Context-Aware Deep Lagrangian Networks for Model Predictive Control

Lucas Schulze[1], Jan Peters[1,2,3,4], Oleg Arenz[1]

Abstract— Controlling a robot based on physics-consistent dynamic models, such as Deep Lagrangian Networks (DeLaN), can improve the generalizability and interpretability of the resulting behavior. However, in complex environments, the number of objects to potentially interact with is vast, and their physical properties are often uncertain. This complexity makes it infeasible to employ a single global model. Therefore, we need to resort to online system identification of context-aware models that capture only the currently relevant aspects of the environment. While physical principles such as the conservation of energy may not hold across varying contexts, ensuring physical plausibility for any individual context-aware model can still be highly desirable, particularly when using it for receding horizon control methods such as model predictive control (MPC). Hence, in this work, we extend DeLaN to make it context-aware, combine it with a recurrent network for online system identification, and integrate it with an MPC for adaptive, physics-consistent control. We also combine DeLaN with a residual dynamics model to leverage the fact that a nominal model of the robot is typically available. We evaluate our method on a 7-DOF robot arm for trajectory tracking under varying loads. Our method reduces the end-effector tracking error by 39%, compared to a 21% improvement achieved by a baseline that uses an extended Kalman filter.

I. INTRODUCTION

Representative models are essential for control of dynamic systems, whether for learning approaches, such as reinforcement or imitation learning in simulation, or for model-based techniques, like model predictive control (MPC). White-box models can be derived from first principles, but they often require significant engineering effort and may not generalize well to dynamically changing environments. To address these challenges, online system identification (SysID) continuously adapts the model using online observations, improving its accuracy and robustness during operation.

Classical adaptive control techniques online identify the model's parameters based on a linear regressor [1], [2], [3], [4]. However, these methods rely on prior knowledge, such as a known kinematic tree, and assume persistent excitation [5]. Additionally, they are typically designed for specific types of model mismatches, limiting their generalization across different systems and conditions.

On the other hand, black-box approaches leverage recorded data to model the environment using nonparametric models, such as Gaussian process regression (GPR) [6], [7], [8] or locally weighted projection regression [9], or parametric models such as neural networks [10]. However, these techniques typically perform well only within the trained domain and require large amounts of data.

A powerful technique to increase the accuracy of the learned model in the small data regime, is to employ physics-informed inductive biases [11]. To address physical plausibility in offline SysID, grey-box methods such as Deep Lagrangian Networks (DeLaN) were proposed to combine Lagrangian mechanics with deep learning [12], [13]. The resulting model from DeLaN is interpretable and can be used as a forward and inverse model. A similar approach is used in Hamiltonian Neural Networks (HNN) [14], which are based on Hamiltonian mechanics. Both DeLaN and HNN can be applied to model-based control [15], [16].

However, these offline methods are not suitable for environments where the robot manipulates different objects, as they cannot account for the resulting changes in the robot's dynamics. Therefore, we propose an extension of DeLaN to enable physically consistent online SysID by learning a contextual DeLaN model that obtains a latent representation of the environment as additional input, which can be identified online. This modification allows us to generalize across multiple and time-varying environments while maintaining a physically plausible model at any given time. By using a fixed model during each MPC iteration, which might be updated between iterations, we impose the prior belief that the dynamic properties will remain constant over the prediction horizon. Yet, this approach still enables swift adaptation whenever this prior belief is violated. We argue that this method is particularly fruitful in settings where changes in dynamics are sparse over time, e.g., when picking up or dropping objects.

Furthermore, we use a nominal model as an inductive bias to focus on learning the unknown dynamics of the system, as certain elements may remain unchanged compared to the nominal case. Learning a residual dynamics model that incorporates prior dynamics has been shown to improve data efficiency and generalization [17]. In the context of deep learning, learning a residual model has similar benefits [18]. It also enables the use of smaller networks, which is essential for fast evaluation and optimization, which is a key requirement of MPC.

Our work is within the context of meta-learning, i.e., learning how to optimize a meta-objective over different tasks using previous experience [19]. Many approaches combine this concept with adaptive control, relying on GPR [20], [21], [22], non-linear basis functions [23], [24], or feed-forward

[1]Department of Computer Science, Technical University of Darmstadt, Germany. [2]Hessian.AI. [3]German Research Center for AI (DFKI), Research Department: Systems AI for Robot Learning. [4]Robotics Institute Germany (RIG).
Corresponding author: `lucas.schulze@tu-darmstadt.de`

neural networks [25], [26]. Similar meta-learning methods have been applied to physics-informed neural networks (PINNs) [27], iMODE [28] learns a more general PINN that incorporates an additional latent variable to describe the underlying system, similar to our approach. However, these methods do not guarantee physical consistency.

Similarly, while residual models have been previously employed for MPC [29], [30], these methods did not use physically plausible models for better data efficiency. Furthermore, while [31] proposes an adaptive controller based on Hamiltonian neural ordinary differential equations to learn disturbance features, ensuring physical consistency, their method is only applied to a feedback energy control strategy, and the results are demonstrated only in simulation.

We demonstrate the adaptability of our method in joint trajectory tracking of a 7-DOF Franka Emika Panda under an unknown payload, both in simulation and on the real robot. To demonstrate the benefits of identifying an entire dynamics model and not only gravitational compensation for agile motions, we compare it to an extended Kalman filter (EKF) that directly estimates the force at the end-effector. As shown in our experiments, we outperform the tracking and prediction error of both, the nominal model and the EKF. On the real robot, our method is applied zero-shot, that is, we apply it using a model trained solely with simulated data.

A. Contributions

Our main contributions are as follows:
- We combine DeLaN with a nominal model to only predict the unmodeled dynamics.
- We extend DeLaN to a contextual setting, enabling us to learn a single network for different dynamics.
- By training our contextual DeLaN along with a history encoder in simulation, we are able to online identify a physically plausible dynamics model.
- We combine the aforementioned contributions for MPC by using recent software frameworks [32], [33], [34], [30] to integrate our contextualized DeLaN network with an optimizer suitable for real-time optimization.
- We demonstrate the benefits of our learning-based control method for agile manipulation tasks, by performing zero-shot real robot experiments from simulated data.

The remainder of this paper is organized as follows: Section II briefly reviews DeLaN and presents our extensions for the residual and contextual setting, along with their application for MPC: Context-Aware Deep Lagrangian MPC (CaDeLaC). Section III presents our experimental setting as well as the results. Finally, Section IV summarizes our findings.

