Learning by doing: Online Learning to Compensate Gravity with a Computed Torque Controller using Lagrangian Neural Networks

Manuel Weiss[1,2*], Alexander Pawluchin[1], Arnold Schwarz[1], Thomas Seel[2], Ivo Boblan[1]

***Abstract*— This paper investigates a novel approach for continuous gravity compensation in robotic systems using a Lagrangian Neural Network (LNN)-based Computed Torque Controller (CTC). Specifically, we use LNNs to model the system's dynamics and adaptively improve the performance of the CTC. Unlike traditional methods relying on predefined models or extensive offline training, this approach enables the controller to learn and adapt in real-time, without prior model knowledge, and within seconds during operation. The proposed approach focuses on energy-based representations and uses Lagrangian mechanics to ensure that the learned dynamics are physically interpretable and reliable, reducing the need for extensive training datasets. The LNN-CTC utilizes an online learning mechanism to continuously update the LNN-based on real-time data, ensuring accurate gravity compensation without prior model knowledge. Real-world experiments on a humanoid robotic arm with 4 degrees of freedom show that the LNN-based online learning CTC effectively compensates for gravity and adapts to changes in mass within 25 s. We compare the proposed controller to a conventional model-based controller that relies on precise parameter knowledge and demonstrate that the LNN-CTC achieves similar trajectory tracking accuracy with significantly less data, no prior knowledge, and rapid adaptation to dynamic changes, such as added payloads without requiring parameter identification. This work contributes an adaptive real-time control framework that compensates for gravity. It overcomes high modeling efforts and large datasets, showing promise for scalability and generalization in autonomous systems and advanced robotics in uncertain environments.**

I. Introduction

Robotics has evolved into an important field with diverse applications ranging from industrial automation to healthcare and space exploration. A fundamental challenge in robotics is the precise and efficient control of robotic manipulators, especially in the presence of gravitational forces that affect the accuracy and performance of these systems [1]. Conventional control methods often rely on predetermined models of robot dynamics, which can be inadequate due to unmodeled dynamics, parameter uncertainties, and external disturbances [1], [2]. Consequently, there is a need for adaptive and robust control strategies that can compensate for these variations in real-time [3].

Various control strategies have been developed to solve the problems of dynamic modeling and gravity compensation. These approaches aim to improve system performance by accurately predicting and compensating for the effects of gravity on the system's dynamics. Model-based controllers, such as Computed Torque Control (CTC) and inverse dynamics, are the most commonly used techniques [4]. In addition, PID controllers are used because they are simple and effective for many applications but are limited in dealing with complex nonlinearities and dynamic uncertainties [5]. Gravity compensation constitutes a widely recognized methodology in robotic engineering that aims to achieve an equilibrium state throughout the range of motion, thus effectively mitigating the mechanical load imposed on actuators [1]. However, its effectiveness depends on the accuracy of the dynamic model, which is often difficult to obtain and maintain [4], [6]. Any discrepancies between the model and the actual system can lead to errors in gravity compensation, affecting the robot's performance and stability. Changes in dynamics, for example, due to altered gravitational forces of the end effector after changing tools or during manipulation tasks, require dynamic models of changes or force / torque sensors to measure changes or observers [7].

In recent years, adaptive control techniques and machine learning methods have been investigated to improve the robustness and adaptability of robot controllers. These methods aim to learn and dynamically adapt control parameters in response to changing conditions. Learning-based controllers, Reinforcement Learning (RL)- and Deep Neural Network (DNN)-based controllers have shown promise in this area, as they utilize large data sets and online learning capabilities to improve control performance [8], [9], [10], [11]. However, these approaches often require significant computational resources and long training times, which can be impractical for online learning in real-time applications [12]. Lagrangian Neural Networks (LNN) represent a significant advance in this area by incorporating the principles of Lagrangian mechanics into the learning process [13]. This integration ensures that the learned models adhere to fundamental physical laws, such as the conservation of energy and momentum, while accurately representing the system dynamics. By embracing these principles, the models generated not only comply with essential physical laws such as energy and momentum conservation, but also effectively capture the intricate dynamics of the system [13], [14].

