few operating points, and trained regressors, therefore, only perform well within these data regions. PINNs [8] provide an excellent opportunity in this *small-data regime* to incorporate existing domain knowledge and thus regularize the training process by additional physics-based loss terms. Here, the basic idea is that the loss function for optimizing the network weights does not only consist of a data loss $\mathcal{L}_\mathrm{d}$ as in RNNs but also takes into account additional knowledge using physics $\mathcal{L}_\mathrm{p}$ and initial-condition/boundary $\mathcal{L}_0$ losses. The additional losses are evaluated on a finite set of *sampled collocation points* in the entire input space and do not require additional real-world datasets. This leads to encoding physical knowledge into the NNs to achieve broad generalizability.

Based on this idea, Antonelo et al. [10] reformulate such PINN for control (PINC) to enable simulations with variable, long-range time horizons. These PINCs are designed for fast MPC of dynamical systems with state-space models from first principles

$$\hat{\dot{\boldsymbol{x}}}(t) = \boldsymbol{f}_\mathrm{FP}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2}$$

with states $\boldsymbol{x}(t)$ and inputs $\boldsymbol{u}(t)$. This is done by the use of several NN inputs such as the time t, initial state $\boldsymbol{x}_0$, and control action $\boldsymbol{u}_0$. Prediction for indefinitely long horizons is realized by recursively feeding back the PINN output

$$\hat{\boldsymbol{x}}(T_\mathrm{s}) = \boldsymbol{f}_\mathrm{PINN}(T_\mathrm{s}, \boldsymbol{x}_0, \boldsymbol{u}_0) \approx \boldsymbol{x}(T_\mathrm{s}) \tag{3}$$

as initial value $\boldsymbol{x}_0$ at the subsequent time step with new input $\boldsymbol{u}_0$. The latter is considered constant within each time interval $t{\in}[0, T_\mathrm{s})$ (zero-order hold assumption). Therefore, the NN is only trained for small time steps, and forecasting is done via *self-loop prediction*, which is similar to RNNs.

The physics loss is determined by means of collocation points evaluated with (3) and compared with (2) using automatic differentiation $\frac{\partial \hat{\boldsymbol{x}}}{\partial t}$. In addition, the initial condition

$$\boldsymbol{x}_0 = \boldsymbol{f}_\mathrm{PINN}(0, \boldsymbol{x}_0, \boldsymbol{u}_0) \tag{4}$$

is learned using an initial-condition loss. In fact, no real data was used for training a surrogate model to solve the ODEs of two benchmark systems (Van-der-Pol oscillator and four-tank system) in [10]. Their main motivation is the development of an efficient-to-evaluate model instead of the time-consuming numerical integration of (2). This is due to the fact that PINNs are trained to directly compute $\hat{\boldsymbol{x}}(T_\mathrm{s})$, cf. (3), while integrators such as Runge-Kutta methods require considerably smaller step sizes due to instabilities.

The high prediction speed is accompanied by a considerably increased training effort of several days compared to RNNs. Especially for dynamical systems with several states, the PINC training is very time-consuming. To counter this, the domain-decoupled PINN (DD-PINN) was proposed in [11], which approximates the solution

$$\hat{\boldsymbol{x}}(T_\mathrm{s}) = \boldsymbol{f}_\mathrm{PINN}(T_\mathrm{s}, \boldsymbol{x}_0, \boldsymbol{u}_0) = \boldsymbol{x}_0 + \boldsymbol{a}(\boldsymbol{\alpha}(\boldsymbol{x}_0, \boldsymbol{u}_0), T_\mathrm{s}) \tag{5}$$

with an ansatz function

$$\boldsymbol{a}(\boldsymbol{\alpha}, t){=}\sum_{i=1}^{n_\mathrm{a}} \boldsymbol{\alpha}_{1i}{\odot}\big(\exp{(-\boldsymbol{\alpha}_{4i}t)}{\odot}\sin{(\boldsymbol{\alpha}_{2i}t{+}\boldsymbol{\alpha}_{3i})} - \sin{(\boldsymbol{\alpha}_{3i})}\big) \tag{6}$$

using Hadamard product ($\odot$) and element-wise functions $\exp{(\cdot)}$ and $\sin{(\cdot)}$. As solution [11], a superposition of damped harmonic oscillations with amplitude $\boldsymbol{\alpha}_{1i}$, damping term $\exp{(-\boldsymbol{\alpha}_{4i}t)}$ and oscillation term $\sin{(\boldsymbol{\alpha}_{2i}t{+}\boldsymbol{\alpha}_{3i})}$ with angular frequency $\boldsymbol{\alpha}_{2i}$ and phase shift $\boldsymbol{\alpha}_{3i}$ is assumed. The subtraction of $\sin{(\boldsymbol{\alpha}_{3i})}$ in (6) ensures the compliance with the initial condition $\hat{\boldsymbol{x}}(0){\equiv}\boldsymbol{f}_\mathrm{PINN}(0, \boldsymbol{x}_0, \boldsymbol{u}_0){\equiv}\boldsymbol{x}_0$ due to $\boldsymbol{a}(\boldsymbol{\alpha}, 0){\equiv}\boldsymbol{0}$.

In constrast to the PINC (3), only the ansatz vector

$$\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1^\mathrm{T}, \boldsymbol{\alpha}_2^\mathrm{T}, \boldsymbol{\alpha}_3^\mathrm{T}, \boldsymbol{\alpha}_4^\mathrm{T}]^\mathrm{T} \in \mathbb{R}^{4\dim(\boldsymbol{x})n_\mathrm{a}} \tag{7}$$

is predicted by the feedforward NN. Each $\boldsymbol{\alpha}_j{=}[\boldsymbol{\alpha}_{j1}^\mathrm{T}, \ldots, \boldsymbol{\alpha}_{jn_\mathrm{a}}^\mathrm{T}]^\mathrm{T}$ consists of n_a vectors of length $\dim(\boldsymbol{x})$. The key advantage is that the training converges much faster [11]. This is mainly because the computationally expensive automatic differentiation is avoided. Instead, $\frac{\partial \hat{\boldsymbol{x}}}{\partial t}{=}\dot{\boldsymbol{a}}(\boldsymbol{\alpha}, t)$ is calculated in closed form by time differentiation of (6), which results to

$$\dot{\boldsymbol{a}}(\boldsymbol{\alpha}, t){=}\sum_{i=1}^{n_\mathrm{a}} \boldsymbol{\alpha}_{1i}{\odot}\exp{(-\boldsymbol{\alpha}_{4i}t)}{\odot}\big(-\boldsymbol{\alpha}_{4i}{\odot}\sin{(\boldsymbol{\alpha}_{2i}t{+}\boldsymbol{\alpha}_{3i})} + \cos{(\boldsymbol{\alpha}_{2i}t{+}\boldsymbol{\alpha}_{3i})}{\odot}\boldsymbol{\alpha}_{2i}\big) \tag{8}$$

with element-wise function $\cos{(\cdot)}$. As already described in the last paragraph, by decoupling the time domain t from the feedforward NN and constructing the ansatz function (6), the initial condition is always maintained. We therefore do not need a separate initial-condition loss, as $\mathcal{L}_0{\equiv}0$ applies. This further simplifies the network training. Both PINC and DD-PINN are extended for variable domains in this work so that generalization can be achieved even when the system changes after training.

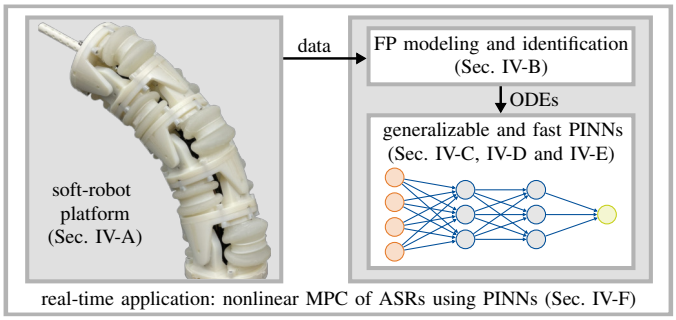


Figure 2. Graphical overview of the article's main part (Sec. IV).

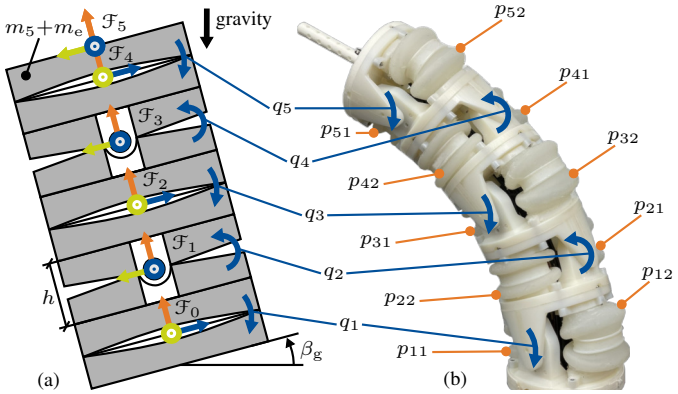


Figure 3. (a) Kinematic chain of the articulated soft robot with n=5 rotational actuators of height h=53.4 mm and coordinate frames $\mathcal{F}_i$. In this work, dynamics are changed by attaching a variable mass m_e to the last segment with mass m_5 and by changing the base orientation β_g against gravity. (b) Real robot with joint angles q_i and antagonistic actuation via pneumatic pressures p_{i1} and p_{i2}.

C. Hyperparameter Optimization

The hyperparameter optimization of the NNs in this work is conducted with the asynchronous successive halving algorithm (ASHA) [64] to obtain optimal network architectures. ASHA samples within the given boundaries of the hyperparameters and starts multiple training runs based on the available computing resources. By monitoring the validation loss of each hyperparameter configuration (trial) during training, poorly performing configurations are stopped by choosing a reduction factor, and the resources are used for new training runs. All trials are minimally trained until the user-defined grace period is reached. The latter is a crucial parameter and should be chosen sufficiently large, as otherwise, good trials with slower convergence will be rejected. In contrast, a grace period that is too long unnecessarily extends the time required for the HPO. After a defined number of trials, the hyperparameters of the training result with the lowest validation loss are chosen as the optimal set.

IV. PINNs for Articulated Soft Robots

The content of this section is graphically illustrated in Fig. 2. After describing the soft-robot platform (Sec. IV-A), a state-space model is obtained using first principles and system identification (Sec. IV-B). Based on this, the generalizable and fast surrogates are introduced (Sec. IV-C) including implementation details (Sec. IV-D) and a note on their practical usability (Sec. IV-E). The proposed PINNs are integrated into a nonlinear MPC as *one possible real-time application* (Sec. IV-F).

A. Soft-Robot Platform

The open-source soft pneumatic robot with antagonistic bellows from [65] is used in this work with n=5 stacked actuators, which is shown in Fig. 3. Antagonistically arranged cast bellows with air pressures $\boldsymbol{p}_i$=$[p_{i1}, p_{i2}]^\mathrm{T}$ actuate each joint i. Bellows pressures $\boldsymbol{p}$=$[\boldsymbol{p}_1^\mathrm{T}, \dots, \boldsymbol{p}_n^\mathrm{T}]^\mathrm{T}$ and joint angles $\boldsymbol{q}$=$[q_1, \dots, q_n]^\mathrm{T}$ are measured. The latter is (first-order) low-pass filtered with 1 Hz, and joint velocities $\dot{\boldsymbol{q}}$ are obtained online via numerical differentiation. Desired bellows pressures $\boldsymbol{p}_\mathrm{d}$ are set using external proportional valves.

Further information regarding the semi-modular ASR can be found in the supporting video of [65]. The published design has been minimally optimized with regard to various points: joint-friction reduction via smaller shaft diameters, reconstruction of the frames to reduce plastic deformation, increased maximum working pressures by using thicker bellows, and larger tube diameters for faster pressure dynamics. The improved version is freely available[3]. To further reduce friction, we removed the cable/tube guides in each actuator.

The additional end-effector mass m_e and the base orientation β_g can be varied, summarized in the *domain variable* $\boldsymbol{\delta}$=$[m_\mathrm{e}, \beta_\mathrm{g}]^\mathrm{T}$. Data $\boldsymbol{D}_\mathrm{t}$ with several training samples of $[\boldsymbol{q}^\mathrm{T}, \dot{\boldsymbol{q}}^\mathrm{T}, \boldsymbol{p}^\mathrm{T}, \boldsymbol{p}_\mathrm{d}^\mathrm{T}]^\mathrm{T}$ in the training domain $\boldsymbol{\delta}_\mathrm{t}$ (constant additional mass and constant base orientation) is acquired for the identification/training of the FP model (Sec. IV-B), black-box model (Sec. III-A) and the hybrid model (Sec. IV-C). *Afterward*, the domain is modified, which aims to represent a change in the real system that occurs after recording training data. Our method is intended to generalize even for such changes that are unknown during training. As a general illustration of our method, ν=$\dim(\boldsymbol{\delta})$=2 different parameters are investigated. This can be extended arbitrarily.

