Scalable Physics-Embedded Neural Networks for Real-Time Robotic Control in Embedded Systems

Zhiwei Zhong
Department of Electrical and Computer Engineering
Northwestern University
Evanston, USA
zhiweizhong2021@u.northwestern.edu

Yuhao Ju
Department of Electrical and Computer Engineering
Northwestern University
Evanston, USA
yuhaoju2017@u.northwestern.edu

Jie Gu
Department of Electrical and Computer Engineering
Northwestern University
Evanston, USA
jgu@northwestern.edu

2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-8717-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MWSCAS60917.2024.10658960

***Abstract*—Physics-embedded neural networks have recently gained significant interest in robotics due to the benefits of combining data-driven machine learning approaches with physics-based modeling methods for real-time control. Despite the improved accuracy over black-box neural networks, existing works have limitations in handling large ranges of system parameters, extracting latent physical parameters, and meeting real-time latency constraints. This paper proposes enhanced physics-embedded neural network models that overcome the scaling limitation of existing models and coupling issues in the extraction of hidden variables, rendering significantly improved model accuracy by more than 95%. A reinforcement learning based neural architecture search engine is developed to meet real-time latency constraints in embedded microprocessors, optimize the solution for scaling issues, and enable the efficient deployment of physics-embedded neural networks into resource-limited edge devices, with 3X searching speed compared with the exhaustive random search method.**

Keywords—physics-embedded neural networks, real-time control, embedded applications, robot dynamics

I. INTRODUCTION

The development of intelligent industrial or humanoid robots has experienced tremendous growth in recent years. One of the leading efforts in robotics is the real-time control of multiple degrees of freedom (DoF) robotic arms, which requires accurate modeling of system physical parameters, low-latency computation, and accountability of variation of latent variables. Conventional robot dynamics are derived based on "first-principle" physics models under precise knowledge of the system. Inaccurate modeling of the dynamic system leads to deviation of movement trajectory, loss of motor efficiency, and reduction of system stability. However, it is challenging to precisely model robot dynamics due to manufacturing tolerance and latent system variables. To overcome this problem, researchers show that machine learning algorithms outperform physics-based methods in complex real-world environments [1]–[3]. However, machine learning based models suffer from requirements of large datasets, poor extrapolation ability, instability, and violation of physical principles.

Recently, the physics-embedded neural network (PENN) has been rapidly developed for existing cyber-physical systems. It combines data-driven machine learning algorithms with "first principle" based domain knowledge to resolve complex real-life problems. In robot dynamics, a significant amount of effort has been spent on using PENN for accurate control of multi-DoF robotic arms. Hamilton neural network (HNN) was proposed to capture the Newtown physics of mechanical systems [4]. Lagrangian neural network (LNN) was proposed to learn robotic dynamics for prediction [5] and control [6]. SIMENS developed an LNN model for a robotic system with improved model accuracy while obeying critical physics laws [7]. Fig. 1 shows the setup of an industrial robotic arm and its related system, where the PENN takes the current robot states as input and sends the estimation of the robot dynamics to a proportional-derivative (PD) controller for control purposes.

There are still missing elements for existing PENN. First of all, all existing works are based on a single configuration of mechanical settings with a limited range of operation, missing the consideration of a large range of system parameters in real-world robots. Second, there is a lack of training methods for the accurate back-tracing of system parameters. Third, the deployment of PENN models into resource-limited embedded systems for real-time applications is missing. This paper addresses the above issues by making the following contributions: (a) A scaling scheme is proposed to achieve a stable, robust, and accurate PENN model for a wide range of system parameters; (b) An extraction and training method is proposed to allow accurate extraction of system parameters including latent variables, resulting in enhanced prediction accuracy; (c) To meet the latency constraints for real-time robot control, a reinforcement learning (RL) based neural architecture search engine was developed to enable implementation of optimized PENN model on resource-limited embedded systems.

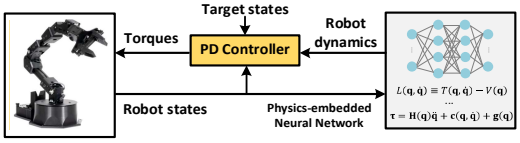


Fig. 1. System configuration of robotic control with physical models combined with physics-embedded neural networks.