In this paper, we propose a novel approach that combines online learning with LNNs to harness the power of a CTC to compensate for gravity with little data and no model knowledge. The core of the proposed approach is the integration of LNNs, which are designed to learn the Lagrangian dynamics of the robotic system directly from the data. Unlike

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

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

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Project-ID 528483508 - FIP 12. Corresponding author {mweiss}@bht-berlin.de

traditional DNNs, LNNs inherently respects the physical laws governing the system, resulting in more accurate and physically consistent models [13]. Integrating LNNs within the CTC framework facilitates the continual refinement of the dynamic model, achieving a level of accuracy comparable to model-based approaches while enhancing the controller's adaptability to varying conditions and payloads, thereby augmenting both precision and robustness.

For instance, in industrial settings, robots often encounter varying payloads and unforeseen disturbances that can compromise their performance without using external force sensors or internal observations based on torques inside the joint. The LNN-CTC can adapt to these changes on the fly, ensuring consistent and reliable operation. Similarly, in autonomous exploration where robots navigate unpredictable environments, the LNN-CTC appears to offer the adaptability that could assist in maintaining control and supporting the achievement of mission objectives.

To validate the effectiveness of the proposed approach, we conduct real-world experiments on a humanoid robot's four Degrees of Freedom (DOF) arm. These experiments compare the controller with a conventional model-based approach and test its adaptability to changes in dynamics, i.e., mass, since mass is the only variable factor affecting gravity balancing. The results show that the LNN-CTC performs similarly to conventional control methods without prior model knowledge or system identification and improved adaptability in a few-shot learning manner.

The main contributions of this work are: First, we present a novel integration of LNNs with CTC, which opens new avenues for adaptive robot control. Second, we develop a data-efficient framework for online learning that enables real-time model updates without a large computational overhead. Third, to our knowledge, this is the first work to use LNNs in control to adapt to changes in dynamics.

The remainder of the paper is organized as follows. Section II presents details of the Digit robot, LNNs and introduces the proposed online learning framework, including the Proportional Derivative (PD) controller and the LNN-CTC based gravity compensation. A series of experiments are included in Section IV, including trajectory tracking and adaptability to changes in dynamics. Section V concludes the paper and discusses future work.

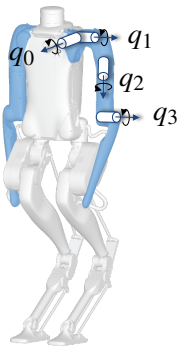
II. Methods

This section introduces the conventions used, the robot model, Lagrangian Neural Networks, and the linearized feedback controller.

A. Robot Model

The robot considered is a humanoid robot with an upper body inspired by human biomechanics (see Fig. 1a). Each of the two arms has four hinge joints, which are actuated by four motors, resulting in four DOF for each arm (see Fig. 1b). A standing controller controls the robot's floating base; thus, the robot arms are considered fixed in the world frame. Let $\mathbf{q}$, $\dot{\mathbf{q}} \in \mathbb{R}^n$, with $n = 1,\ldots,4$ being the number of joints, be the state (position and velocity) and $\boldsymbol{\tau} \in \mathbb{R}^n$ the joint torques in each motor of the robot.

(a)



(b)

Fig. 1: Picture of the upper body of the robot Digit (left) and its kinematic tree structure of the left arm, including the four joints (right).

B. Lagrangian Neural Networks

One essential part of model-based control approaches are models that describe the relationship between control input and the system's state ($\mathbf{q}$, $\dot{\mathbf{q}}$) [15]. Accurate system models rely on notions of the underlying symmetries in the world [13].

One way to describe these symmetries in robotics is through conservation laws, utilizing the Lagrangian to describe systems (1). Lagrange offers a well-known method for modeling rigid robotic systems. The Lagrange function determines the difference between the kinetic energy (T) and the potential energy (V). The underlying model can be set up based on the system's kinematic chain.

$$\mathscr{L} = T - V. \tag{1}$$

From this, the constrained Euler-Lagrange equation (2) can be derived, which describes the path of the system [13].

$$\frac{d}{dt}\frac{\partial \mathscr{L}}{\partial \dot{q}} = \frac{\partial \mathscr{L}}{\partial q} \tag{2}$$

For the control, the Lagrange formalism from [13], [14] is used to describe the relation between the input and the output states within the vectorized form

$$\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) = \frac{\partial^2 \mathscr{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}^2}\ddot{\mathbf{q}} + \frac{\partial^2 \mathscr{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial \mathbf{q}\partial \dot{\mathbf{q}}}\dot{\mathbf{q}} - \frac{\partial \mathscr{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial \mathbf{q}}. \tag{3}$$

Equ. (3) can be rewritten in a more common equation of motion for rigid systems

$$\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{4}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the mass-inertia matrix, $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) \in \mathbb{R}^{n\times n}$ is the Coriolis matrix and $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ are the gravitational forces.