II. CONTEXT-AWARE DeLaN

We will now briefly discuss DeLaN [13], before presenting our extensions to the residual and contextual setting, as well as its application to online system identification for MPC.

DeLaN embeds Lagrangian mechanics into a deep learning framework to learn physically consistent dynamical models. Consider the Lagrangian defined as the difference between the kinetic energy $K = \frac{1}{2}\dot{\mathbf{q}}^{\mathrm{T}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}}$ and the potential energy P. Using the Euler-Lagrange equation, the system's equation of motion is given by

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \underbrace{\dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial\mathbf{q}}\left(\dot{\mathbf{q}}^{\mathrm{T}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^{\mathrm{T}}}_{:=\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}} + \underbrace{\frac{\partial P}{\partial\mathbf{q}}}_{:=\mathbf{g}(\mathbf{q})} = \boldsymbol{\tau},$$

$$(1)$$

where $\boldsymbol{\tau}$ is the joint torque.

DeLaN approximates $\mathbf{H}$ and P by two neural networks. The first one outputs the diagonal $\mathbf{l}_{\mathrm{diag}}$ and off-diagonal $\mathbf{l}_{\mathrm{off}}$ elements of the lower triangular matrix $\mathbf{L}$ given by the Cholesky decomposition of $\mathbf{H}$. Thus, the estimated inertia matrix $\hat{\mathbf{H}}(\mathbf{q}, \boldsymbol{\theta}_{\mathbf{H}})$ is given by

$$\hat{\mathbf{H}}(\mathbf{q}, \boldsymbol{\theta}_{\mathbf{H}}) = \hat{\mathbf{L}}(\mathbf{q}, \boldsymbol{\theta}_{\mathbf{H}})\hat{\mathbf{L}}(\mathbf{q}, \boldsymbol{\theta}_{\mathbf{H}})^{\mathrm{T}}, \quad (2)$$

where an offset ϵ and a softplus function is applied to $\mathbf{l}_{\mathrm{diag}}$ to guarantee $\hat{\mathbf{H}}$ to be positive definite, and $\boldsymbol{\theta}_{\mathbf{H}}$ is the network parameter. The second network directly outputs the potential energy $\hat{P}(\mathbf{q}, \boldsymbol{\theta}_P)$. Therefore, the inverse dynamics model (1) is defined as a function of the joint variables and the parameters of the networks $\boldsymbol{\tau} = f^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\theta}_{\mathbf{H}}, \boldsymbol{\theta}_P)$.

To train the network parameters, a loss function using inverse dynamics error can be employed. The gradient of the loss function is obtained using automatic differentiation in (1) of PyTorch [35].

A. Residual DeLaN

Given a nominal model, the equation of motion of the real system (1) can be rewritten as

$$\hat{\mathbf{H}}(\mathbf{q})\ddot{\mathbf{q}} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \hat{\mathbf{g}}(\mathbf{q}) + \tilde{\boldsymbol{\tau}} = \boldsymbol{\tau}, \quad (3)$$

where $\hat{\mathbf{H}}$, $\hat{\mathbf{C}}$ and $\hat{\mathbf{g}}$ refer to the nominal components, and $\tilde{\boldsymbol{\tau}} = \boldsymbol{\tau} - \hat{\boldsymbol{\tau}}$ denotes the residual torque between actual and nominal torque. Due to the model mismatches, i.e., $\tilde{\mathbf{H}} = \mathbf{H} - \hat{\mathbf{H}}$, $\tilde{\mathbf{g}} = \mathbf{g} - \hat{\mathbf{g}}$, the residual torque can be defined as

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

which has the same form as (1) and can be learned using the DeLaN framework, i.e., $\tilde{\boldsymbol{\tau}} = \tilde{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\theta}_{\mathbf{H}}, \boldsymbol{\theta}_P)$.

Despite the extensive engineering efforts, model mismatches will always arise in real systems due to variations during production, flexible and moving parts, unmodelable internal components, e.g., wires and cables, and additional payloads attached to the robot. Hence, the residual torque is not only a function of joint states but also of the environment.

B. Context-Aware DeLaN

To enable DeLaN to estimate the residual torque $\tilde{\boldsymbol{\tau}}$ across different environments, we condition it on a latent embedding $\mathbf{z}$ of the environment. By training a history-based encoder along with the contextual DeLaN, we are able to infer the latent representation of the environment based on the recently observed states and actions. More specifically, we use an LSTM [36] network as encoder, and feed it a sequence of

joint positions, velocity and measured residual torques $\tilde{\boldsymbol{\tau}}$. Thus, each entry i in the n_h length sequence is:

$$\mathbf{h}_i = \begin{bmatrix} \mathbf{q}_i^{\mathrm{T}} & \dot{\mathbf{q}}_i^{\mathrm{T}} & \tilde{\boldsymbol{\tau}}_i^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}. \tag{5}$$

We chose an LSTM for its ability to capture information from historical data while retaining long-term memory. By conditioning the contextual DeLaN on a learned latent representation, we can also reduce the number of neurons and layers. Please note, that we will only require the contextual DeLaN model during a given MPC iteration, but not the LSTM. The proposed architecture is depicted in Figure 1.
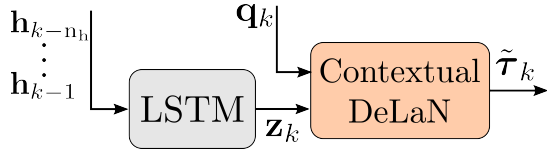

Fig. 1: Our proposed architecture. The orange block is similar to the original DeLaN [13], but with additional input $\mathbf{z}$.

C. Context-Aware DeLaN for MPC

We will now introduce CaDeLaC and discuss how it uses our context-aware and residual DeLaN model for MPC.

MPC is an optimal control technique that computes control actions in a receding horizon fashion by optimizing a cost function based on a system's model prediction [37]. To follow a desired state trajectory, we optimize at each instant k the state and control inputs, $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{u}}_k$, along the prediction horizon n_y with respect to the optimization problem

$$\min_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k} \quad \sum_{i=0}^{n_y} \|\mathbf{x}_{k+i}^{\mathrm{d}} - \mathbf{x}_{k+i}\|_{\mathbf{Q}}^2 + \sum_{i=0}^{n_y-1} \|\mathbf{u}_{k+i}\|_{\mathbf{R}}^2$$
$$s.t. \quad \mathbf{x}_k = \mathbf{x}_0, \tag{6}$$
$$\mathbf{x}_{k+i+1} = f(\mathbf{x}_{k+i}, \mathbf{u}_{k+i}),$$
$$\mathbf{h}(\mathbf{x}_{k+i}, \mathbf{u}_{k+i}) \leq \mathbf{0},$$

where $\mathbf{Q}$ and $\mathbf{R}$ are the weights matrices, $\mathbf{x}_0$ is the current state, $\mathbf{x}_{k+i}^{\mathrm{d}}$ is the desired state at each timestep $k+i$, $f(\mathbf{x}_{k+i}, \mathbf{u}_{k+i})$ defines the system's model and $\mathbf{h}(\mathbf{x}_{k+i}, \mathbf{u}_{k+i})$ are inequality constraints, e.g., actions and state limits, collision avoidance or safety margins.