B. First-Principles Modeling and Identification

1) Robot Dynamics: Forward kinematics of the serial robot are obtained using Denavit-Hartenberg notation. Based on this, dynamics

$$\overbrace{\underbrace{\boldsymbol{\tau}(\boldsymbol{p})=\boldsymbol{g}(\boldsymbol{q})+\boldsymbol{s}(\boldsymbol{q})}_{\mathrm{I}}+\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}+\boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}})+\boldsymbol{d}(\dot{\boldsymbol{q}})}^{\mathrm{II}}+\boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) \quad (9)$$

are modeled by means of Euler-Lagrange equations with gravitational $\boldsymbol{g}$, Coriolis/centrifugal $\boldsymbol{c}$, and inertial torques $\boldsymbol{M}\ddot{\boldsymbol{q}}$ with mass matrix $\boldsymbol{M}$ and accelerations $\ddot{\boldsymbol{q}}$. The latter is only necessary for offline system identification and is determined via offline low-pass filtering and (two-times) numerical differentiation of $\boldsymbol{q}$.

To capture joint friction and the bellows behavior, (9) additionally contains viscous and Coulomb friction

[3] https://tlhabich.github.io/sponge/designs/semi_modular/main

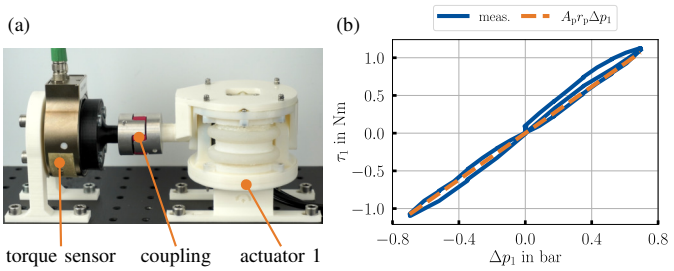


Figure 4. Validation of the actuation model: (a) Experimental setup with torque sensor attached to the first actuator. (b) The mapping from pressures to torque with factor $A_\mathrm{p} r_\mathrm{p}$ applies for the entire pressure range.

$\boldsymbol{d}{=}\boldsymbol{k}_\mathrm{v}{\odot}\dot{\boldsymbol{q}}{+}\boldsymbol{k}_\mathrm{C}{\odot}\tanh(\dot{\boldsymbol{q}}\pi/\dot{q}_\mathrm{C})$, and stiffness torques $\boldsymbol{s}{=}\boldsymbol{k}_\mathrm{s}{\odot}\boldsymbol{q}$. The hyperbolic tangent with threshold $\dot{q}_\mathrm{C}$ is used instead of the discontinuous signum function.

For modeling the contact torques $\boldsymbol{b}{=}[b_1,\dots,b_n]^\mathrm{T}$ at the compliant joint limits, we adapt the nonlinear model of contact normal force [66], which is an improved version of the Hunt-Crossley model [67]. This leads to

$$b_i = \begin{cases} +|\Delta q_i|^{3/2} k_\mathrm{bs} + \sqrt{|\Delta q_i|}\dot{q}_i k_\mathrm{bd} & \text{for } q_i > q_\mathrm{bt} \\ -|\Delta q_i|^{3/2} k_\mathrm{bs} + \sqrt{|\Delta q_i|}\dot{q}_i k_\mathrm{bd} & \text{for } q_i < -q_\mathrm{bt} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

for each joint i with $\Delta q_i{=}q_i{-}q_\mathrm{bt}$, threshold of the soft boundaries q_bt, and contact parameters $\boldsymbol{k}_\mathrm{b}{=}[k_\mathrm{bs}, k_\mathrm{bd}]^\mathrm{T}$.

The left-hand side of (9) consists of pressure differences $\Delta\boldsymbol{p}{=}[\Delta p_1,\dots,\Delta p_n]^\mathrm{T}$ with $\Delta p_i{=}p_{i1}{-}p_{i2}$, which are mapped to joint torques $\boldsymbol{\tau}{=}A_\mathrm{p} r_\mathrm{p}\Delta\boldsymbol{p}$ via pressure area $A_\mathrm{p}{=}639.8\,\mathrm{mm}^2$ and lever arm $r_\mathrm{p}{=}24.1\,\mathrm{mm}$ of each actuator. The simple linear actuation model is validated experimentally by coupling a torque sensor to the first actuator, which is shown in Fig. 4. For an exemplary pressure trajectory, it is shown that the actuation model is valid for the whole pressure range with maximum pressure $p_\mathrm{max}{=}0.7\,\mathrm{bar}$.

2) Identification Parameters: The dynamics equation (9) consists of several unknown parameters

$$\boldsymbol{k} = [\boldsymbol{k}_\mathrm{s}^\mathrm{T}, \boldsymbol{k}_\mathrm{v}^\mathrm{T}, \boldsymbol{k}_\mathrm{C}^\mathrm{T}, \boldsymbol{k}_\mathrm{b}^\mathrm{T}]^\mathrm{T} \in \mathbb{R}^{3n+2}. \tag{11}$$

All other parameters are either adopted from CAD software (actuator height, center-of-gravity positions, and entries of the inertia tensors) or measured (segment masses), whereby the bellows, sensor cables, and tubes are neglected. The kinematic and inertia parameters of the first $n{-}1$ actuators are equal due to the repetitive structure.

Note that due to the casting process with high reproducibility, the contact parameters $\boldsymbol{k}_\mathrm{b}$ are assumed to be identical for all joints. The same simplification could be applied to the parameters $\boldsymbol{k}_\mathrm{s}, \boldsymbol{k}_\mathrm{v}$ and $\boldsymbol{k}_\mathrm{C}$ for all joints. However, the tubes/cables of the last actuator have to overcome considerably higher friction than those of the first actuator. Similarly, the tubes/cables may also influence the joint stiffness. Therefore, we identify different stiffness and friction parameters for each joint.

A three-step least-squares identification is conducted, whereby we choose $\dot{q}_\mathrm{C}{=}1\,^\circ/\mathrm{s}$ and $q_\mathrm{bt}{=}10^\circ$ heuristically. The identification dataset $\boldsymbol{D}_\mathrm{t}$ is split into three subsets: $\boldsymbol{D}_\mathrm{tI}$, $\boldsymbol{D}_\mathrm{tII}$, and $\boldsymbol{D}_\mathrm{tIII}$. Each subset is used to identify different parameters of (11). Such a multi-step identification is common to isolate the influence of the different static/dynamic terms and improve the identification result. The procedure is explained in the following.

3) Identification of Stiffness: The factors $\boldsymbol{k}_\mathrm{s}{=}[k_{\mathrm{s}1}\dots,k_{\mathrm{s}n}]^\mathrm{T}$ are identified using static data inside the soft boundaries: $\boldsymbol{D}_\mathrm{tI}{\subset}\boldsymbol{D}_\mathrm{t}$ with $|\dot{q}_i|{\le}\dot{q}_\mathrm{C}{\approx}0\,{\wedge}\,|q_i|{\le}q_\mathrm{bt}$. The dynamics equation simplifies to (9)-I for these data points, as velocity- and acceleration-dependent terms are assumed to be negligible, and contact torques are zero. This results in the parameter-linear form

$$\boldsymbol{\tau}_{\mathrm{I}j} = \boldsymbol{\tau}_j - \boldsymbol{g}_j = \boldsymbol{Q}_{\mathrm{I}j}\boldsymbol{k}_\mathrm{s} \tag{12}$$

for an arbitrary measurement j. Vertically stacking[4] $\boldsymbol{\tau}_{\mathrm{I}j}$ and $\boldsymbol{Q}_{\mathrm{I}j}{=}\mathrm{diag}(\boldsymbol{q}_j)$ for all measurements leads to $\boldsymbol{\tau}_\mathrm{I}$ and $\boldsymbol{Q}_\mathrm{I}$, respectively. The latent parameters $\boldsymbol{k}_\mathrm{s}{=}\boldsymbol{Q}_\mathrm{I}^\dagger\boldsymbol{\tau}_\mathrm{I}$ are obtained by Moore-Penrose pseudo-inversion ($\dagger$).

4) Identification of Friction: The friction parameters $\boldsymbol{k}_\mathrm{vC}{=}[k_{\mathrm{v}1}, k_{\mathrm{C}1},\dots,k_{\mathrm{v}n}, k_{\mathrm{C}n}]^\mathrm{T}$ are identified using dynamic data inside the soft boundaries: $\boldsymbol{D}_\mathrm{tII}{\subset}\boldsymbol{D}_\mathrm{t}$ with $|\dot{q}_i|{>}\dot{q}_\mathrm{C}{\wedge}|q_i|{\le}q_\mathrm{bt}$. The dynamics equation simplifies to (9)-II for these data points, which can be transformed into the parameter-linear form

$$\begin{aligned}\boldsymbol{\tau}_{\mathrm{II}j} &= \boldsymbol{\tau}_j - \boldsymbol{g}_j - \boldsymbol{s}_j - \boldsymbol{M}_j\ddot{\boldsymbol{q}}_j - \boldsymbol{c}_j \\ &= \underbrace{\begin{bmatrix} \dot{q}_{1j} & f_\mathrm{C}(\dot{q}_{1j}) & \boldsymbol{0} & \boldsymbol{0} \\ & & \ddots & \\ \boldsymbol{0} & \boldsymbol{0} & \dot{q}_{nj} & f_\mathrm{C}(\dot{q}_{nj}) \end{bmatrix}}_{\boldsymbol{Q}_{\mathrm{II}j}\in\mathbb{R}^{n\times 2n}} \boldsymbol{k}_\mathrm{vC}\end{aligned} \tag{13}$$

by using $f_\mathrm{C}(\dot{q}){=}\tanh(\dot{q}\pi/\dot{q}_\mathrm{C})$. Thereby, the stiffness torques $\boldsymbol{s}_j$ are computed with the *previously identified* parameters $\boldsymbol{k}_\mathrm{s}$. Similar to above, $\boldsymbol{\tau}_\mathrm{II}$ and $\boldsymbol{Q}_\mathrm{II}$ are obtained by vertically stacking[4] $\boldsymbol{\tau}_{\mathrm{II}j}$ and $\boldsymbol{Q}_{\mathrm{II}j}$, respectively. This enables the computation of the friction parameters $\boldsymbol{k}_\mathrm{vC}{=}\boldsymbol{Q}_\mathrm{II}^\dagger\boldsymbol{\tau}_\mathrm{II}$.

5) Identification of Contact Parameters: The factors $\boldsymbol{k}_\mathrm{b}$ are identified using data outside the soft boundaries $\boldsymbol{D}_\mathrm{tIII}{\subset}\boldsymbol{D}_\mathrm{t}$ with $|q_i|{>}q_\mathrm{bt}$. The entire dynamics equation (9) applies to these data points, which can be transformed into the parameter-linear form

$$\begin{aligned}\boldsymbol{\tau}_{\mathrm{III}j} &= \boldsymbol{\tau}_j - \boldsymbol{g}_j - \boldsymbol{s}_j - \boldsymbol{M}_j\ddot{\boldsymbol{q}}_j - \boldsymbol{c}_j - \boldsymbol{d}_j \\ &= \underbrace{\begin{bmatrix} \mathrm{sgn}(q_{1j})|\Delta q_{1j}|^{3/2} & \sqrt{|\Delta q_{1j}|}\dot{q}_{1j} \\ \vdots & \vdots \\ \mathrm{sgn}(q_{nj})|\Delta q_{nj}|^{3/2} & \sqrt{|\Delta q_{nj}|}\dot{q}_{nj} \end{bmatrix}}_{\boldsymbol{Q}_{\mathrm{III}j}\in\mathbb{R}^{n\times 2}} \boldsymbol{k}_\mathrm{b}\end{aligned} \tag{14}$$

using *previously identified* parameters of stiffness and friction. The parameters $\boldsymbol{k}_\mathrm{b}{=}\boldsymbol{Q}_\mathrm{III}^\dagger\boldsymbol{\tau}_\mathrm{III}$ are computed in analogy to above[4].

[4]Note that for each datapoint j, the dataset categorization (I, II, III) is done separately for each actuator i. The rows of $\boldsymbol{Q}_\diamond$ and $\boldsymbol{\tau}_\diamond$ are therefore deleted if the conditions for the respective dataset $\boldsymbol{D}_{\mathrm{t}\diamond}$ are not fulfilled. This ensures that all data is used and only occurs once in all datasets. Such separation of the dataset neglects the coupling between the different robot segments. If this simplification does not apply to other systems, suitable identification trajectories must be recorded.

6) Forward Prediction: The inverse dynamics (9) can be transformed into state-space form

$$\hat{\boldsymbol{x}}(t) = \boldsymbol{f}_{\mathrm{FP}\boldsymbol{\delta}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\delta}) \tag{15}$$

with additional domain input $\boldsymbol{\delta}$. We denote $\boldsymbol{x}{=}[\boldsymbol{q}^{\mathrm{T}}, \dot{\boldsymbol{q}}^{\mathrm{T}}]^{\mathrm{T}}{\in}\mathbb{R}^{2n}$ as state of the dynamical system and $\boldsymbol{u}{=}\boldsymbol{p}_{\mathrm{d}}{\in}\mathbb{R}^{2n}$ as system input. Thereby, $\boldsymbol{p}_{\mathrm{d}}{=}\boldsymbol{p}$ is assumed. This is valid due to the fast pressure control of the proportional valves so that the desired and measured pressures match closely with a time delay of $10{-}80\,\mathrm{ms}$. If this simplification does not hold, the pressure dynamics could be modeled via a simple first-order system.