Using this Euler-Lagrange equation (Eq. 4), traditional engineering approaches would calculate $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q})$ and $\mathbf{g}(\mathbf{q})$

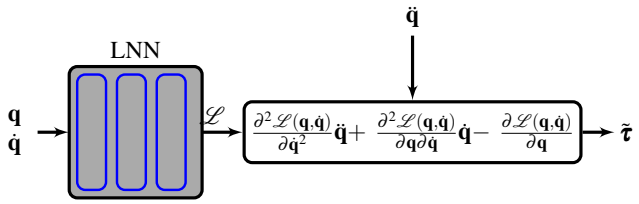


Fig. 2: The structure of the LNN during the training with the full inverse Euler-Lagrange equation as output. During control, the output of the LNN is only the gravitational term of the Euler-Lagrange equation.

from the approximated or measured masses, lengths, and moments of inertia resulting in $\tilde{\mathbf{M}}(\mathbf{q})$, $\tilde{\mathbf{C}}(\mathbf{q})$ and $\tilde{\mathbf{g}}(\mathbf{q})$, with $\tilde{}$ being an approximation [14]. In LNNs, the Lagrangian of the system is represented as a neural network, which means that the network is trained to approximate the Lagrangian function of the system being modeled. Instead of directly learning the dynamics, LNNs learn the underlying physics, i.e., the Lagrangian, from which the dynamics can be derived. We can write the LNN prediction during training

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

Since the controller is only using the gravitational term, the prediction of the controller is defined as

$$\tilde{\mathbf{g}}(\mathbf{q};\theta) = \frac{\partial \mathscr{L}(\mathbf{q},\dot{\mathbf{q}};\theta)}{\partial \mathbf{q}}, \tag{6}$$

depending on the parameters of the neural network θ. This term corresponds to the negative partial derivative of the potential energy component of the Lagrangian with respect to generalized coordinates, which represents the generalized gravitational forces. This approach works with fewer data than traditional DNNs and good predictions [13], [14], [16], [17]. The parameters θ of the LNN can be obtained by minimizing violation of the LNN's prediction of the physical law described by Lagrange mechanics (see Fig. 2). We, therefore, formulate the optimization problem as follows

$$\theta^* = \arg\min_{\theta} L(\tilde{\boldsymbol{\tau}}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}};\theta),\boldsymbol{\tau}), \tag{7}$$

where L is a differentiable loss function. This work uses the Mahalanobis norm as a loss function because the magnitude of the residual might vary between different joints. The LNN consists of three layers of 64 Neurons.

C. Controller

The proposed controller uses gravity compensation and a linear feedback controller to counteract gravitational forces and ensure stable, precise motion by correcting position and velocity errors, enhancing the robot's trajectory tracking performance and accuracy (see Fig. 3). The PD The controller tracks the desired trajectory solely on the basis of the current tracking error. In addition to the feedback term, a model of gravitational torques is used as a CTC to compensate for gravity,which provides good performance for robotic systems and is computationally less expensive than a CTC based

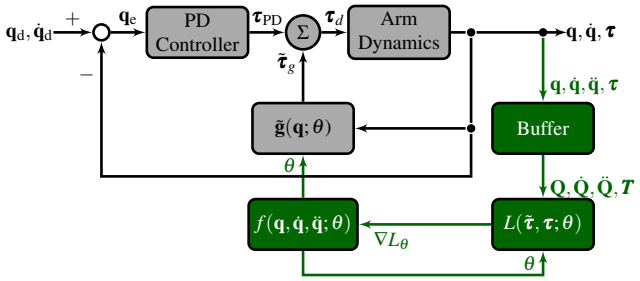


Fig. 3: Control loop using a PD-Controller with a LNN-CTC to compensate the gravity. The training process (green) learns the system dynamics online and updates the model in the control loop.

on the full dynamic model [4]. The proposed LNN-CTC learns the dynamics from past movements over time. An independent PD controller is used for each of the four joints, but the gravity compensation term of the CTC depends on the current configuration of all joints.

