Journal of Process Control 158 (2026) 103634

Contents lists available at ScienceDirect

Journal of Process Control

journal homepage: www.elsevier.com/locate/jprocont

An explicit model predictive control framework based on physics-informed neural networks

Argyri Kardamaki [a], Teo Protoulis [b], Alex Alexandridis [b], Haralambos Sarimveis [a,*]

[a] National Technical University of Athens, School of Chemical Engineering, Iroon Polytechniou 9, Zografou, Athens, 15772, Greece
[b] University of West Attica, Department of Electrical and Electronic Engineering, Thivon 250 & P. Ralli, Aigaleo, Athens, 12244, Greece

ARTICLE INFO

Dataset link: https://github.com/ntua-unit-of-control-and-informatics/pinn-mpc

Keywords:
Explicit control
Model predictive control
Physics-informed neural networks

ABSTRACT

This paper presents a novel control framework that integrates Physics-Informed Neural Networks (PINNs) with Model Predictive Control (MPC) for nonlinear dynamical systems. Unlike traditional MPC, which requires solving optimization problems in real time, the proposed method trains a single feedforward neural network to serve as an explicit controller that directly maps the current state, set-point, and disturbance signals to optimal control actions. The network is trained using a composite loss function that enforces the governing differential equations while incorporating control-oriented objectives such as set-point tracking, control smoothness, and soft constraints on states, inputs, and outputs. The proposed controller is validated on both single-input single-output (SISO) and multi-input multi-output (MIMO) water-tank benchmark systems, demonstrating accurate set-point tracking, effective measured disturbance rejection, and strong generalization across thousands of randomized test scenarios. A runtime comparison with a nonlinear MPC performing online optimization confirms that the explicit PINN-MPC approach achieves comparable control performance while requiring several orders of magnitude less computation time. These results highlight the scalability and computational efficiency of the proposed framework, positioning it as a novel paradigm for real-time control of nonlinear systems.

1. Introduction

Model Predictive Control (MPC) is a widely used framework for constrained multivariable control, applied across domains such as process engineering, robotics, and energy systems [1–7]. With the aid of a model that forecasts the system's dynamic behavior, it solves an optimization problem over a finite horizon, and applies only the first control action in a receding horizon fashion [8]. MPC often assumes linear models for simplicity and computational efficiency. Nonlinear MPC (NMPC) offers increased modeling accuracy but introduces significant computational demands, as it requires solving a nonlinear optimization problem at every sampling instant. If the solver cannot produce a reliable solution in time, the resulting delays or suboptimal control actions may compromise system performance [9]. In practice, the effectiveness of the MPC paradigm is constrained by both the accuracy of the predictive model and its reliance on time-critical online optimization, which can become a major bottleneck for fast dynamic systems and controllers with limited computational capacity.

Explicit MPC (eMPC) addresses this limitation by pre-solving the optimization problem offline across all admissible initial states, yielding a closed-form solution to the underlying Optimal Control Problem (OCP). The resulting control law maps the current system state to control actions using a precomputed function – typically a PieceWise-Affine (PWA) mapping [10]. This framework is especially effective for linear systems with convex cost functions, where the control problem can be reformulated as a multi-parametric linear or quadratic program (mpLP/mpQP) and solved offline to yield an explicit control law [11]. Foundational work by Pistikopoulos et al. formalized the theory of multi-parametric programming and its control applications, including the derivation of solution structures and the development of efficient algorithms [12,13]. Despite these advances, linear eMPC remains subject to the curse of dimensionality, as the number of regions can grow exponentially with the system order and prediction horizon, leading to increased computational effort during region evaluation at runtime [14].

Extending eMPC to nonlinear systems is significantly more challenging. Solving multi-parametric NonLinear Programs (mpNLPs) is generally intractable in closed form, and no general framework exists to derive explicit control laws with guaranteed continuity. This gap has been recently addressed by developing explicit solutions for

* Corresponding author.
E-mail address: hsarimv@mail.ntua.gr (H. Sarimveis).

https://doi.org/10.1016/j.jprocont.2026.103634
Received 22 July 2025; Received in revised form 14 November 2025; Accepted 13 January 2026
Available online 17 January 2026

quadratically constrained and nonlinear problems using multiparametric optimization frameworks [15]. Approximate methods – such as grid-based discretization, lattice-based PWA approximations [16], or feedback synthesis over sampled domains [17] – have been proposed, yet they usually provide only weak continuity guarantees.

Data-driven approaches have emerged as a flexible alternative for modeling and controlling nonlinear systems, as they can learn system dynamics solely from data collected from the process under investigation [18–21]. More specifically, and especially in recent years, machine learning frameworks have been developed with the aim of directly approximating control policies commonly used in process control systems, such as predictive control schemes [22–27]. In particular, neural networks and symbolic regression techniques have been used to approximate NMPC policies offline and deploy them in solver-free form. For example, Chen et al. [28] trained deep networks to approximate optimal control laws for nonlinear systems, while Drgoňa et al. [29] proposed Differentiable Predictive Control (DPC), where a learned model is embedded into a differentiable control architecture trained end-to-end. Although these methods offer fast execution, they do not incorporate the physical principles governing the system and may fail to guarantee constraint satisfaction.

A compelling path to overcoming these limitations lies in hybrid modeling, which integrates mechanistic (first-principles) knowledge with data-driven components to combine predictive accuracy with interpretability [30]. In this work, we specifically investigate the use of Physics-Informed Neural Networks (PINNs) – a powerful subset of hybrid modeling which embed known physical laws directly into the neural network's training process – for designing explicit controllers, particularly in the context of nonlinear dynamical systems. Originally introduced by Raissi et al. [31,32] for solving forward and inverse problems governed by Partial Differential Equations (PDEs), PINNs have shown strong performance in learning system dynamics, even in settings with limited or noisy data.

In control settings, PINNs have primarily been explored as surrogate or auxiliary models for complex dynamics, offering a data-efficient alternative to purely data-driven methods while preserving a conventional control structure. The core function of these approaches is to enhance the predictive accuracy or computational speed of the embedded dynamic model, rather than directly generating the control action. For instance, Lim et al. [33] demonstrated this concept using a control framework, where the network acted as an auxiliary compensator operating in parallel with a conventional Proportional-Integral (PI) controller to account for valve degradation and parameter drift. Liu et al. [34] developed PINN-based models to capture nonconservative robotic dynamics, which were then used to parameterize a conventional model-based feedback controller to improve trajectory tracking. Hart et al. [35] formulated a Lyapunov-based PINN controller where the PINN acted as an adaptive function approximator of unknown dynamics, ensuring stability through its integration into a predefined, Lyapunov-stable control structure. In model predictive frameworks, Antonelo et al. [36] proposed Physics-Informed Neural Nets for Control (PINC) as an efficient long-horizon prediction model for use within MPC, and Patel et al. [37] proposed an MPC framework that employs a PINN to provide the system predictions needed by the control algorithm in a fast, differentiable manner, facilitating the timely and efficient solution of the online nonlinear optimization problem. Collectively, these studies highlight the considerable promise of physics-informed learning in control theory but also reveal a critical gap: existing frameworks employ PINNs to improve predictive models embedded within an established, conventional control architecture; however, to the best of the authors' knowledge, no existing approach has yet demonstrated a fully explicit, control law grounded on physical dynamics.

In this study, we attempt to close this gap by proposing a novel explicit control framework that combines the strengths of PINNs with core principles of MPC, including trajectory optimization over a prediction horizon and the incorporation of input and state constraints. Rather than using PINNs as surrogate models, we train a unified PINN-MPC to directly synthesize control laws that respect both physical dynamics and control objectives. The network is trained end-to-end using a composite loss function that incorporates Ordinary Differential Equation (ODE) residuals to enforce physical consistency, penalties for set-point, and input tracking over a prediction horizon, incremental control terms to promote actuator smoothness, and soft constraints to enforce bounds on inputs, states and outputs.

After training, the resulting PINN-MPC provides a direct mapping from the current state, set-point, input, and measured disturbance to the next control action, enabling very fast real-time inference without the need for online optimization. This design results in a compact and generalizable control law that is well-suited for nonlinear systems, combining physical consistency with real-time applicability. We demonstrate the effectiveness of the proposed approach on two nonlinear fluid-level regulation benchmarks: a Single-Input Single-Output (SISO) system and a Multi-Input Multi-Output (MIMO) system. Both cases are evaluated across a broad range of initial conditions, set-points, and disturbance scenarios. In all configurations, the PINN-MPC achieves fast convergence, smooth actuation, and near-zero steady-state error across thousands of closed-loop simulations, demonstrating its scalability and robustness to increasing system dimensionality.

The remainder of this paper is structured as follows: Section 2 introduces the fundamental principles of PINNs. Section 3 presents the proposed PINN-MPC framework, detailing the controller architecture, loss function formulation, training procedure, and deployment strategy. Section 4 reports the results and discussion, featuring two case studies: a single-tank and a quadruple-tank water system. For each case study, we describe the system dynamics, network configuration, and the hyperparameter tuning process, followed by an evaluation of the controller's closed-loop performance in terms of reference tracking and measured-disturbance rejection. This section also provides a direct runtime comparison between the explicit PINN-MPC and a conventional NMPC implementation. Finally, Section 5 concludes the paper with a summary of key findings and directions for future work.