A straightforward way to integrate our Context-Aware DeLaN would be to infer the residual torque along a future sequence of states and control actions, i.e., predicted in the step before or the reference values. This residual torque could then be given to the MPC as an external torque. However, the controller would only be able to compensate static model errors that do not depend on the system's state. Yet, we typically require state-dependent compensation for model mismatches, for example, due to mismatches in the inertia. A possible solution is to approximate the model using Taylor series [7], [30], but as we will discuss in more detail, we can also directly pass our model to the optimizer.

As we assume that the environment will not change within the prediction horizon, we can infer the latent variable $\mathbf{z}$ before every MPC iteration and use the resultant function

$\tilde{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ as a residual function. Please note, that the network parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\mathbf{H}}, \boldsymbol{\theta}_P\}$ are trained jointly in an initial training phase, but remain constant throughout inference, as illustrated in Algorithm 1 and Algorithm 2.
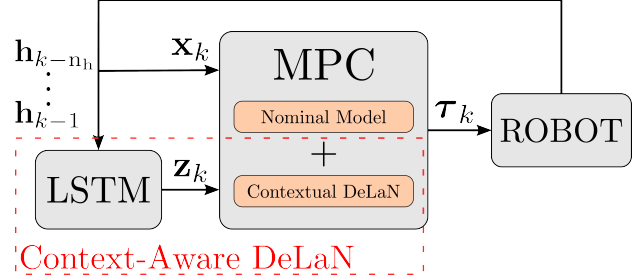

Fig. 2: CaDeLaC: Context-Aware Deep Lagrangian Model Predictive Control. The MPC optimizes considering both nominal and learned models (orange blocks).

Algorithm 1 Training Phase

Require: Stepsize α, Dataset $\mathcal{D} = \{\mathbf{h}_{1:T}^e\}_{e=1}^E$ of state-action sequences collected at different training environments e (e.g., in simulation)

1: **for** every iteration i **do**
2:     **for** every environment $e \in \{1, \ldots, E\}$ **do**
3:         **for** every timestep $t \in \{1, \ldots, T\}$ **do**
4:             $\mathbf{z} \leftarrow \text{LSTM}(\mathbf{h}_0^e, \ldots, \mathbf{h}_t^e)$
5:             $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left\| \tilde{\boldsymbol{\tau}}_t^e - f^{-1}(\mathbf{q}_t^e, \dot{\mathbf{q}}_t^e, \ddot{\mathbf{q}}_t^e; \boldsymbol{\theta}, \mathbf{z}) \right\|^2$
6:         **end for**
7:     **end for**
8: **end for**

Algorithm 2 Control Phase

Require: Network parameters $\boldsymbol{\theta}$ obtained during training phase

1: **for** time step k **do**
2:     $\mathbf{z}_k \leftarrow \text{LSTM}(\mathbf{h}_{k-n_h}, \ldots, \mathbf{h}_{k-1})$
3:     $\mathbf{u}_k \leftarrow$ solve (6) with $f^{-1}(\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k; \boldsymbol{\theta}, \mathbf{z}_k)$
4:     Apply control $\mathbf{u}_k$
5: **end for**

III. EXPERIMENTS

In this section, we evaluate the proposed architecture to improve the joint trajectory tracking on a 7-DOF Franka Emika Panda robot with an unknown payload.

A. Data Collection and Training

We collect data using an LQR based on the nominal model for tracking a joint trajectory of the robot in simulation using MuJoCo [38]. We collect 20 trajectories with a duration of 10 seconds for 100 different payloads attached to the end-effector. The mass and position of the payloads are uniformly sampled from 0 kg to 4 kg, while the 3D coordinates are sampled between -0.3 m and 0.3 m from the center of the

end-effector. Additionally, we simulate the LQR without an additional load. The trajectories are sampled at the same control frequency of 50 Hz as the LQR; thus, the dataset has 1.01 M samples. For system excitation, the joint reference trajectories are a chirp signal of 5th order and 10 seconds following [3], i.e.

$$\mathbf{q}_{\mathrm{d}}(t) = \sum_{i=1}^{n_e} \frac{\mathbf{a}_{ei}}{i\omega_{e}} \sin(i\omega_{e}t) - \frac{\mathbf{b}_{ei}}{i\omega_{e}} \cos(i\omega_{e}t) + \mathbf{q}_0, \quad (7)$$

$$\mathbf{q}_{\min} < \mathbf{q}_{\mathrm{d}} < \mathbf{q}_{\max}, \quad \dot{\mathbf{q}}_{\min} < \dot{\mathbf{q}}_{\mathrm{d}} < \dot{\mathbf{q}}_{\max},$$

where $\omega_{e} = 2\pi/\mathrm{T}_{e}$ with $\mathrm{T}_{e} = 10$ s, $\mathbf{a}_{e}$ and $\mathbf{b}_{e}$ are uniformly sampled. The reference joint velocities $\dot{\mathbf{q}}_{\mathrm{d}}$ are obtained by analytically differentiating the reference joint positions $\mathbf{q}_{\mathrm{d}}$.

In addition to the latent variable $\mathbf{z}$, a transformation input layer is applied to $\mathbf{q}$ in both DeLaN networks, i.e., $\mathbf{T_q}(\mathbf{q}) = [\cos(\mathbf{q}), \sin(\mathbf{q})]$ as proposed in [13]. For the network architectures, we selected an MLP with 2 hidden layers, the first one with 30 neurons, and the second one with 20. The LSTM has 5 hidden layers of 10 neurons each, followed by a dense layer applied to the last temporal output, yielding an output dimension of 10.

To enhance the robustness of our model to noise, we introduced artificial noise into the input dataset. The variances for the different joints and the training hyperparameters are detailed in the Appendix A.

