IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Lagrangian Neural Network-Based Control: Improving Robotic Trajectory Tracking via Linearized Feedback

Manuel Weiss[1,2*], Alexander Pawluchin[1], Jan-Hendrik Ewering[2], Thomas Seel[2], Ivo Boblan[1]

***Abstract*—This paper introduces a control framework that leverages Lagrangian neural networks (LNNs) for computed torque control (CTC) of robotic systems with unknown dynamics. Unlike prior LNN-based controllers that are placed outside the feedback-linearization framework (e.g., feedforward), we embed an LNN inverse-dynamics model within a CTC loop, thereby shaping the closed-loop error dynamics.**

This strategy, referred to as LNN-CTC, ensures a physically consistent model and improves extrapolation, requiring neither prior model knowledge nor extensive training data. The approach is experimentally validated on a robotic arm with four degrees of freedom and compared with conventional model-based CTC, physics-informed neural network (PINN)-CTC, deep neural network (DNN)-CTC, an LNN-based feedforward controller, and a PID controller. Results demonstrate that LNN-CTC significantly outperforms model-based baselines by up to 30 % in tracking accuracy, achieving high performance with minimal training data. In addition, LNN-CTC outperforms all other evaluated baselines in both tracking accuracy and data efficiency, attaining lower joint-space RMSE for the same training data. The findings highlight the potential of physics-informed neural architectures to generalize robustly across various operating conditions and contribute to narrowing the performance gap between learned and classical control strategies.

***Index Terms*—Machine Learning for Robot Control, Model Learning for Control, Motion Control.**

I. Introduction

The ever-increasing complexity of robotic systems demands advanced, data-efficient control methods to handle nonlinear dynamics and ensure precision in real-world applications such as healthcare, manufacturing, and autonomous navigation [1].

Linearized feedback control transforms nonlinear robot dynamics into linear systems, enabling systematic stability analysis and precise trajectory tracking. This approach is particularly advantageous in applications that demand interpretability, robustness, and tunable performance. However, its effectiveness hinges on accurate dynamics models. For complex robots, such fidelity is often impractical, and the necessary modeling simplifications degrade regulation and tracking performance [2], [3]. Computed torque control (CTC) is a standard feedback-linearization method in robotics. It uses a robot model to cancel inertia, Coriolis/centrifugal, and gravity terms, thereby reducing the closed-loop error dynamics to a decoupled linear form. In response to these model-fidelity limitations, learning-based controllers fit policies or dynamics directly from data. Deep reinforcement learning (RL) often requires tens of millions of interactions to obtain effective locomotion policies. On hardware, this translates to extensive data collection [4]. Beyond RL-specific data demands, noisy supervision (e.g., imperfect rewards or demonstrations) further hurts sample efficiency, underscoring that data quality is as critical as data quantity [5]. More generally, deep neural network (DNN)-based controllers tend to be black-box and hard to interpret and often generalize poorly outside of the training distribution. As a result, performance is typically reliable only within the regions represented in the training data [6], [7], [8].

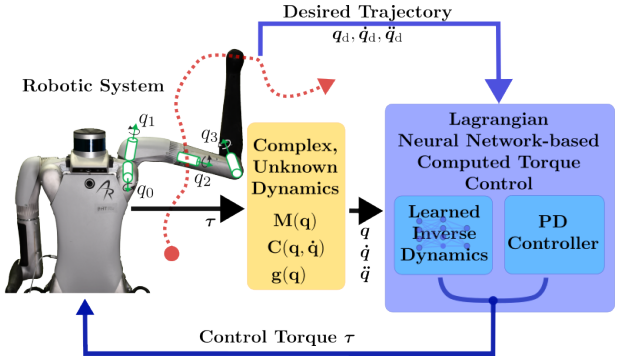


Fig. 1. Schematic of the proposed LNN-CTC framework. An LNN learns the inverse dynamics $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, $\mathbf{g}(\mathbf{q})$ to compute torque commands $\boldsymbol{\tau}$, combined with PD feedback. This enables accurate trajectory tracking $(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)$ without requiring an explicit analytic model.

Recent advances enable learning of robot dynamics with physics-informed models such as physics-informed neural networks (PINNs) [9], [10] and LNNs [11], [12], embedding mechanics priors to improve data efficiency and generalization [9]. Their effectiveness has been shown in control, e.g., PINNs for state prediction in model predictive control (MPC) [9] and accurate continuum robot modeling [10], while structured mechanical models (SMMs) and LNNs outperform black-box DNNs under limited data across simulated robots [13]. LNNs are specialized PINNs tailored to systems governed by Lagrangian mechanics, enforcing physical consistency and enabling robust extrapolation from small datasets [11]. However, in prior work, LNN controllers are typically placed outside the feedback-linearization framework, used either as feedforward terms or within low-rate MPC. While feedforward LNNs provide fast tracking, they offer limited disturbance rejection when the state deviates from the reference. The adaptive LNN-MPC runs at 50 Hz due to solver latency and model-update overhead and therefore relies on a secondary proportional derivative (PD) loop at 1 kHz for real-

This work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – FIP–12 Project-ID 528483508.

[1]Compliant Robotics Lab, Berlin University of Applied Sciences and Technology, Germany.

[2]Institute of Mechatronic Systems, Leibniz University Hannover, Germany.

*Corresponding author name.surname@bht-berlin.de

Digital Object Identifier (DOI): see top of this page.

time stabilization [14]. As a result, this control strategy may not be suitable for highly dynamic systems.

To our knowledge, this work is the first to embed an inverse-dynamics LNN inside CTC, altering the closed-loop dynamics rather than adding a torque bias. When accurate, the LNN achieves decoupled linear error dynamics and improves rejection of disturbances. This changes the error dynamics and, therefore, the controller's stability, both of which are determined by the LNN, in contrast to feedforward control. Because LNNs parameterize the Lagrangian energy of a system, it is possible to extract the exact terms required by CTC, making them a structure-preserving model replacement (Fig. 1).