2. Preliminaries

In 2017, Raissi et al. [31,32] introduced PINNs as supervised deep neural networks trained to satisfy physical laws expressed in the form of differential equations. Their approach naturally handles ODEs as a special case of PDEs, where spatial derivatives are absent. A general form of a system of ODEs is given by:

$$\boldsymbol{x}_t + \mathcal{N}[\boldsymbol{x}] = 0, t \in [0, T] \tag{1}$$

where $\boldsymbol{x}$ denotes the solution and $\mathcal{N}[\cdot]$ is a nonlinear differential operator. We define as $\mathcal{F}(\boldsymbol{x})$ the left-hand side of (1) so that:

$$\mathcal{F}(\boldsymbol{x}) = \boldsymbol{x}_t + \mathcal{N}[\boldsymbol{x}] \tag{2}$$

PINNs are deep neural networks with trainable weights and biases that learn to approximate the solutions $\boldsymbol{x}(t)$ of differential equations, denoted as $\hat{\boldsymbol{x}}(t)$, by minimizing a suitably constructed loss function that typically takes the form of (3):

$$MSE = MSE_x + MSE_{\mathcal{F}} \tag{3}$$

where

$$MSE_x = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{1}{N_{\text{tr}}}\sum_{j=1}^{N_{\text{tr}}}|\hat{x}_i(t_j^{\text{tr}}) - x_i(t_j^{\text{tr}})|^2 \tag{4}$$

and

$$MSE_{\mathcal{F}} = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{1}{N_{\text{col}}}\sum_{j=1}^{N_{\text{col}}}|\mathcal{F}(\hat{x}_i(t_j^{\text{col}}))|^2 \tag{5}$$

where: N_{tr}, N_{col}, and N_x denote the number of training data samples, the number of collocation points, and the number of states, respectively; $x_i(\cdot)$ represents the ith state; $\{(t_j^{\text{tr}}, x_i(t_j^{\text{tr}}))\}_{i=1}^{N_{\text{tr}}}$ correspond to the

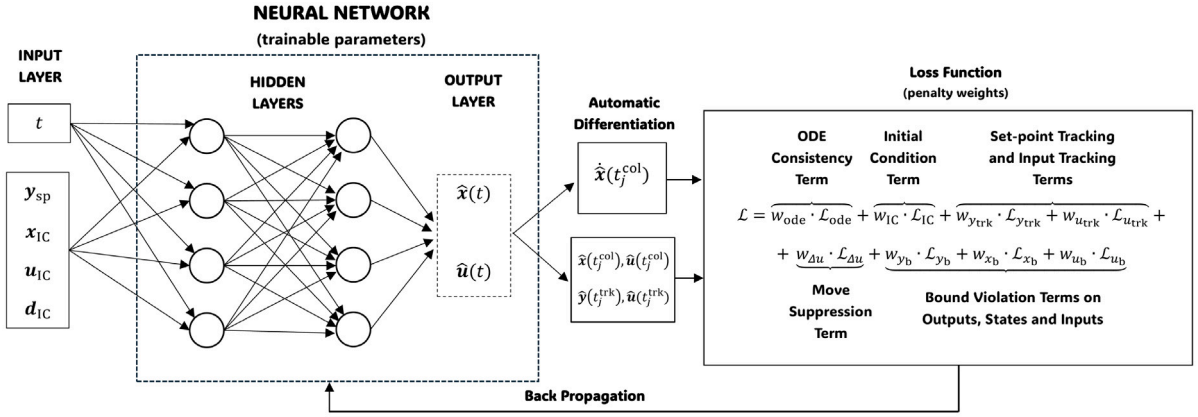

Fig. 1. PINN-MPC training structure.

collected training data for each state x_i at time points $\{t_j^{\text{tr}}\}_{j=1}^{N_{\text{tr}}}$, while $\hat{x}_i(t_j^{\text{tr}})$ are the corresponding predictions of the network; $\hat{x}_i(t_j^{\text{col}})$ are the predictions of the network at the collocation time points $\{t_j^{\text{col}}\}_{j=1}^{N_{\text{col}}}$, which are used to enforce the governing ODE via (2).

In essence, the first term, MSE_x in (4), minimizes the deviation between the predicted and collected data, while the second term, MSE_F in (5), enforces physical consistency by minimizing the ODE residuals at the collocation points. This formulation allows PINNs to generalize from limited data by incorporating physical knowledge directly into the training objective [31,32], which is not the case for conventional neural network modeling approaches.

3. Proposed PINN-MPC framework

3.1. Structure of the PINN-MPC

Building on the principles of PINNs and MPC, we propose a novel control framework that synthesizes optimal control actions by learning trajectory behavior governed by the system's underlying dynamics. In contrast to traditional MPC, which requires solving an optimization problem at each sampling instant, the proposed PINN-MPC method trains a single network offline to generate both optimal state and control trajectories directly from initial conditions and set-point targets.

For the purposes of this paper, we consider MIMO systems whose underlying dynamics are described by the following form:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{d}(t)\big) \tag{6a}$$

$$\boldsymbol{y}(t) = \boldsymbol{h}\big(\boldsymbol{x}(t)\big) \tag{6b}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{N_x}$ denotes the state vector (with N_x the number of states), $\boldsymbol{u}(t) \in \mathbb{R}^{N_u}$ denotes the control-input vector (with N_u the number of manipulated inputs), $\boldsymbol{y}(t) \in \mathbb{R}^{N_y}$ denotes the output vector (with N_y the number of controlled outputs), and $\boldsymbol{d}(t) \in \mathbb{R}^{N_d}$ denotes a piecewise-constant disturbance signal that is considered to be available for measurement in the context of this work (with N_d the number of measured disturbances). The vector field $\boldsymbol{f} : \mathbb{R}^{N_x+N_u+N_d} \to \mathbb{R}^{N_x}$ and the output mapping $\boldsymbol{h} : \mathbb{R}^{N_x} \to \mathbb{R}^{N_y}$ are assumed to be smooth and continuous, representing the system dynamics and the relationship between the states and controlled outputs, respectively.

Fig. 1 illustrates the proposed architecture of the PINN-MPC. In the generalized setting with N_x system states, N_u manipulated inputs, and N_y controlled outputs, the network takes as inputs the time t, the set-point vector $\boldsymbol{y}_{\text{sp}} \in \mathbb{R}^{N_y}$, the initial state vector $\boldsymbol{x}_{\text{IC}} \in \mathbb{R}^{N_x}$, and the initial control input vector $\boldsymbol{u}_{\text{IC}} \in \mathbb{R}^{N_u}$. To account for N_d measured disturbances affecting the plant dynamics, an additional input vector signal, $\boldsymbol{d}_{\text{IC}} \in \mathbb{R}^{N_d}$, is introduced, which is assumed to remain constant throughout the prediction horizon.

The output layer of the proposed PINN-MPC produces two outputs: the predicted state trajectory $\hat{\boldsymbol{x}}(t; \boldsymbol{y}_{\text{sp}}, \boldsymbol{x}_{\text{IC}}, \boldsymbol{u}_{\text{IC}}, \boldsymbol{d}_{\text{IC}}) \in \mathbb{R}^{N_x}$ and the corresponding control trajectory $\hat{\boldsymbol{u}}(t; \boldsymbol{y}_{\text{sp}}, \boldsymbol{x}_{\text{IC}}, \boldsymbol{u}_{\text{IC}}, \boldsymbol{d}_{\text{IC}}) \in \mathbb{R}^{N_u}$. The conditioning vectors $(\boldsymbol{y}_{\text{sp}}, \boldsymbol{x}_{\text{IC}}, \boldsymbol{u}_{\text{IC}}, \boldsymbol{d}_{\text{IC}})$ remain fixed inputs to the network throughout the prediction horizon, while the time variable t evolves continuously within this interval. For brevity, in the following equations we denote these predictions simply as $\hat{\boldsymbol{x}}(t)$ and $\hat{\boldsymbol{u}}(t)$, with the understanding that they depend implicitly on the conditioning vectors.

This formulation represents a significant departure from standard PINN configurations, which do not include the control signal in the network output. The predicted state is differentiated with respect to time using automatic differentiation, and the resulting derivative is incorporated into the loss function, alongside additional terms, to optimize the network parameters.

3.2. Loss function formulation

An overview of the loss function is provided in Fig. 1. For further analysis, its structure is divided into two parts: the physics-informed loss terms and the control-oriented loss terms.

(a) Physics-Informed Loss Terms:

The physics-informed terms correspond to the components used in the standard PINN formulations presented in Eqs. (4) and (5). These terms guide the network to produce solutions that are physically meaningful and compliant with the underlying differential equations:

- **ODE consistency term:** The first term, detailed in Eq. (7), enforces consistency with the known system dynamics by penalizing residuals of the governing ODE across selected *collocation points*:

$$\mathcal{L}_{\text{ode}} = \frac{1}{N_x}\sum_{i=1}^{N_x}\left(\frac{1}{N_{\text{col}}}\sum_{j=1}^{N_{\text{col}}}\left(\dot{\hat{x}}_i(t_j^{\text{col}}) - f_i(\hat{\boldsymbol{x}}(t_j^{\text{col}}), \hat{\boldsymbol{u}}(t_j^{\text{col}}), \boldsymbol{d}_{\text{IC}})\right)^2\right) \tag{7}$$

where N_{col} is the number of collocation points, and t_j^{col} denotes the jth collocation time point. Collocation points are time instants within the prediction horizon where the residuals of the differential equations are evaluated to enforce physical consistency during training. The terms $\hat{\boldsymbol{x}}(t_j^{\text{col}})$ and $\hat{\boldsymbol{u}}(t_j^{\text{col}})$ represent the network predictions for the vector of states and the vector of manipulated variables at time t_j^{col}, $\dot{\hat{x}}_i(t_j^{\text{col}})$ is the ith predicted state derivative obtained via automatic differentiation at t_j^{col}, and $f_i(\hat{\boldsymbol{x}}(t_j^{\text{col}}), \hat{\boldsymbol{u}}(t_j^{\text{col}}), \boldsymbol{d}_{\text{IC}})$ denotes the ith component of the vector field $\boldsymbol{f}(\cdot)$ from Eq. (6) evaluated at t_j^{col}, under the assumption – stated earlier – that the disturbance vector remains constant and equal to $\boldsymbol{d}_{\text{IC}}$ over the entire prediction horizon. As noted earlier, $\hat{\boldsymbol{x}}(t_j^{\text{col}})$ and $\hat{\boldsymbol{u}}(t_j^{\text{col}})$ denote shorthand forms of the network predictions conditioned on $(\boldsymbol{y}_{\text{sp}}, \boldsymbol{x}_{\text{IC}}, \boldsymbol{u}_{\text{IC}}, \boldsymbol{d}_{\text{IC}})$, and this convention applies to all subsequent equations.