II. PHYSICS-EMBEDDED NEURAL NETWORK MODELS

A. Robot Dynamics and Lagrangian Mechanics

Robot dynamics refers to the relationship between forces applied to robotic systems and the stimulated acceleration. Manipulation of a rigid-body robot with n joints is governed by:

$$\boldsymbol{\tau} = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) \quad (1)$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are $n \times 1$ vectors referring to joint position, speed, and acceleration, respectively. $\mathbf{H}(\mathbf{q})$ is the symmetric and positive definite $n \times n$ mass matrix; $\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}}$ stands for inertia force; $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$ is the velocity-product term $(n \times 1)$ and

$\mathbf{g}(\mathbf{q})$ stands for the joint torques ($n \times 1$) caused by gravity; $\boldsymbol{\tau}$ is an $n \times 1$ vector representing torques applied to each joint. Lagrangian mechanics (2) describes system energy, where $T = 1/2\,\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})\dot{\mathbf{q}}$ is the kinetic energy and V is the potential energy with $dV/d\mathbf{q} = \mathbf{g}(\mathbf{q})$.

$$L(\mathbf{q}, \dot{\mathbf{q}}) \equiv T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}) \tag{2}$$

Once a robotic system is expressed by (2), its dynamics could be derived by using the Euler-Lagrange equation:

$$\tau_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} \tag{3}$$

where i is the index of robot joints. Replacing L in (3) with $T(\mathbf{q}, \dot{\mathbf{q}})$ and $V(\mathbf{q})$ yields the expression:

$$\boldsymbol{\tau} = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}(\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})\dot{\mathbf{q}})\right)^T + \mathbf{g}(\mathbf{q}) \tag{4}$$

The velocity-product term (5) represents torques generated by Coriolis and Centripetal forces, where $\dot{\mathbf{H}}(\mathbf{q}) \equiv \boldsymbol{\partial}\mathbf{H}(\mathbf{q})/\boldsymbol{\partial t}$.

$$\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}(\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})\dot{\mathbf{q}})\right)^T \tag{5}$$

B. Lagrangian Neural Network

To learn the dynamics of a robot system, a common choice is to train a black-box model (e.g., a multilayer perceptron) using data $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$ measured from robot joints, which ignores the underlying physics and thus suffers from lower accuracy when extrapolated into unseen data ranges. To solve this problem, LNN models are proposed [6]. By incorporating Lagrangian mechanics into neural networks, LNN has demonstrated advantages over black-box models in learning robot dynamics in terms of efficiency and accuracy [5] [6].

In the LNN, a feed-forward neural network is used to estimate $\mathbf{H}(\mathbf{q})$ and $\mathbf{g}(\mathbf{q})$ in (4), as shown in Fig. 2(a). Joint position $\mathbf{q}$ is the input to the neural network. $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and the outputs of the neural network are inputs to the mathematic transformation, yielding the estimated torque $\hat{\boldsymbol{\tau}}$**.** The derivatives of $\mathbf{H}(\mathbf{q})$ with respect to $\mathbf{q}$ ($\partial\mathbf{H}/\partial\mathbf{q}$) and time ($\dot{\mathbf{H}}$) are achieved by introducing customized neurons [6]. Since $\mathbf{H}(\mathbf{q})$ is a symmetric and positive definite matrix, the outputs of the neural network are set to be a lower-triangular matrix $\mathbf{L}(\mathbf{q})$ so that $\mathbf{H}(\mathbf{q}) = \mathbf{L}(\mathbf{q})\mathbf{L}(\mathbf{q})^{\mathbf{T}}$, which conforms to the physics feature that the kinetic energy $(1/2\,\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})\dot{\mathbf{q}})$ always being non-negative.

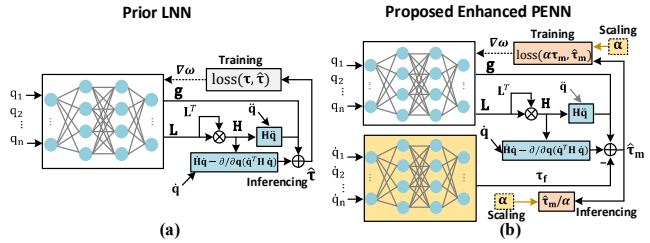


Fig. 2. Architecture and dataflow of (a) the prior LNN [6] and (b) the proposed enhanced PENN with scalability and friction extraction.

III. Proposed PENN Model with Scalability

A. Physical Limitation

The mass matrix $\mathbf{H}$ and gravity torque $\mathbf{g}$ are functions of not only joint angles $\mathbf{q}$ but the mass, center of mass, and three-dimensional shapes of each part of a robot, making it impractical to derive closed-form expressions when encountering complex robotic systems. However, numerical values of $\mathbf{H}$ and $\mathbf{g}$ can be learned from data samples $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$ measured from motions of robots. By minimizing $loss(\boldsymbol{\tau}, \hat{\boldsymbol{\tau}})$ using gradient descent techniques, LNN can learn and estimate the hidden $\mathbf{H}$ and $\mathbf{g}$ without seeing their ground truth according to robot dynamics in (4). However, the numerical range of $\mathbf{H}$ and $\mathbf{g}$ in different robots could have large variances due to diverse physical configurations, resulting in low training efficiency and accuracy. We experiment with seven robots with different $\mathbf{H}$ and $\mathbf{g}$ to validate this issue. The robot models are created by MATLAB Robotics System Toolbox, controlled by a *Computed Torque Controller*, and simulated using ode45 ODE solver.