Conventional numerical integration of (15) by using Runge-Kutta methods can be used to predict the evolution of the states. However, very small step sizes $\leq 100\,\mu\mathrm{s}$ are necessary for stable forward simulation of the system due to the stiff ODE. This is impractical for use in real-time nonlinear MPC not only due to the excessive computation time, which is further evaluated in Sec. V-C. Also, to enable online optimization during control, the prediction horizon is usually bounded to a few time steps. The fine discretization of the model would, therefore, lead to a very short prediction horizon, which considerably reduces the time available to solve an MPC problem. Thus, *a fast-to-evaluate surrogate model with large time steps is necessary for real-time control*.

C. Generalizable and Fast PINNs

Both first-principles (Sec. IV-B) and data-driven modeling (Sec. III-A) have disadvantages. The former has good generalizability. However, prediction with numerical integration is computationally expensive. The latter learns input-output relationships purely from real data, ignoring any physical principles. This leads to poor generalizability, as trained models are strongly overfitted to one (training) domain of the system. However, numerical integration with fine discretization is not necessary, as it enables fast inference with large time steps. We combine the advantages of both approaches in the following. For this purpose, physics-informed neural networks provide an excellent architecture.

The original approach for state-space modeling using PINNs (3) consists of a network with three inputs: time t, initial state $\boldsymbol{x}_0$, and input signal $\boldsymbol{u}_0$. It is trained on artificially sampled collocation points, for which the ODE does not need to be solved. Real-world data can additionally be used if the first-principles model is not accurate enough. According to [68], a trustworthy and reliable physics-informed model should be able to extrapolate to systems with different parameters, external forces, or boundary conditions while maintaining high accuracy. However, both PINC [10] and DD-PINN [11] represent the state space for one selected parameterization of the dynamical system, but not if this domain changes after training. For example, the NNs can be trained only for *one* domain (e.g., one particular mass at the end effector m_{e} and base orientation β_{g}). Instead, we would like to train a PINN that generalizes to *all* realistic domains during inference to avoid time-consuming data acquisition and retraining. For this, the domain $\boldsymbol{\delta}$ is defined as another input of the PINN, which can be any quantity of the first-principles model.

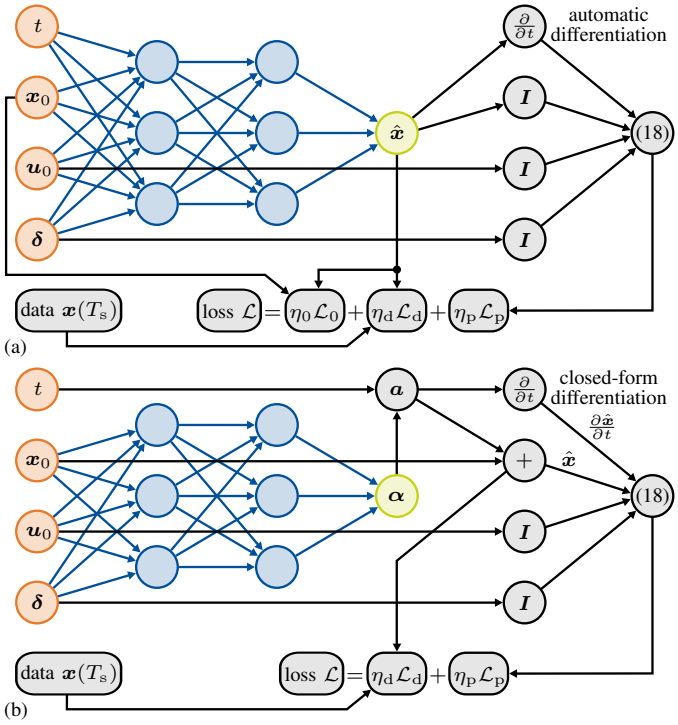


Figure 5. PINN structures with inputs in orange, feedforward network in blue with output in green. Both networks are extended by an additional domain input $\boldsymbol{\delta}$: (a) The PINC directly predicts the state $\hat{\boldsymbol{x}}$. During training, the network requires computationally expensive automatic differentiation for each collocation point (in each training epoch) and contains an additional loss term $\mathcal{L}_0$ for the initial condition. (b) The DD-PINN predicts the ansatz vector $\boldsymbol{\alpha}$ of an ansatz function $\boldsymbol{a}(\boldsymbol{\alpha}, t)$. The latter can be differentiated in closed form. Also, $\boldsymbol{a}(\boldsymbol{\alpha}, 0){\equiv}\boldsymbol{0}$ applies so that no initial-condition loss is necessary. Both drastically speed up the training time.

Similar to (15), the forward pass of the proposed PINN architecture results in

$$\hat{\boldsymbol{x}}(t) = \boldsymbol{f}_{\mathrm{PINN}\boldsymbol{\delta}}(t, \boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\delta}), \tag{16}$$

which is visualized in Fig. 5 for both PINC and DD-PINN. The feedforward NN consists of n_{h} hidden layers, each with n_{n} neurons and hyperbolic tangent activation function. Whereas the feedforward part of the PINC directly predicts $\hat{\boldsymbol{x}}(t)$, the ansatz vector $\boldsymbol{\alpha}$ (7) is computed within the DD-PINN, which is then used to calculate the state

$$\hat{\boldsymbol{x}}(t) = \boldsymbol{x}_0 + \boldsymbol{a}(\boldsymbol{\alpha}(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\delta}), t). \tag{17}$$

Collocation points are sampled in $\mathbb{R}^{1+4n+\nu}$ with user-specified boundaries[5] $\mathcal{T}{\subset}\mathbb{R}$, $\mathcal{X}_0{\subset}\mathbb{R}^{2n}$, $\mathcal{U}_0{\subset}\mathbb{R}^{2n}$ and $\mathcal{D}{\subset}\mathbb{R}^{\nu}$. In this way, *domains can be trained even if no real data is available for that domain*. During prediction, the current domain $\boldsymbol{\delta}$ can then be specified. We use latin hypercube sampling within the range ± 1 for sampling the scaled quantities t, $\boldsymbol{u}_0$, and $\boldsymbol{\delta}$. For the scaled states $\boldsymbol{x}_0$, we utilize a multivariate normal distribution with zero mean and standard deviation of 0.4. This importance sampling has the advantage that there are fewer collocation points at the broadly selected sampling boundaries,

[5]All inputs and outputs of the NN are scaled between -1 and 1 by using this user-specified minimum/maximum values. For the sake of clarity, we do not introduce new symbols for each variable since min-max scaling is straightforward and must be considered during implementation.

and, for example, more samples are taken to consider the high slope of the Coulomb friction term for $\dot{q}_i \approx 0$.

Note that we could also provide the RNN with an additional domain input, i.e., $\boldsymbol{f}_{\mathrm{RNN}\boldsymbol{\delta}}(\boldsymbol{h}_0, \boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\delta})$. However, this has no added value in the restrictive context of this work, as we only provide training data from one domain $\boldsymbol{\delta}_\mathrm{t}$. Therefore, the RNN would be trained with a permanently constant input $\boldsymbol{\delta}=\boldsymbol{\delta}_\mathrm{t}$. Purely data-based approaches are not able to learn the influence of the domain input here, so that it can be omitted. The domain input for the RNN would only be appropriate for training with datasets from different discrete domains ($\nu>1$). However, we do not believe that it is practical to record training data in all possible domains (cf. Sec. II-B). This, again, underlines the motivation for utilizing physical knowledge.

The total (multi-objective) loss $\mathcal{L}$ is composed of several terms ($\mathcal{L}_\mathrm{d}, \mathcal{L}_\mathrm{p}$ and $\mathcal{L}_0$), whose weights ($\eta_\mathrm{d}, \eta_\mathrm{p}$ and η_0) are calculated after the first training epoch (cf. Sec. IV-D). The mean squared error (MSE) between network *prediction* and *ground truth* is determined, whereby the determination of both quantities varies depending on the loss.

1) Physics Loss $\mathcal{L}_\mathrm{p}$: On collocation points, the residuum

$$\boldsymbol{r}_\mathrm{p} = \frac{\partial \hat{\boldsymbol{x}}}{\partial t} - \boldsymbol{f}_{\mathrm{FP}\boldsymbol{\delta}}(\hat{\boldsymbol{x}}, \boldsymbol{u}_0, \boldsymbol{\delta}) \tag{18}$$

is calculated with the modeled system dynamics (15), zero-order hold assumption for $\boldsymbol{u}_0$, and the time derivative of the predicted states $\frac{\partial \hat{\boldsymbol{x}}}{\partial t} = \frac{\partial}{\partial t}\boldsymbol{f}_{\mathrm{PINN}\boldsymbol{\delta}}$. The latter is done by means of automatic differentiation for the PINC, and can be computed in closed form for the DD-PINN via $\frac{\partial \hat{\boldsymbol{x}}}{\partial t} = \dot{\boldsymbol{a}}$ using (8).

2) Initial-Condition Loss $\mathcal{L}_0$: In order to train a PINN, it is essential to consider the governing initial/boundary conditions. For state-space modeling, this loss is defined for the initial condition $\boldsymbol{x}_0$, which must match the network output $\boldsymbol{f}_{\mathrm{PINN}\boldsymbol{\delta}}(0, \boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\delta})$. The loss is also computed on the sampled collocation points with time t set to zero. For the DD-PINN, this loss is not necessary since $\mathcal{L}_0 \equiv 0$ due to the choice of $\boldsymbol{a}$.

3) Data Loss $\mathcal{L}_\mathrm{d}$: This optional loss term is determined with existing real measurement data in the training domain $\boldsymbol{\delta}_\mathrm{t}$. Similar to black-box learning, the prediction $\hat{\boldsymbol{x}}(T_\mathrm{s}) = \boldsymbol{f}_{\mathrm{PINN}\boldsymbol{\delta}}(T_\mathrm{s}, \boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\delta}_\mathrm{t})$ is compared to the measured ground truth $\boldsymbol{x}(T_\mathrm{s})$ using the specified error metric.

To conclude, the PINN is trained for large timesteps similarly to the RNN. Further, synthetic collocation points in arbitrary domains are sampled during training, for which physical system knowledge is utilized. By means of a weighted, multi-objective loss, it is possible to integrate real system data. Note that there is only good generalization capability for quantities that are considered with the domain input $\boldsymbol{\delta}$. For example, if significantly stiffer bellows are used, the accuracy of the PINN (as for the FP model and RNN) will decrease. The disassembly of one soft actuator after training to an ASR with $n=4$ joints would also result in less accuracy for the PINN (as for the RNN). In contrast, the FP approach has the advantage that the dynamics model for $n=4$ can be symbolically generated and used with the identified parameters. The PINN would have to be retrained with the updated FP model.

Algorithm 1: Training of PINC or DD-PINN

In : $n_\mathrm{e}, n_\mathrm{s}, n_\mathrm{p}, n_0, n_\mathrm{b}, n_\mathrm{a}, n_\lambda, \lambda_0, \lambda_\mathrm{min}, \boldsymbol{X}_\mathrm{d}, \boldsymbol{Y}_\mathrm{d}$
Out: $\boldsymbol{w}$