- **Initial condition term:** The second term, detailed in Eq. (8), penalizes deviations between the PINN-MPC state predictions and the specified initial state conditions:

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_x}\sum_{i=1}^{N_x}\left(\left(\hat{x}_i(0) - x_{i,\text{IC}}\right)^2\right) \tag{8}$$

where $\hat{x}_i(0)$ is the ith predicted state at the start of the prediction horizon, and $x_{i,\text{IC}}$ is the initial condition of the ith state.

(b) Control-Oriented Loss Terms:

The additional loss terms extend the classical PINN formulation by incorporating key elements of optimal control, drawing inspiration from the objective functions and constraints typically employed in MPC configurations [8,38,39].

- **Set-point tracking term:** The set-point tracking term $\mathcal{L}_{y_{\text{trk}}}$ in Eq. (9) penalizes deviations between each predicted output and the corresponding desired reference $y_{l,\text{sp}}$, thereby promoting trajectory tracking across selected look-ahead *tracking* time points, that uniformly span the entire prediction horizon:

$$\mathcal{L}_{y_{\text{trk}}} = \frac{1}{N_y}\sum_{l=1}^{N_y}\left(\frac{1}{N_{\text{trk}}}\sum_{j=1}^{N_{\text{trk}}}\left(\hat{y}_l(t_j^{\text{trk}}) - y_{l,\text{sp}}\right)^2\right) \tag{9}$$

where N_{trk} is the number of tracking time points and t_j^{trk} denotes the jth tracking time point. The tracking time points are chosen as equidistant instants within the prediction horizon, serving as discrete sampling locations where the network predicted outputs are compared with the desired set-points. For convenience, the initial time $t_0^{\text{trk}} = 0$ is included among the tracking points and serves as the reference instant at which the controller computes the first control action for the subsequent control-related loss terms. The term $\hat{y}_l(t_j^{\text{trk}}) = h_l\left(\hat{\boldsymbol{x}}(t_j^{\text{trk}})\right)$ denotes the lth predicted output at time t_j^{trk}, where $h_l(\cdot)$ is the lth component of the output mapping $\boldsymbol{h}(\cdot)$ defined in Eq. (6), and $y_{l,\text{sp}}$ is the desired set-point for the lth output.
- **Input tracking term:** The input tracking term $\mathcal{L}_{u_{\text{trk}}}$ in Eq. (10) penalizes deviations between the predicted control inputs and the corresponding steady-state solutions of Eq. (6) associated with the desired reference vector $\boldsymbol{y}_{\text{sp}}$, evaluated across the tracking time points:

$$\mathcal{L}_{u_{\text{trk}}} = \frac{1}{N_u}\sum_{k=1}^{N_u}\left(\frac{1}{N_{\text{trk}}}\sum_{j=0}^{N_{\text{trk}}}\left(\hat{u}_k(t_j^{\text{trk}}) - u_{k,\text{ss}}^*\right)^2\right) \tag{10}$$

where $\hat{u}_k(t_j^{\text{trk}})$ is the kth predicted control input at time t_j^{trk}, and $u_{k,\text{ss}}^*$ is the corresponding steady state solution for the kth control input.
- **Move suppression term:** The move suppression term $\mathcal{L}_{\Delta u}$ in Eq. (11) is used to enforce symmetric bounds $[z_{\min}, z_{\max}]$, where $z_{\min} = -z_{\max}$, on input incremental changes between successive tracking time instants:

$$\mathcal{L}_{\Delta u} = \frac{1}{N_u}\sum_{k=1}^{N_u}\left(\frac{1}{N_{\text{trk}}}\sum_{j=0}^{N_{\text{trk}}}\hat{B}_{\text{rec}}\left(\Delta\hat{u}_k(t_j^{\text{trk}});\delta\right)\right) \tag{11}$$

where $\Delta\hat{u}_k(t_j^{\text{trk}})$ are the incremental changes for the kth control input at t_j^{trk}, defined by Eq. (12), and $\hat{B}_{\text{rec}}(\cdot;\delta)$ is the recentered barrier function, presented in Eq. (13).

$$\Delta\hat{u}_k(t_0^{\text{trk}}) = \hat{u}_k(t_0^{\text{trk}}) - u_{k,\text{IC}}, \tag{12a}$$

$$\Delta\hat{u}_k(t_j^{\text{trk}}) = \hat{u}_k(t_j^{\text{trk}}) - \hat{u}_k(t_{j-1}^{\text{trk}}), j = 1,\ldots,N_{\text{trk}}. \tag{12b}$$

where $u_{k,\text{IC}}$ is the initial condition for the kth control input.

$$\hat{B}_{\text{rec}}(z;\delta) = \begin{cases} -\log(z - z_{\min}) - \log(z_{\max} - z) + \log(-z_{\min}) \\ \quad + \log(z_{\max}), z - z_{\min} > \delta,\ z_{\max} - z > \delta \\ \beta(z - z_{\min};\delta) - \log(z_{\max} - z) + \log(-z_{\min}) \\ \quad + \log(z_{\max}), z - z_{\min} \le \delta,\ z_{\max} - z > \delta \\ -\log(z - z_{\min}) + \beta(z_{\max} - z;\delta) + \log(-z_{\min}) \\ \quad + \log(z_{\max}), z - z_{\min} > \delta,\ z_{\max} - z \le \delta \\ \beta(z - z_{\min};\delta) + \beta(z_{\max} - z;\delta) \\ \quad - \hat{B}(-z_{\min};\delta) - \hat{B}(z_{\max};\delta), \text{otherwise} \end{cases} \tag{13}$$

where $\beta(z;\delta)$ is a quadratic extension, with $\delta > 0$ denoting the relaxation parameter, as defined in Eq. (14):

$$\beta(z;\delta) = \frac{1}{2}\left(\frac{z - 2\delta}{\delta}\right)^2 - \frac{1}{2} - \log(\delta) \tag{14}$$

This formulation ensures that the penalty is minimized when $\Delta u = 0$ and increases smoothly as Δu approaches either bound. Beyond the bounds, and as $\delta \to 0$, the value of Eq. (13) increases rapidly, steering the optimization process toward regions of the search space that comply with the imposed constraints.
- **Bound violation terms:** The bound violation terms $\mathcal{L}_{y_{\text{b}}}$, $\mathcal{L}_{x_{\text{b}}}$ and $\mathcal{L}_{u_{\text{b}}}$ in Eqs. (16a), (16b) and (16c), are used to enforce upper and lower bound constraints on the controlled outputs, predicted states, and control inputs, respectively.

Inspired by the slack variable approach used in classical MPC formulations [40], we construct the terms $\mathcal{L}_{y_{\text{b}}}$, $\mathcal{L}_{x_{\text{b}}}$ and $\mathcal{L}_{u_{\text{b}}}$ to softly penalize violations of output, state and input constraints. These violations are captured using non-negative slack variables $e_{\min}(t)$ and $e_{\max}(t)$, which quantify the amount by which a variable exceeds its allowed range.

For each predicted output $\hat{y}_l(t)$, state $\hat{x}_i(t)$ and input $\hat{u}_k(t)$ we define:

$$e_{y_{l,\min}}(t) = \max\left(0,\ y_{l,\min} - \hat{y}_l(t)\right),$$
$$e_{y_{l,\max}}(t) = \max\left(0,\ \hat{y}_l(t) - y_{l,\max}\right). \tag{15a}$$

$$e_{x_{i,\min}}(t) = \max\left(0,\ x_{i,\min} - \hat{x}_i(t)\right),$$
$$e_{x_{i,\max}}(t) = \max\left(0,\ \hat{x}_i(t) - x_{i,\max}\right). \tag{15b}$$

$$e_{u_{k,\min}}(t) = \max\left(0,\ u_{k,\min} - \hat{u}_k(t)\right),$$
$$e_{u_{k,\max}}(t) = \max\left(0,\ \hat{u}_k(t) - u_{k,\max}\right). \tag{15c}$$

Each auxiliary variable evaluates to zero when the corresponding constraint is satisfied and grows linearly with the degree of violation otherwise.