B. MPC for Joint Trajectory Tracking

We use the optimization problem presented in Section II-C to track a joint trajectory, by choosing the state $\mathbf{x}_k$ as the concatenation of joint velocities and positions, $\dot{\mathbf{q}}_k$ and $\mathbf{q}_k$, while the actions $\mathbf{u}_k$ correspond to the joint torques $\boldsymbol{\tau}_k$. We use the inequality constraints $\mathbf{h}$ to bound the torques within their limits $\boldsymbol{\tau}_{\min}$ and $\boldsymbol{\tau}_{\max}$.

The nonlinear MPC is implemented using Acados [32] on SQP-RTI mode with the solver HPIPM [33]. The nominal model is obtained as a CasADi expression using Pinocchio [34]. To convert the PyTorch model to CasADI, we use the *Naive* implementation of L4CasADi [30] to convert the MLPs into symbolic expressions.

While direct torque control is feasible in simulation, the control interface of the real robot requires a real-time loop controller at 1 kHz, which is challenging for any controller based on online optimization. Therefore, on the real robot, we use a feedback controller at 1 kHz with a feedforward torque that is updated by MPC at 50 Hz. The same frequencies are used for all evaluated controllers to ensure that performance differences are due only to the different models. Hence, the commanded torque is computed as

$$\boldsymbol{\tau}_{\mathrm{CMD}} = \boldsymbol{\tau}_{\mathrm{MPC}} + \mathbf{K}_{\mathrm{p}}(\mathbf{q}_{\mathrm{d}} - \mathbf{q}) + \mathbf{K}_{\mathrm{d}}(\dot{\mathbf{q}}_{\mathrm{d}} - \dot{\mathbf{q}}) - \hat{\mathbf{g}}(\mathbf{q}), \quad (8)$$

where the feedback gains $\mathbf{K}_{\mathrm{p}}$ and $\mathbf{K}_{\mathrm{d}}$ are small, see Appendix B, and only help to regularize the control action $\boldsymbol{\tau}_{\mathrm{CMD}}$. To account for the robot's internal nominal gravity and friction compensations, we subtract $\hat{\mathbf{g}}(\mathbf{q})$ from $\boldsymbol{\tau}_{\mathrm{CMD}}$. Due to the presence of noise in the measured $\dot{\mathbf{q}}$ and the estimated $\ddot{\mathbf{q}}$ through finite differences, we apply a low-pass filter with a 2 Hz cutoff frequency to the LSTM's output.

C. Extended Kalman Filter

To show that only gravity compensation is insufficient for fast motions, an extended Kalman filter (EKF) is implemented to estimate the load as an external force $\mathbf{f}_{\mathrm{ee}}$ applied at the end-effector, which is added to the nominal model,

$$\hat{\mathbf{H}}(\mathbf{q})\ddot{\mathbf{q}} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \hat{\mathbf{g}}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J}_{\mathrm{ee}}(\mathbf{q})^{\mathrm{T}}\mathbf{f}_{\mathrm{ee}}, \quad (9)$$

where $\mathbf{J}_{\mathrm{ee}}$ is the end-effector Jacobian. Similar to [39], we define the filter state and observation as

$$\mathbf{x}_a = \begin{bmatrix} \mathbf{q}^{\mathrm{T}} & \dot{\mathbf{q}}^{\mathrm{T}} & \mathbf{f}_{\mathrm{ee}}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}, \quad \mathbf{y}_a = \begin{bmatrix} \mathbf{q}^{\mathrm{T}} & \dot{\mathbf{q}}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}, \quad (10)$$

assuming a constant external force, $\dot{\mathbf{f}}_{\mathrm{ee}} = 0$. The estimated torque $\hat{\boldsymbol{\tau}}_{\mathrm{ee}} = \mathbf{J}_{\mathrm{ee}}(\mathbf{q})^{\mathrm{T}}\hat{\mathbf{f}}_{ee}$ is provided to the MPC as an external torque applied to the joints, named EKF-MPC.

D. Simulation - Experiments

We evaluate our method in a wide range of environments. Namely, we evaluate Nominal MPC (using only the nominal model), EKF MPC, and CaDeLaC across 30 unseen environments over 20 trajectories, i.e., in total 600 trajectories of 10 seconds for each controller. The reference trajectories are given by (7) with randomly sampled parameters. The payload parameters were sampled from the same range of mass and positions as for the training set.

To evaluate the performance of the three models, we analyze the Root Mean Square Error (RMSE) of the inferred residual torque. Table I presents the average RMSE of each model for all 1800 trajectories. The EKF reduced the residual torque error for only four of the seven joints. This occurs because the rotation axis of the first, along with the third and last joints in certain configurations, is parallel to gravity. Thus, the gravitational torque due to the load is zero, and only the extra inertia affects these joints, requiring some velocity or acceleration to manifest. Since the Context-Aware DeLaN captures not only the gravitational component, it reduces all the errors significantly, outperforming the EKF.

TABLE I: Average residual torque RMSE $\bar{\tau}_e$ for the three models over the 1800 simulated trajectories.

Model	$\bar{\tau}_{e_1}$ [Nm]	$\bar{\tau}_{e_2}$ [Nm]	$\bar{\tau}_{e_3}$ [Nm]	$\bar{\tau}_{e_4}$ [Nm]	$\bar{\tau}_{e_5}$ [Nm]	$\bar{\tau}_{e_6}$ [Nm]	$\bar{\tau}_{e_7}$ [Nm]
Nominal	2.28	10.55	5.13	11.63	3.21	3.98	2.56
Ours	**0.95**	**1.36**	**1.81**	**1.37**	**0.92**	**0.81**	**0.83**
EKF	10.02	8.40	11.30	7.25	1.87	1.25	2.56

The average position and velocity tracking RMSE are presented in Table II. CaDeLaC outperforms both Nominal MPC and EKF MPC in position tracking. However, for tracking velocity, the proposed controller presented larger errors in the first, second, and third joints than the nominal controller. One possible explanation is that the cost function considers both tracking errors and torque amplitudes, allowing for a different tradeoff due to the new controller's dynamic model. The EKF MPC improved the tracking errors for the last three joints but significantly increased them for the first three joints due to the residual torque estimation and the trade-off within the cost function.

TABLE II: Average tracking RMSE position $\bar{\mathbf{e}}$ and velocity $\dot{\bar{\mathbf{e}}}$ for the three controllers across 30 unseen simulated environments over 20 trajectories.