$\boldsymbol{\eta}, \lambda, \mathcal{L}_0 \leftarrow \{[1,1,1]^\mathrm{T}, \lambda_0, 0\}$;
foreach *epoch* $\in [0, \dots, n_\mathrm{e}-1]$ **do**
  if *epoch* $\bmod n_\mathrm{s} = 0$ **then**
    $\boldsymbol{X}_\mathrm{p} \leftarrow$ Sample n_p points;
    if $n_\mathrm{a} = 0$ **then**
      // PINC
      $\boldsymbol{X}_0 \leftarrow$ Take n_0 points from $\boldsymbol{X}_\mathrm{p}$ and set $t = 0$;
      $\boldsymbol{Y}_0 \leftarrow \boldsymbol{x}_0$ from all samples $\boldsymbol{X}_0$, cf. (4);
      $\boldsymbol{X}, \boldsymbol{Y} \leftarrow$ Shuffle $[\boldsymbol{X}_\mathrm{d}, \boldsymbol{Y}_\mathrm{d}], [\boldsymbol{X}_\mathrm{p}, \boldsymbol{0}], [\boldsymbol{X}_0, \boldsymbol{Y}_0]$;
    else
      // DD-PINN
      $\boldsymbol{X}, \boldsymbol{Y} \leftarrow$ Shuffle $[\boldsymbol{X}_\mathrm{d}, \boldsymbol{Y}_\mathrm{d}], [\boldsymbol{X}_\mathrm{p}, \boldsymbol{0}]$;
    end
    $\boldsymbol{D}'_\mathrm{t}, \boldsymbol{D}'_\mathrm{v} \leftarrow$ Split $\boldsymbol{X}, \boldsymbol{Y}$ in batches of size n_b;
  end
  $\bar{\mathcal{L}}_\mathrm{v}, \bar{\boldsymbol{\mathcal{L}}}_\mathrm{t} \leftarrow \{0, \boldsymbol{0}\}$;
  foreach $\boldsymbol{D}' \in [\boldsymbol{D}'_\mathrm{t}, \boldsymbol{D}'_\mathrm{v}]$ **do**
    foreach $\boldsymbol{B} \in \boldsymbol{D}'$ *with b batches* **do**
      $\mathcal{L}_\mathrm{d} \leftarrow \mathrm{MSE}(\boldsymbol{Y}_{\mathrm{d}\boldsymbol{B}}, \hat{\boldsymbol{Y}}_{\mathrm{d}\boldsymbol{B}})$;
      $\mathcal{L}_\mathrm{p} \leftarrow \mathrm{MSE}(\boldsymbol{F}(\hat{\boldsymbol{Y}}_{\mathrm{p}\boldsymbol{B}}, \boldsymbol{X}_{\mathrm{p}\boldsymbol{B}}), \frac{\partial}{\partial t}\hat{\boldsymbol{Y}}_{\mathrm{p}\boldsymbol{B}})$;
      if $n_\mathrm{a} = 0$ **then**
        $\mathcal{L}_0 \leftarrow \mathrm{MSE}(\boldsymbol{Y}_{0\boldsymbol{B}}, \hat{\boldsymbol{Y}}_{0\boldsymbol{B}})$;
      end
      $\mathcal{L} \leftarrow [\mathcal{L}_\mathrm{d}, \mathcal{L}_\mathrm{p}, \mathcal{L}_0]\boldsymbol{\eta}$;
      if $\boldsymbol{D}' = \boldsymbol{D}'_\mathrm{t}$ **then**
        $\bar{\boldsymbol{\mathcal{L}}}_\mathrm{t} \leftarrow \bar{\boldsymbol{\mathcal{L}}}_\mathrm{t} + b^{-1}[\mathcal{L}_\mathrm{d}, \mathcal{L}_\mathrm{p}, \mathcal{L}_0]^\mathrm{T}$;
        $\boldsymbol{w} \leftarrow$ Optimize network weights with $\mathcal{L}, \lambda$;
      else
        $\bar{\mathcal{L}}_\mathrm{v} \leftarrow \bar{\mathcal{L}}_\mathrm{v} + b^{-1}\mathcal{L}$;
      end
      if *epoch* $= 0$ *and last batch of* $\boldsymbol{D}'_\mathrm{t}$ **then**
        if $n_\mathrm{a} = 0$ **then**
          $\boldsymbol{\eta} \leftarrow \max(\bar{\boldsymbol{\mathcal{L}}}_\mathrm{t})[1/\bar{\mathcal{L}}_\mathrm{td}, 1/\bar{\mathcal{L}}_\mathrm{tp}, 1/\bar{\mathcal{L}}_\mathrm{t0}]^\mathrm{T}$;
        else
          $\boldsymbol{\eta} \leftarrow \max(\bar{\boldsymbol{\mathcal{L}}}_\mathrm{t})[1/\bar{\mathcal{L}}_\mathrm{td}, 1/\bar{\mathcal{L}}_\mathrm{tp}, 1]^\mathrm{T}$;
        end
      end
    end
  end
  if *epoch* $> n_\lambda$ **then**
    $\lambda \leftarrow \mathrm{ReduceLROnPlateau}(\bar{\mathcal{L}}_\mathrm{v}, n_\lambda, \lambda_\mathrm{min})$;
  end
end

D. Implementation

For a comprehensive overview, the PINN training described in the previous section is provided in Algorithm 1 as pseudo-code. We implemented the training in PyTorch [69]. Necessary inputs for training a neural network with n_n neurons and n_h hidden layers are:

- n_e: Number of training epochs
- n_s: The collocation points are resampled all n_s epochs for faster convergence and to prevent overfitting
- n_p, n_0: Number of collocation/initial-condition points
- n_b: Number of collocation, initial-condition and data points in one batch
- n_a: Number of ansatz functions of DD-PINN ($n_\mathrm{a}=0$ implies PINC training)

- n_λ, λ_0, $\lambda_{\min}$: The initial learning rate λ_0 is halved when there is no improvement of the mean validation loss $\bar{\mathcal{L}}_\mathrm{v}$ after n_λ epochs until $\lambda_{\min}$ is reached (line 39)
- $\boldsymbol{X}_\mathrm{d}$, $\boldsymbol{Y}_\mathrm{d}$: Real measurement data from $\boldsymbol{D}_\mathrm{t}$ (optional)

In general, $\boldsymbol{X}_\diamond$ describes the input data with $n_\diamond$ points for the respective loss terms. Associated with this, $\boldsymbol{Y}_\diamond$ is the ground-truth output, which is compared to the network predictions $\hat{\boldsymbol{Y}}_\diamond$. A mini-batch training of a given feedforward network is conducted. After resampling at defined intervals, all data points are split into batches such that each batch consists of points for all losses (lines 3–13). There is no ground truth $\boldsymbol{x}(t)$ necessary when calculating the physics loss. This is indicated in line 8 (or 10) with $\boldsymbol{Y}_\mathrm{p}{=}\boldsymbol{0}$. With a $70/30$ split, the training dataset[6] $\boldsymbol{D}'_\mathrm{t}$ and the validation dataset $\boldsymbol{D}'_\mathrm{v}$ are formed.

For each batch $\boldsymbol{B}$ of the two datasets, the losses are calculated (lines 17–22). The physics loss is determined with modeled dynamics $\boldsymbol{f}_{\mathrm{FP}\boldsymbol{\delta}}(\hat{\boldsymbol{x}}, \boldsymbol{u}_0, \boldsymbol{\delta})$, which is denoted with the function $\boldsymbol{F}(\cdot,\cdot)$ for the entire batch (line 18). Finally, the weights of the network $\boldsymbol{w}$ are updated via the Adam optimizer using the current training loss and the current learning rate.

At the beginning, the weighting factors $\boldsymbol{\eta}{=}[\eta_\mathrm{d}, \eta_\mathrm{p}, \eta_0]^\mathrm{T}$ are initialized to ones. After the first epoch, these are adjusted using the mean losses during all training batches $\bar{\boldsymbol{\mathcal{L}}}_\mathrm{t}$ (line 31 or 33). With these scaling factors, the loss weighting is done during the remaining epochs.

E. Domain Knowledge in Practice

The PINN presentation ends with a brief note regarding the main requirement of our approach: the knowledge of the current domain $\boldsymbol{\delta}$. Such a requirement is also used as a basis in comparable work. For example, various payloads are known to the controller a priori in [36]. The model in [60] also receives the load as input. For easily measurable quantities, these can be determined online in the application, e.g., measurement of the base inclination using an accelerometer. Alternatively, unmeasurable quantities could be estimated online. To this end, the proposed PINN architecture with the domain as input is ideally suited due to the high prediction speed.

F. Real-Time Application: Nonlinear MPC with PINNs

As one possible PINN application, a nonlinear model predictive control of the soft robot is realized. Since there is a high demand for prediction speed, the benefits of fast and accurate PINNs can be illustrated here. Besides considerably improved generalizability of PINNs compared to RNNs, the simple structure of a feedforward NN is another advantage of PINNs. MPC with RNNs requires the correct initialization of the non-measurable hidden states, which complicates their use in MPC [32].

The proposed control architecture is illustrated in Fig. 6. MPC uses the discrete model of the system to predict the behavior for a prediction horizon of m time steps. The MPC solver searches for an input trajectory to minimize a user-defined cost function within this prediction horizon.

[6]We use the $\boldsymbol{D}'_\diamond$ to denote the datasets during PINN training, which are not equal to the measured dataset in the training domain $\boldsymbol{D}_\mathrm{t}$.

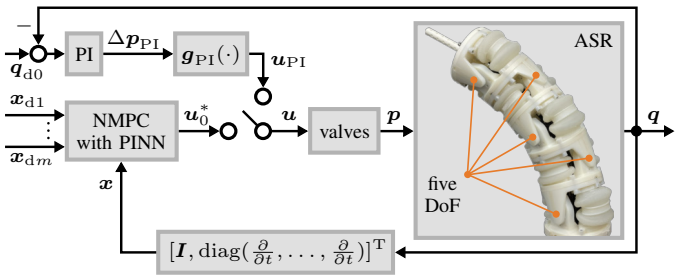


Figure 6. Control architecture: The nonlinear MPC uses the PINN as the dynamics model. Instead, a PI controller [65] is used in this work for comparison. Changing between the two strategies is visualized with the switch.

Only the first input signal $\boldsymbol{u}_0^*$ of this optimized input trajectory $[\boldsymbol{u}_0^*, \ldots, \boldsymbol{u}_{m-1}^*]{\in}\mathbb{R}^{2n\times m}$ is applied to the real system, and the optimization is solved again with current measured states $\boldsymbol{x}$ and new desired states $[\boldsymbol{x}_{\mathrm{d}1}, \ldots, \boldsymbol{x}_{\mathrm{d}m}]{\in}\mathbb{R}^{2n\times m}$ with $\boldsymbol{x}_{\mathrm{d}k}{=}[\boldsymbol{q}_{\mathrm{d}k}^\mathrm{T}, \dot{\boldsymbol{q}}_{\mathrm{d}k}^\mathrm{T}]^\mathrm{T}$ for m future time steps. To improve the speed of the solver, a control horizon of length one is selected. This effectively means that the input is kept constant throughout the prediction horizon so that $\boldsymbol{u}_0{=}\boldsymbol{u}_k \forall k$ applies. Therefore, only one input $\boldsymbol{u}_0$ must be optimized. The optimization problem is formulated as

$$\begin{aligned}\underset{\boldsymbol{u}_0}{\text{minimize}} \sum_{k=1}^{m-1} \big(\|\boldsymbol{q}_{\mathrm{d}k} - \hat{\boldsymbol{q}}_k\|^2_{\boldsymbol{Q}_{\mathrm{s}q}} + \|\dot{\boldsymbol{q}}_{\mathrm{d}k} - \hat{\dot{\boldsymbol{q}}}_k\|^2_{\boldsymbol{Q}_{\mathrm{s}\dot{q}}}\big) + \\ \|\boldsymbol{q}_{\mathrm{d}m} - \hat{\boldsymbol{q}}_m\|^2_{\boldsymbol{Q}_{\mathrm{t}q}} + \|\dot{\boldsymbol{q}}_{\mathrm{d}m} - \hat{\dot{\boldsymbol{q}}}_m\|^2_{\boldsymbol{Q}_{\mathrm{t}\dot{q}}} + \sum_{k=0}^{m-1} \|\boldsymbol{u}_k\|^2_{\boldsymbol{R}_\mathrm{s}} \end{aligned} \tag{19}$$

subject to[7] $\hat{\boldsymbol{x}}_{k+1}{=}\boldsymbol{f}_{\mathrm{PINN}\boldsymbol{\delta}}(T_\mathrm{s}, \hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{\delta})$, $\boldsymbol{u}_{\min}{\leq}\boldsymbol{u}_k{\leq}\boldsymbol{u}_{\max}$, and $\hat{\boldsymbol{x}}_0$ obtained from measurements for each MPC cycle.

Since the PINN uses scaled inputs/outputs, the entire optimization problem is formulated with scaled (unitless) quantities. The diagonal weighting matrices for the stage costs of the positions $\boldsymbol{Q}_{\mathrm{s}q}$, velocities $\boldsymbol{Q}_{\mathrm{s}\dot{q}}$ and inputs $\boldsymbol{R}_\mathrm{s}$, and for the terminal cost of the positions $\boldsymbol{Q}_{\mathrm{t}q}$ and velocities $\boldsymbol{Q}_{\mathrm{t}\dot{q}}$ consist of constant diagonal entries $Q_{\mathrm{s}q}$, $Q_{\mathrm{s}\dot{q}}$, R_s, $Q_{\mathrm{t}q}$, and $Q_{\mathrm{t}\dot{q}}$. The input limits are $\boldsymbol{u}_{\min}$ and $\boldsymbol{u}_{\max}$, which can be obtained by min-max scaling of the system-specific limits with a pressure range of $0{-}0.7\,\mathrm{bar}$. The defined pressure limits inherently eliminate any risk of system damage. Consequently, imposing state constraints (on position $\boldsymbol{q}$ or velocity $\dot{\boldsymbol{q}}$) would offer no practical benefit in this application. Instead, it would increase the computational burden of the online optimization process. We, therefore, refrain from imposing state constraints in the MPC. The nonlinear MPC problem was implemented with CasADi [70] using the interior-point method.

For a comparison, we use the PI controller from [65] with weighting matrices ($\boldsymbol{Q}_\mathrm{P}$ and $\boldsymbol{Q}_\mathrm{I}$), anti-windup and output saturation. This outputs the desired pressure differences $\Delta\boldsymbol{p}_\mathrm{PI}{\in}\mathbb{R}^n$ of all bellows pairs, which is mapped to each antagonist and agonist via

$$\begin{aligned}\boldsymbol{u}_\mathrm{PI} &= \boldsymbol{g}_\mathrm{PI}(\Delta\boldsymbol{p}_\mathrm{PI}) \\ &= \bar{\boldsymbol{p}} + [\Delta p_{\mathrm{PI}1}, -\Delta p_{\mathrm{PI}1}, \ldots, \Delta p_{\mathrm{PI}n}, -\Delta p_{\mathrm{PI}n}]^\mathrm{T}/2\end{aligned} \tag{20}$$

[7]For the sake of clarity, we use the index notation $\hat{\boldsymbol{x}}_k{=}\hat{\boldsymbol{x}}(kT_\mathrm{s})$.