The bound violation terms $\mathcal{L}_{y_{\text{b}}}$, $\mathcal{L}_{x_{\text{b}}}$ and $\mathcal{L}_{u_{\text{b}}}$ are finally constructed by squaring and averaging the violations across the tracking time instants. Therefore, these terms apply squared penalties only when violations occur, thereby promoting feasibility without imposing hard constraints.

$$\mathcal{L}_{y_{\text{b}}} = \frac{1}{N_y}\sum_{l=1}^{N_y}\left(\frac{1}{N_{\text{trk}}}\sum_{j=1}^{N_{\text{trk}}}\left(e_{y_{l,\min}}^2\left(t_j^{\text{trk}}\right) + e_{y_{l,\max}}^2\left(t_j^{\text{trk}}\right)\right)\right) \tag{16a}$$

$$\mathcal{L}_{x_{\text{b}}} = \frac{1}{N_x}\sum_{i=1}^{N_x}\left(\frac{1}{N_{\text{trk}}}\sum_{j=1}^{N_{\text{trk}}}\left(e_{x_{i,\min}}^2\left(t_j^{\text{trk}}\right) + e_{x_{i,\max}}^2\left(t_j^{\text{trk}}\right)\right)\right) \tag{16b}$$

$$\mathcal{L}_{u_{\text{b}}} = \frac{1}{N_u}\sum_{k=1}^{N_u}\left(\frac{1}{N_{\text{trk}}}\sum_{j=0}^{N_{\text{trk}}}\left(e_{u_{k,\min}}^2\left(t_j^{\text{trk}}\right) + e_{u_{k,\max}}^2\left(t_j^{\text{trk}}\right)\right)\right) \tag{16c}$$

(c) Overall Loss Function:

Together, all the previously defined loss terms guide the network to produce control trajectories that not only satisfy the underlying physics

but also align with the practical objectives of OCPs. The overall loss function is presented in Eq. (17):

$$\mathcal{L} = w_{\text{ode}}\,\mathcal{L}_{\text{ode}} + w_{\text{IC}}\,\mathcal{L}_{\text{IC}} + w_{y_{\text{trk}}}\,\mathcal{L}_{y_{\text{trk}}} + w_{u_{\text{trk}}}\,\mathcal{L}_{u_{\text{trk}}} + w_{\Delta u}\,\mathcal{L}_{\Delta u} + w_{y_{\text{b}}}\,\mathcal{L}_{y_{\text{b}}} + w_{x_{\text{b}}}\,\mathcal{L}_{x_{\text{b}}} + w_{u_{\text{b}}}\,\mathcal{L}_{u_{\text{b}}} \tag{17}$$

where $w_{\text{ode}}, w_{\text{IC}}, w_{y_{\text{trk}}}, w_{u_{\text{trk}}}, w_{\Delta u}, w_{y_{\text{b}}}, w_{x_{\text{b}}}$ and $w_{u_{\text{b}}}$ are the non-negative scalar weights that balance the contribution of each loss term. These weights are hyperparameters that can be tuned based on the relative importance of each objective in a given control problem, allowing to prioritize physical consistency, control feasibility, or tracking performance depending on the system dynamics and control requirements.

Remark 1. To address the move suppression term, we adopted relaxed barrier functions within the loss function, following established approaches from the MPC literature [41–43]. These functions are commonly used in constrained optimization to softly enforce variable bounds by penalizing violations directly in the objective. In the context of MPC, they offer a smooth and differentiable alternative to hard constraints, making them particularly suitable for integration into gradient-based training frameworks such as the one employed in this work.

Remark 2. It is important to highlight that, by construction, the proposed overall loss function possesses several key properties. First, Eq. (17) is continuously differentiable, enabling the use of gradient-based solvers that rely on explicit computation of loss gradients to efficiently explore the search space. Furthermore, the structure of Eq. (17) ensures that its global minimum, $\mathcal{L}^* = 0$, is achieved only when all objectives and constraints are simultaneously satisfied. Specifically, this minimum is reached only when each individual term in (17) evaluates to zero, which corresponds to ideal modeling accuracy, perfect tracking, and full constraint compliance. These properties establish Eq. (17) as a well-defined loss function, tailored to address the goals of the proposed framework.

Remark 3. Eq. (10) requires knowledge of the steady state of the control inputs $u^*_{\text{k,ss}}$. By virtue of (6), and assuming that the constant disturbance vector $\boldsymbol{d}$ is known and the desired output set-point $\boldsymbol{y}_{\text{sp}}$ is specified, the steady-state with respect to the state and input vectors $(\boldsymbol{x}_{\text{ss}}, \boldsymbol{u}_{\text{ss}})$ is defined as

$$(\boldsymbol{x}_{\text{ss}}, \boldsymbol{u}_{\text{ss}}) := \left\{(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^{N_x \times N_u} \,:\, \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d}) = \boldsymbol{0},\ \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{y}_{\text{sp}}\right\} \tag{18}$$

When a closed-form solution for $(\boldsymbol{x}_{\text{ss}}, \boldsymbol{u}_{\text{ss}})$ is not available, the equilibrium can be determined numerically by solving the system of nonlinear equations or by formulating an equivalent constrained optimization problem that enforces the above steady-state conditions within the feasible sets of the system states and control inputs [44–46].

3.3. PINN-MPC training procedure

Training the proposed PINN-MPC aims to determine the network parameters – namely, the weights and biases across all layers – that minimize the overall loss function defined in Eq. (17). This process requires careful consideration of both the temporal discretization scheme and the optimization strategy, as these directly influence the model's ability to learn physically consistent dynamics and control-relevant behaviors. The training procedure therefore consists of two main components: the design of suitable time discretization schemes for collocation and tracking points, and the implementation of a structured two-phase optimization strategy to achieve stable and constraint-aware learning.

The choice of time discretization plays a key role in shaping the network's learning behavior. The collocation points $\{t_j^{\text{col}}\}_{j=1}^{N_{\text{col}}}$ are sampled non-uniformly over the prediction horizon (0, T] with a higher density of points allocated near the beginning of the horizon to capture the rapid transients and dynamic shifts in system behavior. This is achieved by dividing the horizon into segments with different sampling resolutions: fine spacing for early times, medium spacing for mid-range, and coarse spacing toward the end. In contrast, the tracking points $\{t_j^{\text{trk}}\}_{j=0}^{N_{\text{trk}}}$, used in the control-oriented terms, are sampled uniformly at intervals of T_{s} to mimic the structure of traditional MPC dynamic state prediction.

The PINN-MPC parameters, including all network weights and biases, are optimized using a two-phase training procedure, with the Adam optimizer applied in both phases. Adam is a widely used first-order optimizer that combines the advantages of momentum and adaptive learning rates, enabling rapid exploration of the loss landscape [47].

The training data consist of episodes, each representing a distinct control scenario defined by conditioning vectors $(\boldsymbol{y}_{\text{sp}}, \boldsymbol{x}_{\text{IC}}, \boldsymbol{u}_{\text{IC}}, \boldsymbol{d}_{\text{IC}})$ that specify the set-point and initial conditions on states, inputs, and disturbances. An epoch corresponds to one complete training cycle over a batch of bs episodes, including forward and backward passes for all loss terms followed by parameter updates. The training proceeds for N_{eps} epochs per phase, progressively refining the network parameters toward convergence. The bs episodes used in each epoch are independently sampled, ensuring variability across training. Therefore, the total number of distinct training episodes amounts to $bs \times N_{\text{eps}}$.

In the first phase, the network is trained for N_{eps} epochs with a predefined batch size bs and learning rate lr_1, using only a subset of the loss terms: $\mathcal{L}_{\text{ode}}$, $\mathcal{L}_{\text{IC}}$, $\mathcal{L}_{y_{\text{trk}}}$, and $\mathcal{L}_{u_{\text{trk}}}$. The weights for the remaining loss terms - $w_{y_{\text{b}}}$, $w_{x_{\text{b}}}$, $w_{u_{\text{b}}}$, and $w_{\Delta u}$ - are temporarily set to zero, allowing the optimizer to freely explore the parameter space without constraint penalties. This initial stage enables the model to learn the underlying system dynamics and tracking behavior before constraints are imposed. In the second phase, the same training data and number of epochs are used, but a new Adam optimizer with learning rate lr_2 is initialized, and all loss terms are activated. This stage refines the previously learned solution to enforce output, state, and input constraints, while also promoting smooth control trajectories.

During each epoch, all loss components are computed for every episode in the batch, producing one scalar loss value per episode per component. These values are averaged across the bs episodes to obtain a single aggregate value for each loss component. The total loss for the epoch is then calculated as the weighted sum of these averaged components, as defined in (17).

All training-related hyperparameters are summarized in Table 1.

Remark 4. To facilitate the applied optimizer, i.e., Adam, in computing well-defined and stable gradients, we constructed the loss function to be continuously differentiable, as previously described. Moreover, to enforce perfect alignment with the differentiability properties that help the optimizer in exploring the available search space, we adopted the hyperbolic tangent as the activation functions of the PINN. The latter guarantees that the employed neural network model is a uniformly continuous model that produces smooth outputs

Remark 5. Unlike conventional PINNs that rely on experimental or pre-simulated data to match predicted outputs with observations, the proposed PINN-MPC framework replaces such data with the set-points for the controlled variables and the corresponding steady-state values for the manipulated variables. These reference values are enforced at the tracking time instants $\{t_j^{\text{trk}}\}_{j=0}^{N_{\text{trk}}}$, providing supervised signals that guide the network to align its predicted trajectories with the target control objectives. Consequently, the training process does not rely on any external dataset

3.4. PINN-MPC deployment

During training, the PINN-MPC learns to generate complete trajectories for both the state and control variables over the interval $[0, T]$. At runtime, however, the controller operates in a receding-horizon fashion

Table 1
Summary of the PINN-MPC training hyperparameters.