Controller	$\bar{e}_1$ [rad]	$\bar{e}_2$ [rad]	$\bar{e}_3$ [rad]	$\bar{e}_4$ [rad]	$\bar{e}_5$ [rad]	$\bar{e}_6$ [rad]	$\bar{e}_7$ [rad]
Nominal	0.024	0.081	0.053	0.123	0.155	0.164	0.145
CaDeLaC	**0.022**	**0.040**	**0.025**	**0.036**	**0.037**	0.070	**0.028**
EKF MPC	0.093	0.083	0.114	0.081	0.101	**0.066**	0.157

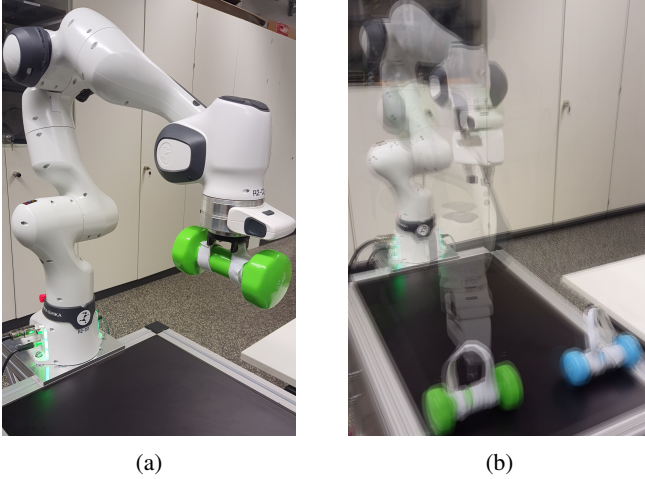| Controller	$\dot{\bar{e}}_1$ [rad/s]	$\dot{\bar{e}}_2$ [rad/s]	$\dot{\bar{e}}_3$ [rad/s]	$\dot{\bar{e}}_4$ [rad/s]	$\dot{\bar{e}}_5$ [rad/s]	$\dot{\bar{e}}_6$ [rad/s]	$\dot{\bar{e}}_7$ [rad/s]
Nominal	**0.059**	**0.084**	**0.096**	**0.102**	0.212	0.215	0.221
CaDeLaC	0.098	0.097	0.106	0.105	**0.152**	**0.163**	**0.128**
EKF MPC	0.168	0.152	0.192	0.166	0.201	0.176	0.267

Fig. 3: Hardware experiments: (a) 3 kg gym weight attached to 3D-Printed holder; (b) The robot executes a sequence of two pick-and-place tasks with varying loads.

E. Hardware Experiments

1) High-speed trajectory: To evaluate our method in agile motions where gravity compensation alone might be insufficient, we tested all three controllers on high-speed trajectory tracking near the joint velocity limits with unseen payloads. Each controller was set to track the specified trajectory with three different gym weights: 1 kg, 2 kg, and 3 kg, resulting in a total of 9 trajectories. To ensure a secure attachment, we removed the robot's gripper and replaced it with a 3D-printed holder, depicted in Figure 3a. Since the gripper weighs approximately 0.7 kg, the resulting payloads were 0.3 kg, 1.3 kg, and 2.3 kg, respectively.

The high-speed trajectory is defined as:

$$\mathbf{p}_{\text{ref}}(t) = \begin{bmatrix} 0 \\ a_1 \sin(\omega_I t)\cos(\omega_I t) \\ a_2 \sin(\omega_I t) \end{bmatrix} + \mathbf{p}_0, \qquad (11)$$

where $\mathbf{p}_0$ is the initial end-effector position, $\omega_I = 2\pi f_I$, with $f_I = 0.7$ Hz, $a_1 = 0.40$, and $a_2 = 0.15$. The reference joint positions were obtained using inverse kinematics. The reference joint velocities were then computed by finite difference, resulting in the following peak values: $\max(|\dot{\mathbf{q}}_{\text{ref}}|) = [2.0, 2.22, 1.99, 1.3, 0.37, 0.95, 0.0]$.

The inferred residual torque RMSE results are presented in Table III. Since the additional payload is only 0.3 kg for the 1 kg gym weight, the ratio between the residual torque and noise is small. As a result, both the EKF and our model do not improve significantly the residual torque. Similar to the simulation results, the EKF was able to improve the residual torques for five of the seven joints, with no significant improvement in the first and third joints. Our model presented an overall better estimation, as it also reduced the error for the first joint, while the errors for the last three joints remained small. The residual torque results were not as good as in the simulation, mainly due to noise, friction, and an imperfect rigid attachment of the weights.

TABLE III: Average residual torque RMSE $\bar{\tau}_e$ for the three models over the 9 high-speed trajectories.

Model	$\bar{\tau}_{e_1}$ [Nm]	$\bar{\tau}_{e_2}$ [Nm]	$\bar{\tau}_{e_3}$ [Nm]	$\bar{\tau}_{e_4}$ [Nm]	$\bar{\tau}_{e_5}$ [Nm]	$\bar{\tau}_{e_6}$ [Nm]	$\bar{\tau}_{e_7}$ [Nm]
Nominal 1 kg	**3.03**	4.05	**3.58**	2.58	0.47	**0.70**	**0.14**
Ours 1 kg	3.95	4.12	4.27	**2.47**	0.57	1.02	0.19
EKF 1 kg	4.32	**3.56**	4.83	2.67	**0.43**	0.83	**0.14**
Nominal 2 kg	7.27	8.70	8.08	6.40	**1.00**	1.55	0.42
Ours 2 kg	**6.36**	6.83	**6.90**	4.45	1.24	1.86	0.53
EKF 2 kg	7.26	**6.04**	7.93	**4.12**	1.01	**1.47**	**0.42**
Nominal 3 kg	10.96	11.69	12.13	9.31	1.43	2.11	**0.56**
Ours 3 kg	**9.40**	**5.92**	**9.68**	4.95	1.72	2.28	0.84
EKF 3 kg	10.00	5.97	10.84	**4.70**	**1.37**	**1.85**	**0.56**

The tracking errors are presented in Table IV. Our controller outperformed position and velocity tracking for almost all the joints when compared to EKF MPC and Nominal MPC. Only the position tracking for q_2 and q_6 presented larger errors, probably due to trade-offs in the cost function, as mentioned before. As the last joint was the only one with a constant reference, its error analysis is irrelevant as they are very small. The EKF MPC still presented an overall performance than the nominal one, but it was not able to improve the errors for the first and third joints.

The first three cycles of the trajectories for each load, around 4.2 seconds, are presented in Figures 4a-4c. Our method achieves the best tracking performance, reducing the RMSE in Cartesian space by 28.24%, 57.93%, and 62.10% for 1 kg, 2 kg, and 3 kg, respectively, compared to 14.80%, 33.58%, and 35.62% achieved by the EKF MPC.