This paper introduces LNN-CTC and demonstrates its effectiveness across multiple CTC architectures by comparing it with existing control strategies, including model-based CTC, LNN feedforward control and a proportional integral derivative (PID) control as a baseline. In particular, this paper proposes incorporating the LNN inverse-dynamics model directly into the CTC law (both full-model and feedback-linearization-only variants), so that when the learned model is accurate, the closed-loop reduces to decoupled linear error dynamics. With a bounded approximation error, the loop behaves as a linear system with an additive disturbance (standard CTC robustness). This tight integration is fundamentally different from existing LNN feedforward controllers, which do not alter the closed-loop linearization. Empirically, on a 4-degrees of freedom (DOF) arm, LNN-CTC trained with only 100 s of hardware data reduces the root mean square (RMS) error from 2.399° to 1.577° (-34.3 %) relative to model-based CTC with identical gains and across ten 130 s trajectories, and outperforms LNN feedforward and pure PD. These results indicate that placing a physics-structured learner inside the feedback-linearized loop is both practically impactful and data-efficient.

The paper is structured as follows. In Section II, the proposed LNN-CTC approach is presented, including the considered class of robotic systems. The experimental setup is introduced in Section III-A. In Section III, the results are presented and a conclusion is drawn in Section IV.

II. Methodology

This section introduces the conventions used, preliminary work on LNNs, and CTCs.

A. Modeling the Robot's System Dynamics

Consider a robotic system with n-DOF. Let $\mathbf{q} \in \mathbb{R}^n$ denote the vector of joint positions, $\dot{\mathbf{q}} \in \mathbb{R}^n$ the vector of joint velocities, and $\ddot{\mathbf{q}} \in \mathbb{R}^n$ the vector of joint accelerations. The state of the system is given by the pair $(\mathbf{q}, \dot{\mathbf{q}})$, and the control input is the vector of joint torques $\boldsymbol{\tau} \in \mathbb{R}^n$ produced by the actuators. The dynamic behavior of the robot is described by the dynamic equations of motion

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}), \tag{1}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ is the mass-inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n \times n}$ contains the centrifugal and Coriolis forces, and $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ are the gravitational forces. $\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ denotes the actuator torque vector specified by the inverse dynamics for the current state and acceleration. $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{g}(\mathbf{q})$ in (1) are dependent on the robotic state configuration and need to be known or identified for diverse model-based control approaches. For instance, the mass matrix accounts for the inertial properties of the robot and varies with the configuration due to the mass distribution. The centrifugal and Coriolis matrix $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ includes terms that arise from the robot's motion and can significantly affect the dynamics at higher speeds. The gravitational vector $\mathbf{g}(\mathbf{q})$ represents the torques required to counteract gravity in different configurations. To simplify the notation we combine Coriolis/centrifugal and gravitational terms, and define

$$\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) := \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}). \tag{2}$$

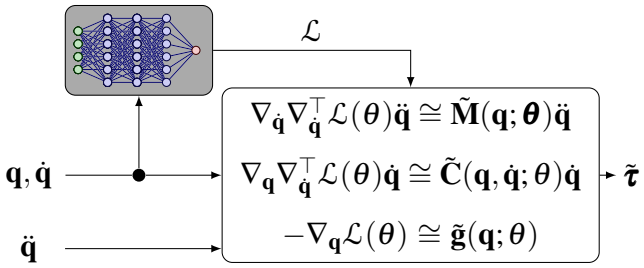


Fig. 2. Architecture of the LNN. The LNN takes as input the generalized coordinates $\mathbf{q}$ and generalized velocities $\dot{\mathbf{q}}$. These inputs are processed by the LNN to produce a Lagrangian $\mathcal{L}$, which is differentiated to obtain the generalized forces or torques $\tilde{\boldsymbol{\tau}}$ (Eq. (5)).

B. Lagrangian Neural Networks

An LNN is a neural network designed to model physical systems by leveraging the principles of Lagrangian mechanics (Fig. 2). This method enables data efficiency and generalization beyond the training data distribution by restricting learning to the class of systems that can be modeled by Lagrangian mechanics, thus structuring and guiding the learning process. As a result, the obtained neural network respects the underlying physical principles, for example, the conservation of energy [11], [12], [15].

We define the Lagrangian

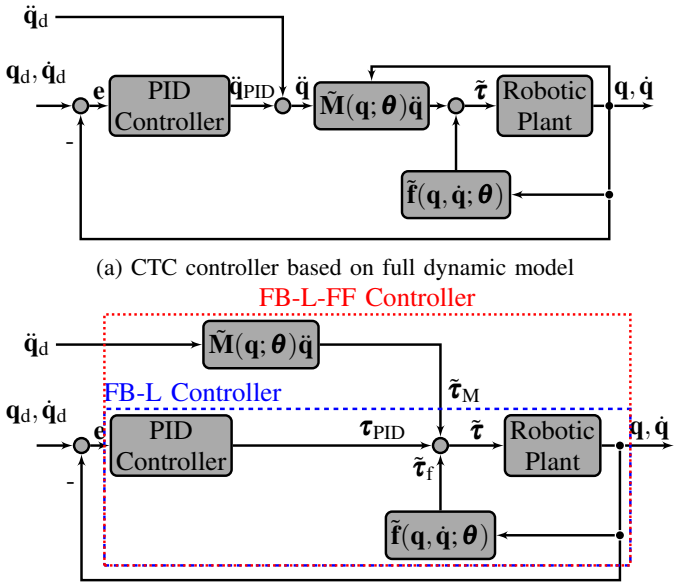
$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) \equiv T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}, \dot{\mathbf{q}}), \tag{3}$$

where T is the kinetic energy and V is the potential energy. It can be shown that the robot's inverse dynamics (Eq. 1) can be expressed in terms of the Lagrangian $\mathcal{L}$ as

$$\begin{aligned}\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = &\left(\nabla_{\dot{\mathbf{q}}} \nabla_{\dot{\mathbf{q}}}^\top \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})\right) \ddot{\mathbf{q}} \\ &+ \left(\nabla_{\mathbf{q}} \nabla_{\dot{\mathbf{q}}}^\top \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})\right) \dot{\mathbf{q}} - \nabla_{\mathbf{q}} \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}),\end{aligned} \tag{4}$$

which relates changes in the system state to the control inputs [6]. The proposed control methodology addresses the robot's inverse dynamics, which can be reformulated as Eq. (1). Because $\mathbf{M}(\mathbf{q})$ and $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})$ are traditionally calculated from measured masses, lengths, and moments of inertia, we instead express the inverse dynamics using an LNN as

$$\tilde{\boldsymbol{\tau}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \boldsymbol{\theta}) = \tilde{\mathbf{M}}(\mathbf{q}; \boldsymbol{\theta})\ddot{\mathbf{q}} + \tilde{\mathbf{f}}(\mathbf{q}, \dot{\mathbf{q}}; \boldsymbol{\theta}) \tag{5}$$