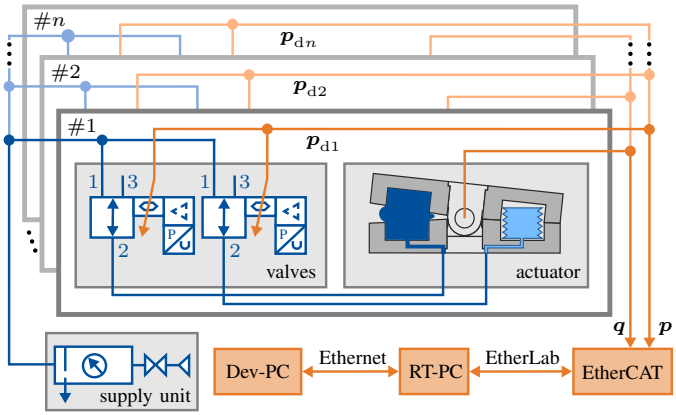


Figure 7. Test-bench architecture: Orange color represents communication elements and blue color represents pneumatic components.

with $\bar{\boldsymbol{p}} = \frac{p_{\max}}{2}[1,\dots,1]^{\mathrm{T}}$.

V. Experiments

After describing the pneumatic test bench (Sec. V-A), the identification and HPO results are presented (Sec. V-B). All these optimized (first-principles, hybrid, and black-box) models are then experimentally validated (Sec. V-C). Afterwards, experiments on learning-based MPC are conducted with the proposed PINN (Sec. V-D). The section ends with an overall discussion of the experiments (Sec. V-E).

A. Test Bench

The used test bench was presented in [65] and is only briefly described below. The main structure is illustrated in Fig. 7.

1) Components: The pneumatic system consists of a supply unit (pressure supply, shut-off valve, and filter regulator) and three-way proportional piezo valves (Festo VEAA-B-3-D2-F-V1-1R1, resolution: 5 mbar) with integrated pressure control for each bellows. Industrial compressors generate the compressed air centrally with negligible pressure fluctuations. Each actuator is equipped with a Hall encoder (Megatron ETA25K, resolution: 0.09°).

2) Communication: The EtherCAT protocol is used with the corresponding open-source tool EtherLab, which was modified with an external-mode patch and a shared-memory real-time interface[8] for Matlab/Simulink. This enables reading in or setting several values (current pressures $\boldsymbol{p}$, joint angles $\boldsymbol{q}$, and desired pressures $\boldsymbol{p}_{\mathrm{d}}$) with a cycle time of 1 ms via the EtherCAT real-time bus with input and output terminals (Beckhoff EL3702 and EL4102). To avoid damage, an output saturation of the desired pressures $\boldsymbol{p}_{\mathrm{d}}$ is implemented with limits $0{-}p_{\max}$.

Data can be logged, and settings can be altered during runtime with the development computer (Dev-PC, 3.6 GHz Intel Core i7-12700K CPU with 16 GB RAM) using Matlab/Simulink 2018b. The compiled model is executed on the real-time computer (RT-PC, 4.7 GHz Intel Core i7-12700K CPU with 16 GB RAM).

[8]https://github.com/SchapplM/etherlab-examples

3) MPC Implementation: CasADi was integrated using its fast C++ API for real-time control instead of using Python. The MPC was decoupled from the real-time system using a ROS service. Once the MPC problem is solved, the modified $\boldsymbol{u}_0^*$ is used. Therefore, no fixed MPC frequency was specified, but rather, the fastest possible frequency was set automatically depending on the choice of MPC parameters. The trained PINN is exported from PyTorch and manually recreated in CasADi. Due to the use of the widespread ROS framework, this implementation[1] can also be beneficial for other applications.

B. Model Identification/Learning

All approaches receive an identical dataset of 15 min for identification/learning in the training domain $\boldsymbol{\delta}_{\mathrm{t}}{=}[0\,\mathrm{g}, 0°]^{\mathrm{T}}$. The data was logged at a frequency of 1 kHz and then downsampled to 50 Hz. Random pressure combinations limited to 0.7 bar with a linear transition of 1 s were applied to each bellows, with each combination being held for 3 s.

1) Parameter Identification: With the chosen parameters ($\dot{q}_{\mathrm{C}}{=}1\,°/\mathrm{s}$ and $q_{\mathrm{bt}}{=}10°$), the datasets $\boldsymbol{D}_{\mathrm{tI}}$, $\boldsymbol{D}_{\mathrm{tII}}$, and $\boldsymbol{D}_{\mathrm{tIII}}$ comprise 40 %, 17 %, and 43 % of all datapoints, respectively. Various combinations of $\dot{q}_{\mathrm{C}}$ and q_{bt} were tested, whereby the identified parameters only changed marginally.

The results are presented in Table I. For both the stiffness $\boldsymbol{k}_{\mathrm{s}}$ and the friction parameters $\boldsymbol{k}_{\mathrm{v}}$ and $\boldsymbol{k}_{\mathrm{C}}$, a trend can be seen that these increase slightly towards the distal end. As already mentioned, this is attributed to the routing of the cables and tubes in the robot body. The parameters shown remain unchanged after this offline identification and serve as the training foundation for the PINNs. To illustrate the effects of the various parameters, Fig. 8 shows the characteristics of different positions and velocities of an exemplary joint after identification.

2) PINC vs. DD-PINN: To first get an impression of the training, the proposed PINC and DD-PINN with domain input $\boldsymbol{\delta}$ were each trained with identical parameters ($n_{\mathrm{s}}{=}250$, $n_{\mathrm{p}}{=}100\,000$, $n_{\mathrm{b}}{=}512$, $n_{\lambda}{=}50$, $\lambda_0{=}5{\times}10^{-4}$, $\lambda_{\min}{=}5{\times}10^{-5}$). Both trainings were conducted on one core with 8 GB RAM of a computing cluster (2.6 GHz Intel Xeon Gold 6442Y CPU). The PINC receives $n_0{=}0.2n_{\mathrm{p}}$ initial-condition points. In contrast, the DD-PINN does not require these points, as $\mathcal{L}_{\mathrm{v}0}{\equiv}\mathcal{L}_{\mathrm{t}0}{\equiv}0$ applies due to the structural property of the used ansatz function (6) with $n_{\mathrm{a}}{=}50$. A simple feedforward NN with $n_{\mathrm{n}}{=}100$ neurons and $n_{\mathrm{h}}{=}2$ hidden layers is trained. The

Table I
Identified parameters $\boldsymbol{k}$ of (9)

param.	unit	$i{=}1$	$i{=}2$	$i{=}3$	$i{=}4$	$i{=}5$
$k_{\mathrm{s}i}$	Nm/°	0.035	0.034	0.043	0.035	0.041
$k_{\mathrm{v}i}$	Nms/°	0.008	0.008	0.010	0.011	0.011
$k_{\mathrm{C}i}$	N m	0.171	0.214	0.233	0.204	0.232
k_{bs}	$\mathrm{Nm}/(°)^{3/2}$	0.010				
k_{bd}	$\mathrm{Nms}/(°)^{3/2}$	0.005				

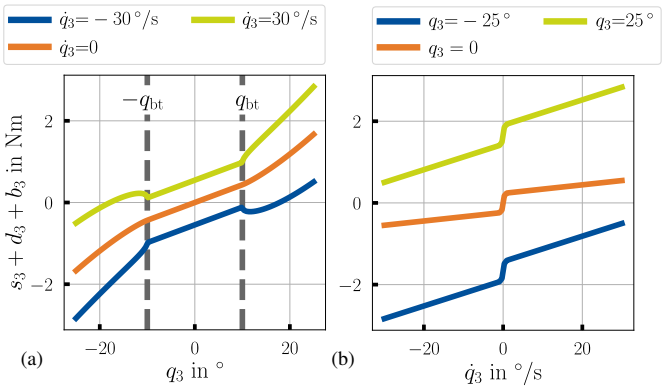


Figure 8. Stiffness, damping, and contact characteristics of joint i=3 with identified parameters for a simulative variation of (a) position q_3 with a kink at the threshold of the soft boundaries $\pm q_{bt}$ and (b) velocity $\dot{q}_3$. Curves illustrate the identified parameters from joint i=3, and are qualitatively similar for the other joints.

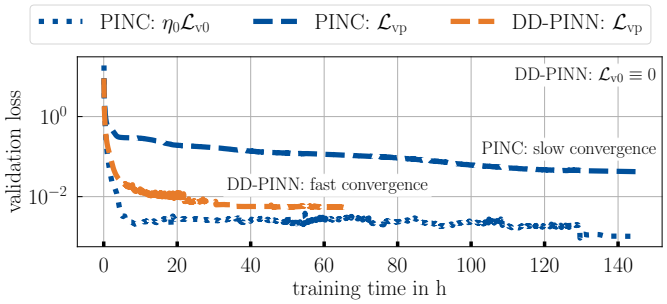


Figure 9. Training convergence of two PINN architectures (PINC and DD-PINN), which are adapted for our system and extended to enable additional domain input $\boldsymbol{\delta}$. The validation physics losses $\mathcal{L}_{vp}$ and validation initial-condition loss $\mathcal{L}_{v0}$ are plotted over training time. Both networks are trained on the same hardware with identical parameters. For the PINC, two competing loss terms must be minimized, whereby the initial-condition loss is weighted with η_0 (η_p=1). The DD-PINN has a considerably improved convergence, which is indicated by the *lower physics loss*. Also, the initial-condition loss from the DD-PINN ($\mathcal{L}_{v0}\equiv$0) is always less than PINC's $\mathcal{L}_{v0}$. In order to also use the PINC as a reference model in this work, it was trained for a considerably longer duration of 144 h (six days). Even after this time, convergence is not yet complete.

user-specified boundaries are set to

$$\begin{aligned} \mathcal{T} &= [0, \kappa T_s], \\ \mathcal{X}_0 &= [-\kappa q_{max}, \kappa q_{max}]^n \times [-\kappa \dot{q}_{max}, \kappa \dot{q}_{max}]^n, \\ \mathcal{U}_0 &= [0, \kappa p_{max}]^{2n} \text{ and} \\ \mathcal{D} &= [0, \kappa m_{emax}] \times [0, \kappa \beta_{gmax}] \end{aligned} \tag{21}$$

with q_{max}=25°, $\dot{q}_{max}$=30 °/s, p_{max}=0.7 bar, m_{emax}=200 g and β_{gmax}=90°. The limits are system-specific, and the factor κ=1.25 is used to train for slightly larger ranges, also done in [51]. We chose the sample time T_s=20 ms, which is further discussed in Sec. V-B3.

Note that no real data points ($n_d \equiv 0$) are used for all PINNs in this work, similar to [10], [11]. In our case, adding data points has no advantage due to the high accuracy of the first-principles model (cf. Sec. V-C2), which is *already identified with the real data* from the training domain. Moreover, this would slow down and complicate the training even further due to the additional loss term.

The training progress for both networks is illustrated in Fig. 9. It can be seen that the DD-PINN converges much faster for two main reasons. First, no initial-condition loss needs to be minimized. Second, $\frac{\partial \hat{\boldsymbol{x}}}{\partial t}$ is calculated in closed form for all n_p collocation points so that no computationally expensive automatic differentiation is required. The latter considerably influences the average duration of one training epoch $\bar{T}_e$, allowing it to be reduced by 38.8 % (PINC: $\bar{T}_e$=260.9 s, DD-PINN: $\bar{T}_e$=159.7 s). Thus, the DD-PINN was trained for 1500 epochs, while the PINC could only be trained for 919 epochs within the recorded training time of approx. 67 h (2.8 days). A systematic HPO can be enabled in a reasonable time due to this considerably faster convergence of the DD-PINN. This is presented in the following.

3) Hyperparameter Optimization: Compared to RNNs, the training of PINNs is very time-consuming and requires considerable resources of computing hardware. We suspect that this is the main reason why systematic hyperparameter optimizations of PINNs are relatively rare in the state of the art. This is in line with a recent overview [48], which declares such HPO as an open challenge. However, since many hyperparameters significantly influence the network performance, we perform an HPO. For this purpose, the computing cluster from the previous section was used, which allows several DD-PINNs to be trained in parallel.

Before an HPO can be carried out, the most crucial training parameters must first be defined. For the DD-PINN, these are n_n, n_h, λ_0 and n_a. All other parameters are taken from the previous section. At this point, it must be noted that T_s is also a crucial parameter for the PINN performance. For large sampling times T_s, the prediction speed is improved, but the accuracy degrades. The opposite is true for a small T_s. This tradeoff was already highlighted in [71] for the PINC and applies analogously to the DD-PINN. Since the physics loss also shows a tendency to decrease as T_s decreases, integration into an HPO with ASHA is not recommended. It would always tend towards a very small T_s without considering the prediction speed. This parameter was therefore tuned iteratively, and a sample time T_s=20 ms proved to be suitable. The later use in the MPC was also taken into account here, as the sample times must not be too small. Otherwise, there would be problems similar to those with the FP model (cf. Sec. IV-B).