The neural network, as part of the MPC's model, increases the optimization complexity and, consequently, the computational time. In Table V, we present the average total time $\bar{t}_{\text{total}}$ from the start of the MPC update to the moment the computed torque is sent, and the model time $\bar{t}_{\text{model}}$ spent by the solver on model discretization and simulation. As the EKF does not add any complexity to the model, its times are comparable to the nominal one. CaDeLaC increased the total time by a factor of 4 due to the extra complexity of the model, as demonstrated by a higher $\bar{t}_{\text{model}}$. However, the delay introduced by the additional computational time did not affect the hardware experiments, as they were almost 50% below the control period of 20 ms. Also, due to the
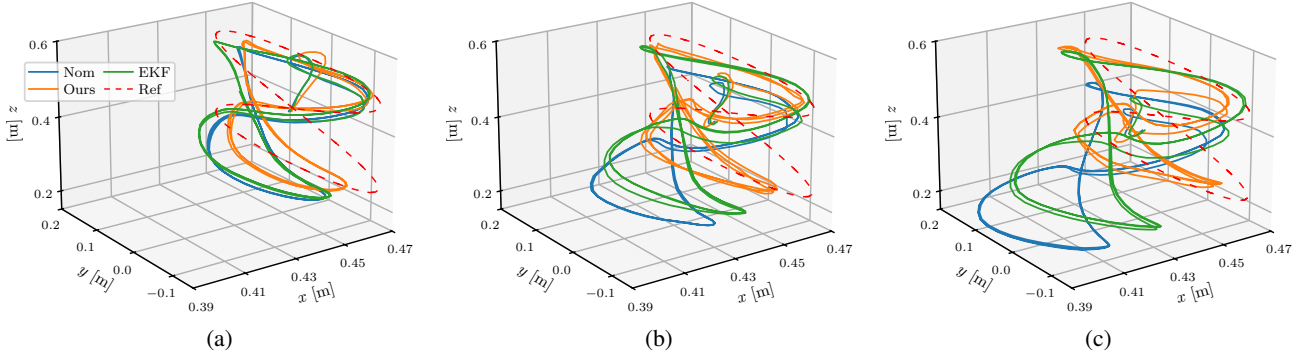
Fig. 4: End-effector trajectory in high-speed experiments on the real robot. (a) 1 kg: CaDeLaC reduced the RMSE by 28.24% (EKF MPC: 14.80%); (b) 2 kg: 57.93% (33.58%); (c) 3 kg: 62.10% (35.62%).

TABLE IV: Tracking position e and velocity ė RMSE for the three controllers over high-speed trajectories with three different loads.

Controller	e_1 [rad]	e_2 [rad]	e_3 [rad]	e_4 [rad]	e_5 [rad]	e_6 [rad]	e_7 [rad]
Nominal 1 kg	0.032	**0.020**	0.024	0.054	0.017	0.015	0.002
CaDeLaC 1 kg	0.032	0.034	0.028	**0.036**	0.009	0.019	**0.001**
EKF MPC 1 kg	**0.028**	0.025	**0.023**	0.046	**0.008**	**0.014**	0.003
Nominal 2 kg	0.044	0.027	0.035	0.091	0.016	0.026	0.007
CaDeLaC 2 kg	**0.029**	0.052	**0.020**	**0.036**	**0.011**	0.022	0.005
EKF MPC 2 kg	0.040	**0.021**	0.033	0.058	**0.011**	**0.014**	**0.005**
Nominal 3 kg	0.053	0.057	0.045	0.124	0.022	0.031	**0.002**
CaDeLaC 3 kg	**0.033**	0.076	0.028	**0.031**	**0.014**	0.027	0.009
EKF MPC 3 kg	0.050	**0.017**	0.047	0.069	0.020	**0.016**	0.004

Controller	$\dot{e}_1$ [rad/s]	$\dot{e}_2$ [rad/s]	$\dot{e}_3$ [rad/s]	$\dot{e}_4$ [rad/s]	$\dot{e}_5$ [rad/s]	$\dot{e}_6$ [rad/s]	$\dot{e}_7$ [rad/s]
Nominal 1 kg	0.379	0.243	0.317	0.150	0.100	0.133	**0.008**
CaDeLaC 1 kg	**0.326**	**0.196**	**0.261**	**0.128**	0.105	0.162	0.015
EKF MPC 1 kg	0.368	0.221	0.322	0.166	**0.081**	**0.122**	0.009
Nominal 2 kg	0.513	0.287	0.454	0.202	**0.137**	0.179	0.040
CaDeLaC 2 kg	**0.345**	**0.228**	**0.301**	**0.167**	**0.137**	0.255	0.052
EKF MPC 2 kg	0.506	0.264	0.467	0.226	0.140	**0.158**	**0.037**
Nominal 3 kg	0.603	0.293	0.554	0.254	**0.201**	0.236	**0.035**
CaDeLaC 3 kg	**0.420**	**0.260**	**0.405**	**0.213**	0.232	0.311	0.124
EKF MPC 3 kg	0.611	0.283	0.597	0.305	0.217	**0.234**	0.040

real-time kernel and the SQP-RTI, the total time variance was low, around 10^{-7}.

TABLE V: Average total and model computational times in high-speed trajectories.

Controller	$\bar{t}_{total}$[ms]	$\bar{t}_{model}$[ms]
Nominal	**2.59**	1.22
CaDeLaC	10.57	8.50
EKF MPC	2.70	**1.19**

2) Pick-and-place task: To demonstrate that our method can adapt to dynamic changes in the environment, we define a *pick-and-place* task, which consists of two repeated phases. In each phase, the robot must approach a gym weight, grasp it, perform a rapid motion in the air, and place it in a location different from the initial one. During the trajectory, which lasts 26 seconds, the system's dynamics change four

times. The robot waits 3 seconds at each pick-and-place position to ensure a safe grasp and disengagement of the strap. Thus, only 14 seconds of the full trajectory are spent in motion. For this experiment, we reattached the gripper to the end-effector and attached straps to the gym load, since the gripper does not have sufficient force to directly hold the gym weight. The experimental setup is depicted in Figure 3b. The slack caused by holding the strap introduced an additional challenge for the controller, as it allows the gym weight to swing, changing the relative position of the load's center of mass. The sequence of weights during the task was 3 kg and 2 kg. Since the gripper was reattached, the additional loads relative to the nominal model are 2 kg and 3 kg, with the latter being the maximum payload specified by the manufacturer.

The results of the evaluated models are presented in Tables VI. Similar to the simulation results and as previously mentioned, the EKF could not capture the residual torque for the first and third joints. While Context-Aware DeLaN showed significant improvement across all joints, except for the last joint, which already had a small error.