Table I shows the error of $\hat{\boldsymbol{\tau}}$ using the prior LNN to learn the dynamics of the seven 3-DoF robot models with a diverse range of $\mathbf{H}$ and $\mathbf{g}$. Take *Robot-a* as an example, the lengths of all its links are 0.1 meters; the mass of the links are 0.175 kg, 0.15 kg, and 0.125 kg, respectively; elements of $\mathbf{H}$ and $\mathbf{g}$ are of magnitude 10^{-3} in average; error of $\hat{\boldsymbol{\tau}}$ is the highest at 96.3%. Values of $\mathbf{H}$ and $\mathbf{g}$ of *Robot-b* to *Robot-g* increase gradually to about 8000 times of *Robot-a*. Table I shows when the $\mathbf{H}$ or $\mathbf{g}$ are of near-zero values, the error of $\hat{\boldsymbol{\tau}}$ is the largest (96.3%). This is because the near-zero ground truth prevents the network weights from updating significantly during optimization.

TABLE I. Error of Estimated Torques of Seven 3-DoF Robots

Robot	*a*	*b*	*c*	*d*	*e*	*f*	*g*
magnitude	0.001	0.044	0.02	0.09	0.41	1.78	8
Error	96.3%	34.1%	10.9%	0.37%	0.9%	2.28%	6.38%
α	1000	150	30	10	0.5	0.3	0.2
$Error_\alpha$	0.62%	0.57%	0.65%	0.79%	0.55%	0.65%	0.69%

Conventionally, normalization and standardization are used to preprocess data in machine learning tasks. However, PENN represents physical characteristics of robot systems, e.g., mass matrix. Any processing methods on input data might break the physical relationship between inputs and outputs of the model, making it fail to represent the real-world physics system. In addition, the data range and distribution of many physical features are hard to measure (e.g., $\mathbf{H}$ and $\mathbf{g}$). Therefore, any preprocessing on input and output data of the PENN models needs to be scrutinized based on the physical impacts.

B. Proposed PENN Model Overcoming Scaling Issue

In this section, we propose to enhance the PENN model by scaling hidden features of robots ($\mathbf{H}$ and $\mathbf{g}$). For any robot with n links, its mass matrix could be expressed by:

$$\mathbf{H}(\mathbf{q}) = \sum_i^n [\boldsymbol{J}_{ib}^T(\mathbf{q}) \cdot \boldsymbol{\delta}_i \cdot \boldsymbol{J}_{ib}(\mathbf{q})] \tag{6}$$

where $\boldsymbol{J}_{ib}(\mathbf{q})$ is the body Jacobian (irrelevant to mass), and $\boldsymbol{\delta}_i$ is the spatial inertia matrix of i-th link in a linear relationship to a 3×3 inertia matrix $\boldsymbol{I}_b$ of the link. Elements in $\boldsymbol{I}_b$ conform to:

$$\int_B \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\boldsymbol{\rho}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x})\boldsymbol{dV} \tag{7}$$

where $\boldsymbol{\rho}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x})$ is the mass density function, $\boldsymbol{B}$ refers to the body of the link and $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ is related to distance. Therefore, each element of $\mathbf{H}$ is in a linear relationship to the mass density of the robot. Similar properties also apply to $\mathbf{g}$. Hence, keeping the other configurations of a robot unchanged, linear scaling of the robot's mass or mass density leads to the linear scaling of $\mathbf{H}$ and $\mathbf{g}$. This also leads to the linear scaling of $\boldsymbol{\tau}$ since it is in a linear relationship to $\mathbf{H}$ and $\mathbf{g}$ according to (4).

Hence, given a dataset $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$ measured from a robot, scaling $\boldsymbol{\tau}$ by a factor $\boldsymbol{\alpha}$ leads to a new dataset $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\alpha} \times \boldsymbol{\tau})$ which conforms to the dynamics of a new robot whose mass density is $\boldsymbol{\alpha}$ times of the former one and the other physical features remain identical. The robot dynamic (4) thus becomes:

$$\boldsymbol{\alpha} \times \boldsymbol{\tau} = \mathbf{H}'(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}'(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}(\dot{\mathbf{q}}^T\mathbf{H}'(\mathbf{q})\dot{\mathbf{q}})^T + \mathbf{g}'(\mathbf{q})\right) \tag{8}$$

where $\mathbf{H}'(\mathbf{q})$ and $\mathbf{g}'(\mathbf{q})$ are physical features of the new robot. When a PENN fails to learn from a dataset $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$ of a real-world robot due to the improper range of hidden physical parameters, we could scale $\boldsymbol{\tau}$ by $\boldsymbol{\alpha}$, mapping the current robot to a virtual one with $\boldsymbol{\alpha}$ times mass density. The enhanced PENN is shown in Fig. 2(b) (the extra neural network for friction $\boldsymbol{\tau}_\mathbf{f}$ prediction will be introduced in Section IV). After training, the output of PENN model will be scaled by $\boldsymbol{\alpha}^{-1}$ to generate target torques (9) for real-time control. Since the ground truth of $\mathbf{H}$ and $\mathbf{g}$ are unknown, the proper value of $\boldsymbol{\alpha}$ cannot be determined in advance. Thus, an automatic search mechanism is required.