During the HPO, 35 networks are trained simultaneously, with a total of 100 trials being examined. As PINN training converges slowly, ASHA's grace period is set to 500 epochs with a reduction factor of 2. To keep the computation time reasonable, only n_e=1000 epochs are trained during the HPO. The best trial can be further trained afterward until n_e=1500.

The results are visualized in Fig. 10(a). It can be seen that the systematic HPO can considerably minimize the validation loss. Note that with such an HPO, the limits of the hyperparameters must be adjusted iteratively so that no optimal parameters lie at their boundaries. Otherwise, the true optimum of the parameter could be outside the limits. This iterative process, in combination with the HPO duration of approx. six

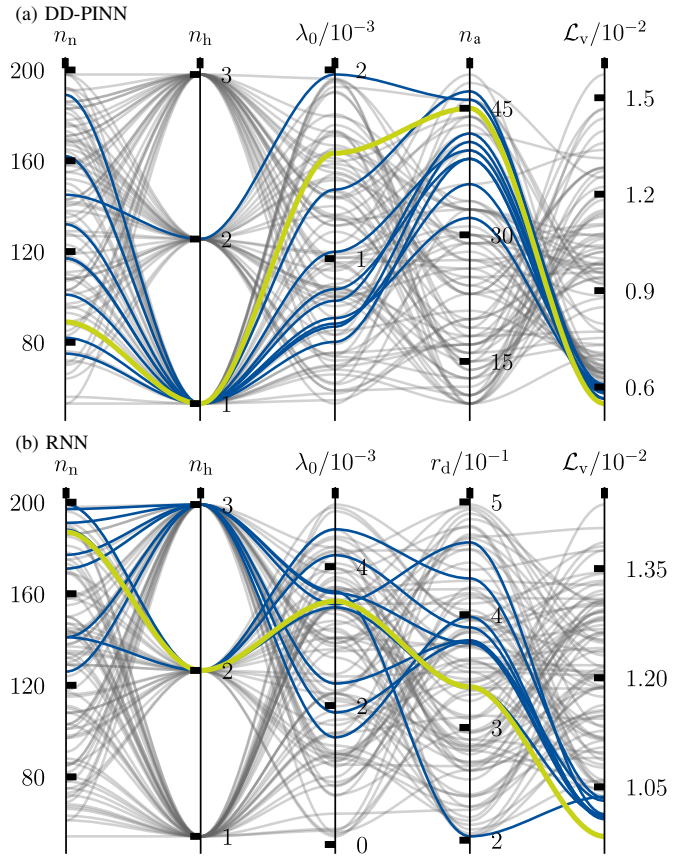


Figure 10. Results of hyperparameter optimization with 100 trials each: Poorly performing trials are shown in gray, the best ten in blue, and the best one in green. The validation loss $\mathcal{L}_\mathrm{v}$ is considerably reduced by systematically tuning the networks' hyperparameters. (a) The optimization of the most important DD-PINN parameters (n_n, n_h, λ_0 and n_a) took approx. six days due to the long training times on the specified cluster hardware. (b) The optimization of the most important RNN parameters (n_n, n_h, λ_0 and r_d) took $<14\,\mathrm{h}$ on the specified cluster hardware with minor performance.

days, therefore, requires a substantial computation and time effort.

The HPO of the RNN is performed similarly, and the results are shown in Fig. 10(b). Since RNNs converge much faster, n_e=300, n_λ=10 are chosen, and ASHA's grace period is set to 100 epochs. The DD-PINN-specific parameter n_a is replaced by the dropout rate r_d as an important training parameter of the RNN. Although significantly inferior cluster hardware (2.3 GHz Intel Cascade Lake Xeon Gold 6230N CPU) was used, only $<14\,\mathrm{h}$ is required for the HPO of the RNNs.

C. First Principles vs. PINNs vs. Black Box

The following experiments underline our *first and second claims* regarding prediction speed and generalizability, which are illustrated in Fig. 1(a). As the model base, we use the identified first-principles model, as well as the DD-PINN and RNN (both hyperparameter-optimized) from the previous section. As already mentioned, the PINC training is too time-consuming for an HPO. In order to also have a PINC reference model, we trained the network even further. This is also shown in Fig. 9. In total, the PINC training of the single configuration took 144 h (six days).

1) Prediction Speed: For a fair comparison of the prediction times, the physics-driven state-space model (15) and the forward passes of the RNN (1) and PINNs (16) were reimplemented and mex-compiled in Matlab R2023a. A 2.5 GHz Intel Core i5-10300H CPU with 16 GB of RAM running Windows was used for evaluation. The explicit Euler method and the Runge-Kutta fourth-order method (RK4) were used for the numerical integration of the physical model. These are often used in the MPC as integrators due to the constant step size.

The results for an exemplary 100 s trajectory of the input signals are listed in Table II. The time for each prediction horizon of T_s=20 ms was measured for all methods. The statistical data (mean, maximum and minimum), therefore, refer to 5000 time measurements, whereby a different number of function calls n_calls are required per measurement depending on the method.

Due to the stiff ODE (15), fine step sizes (Euler: 20 µs, RK4: 100 µs) are necessary for robust forward simulation of the system. In contrast, only *one* function call of the networks' forward passes is required for a horizon of T_s. This demonstrates one key advantage of using model-learning approaches. On average, the DD-PINN is significantly faster than numerical integration using explicit Euler or RK4 by a factor of 467 and 377, respectively. Due to the simpler network structure, the DD-PINN is also slightly faster than the RNN, which is also presented in Table II. The PINC is slightly faster than the DD-PINN, as the PINC's forward pass is simpler without the ansatz function used.

With regard to a real-time nonlinear MPC, the results show that the numerical integration of the physics-driven state-space model is unsuitable. The required fine temporal discretization of the physics model and, thus, the high number of function calls is problematic. The number of discrete time steps m in the prediction horizon of the MPC influences the solution times considerably and is, therefore, usually set to a few time steps, e.g., m=5. This would lead to very short prediction horizons (Euler: 100 µs, RK4: 500 µs), which results in difficulties such as aggressive MPC actions. Online optimization cannot be solved in such a short time frame, as solver frequencies of 2−10 kHz would be required. Even if we ignore this aspect, and use constant input signals for long segments of the prediction horizon — the long prediction times prevent the realization of fast real-time applications.

2) Generalizability: For all test datasets, random pressure combinations limited to 0.7 bar with a linear transition of 1 s were applied to each bellows. In contrast to the training data, each combination is held for only 1 s, which leads to more

Table II
COMPUTATION TIME FOR PREDICTING A HORIZON OF T_s = 20 ms. STATISTICS WERE COMPUTED FOR A 100 s INPUT TRAJECTORY.

method	mean/ms	max/ms	min/ms	n_calls
DD-PINN	0.04	0.30	0.03	**1**
PINC	**0.02**	**0.25**	**0.01**	**1**
RNN	0.08	1.46	0.05	**1**
Euler	18.67	35.74	17.37	1000
RK4	15.07	27.96	13.98	200

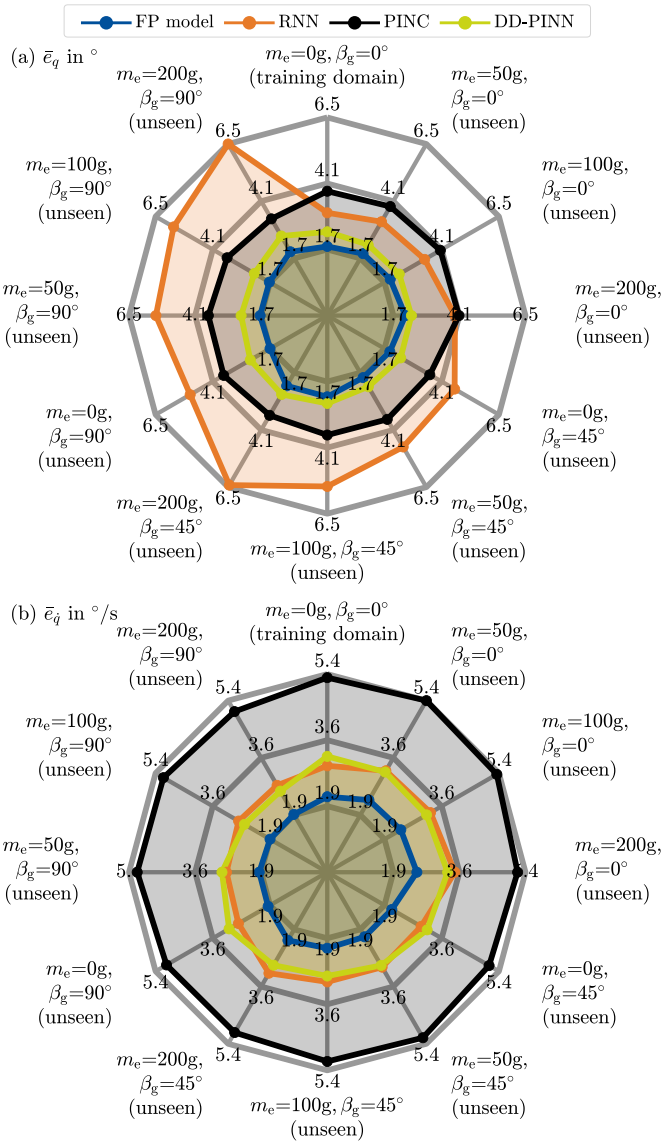


Figure 11. Accuracy of FP model, RNN, and PINNs (PINC and DD-PINN) on the training domain $\boldsymbol{\delta}_t=[0\,\mathrm{g}, 0°]^T$ and eleven unseen test domains $\boldsymbol{\delta}\neq\boldsymbol{\delta}_t$ with 10 min duration of each test dataset. The ODE of the FP model is numerically integrated with an advanced implicit solver to achieve results of the highest possible accuracy. (a) Mean absolute error (MAE) $\bar{e}_q$ between true and predicted positions averaged over all n joints. The DD-PINN is only marginally inferior to the FP model, whereas the RNN has large errors due to poor generalization. The PINC shows better generalizability than the RNN. However, the PINC is less accurate than the DD-PINN, although it has been trained considerably longer. (b) MAE $\bar{e}_{\dot{q}}$ between true and predicted velocities averaged over all n joints. DD-PINN and RNN are only marginally less accurate than the FP model, whereby the error of the RNN increases with higher mass m_e due to poor generalization. Again, the PINC does not show satisfactory accuracy.

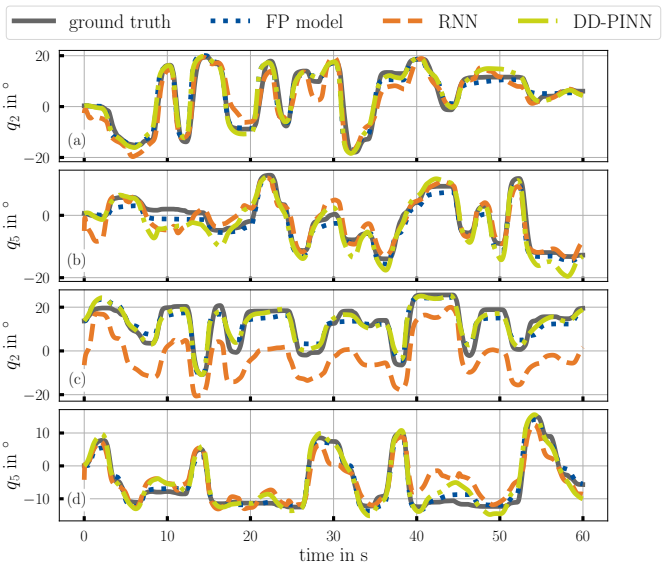


Figure 12. Prediction results for q_2 and q_5 on test datasets in (a)–(b) training domain $\boldsymbol{\delta}=\boldsymbol{\delta}_t=[0\,\mathrm{g}, 0°]^T$ and (c)–(d) unseen test domain $\boldsymbol{\delta}=[100\,\mathrm{g}, 45°]^T$. Within the unseen test domain, the dynamics of joint i=2 are mainly changed due to gravity. This is taken into account by the FP model and the DD-PINN due to the exploited physical knowledge. During training, real-world data in this test domain was not available. Therefore, large deviations occur with the RNN due to poor generalization.

dynamic motions.

Many publications only investigate a generalization for changed input trajectories, for which this would already be sufficient. However, we also want to investigate the generalizability to changes in system dynamics after training-data acquisition. For this purpose, test datasets with a duration of 10 min for twelve different domains $\boldsymbol{\delta}\in[0\,\mathrm{g}, 50\,\mathrm{g}, 100\,\mathrm{g}, 200\,\mathrm{g}]\times[0°, 45°, 90°]$ were recorded. Note that an additional mass of m_e=200 g is approx. 20 % of the total mass of the robot, and therefore, represents a considerable system change. In order to also investigate possible bellows changes over time, the test datasets were recorded *more than two months later* than the training dataset. The robot continued to operate during this time, which required bellows to be replaced due to cracks. We believe this is important because the identified parameters listed in Table I are fixed and serve as the basis for training the PINN.

The results for all test datasets are visualized in Fig. 11. To have a first-principles baseline with the highest possible accuracy, an implicit Runge-Kutta method (Radau IIA of order five [72]) is used for numerical integration. This is more advanced than explicit Euler or RK4, and also has the advantage that no discrete step size has to be chosen/tuned.