Loss weights	
w_{ode}	Penalizes mismatch with system ODEs
w_{IC}	Enforces initial conditions at start of prediction horizon
$w_{y_{\text{trk}}}$, $w_{u_{\text{trk}}}$	Penalize deviations from set-points and solutions
$w_{\Delta u}$	Penalizes large derivatives in control inputs
$w_{y_{\text{b}}}$, $w_{x_{\text{b}}}$, $w_{u_{\text{b}}}$	Penalize output, state and input bound violations
Training settings	
T	Prediction horizon
T_{s}	Controller sampling time
N_{col}	Number of collocation points
N_{trk}	Number of tracking points
bs	Number of episodes per batch in each phase
lr_1, lr_2	Learning rates in phase 1 and phase 2
N_{eps}	Number of training epochs in each phase

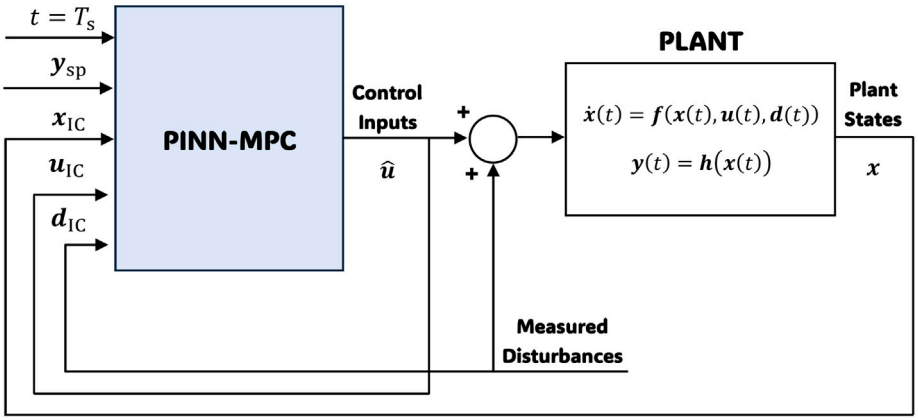

Fig. 2. PINN-MPC closed-loop implementation.

without requiring any online optimization. As illustrated in Fig. 2, it is queried once per control step, along with the current set-point vector $\boldsymbol{y}_{\text{sp}}$, the measured state vector $\boldsymbol{x}_{\text{IC}}$, the initial control input vector $\boldsymbol{u}_{\text{IC}}$, and, when applicable, the disturbance signal vector $\boldsymbol{d}_{\text{IC}}$. The controller returns only the control action vector $\hat{\boldsymbol{u}}$, which is applied to the plant and held constant over the sampling interval T_{s}. At the end of each interval, the system state is re-measured, and the procedure is repeated, thereby implementing the receding-horizon principle characteristic of MPC.

4. Results and discussion

4.1. Case study 1: Single water tank

4.1.1. Case study description

To evaluate the proposed PINN-MPC, we consider a nonlinear water tank system, commonly used in control benchmarks, as illustrated in Fig. 3. The tank receives inflow through a controllable source and a measured disturbance, modeled as an additive inflow perturbation. The evolution of the water level is described by the following first-order nonlinear ODE:

$$\dot{x}(t) = \frac{1}{A}\left(u(t) + d(t) - K\sqrt{x(t)}\right) = f(x(t), u(t), d(t)) \tag{19a}$$

$$y(t) = x(t) \tag{19b}$$

where:

- $x(t) = y(t) \geq 0$ is the water level in the tank (m),
- $u(t)$ is the inflow rate (m^3/s),
- $d(t)$ is the measured inflow disturbance (m^3/s),
- $A = 1.0$ is the tank's cross-sectional area (m^2),

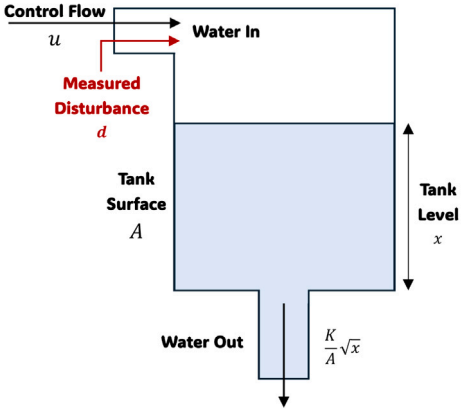

Fig. 3. Nonlinear water tank control problem.

- $K = 0.7$ is the outflow coefficient ($\text{m}^{2.5}/\text{s}$).

The outflow term $K\sqrt{x(t)}$ introduces nonlinear dynamics and depends on the instantaneous water level. The control input $u(t)$ is treated as a direct inflow and serves as the manipulated variable, constrained to the interval $[u_{\min}, u_{\max}] = [0, 1]$ m^3/s to reflect physical limitations of the inflow system. The controlled output, which coincides with the state, $y(t) = x(t)$, is constrained in the range $[y_{\min}, y_{\max}] = [0, 3]$ m. Additionally, the control increments are constrained by $|\Delta u| \leq \Delta u_{\max} = 0.2$ m^3/s, with a relaxation parameter of $\delta = 0.01$ used in (11) and (13). The measured disturbance $d(t)$ is assumed to remain constant over the

prediction horizon, with $d(t) = d_{IC}$, and may take values in the interval $[0, 0.4]$ m^3/s.

At steady state $\dot{x}(t) = 0$, which yields the following explicit steady-state relationship for the controlled state x_{ss}, the manipulated input u_{ss}, and the measured disturbance d_{IC}:

$$x_{ss} = y_{ss} = \left(\frac{u_{ss} + d_{IC}}{K}\right)^2 \tag{20}$$

During training, each episode is constructed by sampling the conditioning inputs $(y_{sp}, x_{IC}, u_{IC}, d_{IC})$ from uniform distributions, subject to feasibility constraints. First, u_{IC} and x_{IC} are drawn uniformly from the ranges $[0, 1]$ m^3/s and $[0, 3]$ m, respectively. Disturbances d_{IC} are set to zero with probability of 20%, and otherwise sampled uniformly from $[0, 0.4]$ m^3/s. The set-point y_{sp} is finally sampled uniformly within the range $\left[\left(\frac{d_{IC}}{K}\right)^2, \min(3, \left(\frac{1+d_{IC}}{K}\right)^2)\right]$ m, ensuring that the desired level is achievable given the disturbance and input and output limits.

4.1.2. Network configuration and hyperparameter selection

The PINN-MPC was developed as a fully connected feedforward neural network consisting of 3 hidden layers with 64, 16, and 16 neurons, respectively. Each hidden layer is followed by a *tanh* activation function to introduce nonlinearity. The output layer produces $\hat{x}(t)$ and $\hat{u}(t)$, which represent the predicted state and control trajectories. The network receives five external inputs: t, y_{sp}, x_{IC}, u_{IC}, and d_{IC}, as described earlier, and this structure comprises 1794 trainable parameters, including all weights and biases across layers, which are iteratively adjusted by the Adam optimizer during training to minimize the composite loss function presented in (17), from which the state-violation term $\mathcal{L}_{x_b}$ has been omitted because the system state coincides with the output variable in this case.

A prediction horizon of $T = 25$ s was chosen to capture both the transient and steady-state behavior of the system, whose open-loop response settles in approximately 17.5 s. This aligns with standard MPC design principles by covering the system's dominant dynamics while ensuring computational efficiency.

As far as time discretization is concerned, a total of $N_{col} = 928$ collocation points were generated over the 25-second prediction horizon using three step sizes: 0.01 s, 0.1 s, 1.0 s. Each step size was applied to one-third of the horizon, allocating a higher density of points at the beginning to better capture fast transients and the immediate effects of control actions, while coarser steps were used later to reduce computational cost. In contrast, $N_{trk} = 26$ tracking points were uniformly defined at intervals of $T_s = 1.0$ s, reflecting standard MPC practice in which deviations from the set-points are evaluated and control actions are updated at equidistant discrete time instants.

The remaining parameters, including the loss weights of the composite loss function in (17) and the Adam optimizer settings, were determined through a combination of manual screening and automated hyperparameter optimization using Optuna [48]. Training initially employed a lightweight configuration ($bs = 100$, $N_{eps} = 1500$ in each optimization phase). The learning rates of the two optimization phases were kept fixed, while manual tuning was used to identify admissible ranges for the loss weights. Optuna was then applied to refine and determine the final values of the loss weights. The evaluation criterion involved minimizing both the mean and maximum steady-state offsets across 3000 tracking and 3000 disturbance-rejection episodes. After fixing the loss weights, a second Optuna study was conducted using an extended configuration ($bs = 100$, $N_{eps} = 10000$ in each optimization phase) to tune the Adam learning rates under the same evaluation protocol (mean and maximum offsets on the same 3000 tracking and 3000 disturbance-rejection episodes). The final set of hyperparameters, summarized in Table 2, achieved an effective balance between computational efficiency and closed-loop performance.

4.1.3. Training and validation results

Fig. 4 shows the evolution of the overall loss across the two successive optimization phases, obtained using the optimal set of training hyperparameters and the extended training configuration ($bs = 100$, $N_{eps} = 10000$ in each phase). In the first phase (left panel), only the dynamics and tracking objectives are active, and the loss exhibits a rapid decline over 10000 epochs, demonstrating effective learning of the underlying system behavior. In the second phase (right panel), all constraint penalties are enabled, leading to an anticipated initial spike of the total loss, followed by convergence to a low level by the end of the training process. This two-stage behavior confirms that the model first internalizes the core system dynamics and tracking performance, and then refines its solution to satisfy the soft constraints without compromising overall accuracy.