1) Proportional Derivative (PD) Controller: The desired joint space trajectory is the input for the PD feedback controller. The trajectory consists of a desired joint position and velocity for each joint. The position and velocity error are calculated as follows:

$$\begin{aligned} \mathbf{e}(t) &= \mathbf{q}_\mathrm{d}(t) - \mathbf{q}(t) \\ \dot{\mathbf{e}}(t) &= \dot{\mathbf{q}}_\mathrm{d}(t) - \dot{\mathbf{q}}(t) \end{aligned} \tag{8}$$

where $\mathbf{q}(t)$ and $\dot{\mathbf{q}}(t)$ are the measured joint angles and joint angular velocities, respectively. The desired joint angles and angular velocities are defined as $\mathbf{q}_\mathrm{d}(t)$ and $\dot{\mathbf{q}}_\mathrm{d}(t)$. The output of the PD controller $\boldsymbol{\tau}_\mathrm{PD}(t) \in \mathbb{R}^n$ is calculated as follows:

$$\boldsymbol{\tau}_\mathrm{PD}(t) = \mathbf{K}_\mathrm{P}\mathbf{e}(t) + \mathbf{K}_\mathrm{D}\dot{\mathbf{e}}(t) \tag{9}$$

where $\mathbf{K}_\mathrm{P} \in \mathbb{R}^{n\times n}$ and $\mathbf{K}_\mathrm{D} \in \mathbb{R}^{n\times n}$ are the proportional and derivative gains, respectively.

2) Lagrangian Neural Networks (LNN)-Computed Torque Control (CTC): Gravity compensation controllers in robotics are designed to counteract the gravitational forces acting on a robot and reduce loads on the actuators by achieving an equilibrium throughout the range of motion [1]. One commonly implemented way to compensate for gravity is the CTC [4]. The primary function of a CTC is to linearize the nonlinear dynamics of a system by using feedback linearization, allowing for more straightforward control strategies to be applied. If the dynamic model is poor, the tracking performance of the CTC is poor. Moreover, it leads to an unstable system caused by the direct state feedback.

The gravitational torque is calculated by using

$$\tilde{\boldsymbol{\tau}}_\mathrm{g}(t) = \tilde{\mathbf{g}}(\mathbf{q}(t);\theta). \tag{10}$$

Adding the feedback torque $\boldsymbol{\tau}_\mathrm{PD}(t)$ and the computed gravitational torque $\tilde{\boldsymbol{\tau}}_\mathrm{g}(t)$ the total control torque is computed as

$$\boldsymbol{\tau}_\mathrm{d}(t) = \boldsymbol{\tau}_\mathrm{PD}(t) + \tilde{\boldsymbol{\tau}}_\mathrm{g}(t). \tag{11}$$

The gravitational part of the controller achieves an equilibrium for the robot while the $\boldsymbol{\tau}_{\text{PD}}$ tracks the trajectory by compensating for other model dynamics, like inertia and friction inside the joints.

3) Online learning: The proposed control strategy is designed to update the parameters of the LNN while running (see Fig. 3, green). Online learning is, therefore, based on data gathered while executing a trajectory. A small batch size is used to avoid overfitting and to escape local minima with noisier gradient estimates. To create the batches, the output data of the system is stored in a buffer $\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$. We define the buffer

$$\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau}) := \begin{bmatrix} [\mathbf{q}_{t-50}, \ldots, \mathbf{q}_t]^T \\ [\dot{\mathbf{q}}_{t-50}, \ldots, \dot{\mathbf{q}}_t]^T \\ [\ddot{\mathbf{q}}_{t-50}, \ldots, \ddot{\mathbf{q}}_t]^T \\ [\boldsymbol{\tau}_{t-50}, \ldots, \boldsymbol{\tau}_t]^T \end{bmatrix} \tag{12}$$

to hold 50 s of data, resulting in $\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau}) \in \mathbb{R}^{4n \times m}$ with $m = 10{,}000$ being the number of samples obtained over the last 50 s. A queue of a fixed size is used as a buffer.

To fill the buffer during the initial phase, until the first update of the LNN, the randomly initialized LNN-CTC and PD controller move the robot arm. The training loop starts when the initial phase of 5 s has filled the buffer. Batches of b random samples are drawn from the buffer, where $b = 400$. One batch consists of the submatrices $\mathbf{Q}, \dot{\mathbf{Q}}, \ddot{\mathbf{Q}}$ and $\boldsymbol{T}$ from $\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\tau})$. Each of these submatrices is of dimension $\in \mathbb{R}^{n \times b}$, where $\ddot{\mathbf{q}}$ is obtained by the numerical derivative (first-order approximation) of $\dot{\mathbf{q}}$. The training loop updates the controller's LNN parameter after every 1000 iterations. We use AdaBelief [18] as an optimizer with weight decay.