(a) CTC controller based on full dynamic model

(b) Linearized Feedback only CTC controller

Fig. 3. Control loops for two common CTC architectures: (a) Full dynamic model CTC using PID control to compute desired joint accelerations $\ddot{\mathbf{q}}_\mathrm{d}$. (b) Feedback-Linearization-based CTC (FB-L), which uses PID control to compute the required torque. The FB-L architecture can be enhanced by incorporating a feedforward term (FB-L-FF) based on the inertia matrix to add necessary torque for tracking $\ddot{\mathbf{q}}_\mathrm{d}$.

with $\tilde{}$ denoting approximations. $\boldsymbol{\theta}$ makes explicit that $\tilde{\boldsymbol{\tau}}$ depends on the parameters of the LNN (see Fig. 2).

Because the LNN learns the underlying equations of motion instead of a general state transition function, LNNs need less data than traditional DNNs for good predictions [7], [11], [12], [16]. To obtain the parameters $\boldsymbol{\theta}$ of the LNN, the discrepancy between the LNN's predictions with respect to the true torque is minimized:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\tilde{\boldsymbol{\tau}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \boldsymbol{\theta}), \boldsymbol{\tau}_\mathrm{meas}), \tag{6}$$

where L is a differentiable loss function, reflecting the mismatch between predicted torque and the measured torque ($\boldsymbol{\tau}_\mathrm{meas}$) by the built-in sensors.

C. Controller

CTC is a widely used control technique for robotics and other nonlinear systems based on feedback linearization. Using an inverse robot model, the control task is transformed into a linearized problem, enabling the use of linear controllers [17], [18], [19], [20]. The fundamental concept of CTC can be implemented through various structural configurations that differ in their formulation and placement of model-based components within the control loop, but all incorporate a PID controller as an essential feedback element. The PID component allows the system to actively respond to tracking errors irrespective of the specific computed torque structure employed, ensuring robustness against model inaccuracies and external disturbances.

The PID input is the current state and the desired joint-space trajectory ($\mathbf{q}_\mathrm{d}$, $\dot{\mathbf{q}}_\mathrm{d}$, and $\ddot{\mathbf{q}}_\mathrm{d}$) provided by a trajectory planner. The controller computes the error, its derivative, and its integral as follows

$$\begin{aligned} \mathbf{e}_\mathrm{P}(t) &= \mathbf{q}_\mathrm{d}(t) - \mathbf{q}(t), \\ \mathbf{e}_\mathrm{I}(t) &= \int_0^t \mathbf{e}_\mathrm{P}(t)\mathrm{d}t, \\ \mathbf{e}_\mathrm{D}(t) &= \dot{\mathbf{q}}_\mathrm{d}(t) - \dot{\mathbf{q}}(t). \end{aligned} \tag{7}$$

The PID controller outputs either the feedback joint acceleration $\ddot{\mathbf{q}}_\mathrm{PID}(t)$ or the control torque $\boldsymbol{\tau}_\mathrm{PID}(t)$, depending on the chosen controller setting, computed as

$$\mathbf{u}_\mathrm{PID}(t) = \mathbf{K}_\mathrm{P}\mathbf{e}_\mathrm{P}(t) + \mathbf{K}_\mathrm{I}\mathbf{e}_\mathrm{I}(t) + \mathbf{K}_\mathrm{D}\mathbf{e}_\mathrm{D}(t), \tag{8}$$

where $\mathbf{u}_\mathrm{PID}$ denotes the controller output ($\mathbf{u}_\mathrm{PID} \equiv \ddot{\mathbf{q}}_\mathrm{PID}$ or $\mathbf{u}_\mathrm{PID} \equiv \boldsymbol{\tau}_\mathrm{PID}$), the diagonal matrices $\mathbf{K}_\mathrm{P}, \mathbf{K}_\mathrm{I}, \mathbf{K}_\mathrm{D} \in \mathbb{R}^{n\times n}$ are the proportional, derivative, and integral gains, respectively.

1) CTC: CTC is a control approach for handling nonlinear dynamics in robotic systems that ensures accurate trajectory tracking by calculating the required torque to follow a desired trajectory through model inversion. This decouples the nonlinear dynamics of the system, resulting in a linearized model that is simpler to control. A standard feedback controller can then efficiently regulate the decoupled joint variables, leading to enhanced control precision [19]. In this work, two CTC architectures are considered: full-model (control variable $\ddot{\mathbf{q}}$) and feedback-linearization CTC (control variable $\boldsymbol{\tau}$).

2) CTC with full dynamic model: The full dynamic model CTC consists of two components. The first component determines a compensation torque $\tilde{\boldsymbol{\tau}}_\mathrm{f}$ for (possibly nonlinear) effects in the system behavior, such as friction, Coriolis forces, and gravitational effects at the current operating point. It is solely dependent on the current state of the system (see Fig. 3a). The second component, $\tilde{\mathbf{M}}(\mathbf{q};\boldsymbol{\theta})\ddot{\mathbf{q}}$, computes the torque required to achieve the desired joint acceleration at the current operating point. Therefore, it allows control in the joint space by providing the linear transformation between joint accelerations and torques. Here, $\ddot{\mathbf{q}}$ represents a sum of the feedforward acceleration $\ddot{\mathbf{q}}_\mathrm{ff}$ and the feedback acceleration $\ddot{\mathbf{q}}_\mathrm{PID}$, generated by the PID controller.