$$\hat{\boldsymbol{\tau}} = \boldsymbol{\alpha}^{-1}[\mathbf{H}'(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}'(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}(\dot{\mathbf{q}}^T\mathbf{H}'(\mathbf{q})\dot{\mathbf{q}})\right)^T + \mathbf{g}'(\mathbf{q})\Big)] \quad (9)$$

C. Experimental Results

The proposed scaling solution could be applied to any rigid-body robot. We tested it on seven 3-DoF robot arms in Table I. Dataset $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$ is sampled from the simulation of motions of robots. The feed-forward neural network in the PENN shown in Fig. 2(b) has 4 hidden layers with 64 neurons in each layer (the friction extraction NN is not used). After applying different scaling factors $\boldsymbol{\alpha}$, Error_α of all robots decrease to less than 0.8%. For *Robot-a*, three curves of $\hat{\boldsymbol{\tau}}$ lasted for seven seconds are shown in Fig. 3. For each joint, PENN with $\boldsymbol{\alpha} = 1$ has the largest deviation from the ground truth. For $\boldsymbol{\alpha} = 10$, the deviation is greatly reduced. As $\boldsymbol{\alpha}$ increase to 1000, the error is reduced by more than 95% (from 96.3% to 0.62%). A reinforcement learning based method is proposed in Section V to search for the optimized value of the scaling factor $\boldsymbol{\alpha}$.

IV. Extraction of Latent Variables

A. Latent Features and Coupling Issue

Typically, torques applied on robot joints come from joint motors, gravity, and friction, which could be reformulated as:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_\mathbf{m} + \boldsymbol{\tau}_\mathbf{f} = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) \quad (10)$$

where $\boldsymbol{\tau}_\mathbf{m}$ and $\boldsymbol{\tau}_\mathbf{f}$ refer to the torques from joint motors and friction, respectively. The value of $\boldsymbol{\tau}_\mathbf{m}$ is known since it is determined by controllers. However, $\boldsymbol{\tau}_\mathbf{f}$ is a latent variable with unknown values determined by complex system characteristics, such as lubrication, temperature, and manufacturing tolerance. Therefore, in real-world applications, the available data would be $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau}_\mathbf{m})$ ($\boldsymbol{\tau}_\mathbf{f}$ is missing). In this paper, as an example, we consider the viscous friction model $\boldsymbol{\tau}_\mathbf{f} = -\boldsymbol{\beta} \circ \dot{\mathbf{q}}$ where $\boldsymbol{\beta}$ is the friction coefficient and $\circ$ is elementwise multiplication [8]. Note that latent variables are not limited to this friction model.

To solve this issue, an additional neural network is developed in the proposed PENNs to capture the latent feature and the dynamics is expressed by (11), where $\mathbf{f}(\dot{\mathbf{q}})$ is an introduced neural network that is used to estimate friction torque $\boldsymbol{\tau}_\mathbf{f}$. Fig. 2(b) shows the proposed enhanced PENN model that includes the scaling operation, the neural network for of $\mathbf{H}$ and $\mathbf{g}$, and the introduced neural network for friction. However, since there are two neural network in the PENN, and their optimization depends on one single function $loss(\boldsymbol{\tau}_\mathbf{m}, \hat{\boldsymbol{\tau}}_\mathbf{m})$, the trained PENN may suffer from mutually canceling issues, i.e., the outputs of two neural networks could both have large deviation from the corresponding ground truth while the linear combination of them compensates the loss. This is because the loss function could only consider the available ground truth $\boldsymbol{\tau}_\mathbf{m}$,

$$\hat{\boldsymbol{\tau}}_\mathbf{m} = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}(\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})\dot{\mathbf{q}})\right)^T + \mathbf{g}(\mathbf{q}) - \mathbf{f}(\dot{\mathbf{q}}) \quad (11)$$

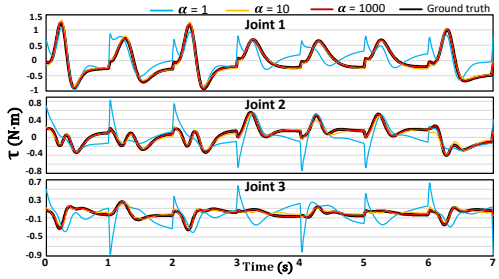


Fig. 3. Estimation of torque $\boldsymbol{\tau}$ on joints of a 3-DoF robot (*Robot-a*) with PENN trained with scaling factors 1, 10, and 1000.