Fig. 11(a) demonstrates the advantage of the PINNs over the RNN. The more changes are present after training, the larger the RNN error becomes: For m_e=200 g and $\beta_g\in[45°, 90°]$, this position error is more than twice as large compared to the performance in the training domain $\boldsymbol{\delta}_t$. In contrast, our proposed DD-PINN consistently achieves an accuracy that is only slightly worse than the FP model, which has to be integrated with a substantial time effort. Even in the training domain, the DD-PINN achieves a lower position error than the RNN. Also, the DD-PINN is considerably more accurate than the PINC. Although the PINC was trained for six days, it has obviously not yet finally converged.

There are somewhat different results on the velocity level, which are shown in Fig. 11(b). Since the base orientation β_g only has a minor influence on the velocity, the poor generalizability of the RNN is less present. However, the mass m_e

Table III
TRACKING ERRORS AND NMPC FREQUENCY FOR CONTROL EXPERIMENTS IN DIFFERENT DOMAINS $\boldsymbol{\delta}=[m_\mathrm{e}, \beta_\mathrm{g}]^\mathrm{T}$ WITH 40 s DURATION EACH.

$\beta_\mathrm{g}/°$	m_e/g	controller	$\bar{e}_q/°$	$f_\mathrm{NMPC}/\mathrm{Hz}$
0	0	DD-PINN+NMPC	**2.60**	46.4
		PI [65]	3.04	–
	200	DD-PINN+NMPC	**2.29**	46.8
		PI [65]	3.23	–
45	0	DD-PINN+NMPC	**2.88**	47.6
		PI [65]	3.20	–
	200	DD-PINN+NMPC	**2.80**	46.3
		PI [65]	3.22	–
90	0	DD-PINN+NMPC	**2.66**	47.7
		PI [65]	2.93	–
	200	DD-PINN+NMPC	**2.98**	46.8
		PI [65]	3.36	–

influences the velocity. Therefore, at higher inertia, a higher prediction error of the RNN can be seen.

The PINC clearly shows the worst accuracy at the velocity level, which can be explained by the suboptimal convergence. A systematic HPO of the PINC could find better network configurations, but is not realistic due to the excessive training time. Since the DD-PINN has a comparable prediction speed with considerably better accuracy compared to the PINC, it is used for the rest of this work.

To elaborate further, Fig. 12 shows prediction results for a short excerpt of tests in two different domains. First, the RNN often shows poor accuracy at the beginning because the unmeasurable hidden states have to be initialized first. Moreover, it can be seen that the prediction of q_2, in particular, is worse for new domains, as this is most affected by the system change. Such disadvantages do not exist with the DD-PINN.

D. Control Results

In final control experiments with the soft robot, the *third claim* is confirmed. It should first be mentioned that all controller gains were manually tuned once in the training domain and then remained unchanged. The PI gains are set to $\boldsymbol{Q}_\mathrm{P}=\mathrm{diag}(40, 60, 110, 110, 100)\mathrm{mbar}/°$ and $\boldsymbol{Q}_\mathrm{I}=\mathrm{diag}(40, 40, 80, 60, 110)\mathrm{mbar}/(°\mathrm{s})$. For the nonlinear MPC problem, a prediction horizon with $m=3$ steps was chosen and the unitless gains were tuned to $Q_{\mathrm{s}q}=Q_{\mathrm{t}q}=0.7$ and $Q_{\mathrm{s}\dot{q}}=Q_{\mathrm{t}\dot{q}}=R_\mathrm{s}=0.01$. Automated tuning could further improve the control performance.

Experiments were conducted in six domains with different desired trajectories of the positions. For this purpose, linear trajectories were generated, whereby both the rise time in the range $0.4-1.6\,\mathrm{s}$ and the height of the amplitudes in the range $\pm 18°$ are determined randomly. In order to reduce the sharp transitions between the linear trajectories, the reference was slightly smoothed using a moving mean. The average of the absolute joint velocities during all experiments was $11.70\,°/\mathrm{s}$. Thus, the resulting trajectories correspond to dynamic movements of the robot, whereby some sections are

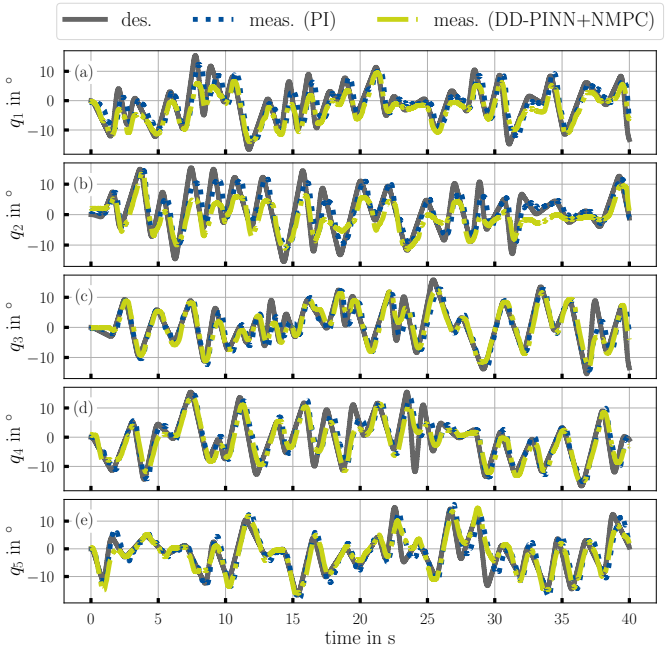


Figure 13. (a)–(e) Control results for a randomly selected trajectory of all n joints and $\boldsymbol{\delta}=[0\,\mathrm{g}, 90°]^\mathrm{T}$ using the proposed DD-PINN as a model within NMPC and comparison with PI control.

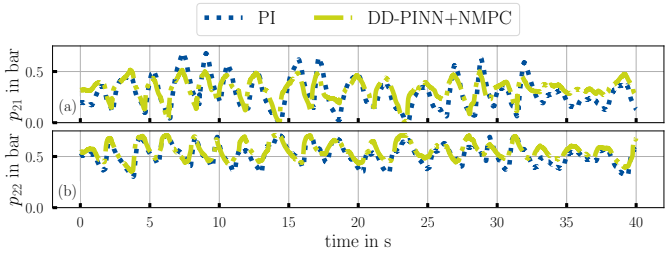


Figure 14. Measured pressures (a) p_{21} and (b) p_{22} during the control experiment with $\boldsymbol{\delta}=[0\,\mathrm{g}, 90°]^\mathrm{T}$ for NMPC with the DD-PINN and PI control. The NMPC enables a more dynamic response due to the used model knowledge and online optimization.

not even feasible. Please watch the supporting video of this article to see the soft robot and its movements during control.

The results are presented in Table III. In general, accurate trajectory tracking is realized with the nonlinear MPC, and the fast PINNs enable control frequencies f_NMPC up to $47.7\,\mathrm{Hz}$. It can be seen that the NMPC approach achieves better tracking results due to its predictive nature, i.e., anticipating future system behavior and optimizing the input signal accordingly, rather than merely reacting to past control errors. On average, an improvement of $14.6\,\%$ can be realized by using the proposed DD-PINN+NMPC. The highest improvement of $29.1\,\%$ was achieved for $\boldsymbol{\delta}=[200\,\mathrm{g}, 0°]$. Thereby, especially poor tracking of the first joint was present during PI control due to the challenging task of balancing against gravity.

It must be noted that the MPC runs with a significantly lower control frequency than the PI control ($1\,\mathrm{kHz}$). Better computing hardware in the future will improve the applicability of such MPC approaches further. For example, this would

increase f_{NMPC}, which will further improve the tracking performance. However, the realized real-time nonlinear MPC with up to $47.7\,\mathrm{Hz}$ is already a notable research achievement, which is only made possible by using PINNs. Depending on the system dynamics, the tracking accuracy of the MPC can also be higher in comparison to the PI control. If state constraints were required for other systems, these could also be easily handled using the MPC framework.

Exemplary control results for one test domain are visualized in Fig. 13. In some areas it can be seen that the steep changes can only be roughly tracked with a slight delay. In addition, for joint two in the present domain, several sections of the desired trajectory are not feasible due to gravity. This is illustrated in Fig. 14 with the corresponding pressure curves, which run into saturation. It can also often be seen that the PI controller has a delay and that the MPC reacts faster due to its predictive nature.

E. Overall Discussion

Three key claims of the article were given in Sec. I. Results show that the accurate physics-driven model can be accelerated by the fast surrogate by a factor of 467. This comes at the cost of a slightly lower accuracy. The hybrid model only requires data from one domain due to the use of prior physical knowledge and still allows a generalization to unknown dynamics. Such a key advantage becomes apparent when comparing the proposed DD-PINN with a pure black-box model. Finally, a real-time application is used to illustrate what becomes possible — the generalizable and fast DD-PINN enables a high control frequency and accurate control for articulated soft robots with different payloads or base orientations.

Although the scenario presented in Fig. 1 is more academic, this article provides important insights for real-world applications. In principle, changing conditions are always to be expected in reality, so generalizable models have great potential. Grasping different masses or acting with a variable base orientation is therefore possible with model-based control and the proposed DD-PINN. With longer versions of the snake robot, it is also possible that not every actuator is equipped with an encoder due to the cost. Instead, the states could be obtained in real-time using a PINN-based estimator. Furthermore, fast simulations are possible with PINNs. This can be useful for a fast design optimization of the system or time-efficient (simulation-based) reinforcement learning, for example. In general, utilizing PINNs on embedded or low-power computing platforms in wearable devices is also conceivable due to the simple feedforward structure.

VI. Conclusion

Model-based estimation and control of soft robots require forward models that generalize to several system dynamics and provide a high prediction speed for real-time applications. However, data-hungry black-box learning and slow physics-based models cannot fulfill both requirements. In a literature overview, we analyze the current trend of hybrid model learning and identify a research gap regarding physics-informed neural networks for *real-world, multi-DoF soft robots in small-data regimes*. The latter is crucial: Real-world data is expensive due to the recording effort or possible maintenance/damage, and it is unrealistic that all future system conditions can be considered during training-data acquisition.

We extend two existing PINN architectures with domain knowledge and perform hybrid model learning for articulated soft robots. In two hours of experiments, the presented model was tested in various system domains that were not seen during the recording of the training data. Core results of our work (Table II and Fig. 11) show that the proposed DD-PINN outperforms an elaborately identified first-principles model and a recurrent neural network *when generalizability and prediction speed are considered jointly*. As one possible real-time application, the DD-PINN enables nonlinear model predictive control and, thus, accurate position tracking in several system domains for dynamic references.

Diverse future research directions arise from our work. The method is applicable to (soft) continuum robots, where dynamical models from Cosserat rod theory are accurate but require high computational costs. Since the domain variables only need to appear in the dynamics equation, the PINN could be trained for other variable quantities, such as external contacts. This could, in principle, be used for contact estimation during operation. In order to further investigate the extent to which generalization is possible, considerably more than two domain variables should be examined. In general, due to the required knowledge of the domain variables, online estimation is promising for unmeasurable quantities. Furthermore, different ansatz functions of the DD-PINN could be compared, or suitable ansatz functions could be found using symbolic regression. Also, the feedforward structure of the DD-PINN could be replaced by an RNN to further improve the prediction accuracy. Since the DD-PINN training is still demanding, approaches from meta or transfer learning could be used to enhance generalization for system changes beyond the considered domain variables. Regarding control, GPU approaches [26], [73] could further improve the performance, and the combination with online learning could enable adaptivity. Besides soft robotics, our architecture can be applied to various dynamical systems to provide generalizable and fast surrogates for real-time estimation and control.

Acknowledgment

We thank Mehdi Belhadj for collecting the datasets.

References

[1] D. Rus and M. T. Tolley, “Design, fabrication and control of soft robots,” *Nature*, pp. 467–475, 2015.

[2] M. S. Xavier et al., “Soft pneumatic actuators: A review of design, fabrication, modeling, sensing, control and applications,” *IEEE Access*, vol. 10, pp. 59 442–59 485, 2022.

[3] O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks, Fuzzy Models, and Gaussian Processes*, 2nd ed. Springer Cham, 2020.

[4] 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,” *IEEE Control Systems*, vol. 43, no. 3, pp. 30–65, 2023.

[5] T. G. Thuruthel, Y. Ansari, E. Falotico, and C. Laschi, “Control strategies for soft robotic manipulators: A survey,” *Soft Robotics*, vol. 5, no. 2, pp. 149–163, 2018.

[6] C. Laschi, T. G. Thuruthel, F. Lida, R. Merzouki, and E. Falotico, "Learning-based control strategies for soft robots: Theory, achievements, and future challenges," *IEEE Control Systems*, vol. 43, no. 3, pp. 100–113, 2023.

[7] E. Falotico et al., "Learning controllers for continuum soft manipulators: Impact of modeling and looming challenges," *Advanced Intelligent Systems*, 2024.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[9] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics-informed neural networks: Where we are and what's next," 2022. [Online]. Available: https://arxiv.org/pdf/2201.05624

[10] E. A. Antonelo, E. Camponogara, L. O. Seman, J. P. Jordanou, E. R. de Souza, and J. F. Hübner, "Physics-informed neural nets for control of dynamical systems," *Neurocomputing*, vol. 579, p. 127419, 2024.