3) CTC Feedback-Linearization: The Feedback-Linearization-based CTC (FB-L) (see Fig. 3b) uses the torque $\tilde{\boldsymbol{\tau}}_\mathrm{f}$ as well. In contrast to the full CTC scheme, the architecture employs a PID torque control law

$$\tilde{\boldsymbol{\tau}}(t) = \boldsymbol{\tau}_\mathrm{PID}(t) + \tilde{\boldsymbol{\tau}}_\mathrm{f}(t). \tag{9}$$

For improved performance, if the dynamic model is sufficiently accurate, the control law can be augmented with an additional torque term, $\tilde{\boldsymbol{\tau}}_\mathrm{M} = \tilde{\mathbf{M}}(\mathbf{q};\boldsymbol{\theta})\ddot{\mathbf{q}}_\mathrm{d}$, which compensates for inertia during motion, resulting in a feedback-linearized CTC with acceleration feedforward (FB-L-FF). The resulting control law reads

$$\tilde{\boldsymbol{\tau}}(t) = \boldsymbol{\tau}_\mathrm{PID}(t) + \tilde{\boldsymbol{\tau}}_\mathrm{f}(t) + \tilde{\boldsymbol{\tau}}_\mathrm{M}(t). \tag{10}$$

The proposed controller architectures can be found on GitHub (https://github.com/ma-weiss/Lagrangian-Neural-Network_CTC).

III. Results

This section introduces experimental setups and the generation of training data. The results of the experiments are shown and discussed.

A. Experimental Setup and Training Data

All experiments are conducted on the 4-DOF arm of Agility Robotics' Digit platform (Fig. 1). The robot base is assumed to be stationary. Unless stated otherwise, the trained parameters of the LNN remain fixed. Only the adaptation study involves retraining. The evaluation comprises seven experiments: (1) Model generalization, comparing predicted torques for 250,000 out-of-distribution (OOD) states $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ against an analytic Lagrangian model, (2) Robustness, testing tracking under perturbed link parameters and uniformly distributed external wrenches on the end effector, (3) Gravity compensation, assessing static equilibrium to isolate the potential term V, (4) Dynamic trajectory tracking, evaluating acceleration-based LNN-CTC performance against diverse references with OOD setpoints, (5) Control-scheme comparison, evaluating torque-based LNN-CTC variants (FB-L, FB-L-FF) against different benchmarks, (6) Adaptation, adding a 3.5 kg payload ($\approx$ arm mass 3.6 kg) and fine-tuning the LNN to recover nominal tracking and (7) Cross-robot evaluation in simulation, extending the controller to the UR5 and KUKA IIWA 14 to study scalability. Training data are generated from smooth arbitrary trajectories tracked by a PD controller, with position setpoints sampled as $\mathbf{q} \sim \mathcal{U}(\mathbf{q}_{\min}, \mathbf{q}_{\max})$ and and segment durations drawn from $\mathcal{U}(10\,\text{s}, 25\,\text{s})$. MuJoCo is used to generate simulated data, while analytic models and random poses rely on Pinocchio [21]. The desired trajectories are generated using the Python robotics toolbox [22]. Cross-source evaluation (simulation vs. hardware) further quantifies sim-to-real transfer and inductive bias.

For all experiments, we trained the LNNs using 10 s, 100 s, 1000 s of simulated training data (Fig. 4), as well as LNNs trained with 10 s and 100 s of real-world training data. All data were sampled at 500 Hz. This work uses the Mahalanobis norm as loss function because the magnitude of the residual might vary between different joints. The LNN is a fully connected MLP (4 hidden layers, 64 units each) with Glorot–uniform initialization.

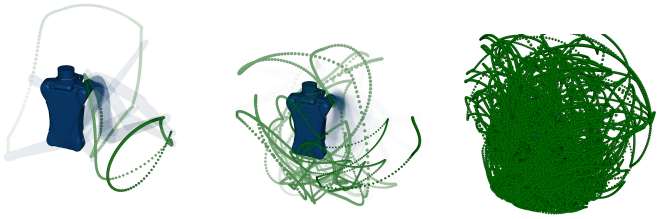
On an Apple M1 Pro CPU, the LNN-CTC inference is approximately 0.2 ms (187.7 MFLOP/s at 500 Hz with 375,324 FLOPs per inference).

This work begins with open-loop adequacy and robustness evaluations, then reports closed-loop performance. The LNN-CTC achieves lower root mean square error (RMSE) than the model-based CTC with identical gains.

B. Open Loop Prediction

To evaluate the generalization capabilities of the LNNs in predicting open-loop Lagrangian dynamics, the robot Lagrangian model is used as a baseline due to its reasonably good performance. Since the model-estimated $\tilde{\boldsymbol{\tau}}$ is only approximate, it serves only as a baseline to measure how close the LNN's predictions are. To compare the LNN with the model, 250,000 arbitrary samples are drawn uniformly from the hyper-rectangle $[2\mathbf{x}_{\min}, 2\mathbf{x}_{\max}]$, where $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. These OOD data exceed hardware-feasible joint states $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. The RMS difference between the model-based Lagrangian torque and the LNN prediction (Eq. 5) is computed for each data point (Fig. 5). As baselines, we include a PINN [23], a DNN to learn the full dynamics and a model-based Lagrangian with a DNN to learn the remaining error (RN) [24]. The baselines were trained with 100 s hardware data. The plot indicates that LNN predictions improve as more training data is used, especially when trained with hardware data. The lower RMS difference for models trained on 1000 s of simulated data and 100 s of hardware data suggests that the LNNs become more accurate in predicting the required torque as the duration of training increases.

In contrast, LNNs trained on limited data (10 s Sim and 10 s HW) exhibit significant RMS differences, implying reduced extrapolation to unseen joint-state combinations. The model trained on 10 s of simulated data shows the highest



(a) 10 s of training data (b) 100 s of training data (c) 1000 s of training data

Fig. 4. Visualization of the trajectories over varying training data length. (a) With 10 s of training data, the sparse trajectories show a limited explored operational region. (b) With 100 s of training data, the trajectories are denser, showing more complex and varied movements with a greater explored operational region. (c) With 1000 s of training data, the behavior exhibits an extensively explored operational region.

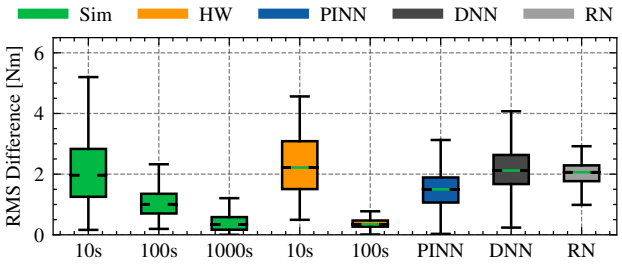


Fig. 5. RMS difference between the LNN-predicted torque and the torque estimated by the model, using 250,000 randomly sampled combinations of $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$. Since the model-estimated $\tilde{\boldsymbol{\tau}}$ is only approximate, it serves only as a baseline to measure how close the LNN's predictions are. The LNNs trained on 1000 s simulated and 100 s hardware data most closely approximate the model. In contrast, the LNNs trained on just 10 s of data exhibit larger offsets, with the simulated version having a greater error. DNN and RN are trained with 100 s hardware data. Bar color encodes the training data source for the inverse-dynamics model.