$\mathbf{g}(\mathbf{q})$, or $\mathbf{f}(\dot{\mathbf{q}})$ is unknown. Though the overall estimation error of $\hat{\boldsymbol{\tau}}_\mathbf{m}$ could be small, it is of high probability that the PENN would fail to discover the latent parameters of the system.

B. Proposed PENN Model for Latent Features Extraction

The proposed PENN contains neural networks to estimate $\mathbf{H}(\mathbf{q})$, $\mathbf{g}(\mathbf{q})$, and $\boldsymbol{\tau}_\mathbf{f}$, respectively, as shown in Fig. 2(b). The coupling issue mostly comes from the prediction of $\mathbf{g}(\mathbf{q})$ and $\boldsymbol{\tau}_\mathbf{f}$. As for $\mathbf{H}(\mathbf{q})$, it is multiplied by $\ddot{\mathbf{q}}$ in (11), and robot states with the same $\mathbf{q}$ and $\dot{\mathbf{q}}$ could be in different $\ddot{\mathbf{q}}$. Consider the below error analysis equations (12) (Δ refers to error) with $\ddot{\mathbf{q}}_\mathbf{b} \neq \ddot{\mathbf{q}}_\mathbf{c}$, the only condition for both equations holding true is that $\Delta\mathbf{H}(\mathbf{q}_\mathbf{a}) = 0$, which means different values of $\ddot{\mathbf{q}}$ could help $\mathbf{H}(\mathbf{q})$ escape from the coupling issue.

$$\begin{aligned}\Delta\mathbf{H}(\mathbf{q}_\mathbf{a})\ddot{\mathbf{q}}_\mathbf{b} + \Delta\mathbf{c}(\mathbf{q}_\mathbf{a}, \dot{\mathbf{q}}_\mathbf{a}) + \Delta\mathbf{g}(\mathbf{q}_\mathbf{a}) - \Delta\mathbf{f}(\dot{\mathbf{q}}_\mathbf{a}) = 0 \\ \Delta\mathbf{H}(\mathbf{q}_\mathbf{a})\ddot{\mathbf{q}}_\mathbf{c} + \Delta\mathbf{c}(\mathbf{q}_\mathbf{a}, \dot{\mathbf{q}}_\mathbf{a}) + \Delta\mathbf{g}(\mathbf{q}_\mathbf{a}) - \Delta\mathbf{f}(\dot{\mathbf{q}}_\mathbf{a}) = 0 \end{aligned} \quad (12)$$

To solve the coupling issue between $\mathbf{g}(\mathbf{q})$ and $\mathbf{f}(\dot{\mathbf{q}})$ and extract the latent variables, we propose another enhanced model named TS-PENN that contains three neural networks for $\mathbf{H}(\mathbf{q})$, $\mathbf{g}(\mathbf{q})$, and $\mathbf{f}(\dot{\mathbf{q}})$, respectively. A two-step (TS) training flow is proposed for TS-PENN, as shown in Fig. 4. In the first step, data samples with zero speed and acceleration are picked from dataset $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau}_\mathbf{m})$. These data are used to train the neural network for $\mathbf{g}(\mathbf{q})$ since it is the only term that is irrelevant to $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$. With $\dot{\mathbf{q}} = 0$ and $\ddot{\mathbf{q}} = 0$, robot dynamics in (11) is simplified to $\hat{\boldsymbol{\tau}}_\mathbf{m} = \mathbf{g}(\mathbf{q})$. In the second step, the remaining two neural networks for $\mathbf{H}(\mathbf{q})$ and $\mathbf{f}(\dot{\mathbf{q}})$ are trained with the rest of the dataset. Since the weights of the neural network $\mathbf{g}(\mathbf{q})$ have been optimized in the first step, their values are not updated anymore. Because the optimization of neural networks for $\mathbf{g}(\mathbf{q})$ and $\mathbf{f}(\dot{\mathbf{q}})$ are not proceeding simultaneously, the coupling issue is eliminated. The scaling factor is still applicable to TS-PENN since $\mathbf{f}(\dot{\mathbf{q}})$ is a linear additive term in (11).

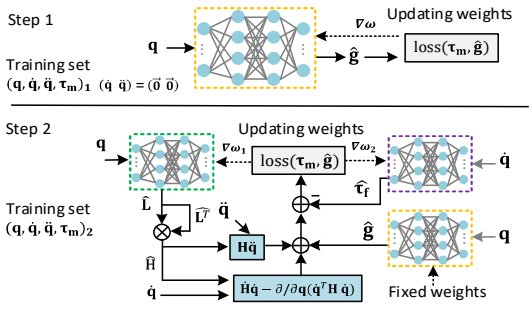


Fig. 4. The two-step training flow for the proposed TS-PENN.