Table 3 summarizes the final values of all loss components at the end of Phase 1 and Phase 2 of training. For each term defined in Eqs. (7)–(11) and Eqs. (16a), (16b), (16c) both the unweighted value (as defined in the corresponding equation, before weighting) and the weighted contribution to the total loss in Eq. (17) are reported. This presentation highlights the relative influence of the weighting factors listed in Table 2 on each component and provides insight into the relative contribution of the different loss terms after training.

The weighting strategy assigns a larger coefficient to $\mathcal{L}_{ode}$ relative to $\mathcal{L}_{IC}$ and the tracking terms $\mathcal{L}_{y_{trk}}$, $\mathcal{L}_{u_{trk}}$, thereby increasing its contribution to the total loss and ensuring that the system dynamics are strongly enforced. After Phase 1, during which the tracking behavior and dynamics are well captured, the PINN-MPC is able to satisfy the output and input constraints effectively in Phase 2 by navigating within the feasible space, as evidenced by the near-zero values of the corresponding loss terms. The smoothness penalty $\mathcal{L}_{\Delta u}$ decreases substantially from 2.05×10^1 at the beginning of Phase 2 to 1.66×10^{-2} by the end of training, demonstrating the controller's ability to conform to smoothness constraints. The primary terms $\mathcal{L}_{ode}$, $\mathcal{L}_{IC}$, $\mathcal{L}_{y_{trk}}$ and $\mathcal{L}_{u_{trk}}$ remain well balanced in scale, confirming that the introduction of soft constraints does not compromise the satisfaction of the core objectives.

To assess the controller's generalization capability, its performance was evaluated on two independent test suites that were not used during the training process. The first suite consists of 5000 episodes with set-points and initial conditions (y_{sp}, x_{IC}, u_{IC}) sampled uniformly across their admissible ranges, and with no disturbance ($d_{IC} = 0$). Each episode was simulated for 30 s which is sufficient time to reach steady state. The second suite evaluates disturbance rejection: in this case, 5000 episodes were generated with $(y_{sp}, x_{IC}, u_{IC}, d_{IC})$ sampled uniformly across their respective ranges. In each episode, a step disturbance was applied at $t = 0$ s, and the simulation was run for 30 s. For both test suites, the steady-state offset $\Delta y = y(t = 30\text{s}) - y_{sp}$ was recorded for each episode. The mean, minimum, and maximum offsets from the first 5000 disturbance-free episodes are reported in the left column of Table 4, while the corresponding results for the disturbance-rejection tests are shown in the right column.

The results demonstrate that the trained PINN-MPC achieves high set-point tracking performance in the absence of disturbances, with a very small mean steady-state offset and maximum deviation. When subjected to disturbances, the controller maintains strong performance, exhibiting only slightly higher maximum deviations while achieving an even smaller mean tracking offset.

Fig. 5 illustrates the closed-loop performance of the trained PINN-MPC under alternating reference and disturbance steps. In the top panel, the solid blue line represents the tank level $y(t)$, while the black dashed line indicates the desired set-point $y_{sp}(t)$. The middle panel shows the corresponding control action $u(t)$ in purple, and the bottom panel displays the disturbance profile $d(t)$ as a cyan dashed line. Across all randomly selected test scenarios, the controller drives the tank level smoothly and accurately to each new set-point, closely tracking the reference changes indicated in black. Following each disturbance step, it promptly adjusts the inflow to reject the load change and

Table 2
Training hyperparameters in the SISO case study.

Loss weights	Training settings
$w_{\text{ode}} = 131.20$	$T = 25\,\text{s}$
$w_{\text{IC}} = 2.34$	$T_s = 1.0\,\text{s}$
$w_{y_{\text{trk}}} = 6.28,\ w_{u_{\text{trk}}} = 6.73$	$N_{\text{col}} = 928,\ N_{\text{trk}} = 26$
$w_{\Delta u} = 32.52$	$bs = 100$
$w_{y_b} = 325.96$	$N_{\text{eps}} = 10000$
$w_{u_b} = 3243.43$	$lr_1 = 1.01 \times 10^{-3},\ lr_2 = 2.66 \times 10^{-4}$

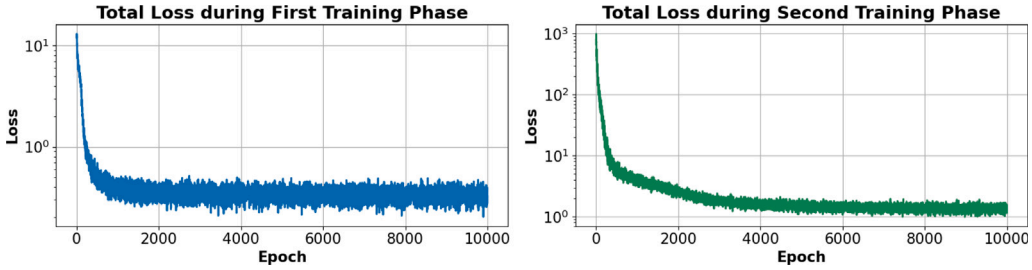

Fig. 4. Evolution of loss over epochs for both training phases in the SISO case study.

Table 3
Loss terms at the end of phase 1 and phase 2 in the SISO case study.

Eq.	Term	Phase 1		Phase 2	
		Raw (pre-weighted)	Weighted	Raw (pre-weighted)	Weighted
(7)	ODE Residuals ($\mathcal{L}_{\text{ode}}$)	2.27×10^{-5}	2.98×10^{-3}	3.86×10^{-4}	5.06×10^{-2}
(8)	Initial Conditions ($\mathcal{L}_{\text{IC}}$)	2.14×10^{-2}	5.01×10^{-2}	5.51×10^{-2}	1.29×10^{-1}
(9)	Output Tracking ($\mathcal{L}_{y_{\text{trk}}}$)	4.75×10^{-2}	2.98×10^{-1}	1.11×10^{-1}	6.98×10^{-1}
(10)	Input Tracking ($\mathcal{L}_{u_{\text{trk}}}$)	6.00×10^{-3}	4.04×10^{-2}	1.43×10^{-2}	9.61×10^{-2}
(11)	Smoothness ($\mathcal{L}_{\Delta u}$)	–	–	1.66×10^{-2}	5.39×10^{-1}
(16a)	Output Bounds ($\mathcal{L}_{y_b}$)	–	–	0.00	0.00
(16c)	Input Bounds ($\mathcal{L}_{u_b}$)	–	–	0.00	0.00
Total loss (weighted)		3.91×10^{-1}		1.51	

Table 4
Performance metrics for set-point tracking and disturbance rejection in the SISO case study.

Set-point tracking	Disturbance rejection
Mean Tracking Offset = 1.61×10^{-2} m	Mean Tracking Offset = 1.29×10^{-2} m
Max Tracking Offset = 3.22×10^{-2} m	Max Tracking Offset = 4.53×10^{-2} m
Min Tracking Offset = 0.00 m	Min Tracking Offset = 0.00 m

Table 5
Quantitative analysis of different training configurations in the SISO case study. Each configuration is defined by the learning rates (lr_1, lr_2) and the activation schedule of the constraint terms in the loss function across the two-phase optimization schedule. Configuration (1) corresponds to the proposed approach.

Conf.	Learning rate		Constraint terms active		Set-point tracking offset		Disturbance rejection offset	
	lr_1	lr_2	Phase 1	Phase 2	Mean (m)	Max (m)	Mean (m)	Max (m)
(1)	1.01×10^{-3}	2.66×10^{-4}	No	Yes	1.61×10^{-2}	3.22×10^{-2}	1.29×10^{-2}	4.53×10^{-2}
(2)	1.01×10^{-3}	2.66×10^{-4}	Yes	Yes	1.54×10^{-2}	5.69×10^{-2}	3.46×10^{-2}	8.14×10^{-2}
(3)	1.01×10^{-3}	1.01×10^{-3}	No	Yes	1.69×10^{-2}	3.43×10^{-2}	1.64×10^{-2}	7.43×10^{-2}
(4)	2.66×10^{-4}	2.66×10^{-4}	No	Yes	3.16×10^{-2}	5.65×10^{-2}	2.63×10^{-2}	5.59×10^{-2}

restore $y(t)$ to the desired set-point. The control input $u(t)$ consistently satisfies the imposed constraints, as it remains within the admissible range of $[0, 1]\ \text{m}^3/\text{s}$ throughout the simulation. Moreover, the changes between successive control actions stay within the maximum permitted increment of $0.2\ \text{m}^3/\text{s}$, demonstrating the effectiveness of the smoothness-promoting barrier function.