TABLE I. Original simulated robot model vs modified robot model vs. modified robot model with external wrench on the end effector controlled by LNN based on 1000 s sim. training data. For each control architecture 100 new models were sampled.

	Original Model	Modified	External Wrench
CTC	$0.734^\circ \pm 0.059^\circ$	$1.741^\circ \pm 0.171^\circ$	$1.932^\circ \pm 0.172^\circ$
FB-L	$1.137^\circ \pm 0.078^\circ$	$2.680^\circ \pm 0.170^\circ$	$2.875^\circ \pm 0.164^\circ$
FB-L-FF	$1.046^\circ \pm 0.090^\circ$	$2.929^\circ \pm 0.165^\circ$	$3.170^\circ \pm 0.167^\circ$
FF [11]	$1.241^\circ \pm 0.086^\circ$	$3.139^\circ \pm 0.192^\circ$	$3.834^\circ \pm 0.199^\circ$

error, indicating that the quantity and type of training data significantly affect the accuracy and generalization of the LNNs. The baselines show that DNNs require more data than LNNs to achieve similar performance. The baselines perform significantly worse than the LNN trained on the same data.

C. Robustness evaluation

We evaluate robustness to structured uncertainty in simulation by sampling 100 modified robot models from bounded, physically motivated intervals derived from CAD tolerances and identification residuals (link length $\pm 10\,\%$, masses $\pm 15\,\%$, inertias $\pm 25\,\%$, center of mass (COM) ± 15 mm, friction $\pm 50\,\%$). Since parameter modification of the physical robot is not feasible, this is done in simulation. For an additional experiment, we added a uniformly distributed external wrench to the modified model (sampled as $\mathbf{w} \backsim \mathcal{U}(-20, 20)^6$) to validate the robustness of the control architecture. We employ Latin Hypercube sampling to cover the space. The 95th percentile RMSE remained $<3.253^\circ$ for all architectures. Although we report only the LNN trained with 1000 s simulation data here, we hypothesize similar trends for all other LNNs. This remains to be validated. The performance of the LNN-CTC degrades as approximation error increases. A similar effect is visible in the sim-to-real transfer. However, LNN-CTC can handle imperfections in the model better than an LNN used as an FF controller (Tab. I). The performance of FB-L and FB-L-FF indicates that imperfect $\tilde{\mathbf{M}}(\mathbf{q}; \boldsymbol{\theta})$ has a substantial impact on the control performance. The results show that the FF controller has more difficulties in handling disturbances.

Preliminary sensitivity analysis indicates that mass and link length are the most influential. To reduce the performance below that of the PD controller ($3.912^\circ \pm 0.199^\circ$) the mass and the link length must be changed by $\approx 25\,\%$. However, this might be due to the low mass and therefore low torques of the LNN and might differ for other robots. We did not evaluate the robustness to disturbance (impulses, latency), which we leave for future work.

D. Gravity Compensation

The third experiment is conducted to test the ability to compensate for gravitational forces on the robot hardware. In particular, the term $\tilde{\mathbf{g}}(\mathbf{q}; \boldsymbol{\theta})$ should counteract gravitational forces, and we test the approximation based on different amounts of training data. For this, the robot arm is placed in arbitrary positions using a PD controller. These positions should be maintained without feedback if the model is sufficiently good (see Fig. 6). This is only done for LNN and the analytic model, since isolating the gravity term from DNNs is not feasible without effectively implementing a CTC variant.

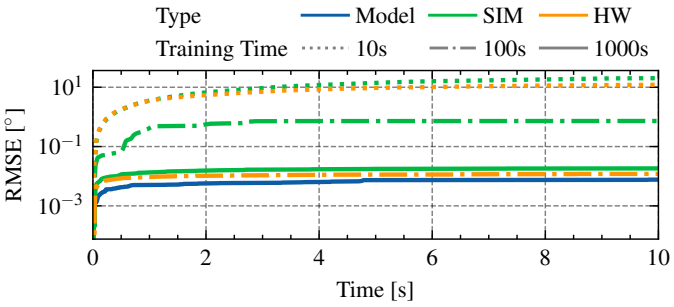


Fig. 6. Gravity compensation performance on hardware using different amounts of LNN training data. Simulation (SIM) and hardware (HW) training data are compared. While 10 s of training data is insufficient for equilibrium, both 100 s of hardware data and 1000 s of simulation data achieve satisfactory gravity compensation.

As a reference, the experiment is also conducted using the original model-based controller. Using only 10 s of simulated data to train the LNN, the controller is not able to keep the robotic arm in equilibrium, and the arm always yields to gravity, resulting in the arm hanging down. Only the elbow joint maintains its position due to its low weight and high friction. With 10 s of training data collected on real hardware, the results are better than with simulated training data. Still, the controller can only achieve equilibrium for the robotic arm close to hanging down. Therefore, within 10 s the exploration of the joint space configuration is not sufficient to extrapolate to configurations not included in the training data. Using 100 s or more data to train the LNN results in an error $< 1^\circ$ (see Fig. 6 and Tab. II). Training data gathered on the real hardware outperform the simulated training data due to the implicit learning of some friction components that are not simulated using the manufacturer's MuJoCo model due to fixed joint damping and friction components in the simulation. As expected, none of the LNN-based gravity models $\tilde{\mathbf{g}}(\mathbf{q}; \boldsymbol{\theta})$ can outperform the model-based approach, since the masses are perfectly known in the model.