C. Experimental Results

We evaluate the prior LNN in Fig. 2(a), the enhanced PENN in Fig. 2(b), and the TS-PENN by learning the dynamics of a 3-DoF robot that considers friction and comparing their performance on estimation of latent variables. For the LNN, its neural network consists of 4 hidden layers and each layer has 96 neurons. For both PENN and TS-PENN, the neural network to estimate friction has 3 hidden layers and each layer has 32 neurons. In PENN, two neural networks are optimized at the same time using the entire dataset; the neural network for $\mathbf{H}(\mathbf{q})$ and $\mathbf{g}(\mathbf{q})$ consists of 4 hidden layers and each has 96 neurons. In TS-PENN, the neural network for $\mathbf{H}(\mathbf{q})$ consists of 4 hidden layers and each has 64 neurons; the neural network for $\mathbf{g}(\mathbf{q})$ also has 4 hidden layers and each contains 32 neurons. Thus, PENN and TS-PENN have the same number of parameters.

The accuracy of the above models is evaluated by controlling the motion of the robot for 50 seconds in a robot system. Fig. 5(a) shows that LNN has the highest mean absolute error (MAE) in the estimation of each latent torque component (except $\mathbf{g}(\mathbf{q})$). The reason is that the LNN conforms to robot dynamics in (4) of which the equality is broken by the introduced friction, making it fail to precisely capture any dynamics of the robot system. PENN and TS-PENN conform to robot dynamics in (11) that takes friction into consideration, resulting in higher accuracy in terms of $\boldsymbol{\tau}_\mathbf{m}$, $\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}}$, and $\mathbf{c}(\mathbf{q},\dot{\mathbf{q}})$ compared with LNN. However, PENN has the largest MAE for $\mathbf{g}(\mathbf{q})$ and $\mathbf{f}(\dot{\mathbf{q}})$ due to the coupling issue, though it has the lowest MAE of 0.044 $\mathrm{N}\cdot\mathrm{m}$ in the estimation of $\boldsymbol{\tau}_\mathbf{m}$. On the other hand, TS-PENN has solved the coupling issue and thus has the smallest MAE in all the torque components except that its MAE of $\boldsymbol{\tau}_\mathbf{m}$ is slightly higher than the PENN model by 0.073 $\mathrm{N}\cdot\mathrm{m}$. Its MAEs of $\mathbf{f}(\dot{\mathbf{q}})$ and $\mathbf{g}(\mathbf{q})$ are reduced by ~99% compared with that of PENN thanks to the two-step training flow. The latent torques estimated by PENN and TS-PENN on the second robot joint are shown in Fig. 5(b) and Fig. 5(c), which illustrates that estimated values of $\mathbf{f}(\dot{\mathbf{q}})$ and $\mathbf{g}(\mathbf{q})$ by PENN both have about -5 $\mathrm{N}\cdot\mathrm{m}$ deviation from the ground truth. According to the dynamics equation (11), $\mathbf{f}(\dot{\mathbf{q}})$ is subtracted from $\mathbf{g}(\mathbf{q})$, resulting in the cancellation of errors.

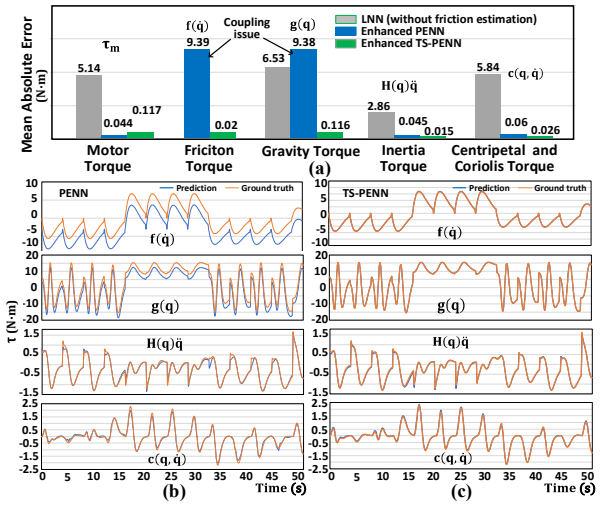


Fig. 5. (a) Mean absolute error comparison of different variables estimated by LNN, PENN, and TS-PENN. Waveforms of estimated latent variables on the second joint by (b) PENN and (c) TS-PENN.

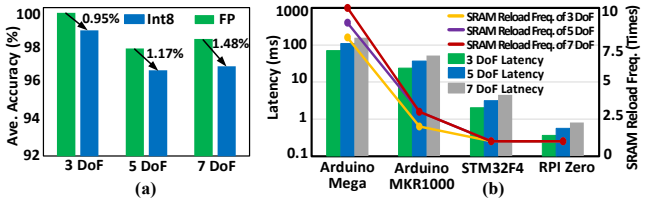


Fig. 6. (a) Average accuracy degradation of 8-bit quantization and (b) latency estimation for 3, 5, and 7-DoF robots by four microcontrollers.