III. Experimental Setup

The proposed control strategy depicted in Fig. 3 is tested on the arm of the Digit robot from Agility Robotics (see Fig. 1). Digit is stabilized with a standing controller during the experiments; therefore, the base is stationary with negligible movement. The arm weighs approximately 3.6 kg, including all four joints up to the torso.

Fig. 4: The robot with the additional weight near the end effector.

To test the online learning capabilities of the LNN-CTC, first, the tracking performance of an arbitrary trajectory is compared to a model-based CTC. Thus, the convergence of the LNN-CTC can be evaluated. The model-based CTC is based on the URDF given by the manufacturer and is modeled using the Rigid Body Algorithms of Pinocchio [19]. Afterward, in a second experiment, the adaptability of the LNN-CTC is evaluated by adding additional weight (0.4 kg, then 1.5 kg approximately 11 % and 42 % of the total arm weight) to the last link of the arm, close to the end effector (see Fig. 4). Since the convergence of the LNN-CTC compared to a model-based approach is shown in the first experiment, this experiment focuses solely on the adaptability after changing the dynamics (i.e., mass) of the robotic arm. After weight attachment, the LNN's weights are updated to fit the new dynamics while tracking a trajectory. The LNN's response to the change in dynamics was evaluated to highlight the fast adaptability of the LNN-CTC. For all experiments, the proportional and derivative gains are set to $\mathbf{K}_{\text{P}} = 50 \cdot \mathbf{I}_4$ and $\mathbf{K}_{\text{D}} = 0.1 \cdot \mathbf{K}_{\text{P}}$. These gains are sufficient to track fast velocities, but cannot compensate for gravity without CTC. These gains are arbitrarily chosen and do not receive additional tuning.

The controller runs at 200 Hz on a laptop equipped with an AMD Ryzen 9 8945HS CPU, and the LNN learning runs on the same machine on the Nvidia 4070 laptop GPU. The weights of the LNN are updated after every 1000 training steps, which results in an update rate between 1.5 Hz to 3 Hz. For safety reasons, Digit is connected to a safety rope (see Fig. 1a).

IV. Results

The learned and model-based CTC are evaluated in two different setups performing the same trajectory-tracking tasks. In the first setup, we assess the arm without any external mass, and in the second scenario, we apply a payload to show the adaptivity. To generate these trajectories, arbitrary reachable Cartesian positions are generated and connected by smooth splines at random time intervals. Then, the robotics toolbox in Python [20] is used to generate smooth splines as joint space trajectories. As expected, the model-based CTC can track the trajectory without significant errors (see Fig. 5). The LNN-CTC converges after approximately 25 s to a similar tracking performance. During the first 5 s, the error of the LNN-CTC is significant since during this time, the PD controller has to compensate for gravity, track the trajectory, and account for the error of the untrained, randomly initialized LNN-CTC. After the first LNN update (after $\approx$ 5 s), the tracking performance increases and becomes better with every update. Since the PD controller is not tuned, both the LNN-based and the model-based controllers always exhibit a small remaining error. However, both errors become equally small over time.

The RMS error in trajectory tracking (see Fig. 5) shows that the performance of the LNN is close to the model-based CTC after 25 s. Over time, the performance of the LNN-based and model-based CTC is becoming increasingly