E. Reference Tracking CTC

To validate the acceleration-based CTCs' performance on different amounts of training data, controllers were tested on 10 arbitrary trajectories of 130 s each. The trajectories vary in speed and amplitude to cover the robot's operational space, deliberately adding partially OOD conditions (higher velocities, with segment durations drawn from $\mathcal{U}(2\,\text{s}, 35\,\text{s})$), diverse motions, and joint configurations not present in the training set. The integral term is omitted from the PID controller, creating a PD controller. All parameters remain fixed throughout the experiments to ensure an unbiased comparison, with model-based, other neural network-based CTC as the baselines. As additional baselines a PINN, a DNN and an RN were trained with 100 s of hardware data. The DNN-based baselines were unstable due to poor extrapolation and to their placement within the feedback loop.

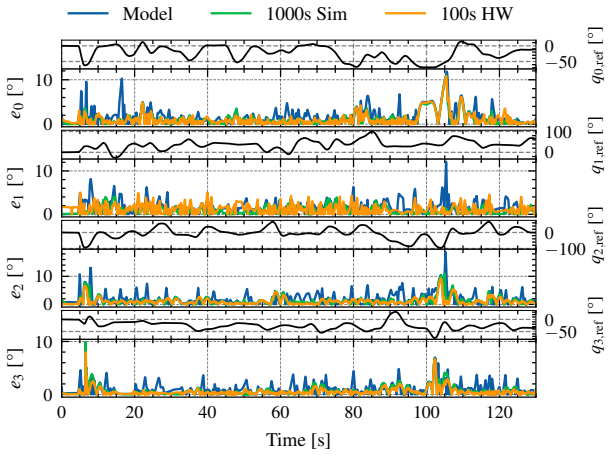


Fig. 7. Tracking performance of model-based CTC with LNN-CTCs using different training data (1000 s simulated data (SIM) and 100 s data gathered on the hardware (HW)).

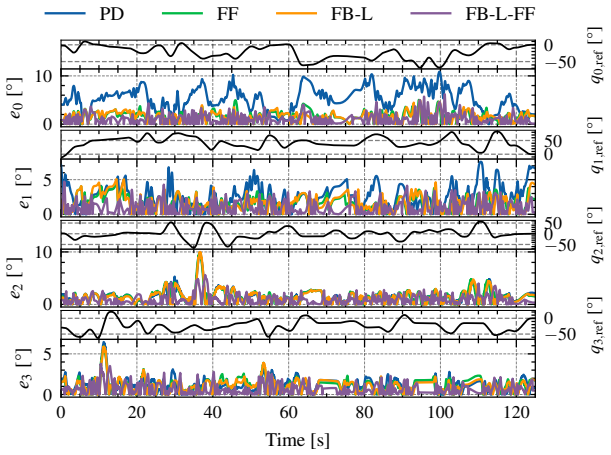


Fig. 8. Comparing a PD controller with various LNN-based control architectures, all trained on 100 s of hardware data and using identical PD gains. The PD controller performs the worst, while the feedforward controller (FF) [11] and the feedback-linearized controller (FB-L) show similar performance. The feedback-linearized controller performs best with $\ddot{\mathbf{q}}_\mathrm{d}$ feedforward (FB-L-FF).

The results in Table II reveal a clear hierarchy in training data effectiveness. LNN-CTC trained on only 10 s of simulated data failed to produce meaningful torques, resulting in poor trajectory tracking performance. However, 10 s of hardware training data achieved a performance approaching the model-based baseline, highlighting the superior representational quality of the real-world data. LNN-CTC trained on 100 s of hardware data outperformed the model-based CTC baseline. This superior performance could be attributed to the implicit learning of unmodeled dynamics, particularly friction and damping terms that are challenging to model analytically but naturally captured in hardware data. The LNN's ability to learn these real-world effects compensates for the limitations of the manufacturer's simulation model.

Fig. 7 compares the tracking performance between model-based CTC and the best-performing LNN-CTC (1000 s simulated and 100 s hardware data). For clarity, only the analytical model is plotted as a baseline. The PINN-CTC performs only marginally better than the model-base and remains far below the LNN-based CTC (Table II), so including it would not change the qualitative comparison. The LNN-based controllers demonstrate consistently lower tracking error across the majority of the trajectory, validating their practical advantage over traditional model-based approaches. The advantage is most evident in high-acceleration segments, where model-based error spikes are markedly reduced.

The results show that LNN-CTC achieve superior performance with minimal training data requirements when using hardware data, while maintaining the interpretability and physical consistency inherent to Lagrangian mechanics. This combination of data efficiency and performance improvement makes LNN-CTC particularly suitable for real-world robots where accurate system models are difficult to obtain.

F. Reference Tracking Feedback-Linearization

To isolate the performance contributions of different control components, we compare the two torque-based LNN-CTC architectures (FB-L and FB-L-FF) with the LNN-based feedforward controller (FF) using identical control parameters. All controllers use unified PD gains to ensure fair comparison: proportional gain $\mathbf{K}_\mathrm{P} = 50 \cdot \mathbf{I}_4$ and derivative gain $\mathbf{K}_\mathrm{D} = 5 \cdot \mathbf{I}_4$ where $\mathbf{I}_4$ denotes the identity matrix of order 4. The integral term is omitted, creating a PD controller configuration. Each controller tracks the same trajectory set used in previous experiments, with all LNNs trained on 10 s of hardware data. The PD controller alone achieves a RMSE of 3.454°, demonstrating limited performance and highlighting the necessity of model-based compensation for accurate trajectory tracking. As additional baselines, we include a PINN, a DNN and an RN.

Figure 8 shows the reference trajectories and tracking errors for the PD, FF, FB-L, and FB-L-FF controllers. PD exhibits the largest and most variable errors, while FF and FB-L produce similar, substantially tighter tracking. FB-L-FF achieves the smallest and most consistent errors across all joints. The superior performance of FB-L-FF demonstrates that the incorporation of the learned inertia matrix $\tilde{\mathbf{M}}(\mathbf{q};\boldsymbol{\theta})$ for acceleration feedforward provides significant benefits. The learned inertia model is more accurate than its model-based counterpart, enabling a better torque computation for desired accelerations. The DNN and the RN can track the trajectories but are less accurate than the LNN with the same training data. The 10 s simulation trained LNN outperforms the PD controller in these architectures, which mainly contributed to the poor fitted $\tilde{\mathbf{M}}(\mathbf{q};\boldsymbol{\theta})$ not being inside the feedback loop.

The results confirm that integrating learned dynamics into feedback-linearization and feedforward control architectures significantly enhances precision and robustness compared to simpler control approaches. The FB-L-FF architecture effectively combines the benefits of disturbance rejection (through feedback) and accurate dynamic compensation (through learned feedforward terms).