V. Implementation of PENN with Neural Architecture Search

A. Neural Architecture Search for Real-time Control Systems

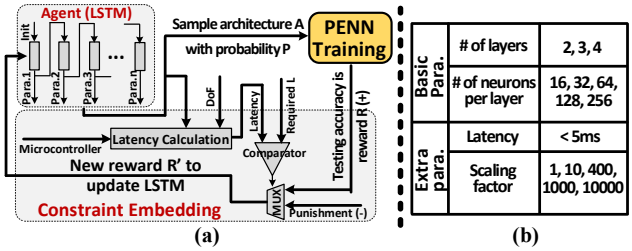
Due to area, power limitations and timing constraints of embedded microprocessors, quantization is typically performed on neural network models for real-time applications. We evaluate 8-bit integer quantization impact on PENN using four low-cost embedded microprocessors (Arduino Mega, MKR1000, STM32F4, and RPI Zero) with 3, 5, and 7-DoF robots. The PENN used for quantization and latency calculation has 4 fully connected layers with 64 neurons in each layer and the scaling factor $\boldsymbol{\alpha}$ = 400. As shown in Fig. 6(a), quantization from 64-bit floating point precision to 8-bit integer only leads to 0.95 ~ 1.48% accuracy degradation for real-time inference of motor torques. For the practical deployment of PENN in embedded systems aimed at real-time control, below 5ms latency is required due to the motor PWM control update rates. Fig. 6(b) illustrates the estimated execution latency and memory reload frequency in different microprocessors. For Arduino Mega (73mW, 8KB SRAM) and MKR1000 (32KB SRAM), the PENN model requires 10~100ms operation latency and 7-10 rounds of memory reload. STM32F4 (140mw, 128KB SRAM) and RPI Zero (449mW, 512MB SRAM) can achieve 0.4~5ms latency without data reloading. The developed PENN models are expected to consider the computational resource of the embedded system, which could be addressed by the neural architecture search (NAS) scheme. However, not all the hyperparameters of PENN can be inserted into existing NAS methods. For example, DARTS [9], MCTS [10], and Bayesian optimization [11] require a fixed sequence of feature maps. Evolution [12] and SMBO [13] are cell-wise learning with a fixed number of cell stacks. For real-time robot control, the PENN size should be relatively small but needs flexibility for constraints from embedded systems. Besides, the proposed scaling factor $\boldsymbol{\alpha}$ is also one optimization target. Therefore, reinforcement learning based searching methods [14] are most suitable for PENN. In this section, we proposed an RL-based NAS engine for PENN models. Compared with existing works, the proposed NAS engine considers the scaling factor and the real-time latency constraint of selected microprocessors.

In the proposed RL-based NAS engine shown in Fig. 7(a), an LSTM is employed as a search agent that generates the hyperparameters for PENN models. Exemplary hyperparameter candidates for a 3-DoF robot are illustrated in Fig. 7(b), including the number of layers, the number of neurons in each layer, latency constraint, and the scaling factors. The list of tokens predicted by LSTM can be named by an action list. In an exemplary action list [16, 32, 32, 0, 1000], the first four parameters indicate the number of neurons in four layers in order (0 means the fourth layer is not needed), and the last parameter 1000 stands for the scaling factor. PENN is a trainable children network specified by the action list and its testing accuracy after

latency constraint embedding is the reward for updating the LSTM to get better architecture. The reinforcement rule [14] to make LSTM get the maximum expected reward is represented by the $J(\theta_c)$ in (13) with the gradient method in (14), where θ_c is the LSTM weights and biases, $a_{1:T}$ is the action list, T is the number of hyperparameters, R' is the reward from PENN after latency constraint embedding, and P is the probability.

$$J(\theta_C) = E_{P(a1:T;\,\theta C)}[R'] \quad (13)$$

$$\nabla_{\theta C} J(\theta_C) = \sum_{t=1}^{T} \nabla_{\theta C} \log P(a_t | a_{(t-1):1};\, \theta_C)\, R' \quad (14)$$



(a) (b)

Fig. 7. (a) RL-based search engine (b) hyperparameter candidates.

In the proposed NAS engine, the real-time constraint is added to limit the latency of embedded systems. The constraint is done by adding (15) into the reward function. $-\alpha$ is a negative value as the punishment for the LSTM agent. $La(a1{:}T)$ is the approximation of the latency of microprocessors. If the current latency is smaller than the required latency L, the error e combined with a scaling parameter λ is a positive reward to the LSTM gradients. If the current latency is larger than the requirement L, the punishment $-\alpha$ will impact the LSTM optimization. The latency approximation is shown in (16), which is related to the picked architecture, the implemented microprocessor, and the DoF (D). f is the clock speed, and γ is the run cycles of multiplication for the selected microprocessor.