TABLE VI: Average residual torque RMSE $\bar{\tau}_e$ for the three models in the pick-and-place task.

Model	$\bar{\tau}_{e_1}$ [Nm]	$\bar{\tau}_{e_2}$ [Nm]	$\bar{\tau}_{e_3}$ [Nm]	$\bar{\tau}_{e_4}$ [Nm]	$\bar{\tau}_{e_5}$ [Nm]	$\bar{\tau}_{e_6}$ [Nm]	$\bar{\tau}_{e_7}$ [Nm]
Nominal	0.69	7.97	0.78	6.74	0.19	1.93	**0.03**
Ours	**0.64**	**2.32**	**0.59**	**1.55**	**0.14**	**0.45**	0.04
EKF	1.70	2.78	1.65	2.21	0.24	0.63	**0.03**

The tracking errors are presented in Table VII. Although the EKF MPC had the best position tracking for q_2, q_4, and q_5, CaDeLaC presented an overall improvement in position and velocity tracking, particularly for q_3. Note that, since almost half of the full trajectory is spent in waiting positions, the RMSE values will not vary much among the controllers compared to a full trajectory in motion.

Regarding the end-effector trajectory, the obtained trajectories for the three controllers are presented in Figure 5. Visually, CaDeLaC demonstrated better tracking, as it reduced the end-effector position RMSE by 39%, compared to only 21% with the EKF.

TABLE VII: Tracking position $\mathbf{e}$ and velocity $\dot{\mathbf{e}}$ RMSE for the three controllers in the pick-and-place task.

Controller	e_1 [rad]	e_2 [rad]	e_3 [rad]	e_4 [rad]	e_5 [rad]	e_6 [rad]	e_7 [rad]
Nominal	**0.011**	0.055	0.005	0.073	0.016	0.052	**0.001**
CaDeLaC	**0.011**	0.044	**0.004**	0.032	0.014	**0.015**	**0.001**
EKF MPC	**0.011**	**0.038**	0.018	**0.028**	**0.007**	**0.015**	**0.001**

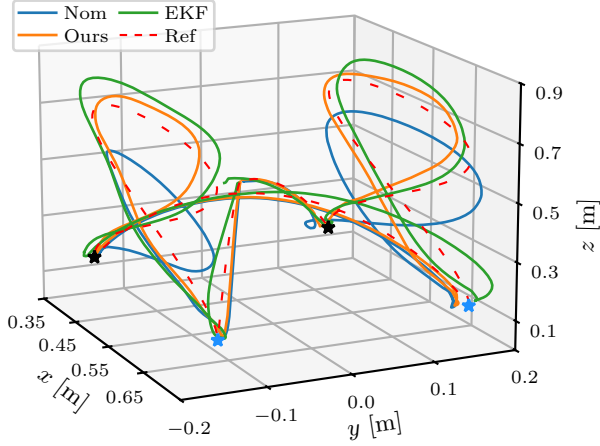| Controller	$\dot{e}_1$ [rad/s]	$\dot{e}_2$ [rad/s]	$\dot{e}_3$ [rad/s]	$\dot{e}_4$ [rad/s]	$\dot{e}_5$ [rad/s]	$\dot{e}_6$ [rad/s]	$\dot{e}_7$ [rad/s]
Nominal	**0.018**	0.121	0.016	0.055	0.012	0.068	**0.001**
CaDeLaC	**0.018**	0.104	**0.014**	**0.049**	0.011	0.061	**0.001**
EKF MPC	0.019	**0.101**	0.023	0.057	**0.008**	**0.048**	**0.001**

Fig. 5: End-effector trajectory in the pick-and-place task for the three evaluated controllers. Blue and black markers indicate the pick and place positions, respectively. Each marker corresponds to a change in the payload and, consequently, in the system's dynamics.

Since the controllers and hardware setup, including the computers and communication architecture, remain the same, we omit the computational time analysis, as it will be identical to the first hardware experiment.

IV. CONCLUSION

In this work, we presented a method for online identification of physics-consistent DeLaN models and their application in model predictive control, named CaDeLaC. To enable online adaptation, we introduced latent environment embeddings as additional inputs to DeLaN and jointly learned a recurrent system identification network alongside contextual DeLaN. To meet the computational requirements of MPC, we proposed modeling only the residual dynamics with respect to a nominal model, enabling good predictive performance with small network sizes. Our real robot experiments demonstrated excellent tracking performance for fast motions under varying loads, outperforming the baselines. For future work, we want to apply our method to higher-dimensional and underactuated systems, such as humanoids. Furthermore, we want to investigate alternative objectives for training the history-based encoder, in particular, using

reinforcement learning to directly optimize it with respect to downstream task performance.

APPENDIX

A. Data Collection and Training

The data collection and training are conducted on an AMD Ryzen 9 5900X processor with an NVIDIA GeForce RTX 4080 GPU and 32 GB of RAM. The parameters for both are presented in Tables VIII and IX.

TABLE VIII: Data collection parameters.

Parameter	Values
$\mathbf{Q}_{\mathrm{LQR}}$	$\mathrm{diag}(100 \times \mathbf{1}_{14})$
$\mathbf{R}_{\mathrm{LQR}}$	$\mathrm{diag}(0.05 \times \mathbf{1}_{14})$
$\mathbf{q}_{\max}$	[2.8973, 1.7628, 2.8973, -0.0698, 2.8973, 3.7525, 2.8973]
$\mathbf{q}_{\min}$	[-2.8973, -1.7628, -2.8973, -3.0718, -2.8973, -0.0175, -2.8973]
$\dot{\mathbf{q}}_{\max}, -\dot{\mathbf{q}}_{\min}$	$2 \times \mathbf{1}_7$
Range $\mathbf{a}_{\mathrm{e}}, \mathbf{b}_{\mathrm{e}}$	$\pm[0.20, 0.35, 0.58, 0.30, 0.58, 0.38, 0.58]$
Samples	1.01 M

TABLE IX: Training hyperparameters.

Parameter	Values
$\mathrm{var}(\dot{\mathbf{q}})$	$10^{-3} \times \mathbf{1}_7$
$\mathrm{var}(\ddot{\mathbf{q}})$	[0.05, 0.05, 0.15, 0.03, 0.15, 0.25, 0.65]
$\mathrm{var}(\boldsymbol{\tau})$	[0.5, 0.1, 0.5, 0.3, 0.01, 0.01, 0.01]
$\mathrm{var}(\tilde{\boldsymbol{\tau}})$	[1.0, 0.5, 1.0, 0.4, 0.02, 0.03, 0.02]
Activation	Tanh
Batch Size	1024
Learning Rate	5×10^{-4}
Epochs	3000
n_{h}	15
ϵ	0.1

B. Experiments setup

The inference and control experiments are performed on an AMD Ryzen 9 3900X processor with 64 GB of RAM, running Ubuntu 22.04 with a real-time kernel 5.15.0. All parameters related to the controllers are presented in Table X.