[11] H. Krauss, T.-L. Habich, M. Bartholdt, T. Seel, and M. Schappler, "Domain-decoupled physics-informed neural networks with closed-form gradients for fast model learning of dynamical systems," in *International Conference on Informatics in Control, Automation and Robotics*. SciTePress, 2024, pp. 55–66.

[12] T.-L. Habich, A. Mohammad, S. F. G. Ehlers, M. Bensch, T. Seel, and M. Schappler, "Experimental data of an articulated soft robot with variable payloads and base orientations," *Institutional Repository of Leibniz University Hannover*, 2025. [Online]. Available: https://doi.org/10.15488/19708

[13] C. Della Santina, M. G. Catalano, and A. Bicchi, "Soft robots," in *Encyclopedia of Robotics*, M. H. Ang, O. Khatib, and B. Siciliano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–15.

[14] R. J. Webster 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.

[15] C. Alessi, C. Agabiti, D. Caradonna, C. Laschi, F. Renda, and E. Falotico, "Rod models in continuum and soft robot control: a review."

[16] E. Coevoet, T. Morales-Bieze, F. Largilliere, Z. Zhang, M. Thieffry, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, J. Dequidt, and C. Duriez, "Software toolkit for modeling, simulation, and control of soft robots," *Advanced Robotics*, vol. 31, no. 22, pp. 1208–1224, 2017.

[17] C. Armanini, F. Boyer, A. T. Mathew, C. Duriez, and F. Renda, "Soft robots modeling: A structured overview," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1728–1748, 2023.

[18] C. M. Best, J. P. Wilson, and M. D. Killpack, "Control of a pneumatically actuated, fully inflatable, fabric-based, humanoid robot," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 1133–1140.

[19] K. Hoffmann, C. Trapp, A. Hildebrandt, and O. Sawodny, "Nonlinear model-based control of a pneumatically driven robot," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 8770–8775, 2023.

[20] S. P. Chhatoi, M. Pierallini, F. Angelini, C. Mastalli, and M. Garabini, "Optimal control for articulated soft robots," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3671–3685, 2023.

[21] X. Wang, Y. Li, and K.-W. Kwok, "A survey for machine learning-based control of continuum robots," *Frontiers in Robotics and AI*, vol. 8, p. 730330, 2021.

[22] D. Braganza, D. M. Dawson, I. D. Walker, and N. Nath, "A neural network controller for continuum robots," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1270–1277, 2007.

[23] 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 *IEEE International Conference on Soft Robotics*, 2018, pp. 39–45.

[24] D. G. Cheney and M. D. Killpack, "MoLDy: Open-source library for data-based modeling and nonlinear model predictive control of soft robots," in *IEEE International Conference on Soft Robotics*, 2024, pp. 958–964.

[25] 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.

[26] P. Hyatt and M. D. Killpack, "Real-time nonlinear model predictive control of robots using a graphics processing unit," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1468–1475, 2020.

[27] T.-L. Habich, S. Kleinjohann, and M. Schappler, "Learning-based position and stiffness feedforward control of antagonistic soft pneumatic actuators using Gaussian processes," in *2023 IEEE International Conference on Soft Robotics*. IEEE, 2023, pp. 1–7.

[28] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015. [Online]. Available: https://arxiv.org/pdf/1506.00019

[29] 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.

[30] ——, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 124–134, 2019.

[31] T. Luong et al., "Long short term memory model based position-stiffness control of antagonistically driven twisted-coiled polymer actuators using model predictive control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4141–4148, 2021.

[32] H. Schäfke, T.-L. Habich, C. Muhmann, S. F. G. Ehlers, T. Seel, and M. Schappler, "Learning-based nonlinear model predictive control of articulated soft robots using recurrent neural networks," *IEEE Robotics and Automation Letters*, vol. 9, no. 12, pp. 11 609–11 616, 2024.

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

[34] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[35] M. Kasaei, K. K. Babarahmati, Z. Li, and M. Khadem, "Data-efficient non-parametric modelling and control of an extensible soft manipulator," in *2023 IEEE International Conference on Robotics and Automation*. IEEE, 2023, pp. 2641–2647.

[36] A. Centurelli, L. Arleo, A. Rizzo, S. Tolu, C. Laschi, and E. Falotico, "Closed-loop dynamic control of a soft manipulator using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4741–4748, 2022.

[37] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2677–2682.

[38] G. Runge, M. Wiese, and A. Raatz, "FEM-based training of artificial neural networks for modular soft robots," in *2017 IEEE International Conference on Robotics and Biomimetics*. Piscataway, NJ: IEEE, 2017, pp. 385–392.

[39] C. C. Johnson, T. Quackenbush, T. Sorensen, D. Wingate, and M. D. Killpack, "Using first principles for deep learning and model-based control of soft robots," *Frontiers in Robotics and AI*, vol. 8, p. 654398, 2021.

[40] X. Huang, Y. Rong, and G. Gu, "High-precision dynamic control of soft robots with the physics-learning hybrid modeling approach," *IEEE/ASME Transactions on Mechatronics*, pp. 1–12, 2024.

[41] R. F. Reinhart, Z. Shareef, and J. J. Steil, "Hybrid analytical and data-driven modeling for feed-forward robot control," *Sensors*, vol. 17, no. 2, 2017.

[42] J. Gao, M. Y. Michelis, A. Spielberg, and R. K. Katzschmann, "Sim-to-real of soft robots with learned residual physics," *IEEE Robotics and Automation Letters*, pp. 1–8, 2024.

[43] G. Lou, C. Wang, Z. Xu, J. Liang, and Y. Zhou, "Controlling soft robotic arms using hybrid modelling and reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 9, no. 8, pp. 7070–7077, 2024.

[44] T. Z. Jiahao, R. Adolf, C. Sung, and M. A. Hsieh, "Knowledge-based neural ordinary differential equations for Cosserat rod-based soft robots," *arXiv*, 2024. [Online]. Available: https://arxiv.org/pdf/2408.07776

[45] K. Neumann, M. Rolf, and J. J. Steil, "Reliable integration of continuous constraints into extreme learning machines," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 21, no. supp02, pp. 35–50, 2013.

[46] A. Tariverdi et al., "A recurrent neural-network-based real-time dynamic model for soft continuum manipulators," *Frontiers in Robotics and AI*, vol. 8, p. 631303, 2021.

[47] 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.

[48] T. X. Nghiem et al., "Physics-informed machine learning for modeling and control of dynamical systems," in *2023 American Control Conference*. IEEE, 2023, pp. 3735–3750.

[49] M. Lutter, K. Listmann, and J. Peters, "Deep Lagrangian networks for end-to-end learning of energy-based control for under-actuated systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 7718–7725.

[50] M. Lutter and J. Peters, "Combining physics and deep learning to learn continuous-time dynamics models," 2021. [Online]. Available: https://arxiv.org/pdf/2110.01894

[51] J. Nicodemus, J. Kneifl, J. Fehr, and B. Unger, "Physics-informed neural networks-based model predictive control for multi-link manipulators," *IFAC-PapersOnLine*, vol. 55, no. 20, pp. 331–336, 2022.

[52] W. Sun, N. Akashi, Y. Kuniyoshi, and K. Nakajima, "Physics-informed recurrent neural networks for soft pneumatic actuators," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6862–6869, 2022.

[53] M. Lahariya, C. Innes, C. Develder, and S. Ramamoorthy, "Learning physics-informed simulation models for soft robotic manipulation: A case study with dielectric elastomer actuators," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2022, pp. 11 031–11 038.

[54] X. Wang et al., "PINN-Ray: A physics-informed neural network to model soft robotic fin ray fingers," 2024. [Online]. Available: https://arxiv.org/abs/2407.08222

[55] S. I. Beaber, Z. Liu, and Y. Sun, "Physics-guided deep learning enabled surrogate modeling for pneumatic soft robots," *IEEE Robotics and Automation Letters*, vol. 9, no. 12, pp. 11 441–11 448, 2024.

[56] M. Bensch, T.-D. Job, T.-L. Habich, T. Seel, and M. Schappler, "Physics-informed neural networks for continuum robots: Towards fast approximation of static Cosserat rod theory," in *2024 IEEE International Conference on Robotics and Automation*. IEEE, 2024, pp. 17 293–17 299.

[57] J. Liu, P. Borja, and C. Della Santina, "Physics–informed neural networks to model and control robots: A theoretical and experimental investigation," *Advanced Intelligent Systems*, 2024.

[58] T. Yoon et al., "Kinematics-informed neural networks: Enhancing generalization performance of soft robot model identification," *IEEE Robotics and Automation Letters*, vol. 9, no. 4, pp. 3068–3075, 2024.

[59] C. A. Mendenhall, J. Hardan, T. D. Chiang, L. H. Blumenschein, and A. B. Tepole, "Physics-informed neural network for scalable soft multi-actuator systems," in *IEEE International Conference on Soft Robotics*, 2024, pp. 716–721.

[60] S. Wang, R. Wang, J. Yang, and L. Hao, "Online incremental dynamic modeling using physics-informed long short-term memory networks for the pneumatic artificial muscle," *IEEE Robotics and Automation Letters*, vol. 9, no. 10, pp. 8435–8442, 2024.

[61] M. Schüssler, T. Munker, and O. Nelles, "Deep recurrent neural networks for nonlinear system identification," in *2019 IEEE Symposium Series on Computational Intelligence*. IEEE, 2019, pp. 448–454.

[62] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014. [Online]. Available: https://arxiv.org/pdf/1409.1259.pdf

[63] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[64] L. Li et al., "A system for massively parallel hyperparameter tuning," *Conference on Machine Learning and Systems*, 2020. [Online]. Available: https://arxiv.org/pdf/1810.05934

[65] T.-L. Habich, J. Haack, M. Belhadj, D. Lehmann, T. Seel, and M. Schappler, "SPONGE: Open-source designs of modular articulated soft robots," *IEEE Robotics and Automation Letters*, vol. 9, no. 6, pp. 5346–5353, 2024.

[66] M. Azad and R. Featherstone, "A new nonlinear model of contact normal force," *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 736–739, 2014.

[67] K. H. Hunt and F. R. E. Crossley, "Coefficient of restitution interpreted as damping in vibroimpact," *Journal of Applied Mechanics*, vol. 42, no. 2, pp. 440–445, 1975.

[68] R. Wang and R. Yu, "Physics-guided deep learning for dynamical systems: A survey," 2021. [Online]. Available: http://arxiv.org/abs/2107.01272v6

[69] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

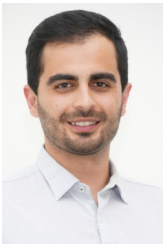
[70] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[71] H. Zeipel, T.-L. Habich, T. Seel, and S. F. G. Ehlers, "Fast and accurate prediction of vehicle dynamics using physics-informed neural networks," *TechrXiv*, 2024. [Online]. Available: https://doi.org/10.36227/techrxiv.173398186.65085317/v1

[72] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, vol. 14.

[73] S. H. Jeon, S. Hong, H. J. Lee, C. Khazoom, and S. Kim, "CusADi: A GPU parallelization framework for symbolic expressions and optimal control," *IEEE Robotics and Automation Letters*, vol. 10, no. 2, pp. 899–906, 2025.

Tim-Lukas Habich received his B. Sc. and M. Sc. degrees in Mechanical Engineering from Leibniz University Hannover (LUH), Germany, in 2017 and 2020, respectively. Currently, he is working towards his Ph. D. degree at the Institute of Mechatronic Systems (imes) at LUH. His research interests include physics-informed machine learning and learning-based control of mechatronic systems.



Aran Mohammad received his B. Sc. and M. Sc. degrees in Mechanical Engineering from LUH, Germany, in 2017 and 2020, respectively. Currently, he is working towards his Ph. D. degree at the imes at LUH. His research interests include physics- and learning-based modeling, optimization and control of mechatronic systems.

Simon F. G. Ehlers received his B. Sc., M. Sc. and Ph. D. degrees in Mechanical Engineering from LUH, Germany, in 2016, 2019 and 2024, respectively. He is currently leading a research group on learning and control at the imes at LUH. His research interests include physics- and learning-based estimator concepts, especially in the field of vehicle dynamics.

Martin Bensch received his B. Sc. and M. Sc. degrees in Mechanical Engineering from LUH, Germany, in 2015 and 2019, respectively. Currently, he is working towards his Ph. D. degree at the imes at LUH. His research interests include continuum robots, machine learning, and path planning for continuum robots.

Thomas Seel studied Engineering Cybernetics at Otto von Guericke University Magdeburg, Germany, and the University of California, USA. He received the Ph. D. degree from Technische Universität Berlin, Germany, in 2016. He is currently director of the imes at LUH. His research interests include dynamic inference and learning in biomedical and mechatronic systems.

Moritz Schappler received his B. Sc. and M. Sc. degrees in Mechatronics Engineering, and Ph. D. degree in Mechanical Engineering from LUH, Germany, in 2012, 2014 and 2025, respectively. He is currently leading a research group on robotic systems at the imes at LUH. His research interests include design, modeling and control of robotic and mechatronic systems with a focus on parallel robots, snake-like robots and methods for design optimization.