TABLE II. RMSE and STD in degrees over 10 trajectories of 130 s of hardware experiments. LNN models were trained on varying amounts of simulated (sim.) or experimental (exp.) data. RMSE is the mean across the 10×130 s trajectories. The red and green lines are barplots of the RMSE indicating the best performing in green. As an additional baseline a PD controller alone achieved a RMSE of 3.454°. (n.a.: not applicable, unst.: unstable closed-loop tracking)

		Gravity Comp.	CTC	FB-L	FB-L-FF	FF [11]
LNN based on sim. training data						
10 s	RMSE ± STD	14.676° ± 12.687°	14.362° ± 0.464°	3.238° ± 0.266°	3.525° ± 0.324°	3.552° ± 0.333°
100 s	RMSE ± STD	0.688° ± 0.595°	1.601° ± 0.330°	2.602° ± 0.203°	2.812° ± 0.186°	2.770° ± 0.208°
1000 s	RMSE ± STD	0.017° ± 0.01°	1.567° ± 0.379°	2.146° ± 0.190°	2.016° ± 0.365°	2.357° ± 0.195°
LNN based on exp. training data						
10 s	RMSE ± STD	1.962° ± 1.53°	1.931° ± 0.344°	3.348° ± 0.183°	3.495° ± 0.175°	3.482° ± 0.203°
100 s	RMSE ± STD	**0.011° ± 0.003°**	**1.577° ± 0.317°**	**2.076° ± 0.190°**	**1.831° ± 0.338°**	**2.085° ± 0.193°**
Baselines / others						
Model	RMSE ± STD	**0.011° ± 0.003°**	2.399° ± 0.530°	**2.076° ± 0.191°**	2.224° ± 0.162°	2.133° ± 0.195°
PINN [23]	RMSE ± STD	n.a.	2.361° ± 0.214°	2.520° ± 0.668°	2.359° ± 0.214°	2.359° ± 0.214°
DNN	RMSE ± STD	n.a.	unst.	3.333° ± 0.416°	3.026° ± 0.336°	3.799° ± 0.234°
RN [24]	RMSE ± STD	n.a.	unst.	2.452° ± 0.267°	2.594° ± 0.309°	2.565° ± 0.225°

G. Payload Adaptation

To assess adaptability, a payload of 3.5 kg, nearly equal to the arm's own mass 3.6 kg, is attached to the end-effector, causing a strong change in the inertial and gravitational dynamics of the system. The pre-trained LNN used in the previous experiments is fine-tuned on the data collected under this new configuration. In ten trials, the controller initially failed to produce meaningful torques and collapsed, but recovered stable tracking within 10 s of additional data. Fig. 9 shows one of the trials. The adapted LNN-CTC reduced the RMSE from above 40° to below 5°, restoring smooth torque profiles without manual reparameterization. These results confirm the capacity of the LNN for fast and data-efficient adaptation to large structural changes, an essential property for practical manipulation tasks.

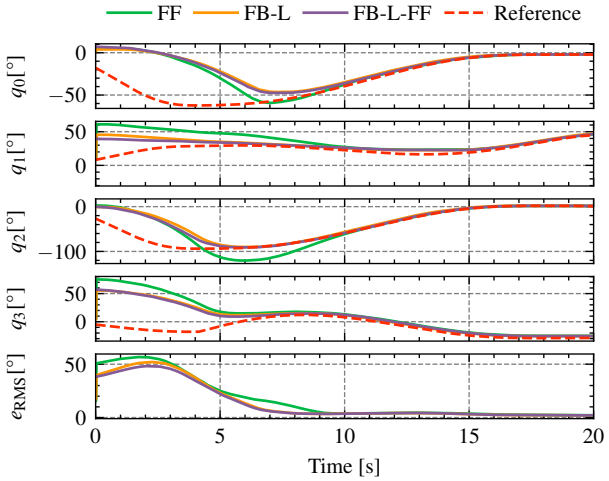


Fig. 9. With a 3.5 kg payload, nearly equal to the arm's mass, the LNN-CTC regained nominal tracking within 10 s of additional data, reducing the RMS joint-space error from over 40° to below 5° and demonstrating rapid, data-efficient adaptation to large dynamic changes.

H. Additional Robots

To examine generalization across morphologies, the feedback-linearized LNN with feedforward compensation (FB-L-FF) was evaluated on two additional robot platforms in simulation of increasing complexity: The 4-DOF Digit arm, the 6-DOF UR5 and the 7-DOF KUKA IIWA 14 with models from the MuJoCo Menagerie. As shown in Table III, all systems achieved stable tracking with limited data, and performance improved consistently with longer training durations. After only 10 s of training data, the joint-space average RMSE remained below 5° across all robots, demonstrating that coarse trend tracking can be achieved with minimal data. With 1000 s of data, the Digit arm reached 1.15°, UR5 1.54°, and KUKA IIWA 14 1.73° mean RMSE over 10 trajectories of 130 s with a similar magnitude of the desired joint velocities and joint accelerations, indicating a clear scaling relationship between model complexity, data requirements and operational space dimensionality. These results confirm that while the LNN-based controller generalizes across platforms, higher DOF systems demand proportionally more data to achieve comparable accuracy, reflecting the inherent increase in dynamic coupling and task complexity. Cross-robot results demonstrate sim-to-sim generalization across morphologies, but sim-to-real transfer on other platforms remains future work.

Overall, these results demonstrate that LNN-based CTC require sufficient training data for stable performance, with hardware-collected data consistently outperforming simulation data of equivalent duration. The FB-L-FF architecture proved to be most effective by combining learned dynamics compensation with acceleration feedforward. For 10 s of simulated training data, gravity compensation cannot maintain equilibrium and LNN-CTC tracking shows a large error (RMSE

TABLE III. Mean joint-space RMSE ± STD of over 10×130 s trajectories for the FB-L-FF controller on three robot platforms in MuJoCo simulation. Results show that coarse trend tracking is achieved with little data, while higher DOF systems require proportionally more data to reach comparable accuracy.