TABLE X: Controller parameters.

Parameter	Values
n_{y}	12
Control frequency	50 Hz
Discretization time	20 ms
$\mathbf{Q}$	$\mathrm{diag}(100 \times \mathbf{1}_4, 50 \times \mathbf{1}_3, 10^3 \times \mathbf{1}_4, 500 \times \mathbf{1}_3)$
$\mathbf{R}$	$\mathrm{diag}(0.2 \times \mathbf{1}_4, 1.0 \times \mathbf{1}_3)$
$\boldsymbol{\tau}_{\max}, -\boldsymbol{\tau}_{\min}$	[87, 87, 87, 87, 12, 12, 12]
$\mathbf{Q}_{\mathrm{EKF}}$	$\mathrm{diag}(\mathbf{1}_7, 10^{-1} \times \mathbf{1}_7, 10^2 \times \mathbf{1}_3)$
$\mathbf{R}_{\mathrm{EKF}}$	$\mathrm{diag}(10^{-2} \times \mathbf{1}_{14})$
$\mathbf{K}_{\mathrm{p}}$	[10, 10, 10, 10, 5, 5, 2]
$\mathbf{K}_{\mathrm{d}}$	[2.5, 2.5, 2.5, 2.5, 1.5, 1.5, 1.5]

REFERENCES

[1] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 3, pp. 49–59, 1987. [Online]. Available: https://doi.org/10.1177/027836498700600303

[2] C. H. An, C. G. Atkeson, and J. M. Hollerbach, "Estimation of inertial parameters of rigid body links of manipulators," in *1985 24th IEEE Conference on Decision and Control*, 1985, pp. 990–995.

[3] J. Huang, D. Tateo, P. Liu, and J. Peters, "Adaptive control based friction estimation for tracking control of robot manipulators," *IEEE Robotics and Automation Letters*, vol. 10, no. 3, pp. 2454–2461, 2025.

[4] J. Foster, S. McCrory, C. DeBuys, S. Bertrand, and R. Griffin, "Physically consistent online inertial adaptation for humanoid loco-manipulation," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 11 278–11 285.

[5] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," *Cognitive processing*, vol. 12, pp. 319–40, 04 2011.

[6] M. S. Duy Nguyen-Tuong and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[7] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2020.

[8] J. Matschek, J. Bethge, and R. Findeisen, "Safe machine-learning-supported model predictive force and motion control in robotics," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 6, pp. 2380–2392, 2023.

[9] G. Petkos and S. Vijayakumar, "Load estimation and control using learned dynamics models," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 1527–1532.

[10] L. F. Recalde, J. Varela, B. S. Guevara, V. Andaluz, and D. Gandolfo, "Adaptive nmpc-rbf with application to manipulator robots," in *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2023, pp. 2475–2482.

[11] J. Watson *et al.*, "Machine learning with physics knowledge for prediction: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2408.09840

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

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

[14] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[15] M. Lutter, K. D. Listmann, and J. Peters, "Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7718–7725, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195874061

[16] T. Duong, A. Altawaitan, J. Stanley, and N. Atanasov, "Port-hamiltonian neural ode networks on lie groups for robot dynamics learning and control," *IEEE Transactions on Robotics*, vol. 40, pp. 3695–3715, 2024.

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

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:206594692

[19] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-Learning in Neural Networks: A Survey," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 44, no. 09, pp. 5149–5169, Sept. 2022. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TPAMI.2021.3079209

[20] R. C. Grande, G. V. Chowdhary, and J. P. How, "Nonparametric adaptive control using gaussian processes with online hyperparameter estimation," *52nd IEEE Conference on Decision and Control*, pp. 861–867, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:7528974

[21] E. Arcari, A. Carron, and M. N. Zeilinger, "Meta learning mpc using finite-dimensional gaussian process approximations," *ArXiv*, vol. abs/2008.05984, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221112284

[22] A. Carron, E. Arcari, M. Wermelinger, L. Hewing, M. Hutter, and M. N. Zeilinger, "Data-driven model predictive control for trajectory tracking with a robotic arm," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3758–3765, 2019.

[23] S. M. Richards, N. Azizan, J.-J. E. Slotine, and M. Pavone, "Adaptive-control-oriented meta-learning for nonlinear systems," *ArXiv*, vol. abs/2103.04490, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232147745

[24] E. Arcari, M. V. Minniti, A. Scampicchio, A. Carron, F. Farshidian, M. Hutter, and M. N. Zeilinger, "Bayesian multi-task learning mpc for robotic mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3222–3229, 2023.

[25] D. Lapandić, F. Xie, C. K. Verginis, S.-J. Chung, D. V. Dimarogonas, and B. Wahlberg, "Meta-learning augmented mpc for disturbance-aware motion planning and control of quadrotors," *IEEE Control Systems Letters*, vol. 8, pp. 3045–3050, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273229398

[26] W. Yu, J. Tan, C. Karen Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Robotics: Science and Systems XIII*, ser. RSS2017. Robotics: Science and Systems Foundation, July 2017. [Online]. Available: http://dx.doi.org/10.15607/RSS.2017.XIII.048

[27] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125

[28] Q. Li, T. Wang, V. Roychowdhury, and M. K. Jawed, "Metalearning generalizable dynamics from trajectories," *Physical Review Letters*, vol. 131, no. 6, p. 067301, 2023.

[29] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh, "Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2819–2826, 2022.

[30] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural-mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, 2023.

[31] T. P. Duong and N. A. Atanasov, "Adaptive control of se(3) hamiltonian dynamics with learned disturbance features," *IEEE Control Systems Letters*, vol. 6, pp. 2773–2778, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:247597120

[32] R. Verschueren *et al.*, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[33] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020, 21st IFAC World Congress.

[34] J. Carpentier *et al.*, "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.

[35] J. Ansel *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] E. F. Camacho and C. Bordons Alba, *Model predictive control*. Springer-Verlag London, 2007.

[38] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[39] L. Roveda, D. Riva, G. Bucca, and D. Piga, "External joint torques estimation for a position-controlled manipulator employing an extended kalman filter," in *2021 18th International Conference on Ubiquitous Robots (UR)*, 2021, pp. 101–107.