$$R' = \begin{cases} -\alpha & if\ La(a1{:}T) > L \\ 100 - \lambda e & Otherwise \end{cases} \quad (15)$$

$$La(a1{:}T) = 1.005\gamma(2D+3)[Da_1 + \sum_{t=1}^{T-1} a_t a_{t+1} + (D^2/2 + 3D/2)a_T] f^{-1} \quad (16)$$

B. Experimental Results

Fig. 8 shows the traces of optimized PENN architecture with different search engines. Blue dots represent the exhaustive search results, which show that the error grows quickly if the PENN size is too large or too small (reflected by latency). The baseline NAS can achieve a better accuracy result by jumping search between the large and small PENN. With the latency constraint, the NAS also uses jumping search at the beginning, but the searching space will be limited to 5ms in the later stage. Fig. 9(a) shows the results of NAS for a 3-DoF robot with PENN on STM32F4. The proposed NAS achieves the same error rate compared with the exhaustive random search but with 3 times faster searching speed. For the scaling factor trace, NAS starts searching from 10000, passes through 400, and finally reaches 1000 as the best result. Fig. 9(b) shows the summary of the benefits of latency constraint and scaling. The latency decreases from 8.58ms to 4.98ms and the error drops from 2.37% to 0.56% by using better scaling factors. Essentially, the proposed NAS methods have achieved the optimized solution without accuracy loss while being able to meet the tight latency constraints.

VI. Conclusions

In this paper, we propose enhanced PENN models with the scaling scheme and the two-step training flow, which enables PENN models to learn real-world robot systems with a wide range of hidden physical parameters and perform extraction of latent parameters accurately (reduce estimation error by more than 95% for both torques and latent physical variables) without breaking the interior physics law. The NAS engine is also proposed to search for low-latency and accurate PENN in resource-limited embedded systems, with 3X searching speed compared with the exhaustive random search method.

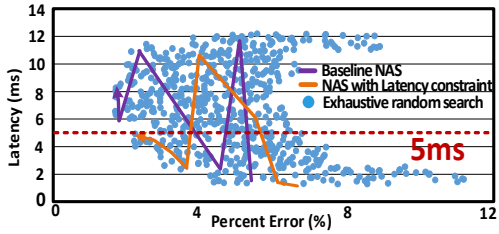


Fig. 8. The searching trace of exhaustive search, baseline NAS, and the NAS with 5ms latency constraint for a 3-DoF robot on STM32F4.

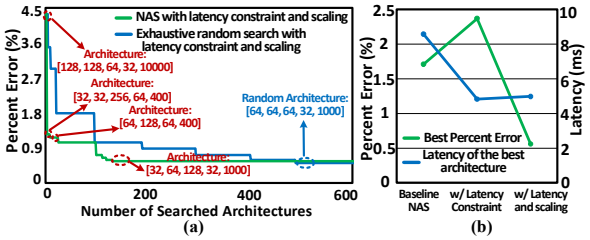


Fig. 9. (a) Error trace of NAS considering scaling factor compared with exhaustive search and (b) the benefits summary with latency constraint and scaling on STM32F4 for a 3-DoF robot.

Acknowledgment

This work was supported by the National Science Foundation under grant number CCF-1846424.

References

[1] A. Sarabakha and E. Kayacan, "Online deep learning for improved trajectory tracking of unmanned aerial vehicles using expert knowledge," in *IEEE International Conference on Robotics and Automation,* 2019.

[2] J. Muliadi and B. Kusumoputro, "Neural network control system of UAV altitude dynamics and its comparison with the PID control system," *J. Adv. Transp.*, vol. 2018, pp. 1–18.

[3] Q. Li, J. Qian, et al. "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5183–5189.

[4] Y. D. Zhong, et al. "Symplectic ODE-Net: Learning hamiltonian dynamics with control," in *International Conference on Learning Representations (ICLR)*, 2019.

[5] M. Cranmer, et al. "Lagrangian neural networks," in *ICLR Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[6] M. Lutter, et al. "Deep lagrangian networks: using physics as model prior for deep learning," in *International Conference on Learning Representations (ICLR)*, 2018.

[7] M. A. Roehrl, et al. "Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics," *IFAC-Pap.*, vol. 53, no. 2, pp. 9195–9200, 2020.

[8] A. Wahrburg, et al. "Motor-current-based estimation of cartesian contact forces and torques for robotic manipulators and its application to force control," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 879–886, 2018.

[9] H. Liu, et al. "DARTS: Differentiable architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

[10] R. Negrinho and G. Gordon, "DeepArchitect: Automatically designing and training deep architectures." 2017. doi: 10.48550/arXiv.1704.08792.

[11] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Advances in Neural Information Processing Systems*, 2018.

[12] E. Real, et al. "Regularized evolution for image classifier architecture search," *Proc. AAAI Conf. Artif. Intell.*, vol. 33, pp. 4780–4789, 2019.

[13] C. Liu *et al.*, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[14] B. Zoph et al. "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations,* 2017.