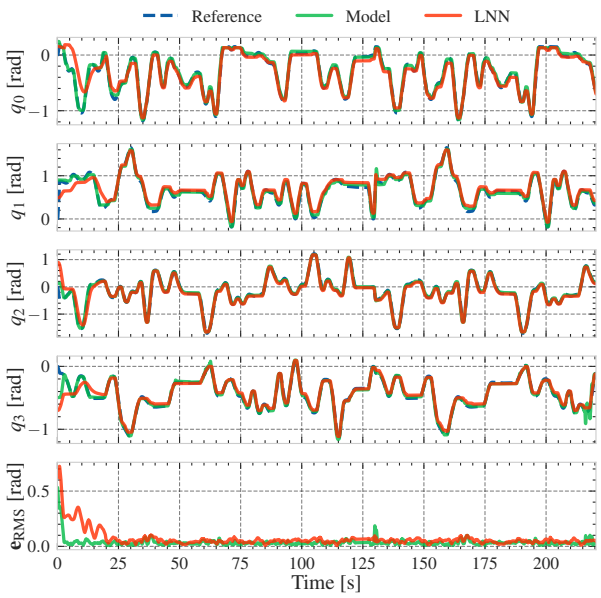


Fig. 5: Comparison of the LNN-based and model-based CTC tracking performance. During the first twenty seconds, the LNN-CTC learns to compensate for gravity. After ≈ 8 updates, the LNN-CTC error converges to the error archived by the model-based approach. The last plot shows the Root Mean Square (RMS) error of all joints in the joint space.

TABLE I: RMS error ($\mathbf{e}_{\text{RMS}}$) of the LNN, model-based CTC and the additional loads after the initial convergence (100 s until 200 s)

	q_0	q_1	q_2	q_3
LNN	0.045 rad	0.051 rad	0.039 rad	0.023 rad
Model	0.032 rad	0.046 rad	0.044 rad	0.024 rad

similar, with the model slightly outperforming the LNN-based approach. This is because the Lagrangian's gravity term is solely based on the joint configuration and mass, which are perfectly known. To further compare the errors, we can calculate the RMS error $\mathbf{e}_{\text{RMS}}$ of the joints (see Tab. I). After 100 s (i.e., after initial convergence and some additional training time), the mean of the joint RMS errors of the LNN-CTC is slightly (0.003 rad) worse than the model-based approach.

In the second experiment, the same trajectory as before is tracked by the LNN-CTC for the first 210 s. After that time, the robot is stopped to add additional weight, e.g., 1.5 kg. With the new weight on the arm, the robot restarts with an arbitrary trajectory. The tracking performance decreases with the added weight, but within 5 s, the LNN can compensate for the new weight (see Fig. 6). Therefore, about three updates are necessary to adapt to the external disturbance. The buffer is cleared after the weight change to speed up learning the new dynamic; otherwise, the old data would fade out over time and prolong the learning process.

While mounting the additional weight to the arm, a PD controller holds the robot in place. After the mounting phase,

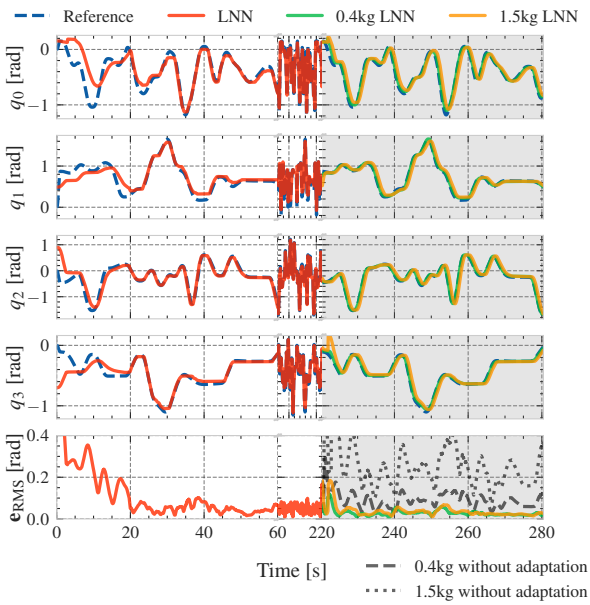


Fig. 6: Tracking performance of the LNN-CTC with additional weight. The gray shaded area indicates the changed weight. The first 210 s is the training trajectory without additional weight (full training trajectory, see Fig. 5). Then, the additional weight is added, and the controller continues. After the weight is added (0.4 kg and 1.5 kg, respectively), the performance decreases, but within 5 s, the LNN can compensate for the new weight. The gray lines on the bottom right are the results with the weight of LNN-CTC not updated after 210 s to highlight the difference in performance for the new weights.

the fitted LNN-CTC and the PD controller are not able to compensate for gravity. This performance decrease is visible in the RMS error (see Fig. 6 right side). After 5 s, the LNN can compensate for gravity with the additional weight with the same performance as before. Online learning, therefore, enables the robot to pick objects with unknown masses and adapt the controller to the changed dynamics of the system. Also, due to online learning and the LNN, unknown joint configurations pose no problem to the controller.