To validate the rationale behind the two-phase optimization approach, in which the constraint-related terms are activated only during the second phase, the training procedure was also executed with these terms activated during both phases (configuration (2)). The corresponding results are presented in Table 5. This configuration yields inferior performance metrics, particularly in disturbance rejection, where increases of 168.2% and 79.7% in the mean and maximum errors, respectively, are observed. In the disturbance-free case, although the mean error is lower by 4.9%, the maximum error increases by 76.7%. Furthermore, the two-phase training procedure with the constraint-related terms activated only during the second phase was additionally evaluated using identical learning rates for both phases (configurations

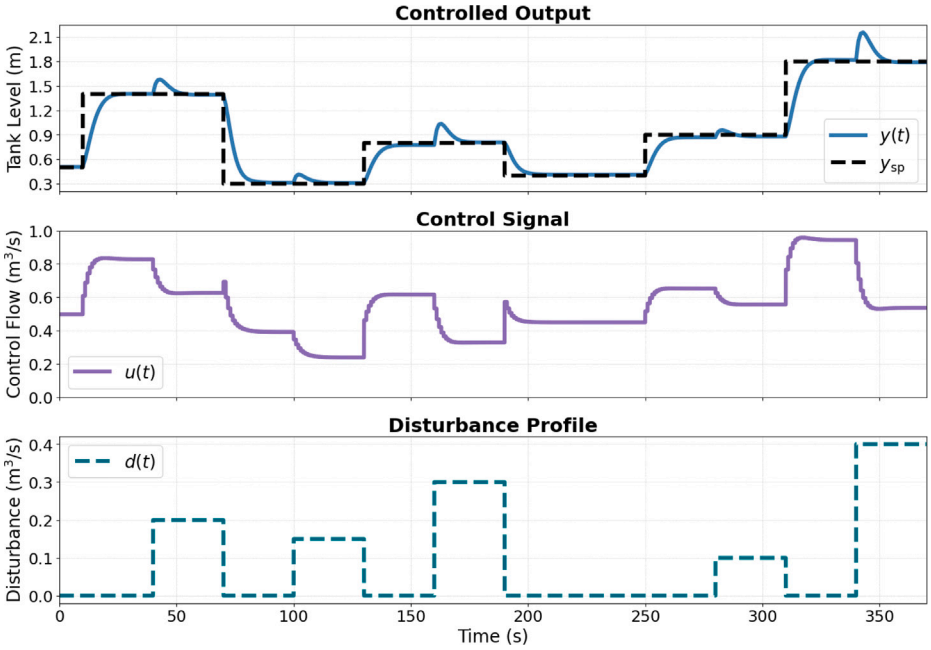

Fig. 5. Set-point tracking and disturbance rejection scenarios in the SISO case study.

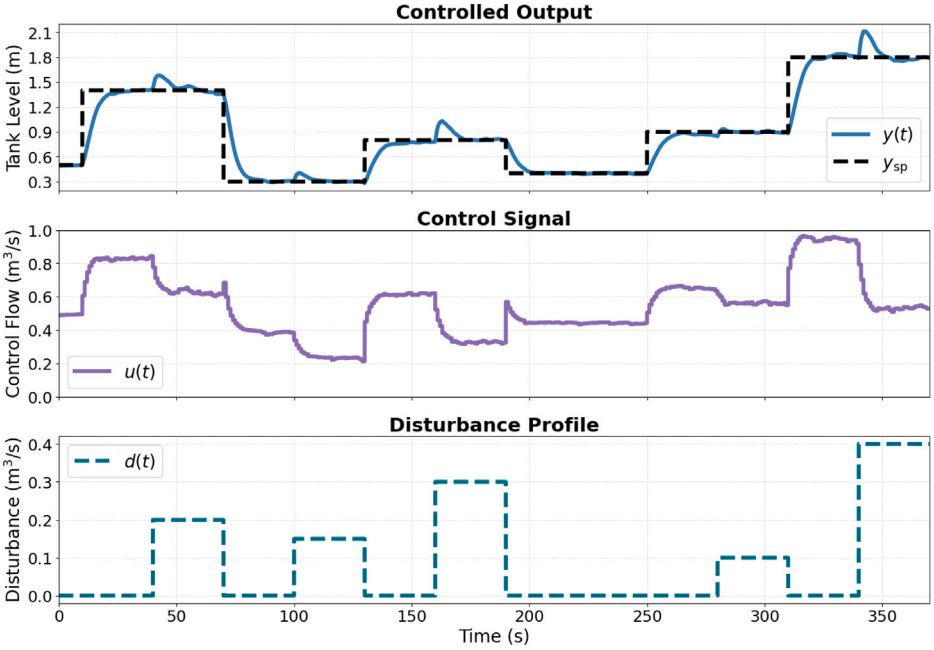

Fig. 6. Set-point tracking and disturbance rejection scenarios with measurement noise in the SISO case study.

(3) and (4)). The results in Table 5 confirm that the proposed configuration, which employs distinct learning rates between the two phases, achieves superior performance compared to these alternative training strategies.

To assess the performance of the trained PINN-MPC under sensing noise, we introduced zero-mean Gaussian noise with standard deviation $\sigma = 0.3$ m to the measured water level $y(t)$. The test is conducted on the same simulations presented in Fig. 5, without retraining the model. The results in Fig. 6 show that the controller remains stable and effective despite the noisy measurements. While minor oscillations appear, the system consistently reaches and maintains the target level, demonstrating strong resilience to significant sensor noise.

It is important to note that the current framework is limited to handling measured disturbances, which are explicitly supplied to the PINN-MPC through its disturbance input channel. Unmeasured disturbances as well as robustness to model mismatch have not been systematically examined. Nevertheless, the controller exhibited tolerance to a specific form of model mismatch – variations in the tank cross-sectional area A – successfully reaching the desired set-point, though with slower transient responses for larger values of A, as shown in Fig. 7. Extending the framework to account for unmeasured disturbances and to improve robustness against model uncertainty remains an important direction for future work.

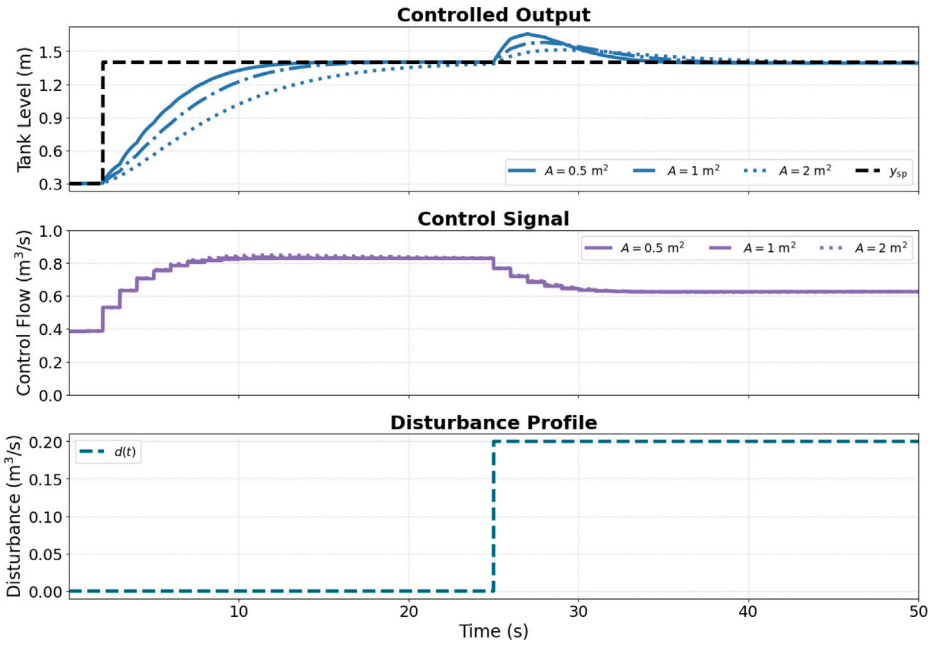

Fig. 7. Closed-loop performance of the PINN-MPC in the SISO case study under model mismatch. The controller was trained with $A = 1.0$ m^2 and evaluated for different tank cross-sectional areas $A = 0.5, 1.0, 2.0$ m^2.

4.2. Case study 2: Quadruple water tank

4.2.1. Case study description

To further evaluate the scalability and generalization capability of the proposed PINN-MPC framework, we extend the analysis to the nonlinear quadruple-tank system, a classical multivariable benchmark for control design [49,50], shown in Fig. 8. The process consists of two pairs of interconnected tanks arranged in lower and upper levels.

Each lower tank receives inflow through a controllable valve, while a fixed portion of this inflow is diverted to the corresponding upper tank according to split ratios γ_1 and γ_2. Similar to the single-tank case, measured disturbances $d_1(t)$ and $d_2(t)$ are modeled as additive inflow perturbations acting on the u_1 and u_2 paths, respectively. The upper tanks drain into the lower ones through gravity, introducing nonlinear coupling and cross-interactions among all four states. The time evolution of the water levels is described by the nonlinear dynamic system below:

$$\dot{x}_1(t) = \frac{1}{A_1}\left(\gamma_1\left(u_1(t)+d_1(t)\right) + K_3\sqrt{x_3(t)} - K_1\sqrt{x_1(t)}\right), \tag{21a}$$

$$\dot{x}_2(t) = \frac{1}{A_2}\left(\gamma_2\left(u_2(t)+d_2(t)\right) + K_4\sqrt{x_4(t)} - K_2\sqrt{x_2(t)}\right), \tag{21b}$$

$$\dot{x}_3(t) = \frac{1}{A_3}\left((1-\gamma_2)\left(u_2(t)+d_2(t)\right) - K_3\sqrt{x_3(t)}\right), \tag{21c}$$

$$\dot{x}_4(t) = \frac{1}{A_4}\left((1-\gamma_1)\left(u_1(t)+d_1(t)\right) - K_4\sqrt{x_4(t)}\right), \tag{21d}$$

$$y_1(t) = x_1(t), \qquad y_2(t) = x_2(t). \tag{21e}$$

where:

- $x_1(t), x_2(t), x_3(t), x_4(t) \geq 0$ are the levels in each tank (m),
- $u_1(t), u_2(t)$ are the valve commands (m^3/s),
- $d_1(t), d_2(t)$ are measured inflow disturbances added on the u_1 and u_2 paths, respectively (m^3/s),
- $\gamma_1, \gamma_2 \in [0, 1]$ are split ratios (here: $\gamma_1 = 0.6, \gamma_2 = 0.5$) to the lower tanks; $1-\gamma_1$, $1-\gamma_2$ go to the upper tanks.
- $A_i = 1.0$ are cross-sectional areas (m^2) and $K_i = 0.7$ are outlet coefficients (m$^{2.5}$/s) for each tank with $i = 1, 2, 3, 4$.