	Digit Arm (4-DOF)	UR5 (6-DOF)	KUKA IIWA 14 (7-DOF)
10 s	2.901° ± 1.309°	3.821° ± 1.671°	4.621° ± 1.972°
100 s	2.071° ± 0.108°	2.890° ± 0.087°	3.015° ± 0.094°
1000 s	1.146° ± 0.090°	1.541° ± 0.065°	1.730° ± 0.107°

14.3°) but small changes in architecture (FB-L/FB-L-FF) achieve better performance than the PD controller (≈3.2° to 3.5° RMSE vs. 3.454°). In the settings and platforms evaluated, an interaction window of 10 s proved to be insufficient. Achieving robustness requires interaction on the order of 100 s.

IV. Conclusion

This paper introduces and evaluates LNNs for linearized feedback control, specifically focusing on CTC. By accurately capturing real-world system behaviors from limited training data and extrapolating to OOD regimes, LNN-CTC significantly enhances control performance.

Experimental validation on a 4-DOF humanoid robotic arm confirms that LNN-CTC achieves superior trajectory tracking precision compared to conventional model-based CTC. Remarkably, this was accomplished using only 100 s of hardware-collected training data. By inherently compensating for unmodeled dynamics, LNN-based controllers outperform their model-based counterparts, demonstrating tracking accuracy improvements exceeding 30 % in certain cases. The results underline the potential of physics-informed neural architectures to offer robust, interpretable, and highly data-efficient control solutions. Such capabilities are particularly beneficial in practical robotics applications, where obtaining complete analytical system models is often infeasible due to system complexity and cost constraints.

To our knowledge, this is the first integration of LNNs into linearized feedback control schemes such as CTC. By unifying physics-informed modeling and real-time feedback control, LNN-CTC combines interpretability, robustness, and plug-and-play scalability.

Future work will apply this strategy to more complex systems and examine its scalability in dynamic robotic environments.

References

[1] S. Nahavandi, R. Alizadehsani, D. Nahavandi, C. P. Lim, K. Kelly, and F. Bello, "Machine learning meets advanced robotic manipulation," Information Fusion, vol. 105, p. 102221, 2024.

[2] Z. Zhang and Z. Chen, "Modeling and control of robotic manipulators based on symbolic regression," IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 5, pp. 2440–2450, 2023.

[3] T. N. Truong, A. T. Vo, and H.-J. Kang, "A model-free terminal sliding mode control for robots: Achieving fixed-time prescribed performance and convergence," ISA Transactions, vol. 144, pp. 330–341, 2024.

[4] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," The Int. Journal of Robotics Research, vol. 40, no. 4-5, pp. 698–721, 2021.

[5] L. P. Vincent Mai, Kaustubh Mani, "Sample efficient deep reinforcement learning via uncertainty estimation," Int. Conf. on Learning Representations (ICLR 2022), 2022. [Online]. Available: https://openreview.net/forum?id=vrW3tvDfOJQ

[6] A. R. Geist and S. Trimpe, "Structured learning of rigid–body dynamics: A survey and unified view from a robotics perspective," GAMM-Mitteilungen, vol. 44, no. 2, 2021.

[7] M. Lutter and J. Peters, "Combining physics and deep learning to learn continuous-time dynamics models," The Int. Journal of Robotics Research, vol. 42, no. 3, pp. 83–107, 2023.

[8] M. Meindl, S. Bachhuber, and T. Seel, "Ai-mole: Autonomous iterative motion learning for unknown nonlinear dynamics with extensive experimental validation," Control Eng. Practice, vol. 145, 2024.

[9] 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.

[10] 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 Int. Conf. on Robotics and Automation (ICRA). IEEE, 2024, pp. 17 293–17 299.

[11] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," in Int. Conf. on Learning Representations, 2019. [Online]. Available: https://openreview.net/forum?id=BklHpjCqKm

[12] M. D. Cranmer, S. Greydanus, S. Hoyer, P. W. Battaglia, D. N. Spergel, and S. Ho, "Lagrangian neural networks," in ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2019. [Online]. Available: https://openreview.net/forum?id=iE8tFa4Nq

[13] J. K. Gupta, K. Menda, Z. Manchester, and M. Kochenderfer, "Structured mechanical models for robot learning and control," in 2nd Conf. on Learning for Dynamics and Control, ser. Proceedings of Machine Learning Research, vol. 120. PMLR, 2020, pp. 328–337.

[14] L. Schulze, J. Peters, and O. Arenz, "Context-aware deep lagrangian networks for model predictive control," 2025.

[15] H. Sharma, D. A. Najera-Flores, M. D. Todd, and B. Kramer, "Lagrangian operator inference enhanced with structure-preserving machine learning for nonintrusive model reduction of mechanical systems," Computer Methods in Applied Mechanics and Eng., 2024.

[16] 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 Int. Conf. on Intelligent Robots and Systems (IROS). IEEE, 2019, pp. 7718–7725.

[17] S. Han, H. Wang, Y. Tian, and N. Christov, "Time-delay estimation based computed torque control with robust adaptive rbf neural network compensator for a rehabilitation exoskeleton." ISA Transactions, 2020.

[18] D. Jorge, G. Pizzuto, and M. Mistry, "Efficient learning of inverse dynamics models for adaptive computed torque control," in 2022 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2022, pp. 11 203–11 208.

[19] K. M. Lynch and F. C. Park, Modern Robotics: Mechanics, Planning, and Control, 1st ed. USA: Cambridge University Press, 2017.

[20] P. Masarati, "Computed torque control of redundant manipulators using general-purpose software in real-time," Multibody System Dynamics, 2014.

[21] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in IEEE Int. Symposium on System Integrations (SII), 2019.

[22] P. Corke and J. Haviland, "Not your grandmother's toolbox–the robotics toolbox reinvented for python," in 2021 IEEE Int. Conf. on Robotics and Automation (ICRA). IEEE, 2021, pp. 11 357–11 363.

[23] A. Al-Shahrabi, M. J. Javid, A. A. Fahmy, C. A. Griffiths, and C. Li, "Torque tracking position control of dlr-hit ii robotic hand using a real-time physics-informed neural network," Applied Mathematical Modelling, vol. 145, p. 116110, 2025.

[24] J. Gruenstein, T. Chen, N. Doshi, and P. Agrawal, "Residual model learning for microrobot control," in 2021 IEEE Int. Conf. Robotics and Automation (ICRA), 2021, pp. 7219–7226.