The experiments on the physical robot show that the LNN-CTC can learn to compensate for gravity despite noisy observations and can be used in real-time by a closed-loop controller to achieve good control performance close to the performance of a model-based approach. The experiments also show the adaptability of the LNN-CTC in online learning approaches.

V. Conclusion

This paper presents an adaptive, real-time control framework that integrates LNN with CTC to achieve continuous gravity compensation in robotic systems. Through real-world experiments on a 4-DOF humanoid robot, we demonstrated the effectiveness of our approach in compensating for gravitational forces, adapting to changes in mass, and maintaining

trajectory tracking accuracy without requiring prior model knowledge. Our results show that after only 25 s, the LNN-CTC system performs comparably to traditional model-based controllers while offering the advantage of rapid adaptation based on only a few seconds of real-world interaction data. There is no sim-to-real gap since all data is gathered in real-time on the robot. Therefore, the proposed control scheme overcomes the need for extensive re-modeling after weight changes as it adapts to changes in dynamics. The results indicate the potential usability of the proposed approach for practical use in various robotic applications, e.g., autonomous robots picking up unknown weights. The learned parameters of the LNN can be saved and reloaded for varying loads on the end effector to save training time.

Future work will extend this method to higher degree-of-freedom systems and explore its performance in more complex dynamic scenarios.

REFERENCES

[1] V. Arakelian, "Gravity compensation in robotics," Advanced Robotics, vol. 30, no. 2, pp. 79–96, 2016.

[2] H. Lin, C.-W. Vincent Hui, Y. Wang, A. Deguet, P. Kazanzides, and K. W. S. Au, "A reliable gravity compensation control strategy for dvrk robotic arms with nonlinear disturbance forces," IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3892–3899, 2019.

[3] C. Yu, Z. Li, D. Yang, and H. Liu, "A fast robotic arm gravity compensation updating approach for industrial application using sparse selection and reconstruction," Robotics and Autonomous Systems, vol. 149, p. 103971, 2022.

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

[5] C. Zhao and L. Guo, "Towards a theoretical foundation of pid control for uncertain nonlinear systems," Automatica, vol. 142, p. 110360, 2022.

[6] R. C. Luo, C. Y. Yi, and Y. W. Perng, "Gravity compensation and compliance based force control for auxiliarily easiness in manipulating robot arm," in 2011 8th Asian Control Conference (ASCC), 2011, pp. 1193–1198.

[7] Y. Yu, R. Shi, and Y. Lou, "Bias estimation and gravity compensation for wrist-mounted force/torque sensor," IEEE Sensors Journal, vol. 22, no. 18, pp. 17 625–17 634, 2022.

[8] J. Fugal, J. Bae, and H. A. Poonawala, "On the impact of gravity compensation on reinforcement learning in goal-reaching tasks for robotic manipulators," Robotics, vol. 10, no. 1, 2021.

[9] A. Perrusquía, J. A. Flores-Campos, and C. R. Torres-San-Miguel, "A novel tuning method of pd with gravity compensation controller for robot manipulators," IEEE Access, vol. 8, pp. 114 773–114 783, 2020.

[10] M. Razmi and C. Macnab, "Near-optimal neural-network robot control with adaptive gravity compensation," Neurocomputing, vol. 389, pp. 83–92, 2020.

[11] A. Ugartemendia, D. Rosquete, J. J. Gil, I. Diaz, and D. Borro, "Machine learning for active gravity compensation in robotics: Application to neurological rehabilitation systems," IEEE Robotics & Automation Magazine, vol. 27, no. 2, pp. 78–86, 2020.

[12] T. Westenbroek, D. Fridovich-Keil, E. Mazumdar, S. Arora, V. Prabhu, S. S. Sastry, and C. J. Tomlin, "Feedback linearization for uncertain systems via reinforcement learning," in 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 1364–1371.

[13] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," in ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations.

[14] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," in International Conference on Learning Representations, 2018.

[15] P. Ioannou and J. Sun, Robust Adaptive Control, ser. Dover Books on Electrical Engineering Series. Dover Publications, Incorporated, 2012.

[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 International Conference on Intelligent Robots and Systems (IROS). IEEE, 2019, pp. 7718–7725.

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

[18] J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan, "Adabelief optimizer: Adapting stepsizes by the belief in observed gradients," in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 18 795–18 806.

[19] 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 International Symposium on System Integrations (SII), 2019.

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