The control objective is to regulate the outputs $\boldsymbol{y}(t) = [x_1(t), x_2(t)]^\top$ to set-points $\boldsymbol{y}_{\text{sp}} = [y_{1,\text{sp}}, y_{2,\text{sp}}]^\top$ with $u_1(t), u_2(t)$ both in the range $[u_{\min}, u_{\max}] = [0, 1]$ m^3/s, while maintaining the upper levels x_3, x_4 below their user-defined upper bounds: $x_{3,\max} = 0.6$ m, $x_{4,\max} = 0.4$ m. The incremental changes in each control input (u_1 and u_2) are constrained at each tracking time instant so that their absolute values remain below 0.2 m^3/s, using Eqs. (11) and (13) with a relaxation parameter of $\delta = 0.01$. The measured disturbances $\boldsymbol{d}(t) = [d_1(t), d_2(t)]^\top$ are assumed to remain constant over the prediction horizon, with $\boldsymbol{d}(t) = \boldsymbol{d}_{\text{IC}}$, and may take values in the interval $[0, 0.4]$ m^3/s.

The system's steady-state is determined by setting $\dot{x}_i(t) = 0$. Solving the system yields the following explicit steady-state relationships for the controlled states $(x_{1,ss}, x_{2,ss})$, the manipulated inputs $(u_{1,ss}, u_{2,ss})$ and the measured disturbances $(d_{1,\text{IC}}, d_{2,\text{IC}})$:

$$x_{1,ss} = \left(\frac{\gamma_1\left(u_{1,ss}+d_{1,\text{IC}}\right) + (1-\gamma_2)\left(u_{2,ss}+d_{2,\text{IC}}\right)}{K_1}\right)^2 \tag{22a}$$

$$x_{2,ss} = \left(\frac{\gamma_2\left(u_{2,ss}+d_{2,\text{IC}}\right) + (1-\gamma_1)\left(u_{1,ss}+d_{1,\text{IC}}\right)}{K_2}\right)^2 \tag{22b}$$

and, assuming $\Delta := \gamma_1 + \gamma_2 - 1 \neq 0$:

$$u_{1,ss} = \frac{\gamma_2 K_1\sqrt{x_{1,ss}} - (1-\gamma_2)K_2\sqrt{x_{2,ss}}}{\Delta} - d_{1,\text{IC}}, \tag{23a}$$

$$u_{2,ss} = \frac{-(1-\gamma_1)K_1\sqrt{x_{1,ss}} + \gamma_1 K_2\sqrt{x_{2,ss}}}{\Delta} - d_{2,\text{IC}}. \tag{23b}$$

Substituting $x_{1,ss} = y_{1,\text{sp}}$ and $x_{2,ss} = y_{2,\text{sp}}$ into (23a) and (23b) yields the closed-form steady-state input references $(u^*_{1,\text{ss}}, u^*_{2,\text{ss}})$ used in loss term (10).

During training, each episode is constructed by sampling the conditioning inputs $(\boldsymbol{y}_{\text{sp}}, \boldsymbol{x}_{\text{IC}}, \boldsymbol{u}_{\text{IC}}, \boldsymbol{d}_{\text{IC}})$ from uniform distributions, subject to feasibility constraints. First, $\boldsymbol{u}_{\text{IC}} = [u_{1,\text{IC}}, u_{2,\text{IC}}]^\top$ are drawn uniformly from the range $[0, 1]$ m^3/s. Next, the disturbances $\boldsymbol{d}_{\text{IC}} = [d_{1,\text{IC}}, d_{2,\text{IC}}]^\top$ are set to zero with probability 20%, and otherwise sampled uniformly from $[0, 0.4]$ m^3/s. The initial states $\boldsymbol{x}_{\text{IC}} = [x_{1,\text{IC}}, x_{2,\text{IC}}, x_{3,\text{IC}}, x_{4,\text{IC}}]^\top$ are drawn uniformly within the disturbance-dependent feasible ranges defined by Eqs. (24a)–(24d). Finally, the set-points $\boldsymbol{y}_{\text{sp}} = [y_{1,\text{sp}}, y_{2,\text{sp}}]^\top$ are constructed to correspond to steady-state conditions. This is achieved by sampling $u_{1,\text{ss}}, u_{2,\text{ss}}$ from the range $[0, 1]$ m^3/s uniformly, and solving Eqs. (22a) and (22b) for $x_{1,\text{ss}}$, $x_{2,\text{ss}}$, given the already selected $d_{1,\text{IC}}$, $d_{2,\text{IC}}$ values. This sampling procedure ensures that all training episodes

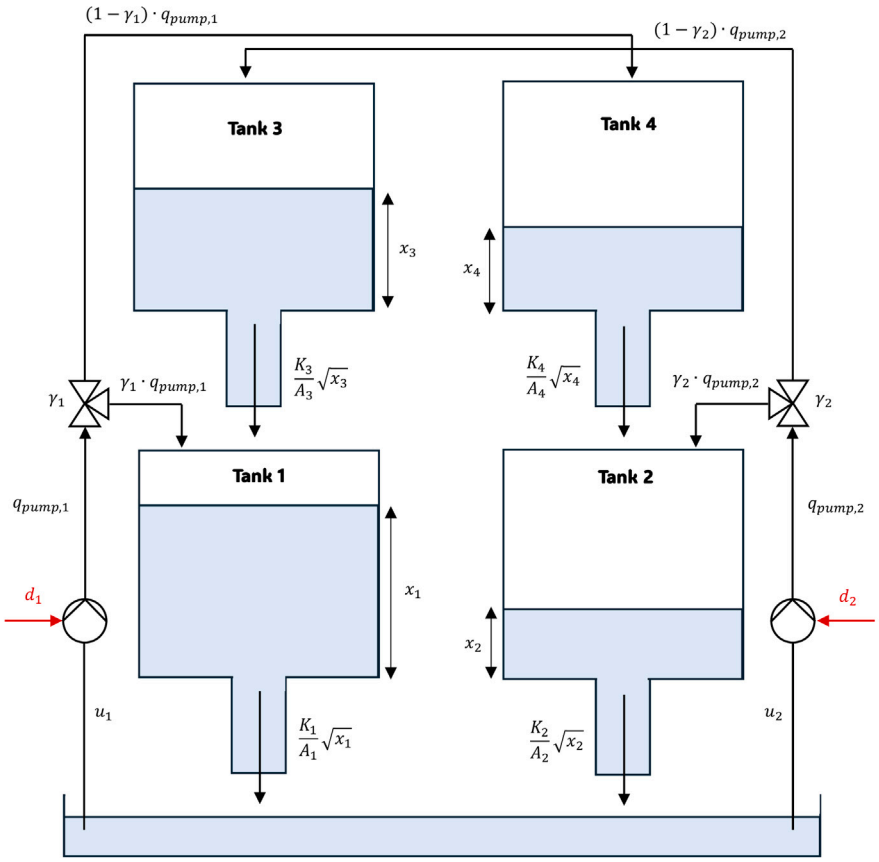

Fig. 8. Quadruple nonlinear water tank control problem.

correspond to physically attainable operating scenarios that comply with the disturbance, actuator, and output limits.

$$x_1 \in \left[\left(\frac{\gamma_1 d_{1,\mathrm{IC}} + (1-\gamma_2)d_{2,\mathrm{IC}}}{K_1}\right)^2, \left(\frac{\gamma_1 + (1-\gamma_2)}{K_1}\right)^2\right], \tag{24a}$$

$$x_2 \in \left[\left(\frac{\gamma_2 d_{2,\mathrm{IC}} + (1-\gamma_1)d_{1,\mathrm{IC}}}{K_2}\right)^2, \left(\frac{\gamma_2 + (1-\gamma_1)}{K_2}\right)^2\right], \tag{24b}$$

$$x_3 \in \left[\left(\frac{(1-\gamma_2)d_{2,\mathrm{IC}}}{K_3}\right)^2, \left(\frac{(1-\gamma_2)}{K_3}\right)^2\right], \tag{24c}$$

$$x_4 \in \left[\left(\frac{(1-\gamma_1)d_{1,\mathrm{IC}}}{K_4}\right)^2, \left(\frac{(1-\gamma_1)}{K_4}\right)^2\right]. \tag{24d}$$

4.2.2. Network configuration and hyperparameter selection

The PINN-MPC architecture for the quadruple-tank system follows the same fully connected feedforward design, comprising 3 hidden layers with 64, 16, and 16 neurons and *tanh* activations. As illustrated in Fig. 1, the same network architecture can be used for both the SISO and MIMO implementations, with vectorized inputs and outputs to accommodate systems of different dimensionality. In the current MIMO configuration, the input vector includes the time variable t, the set-points $\boldsymbol{y}_{\mathrm{sp}} \in \mathbb{R}^2$, the initial conditions $\boldsymbol{x}_{\mathrm{IC}} \in \mathbb{R}^4$ and $\boldsymbol{u}_{\mathrm{IC}} \in \mathbb{R}^2$, and the measured disturbances $\boldsymbol{d}_{\mathrm{IC}} \in \mathbb{R}^2$, resulting in a total of 11 input variables. The outputs correspond to the predicted state and control trajectories, $\hat{\boldsymbol{x}}(t) = [\hat{x}_1(t), \hat{x}_2(t), \hat{x}_3(t), \hat{x}_4(t)]^\top$ and $\hat{\boldsymbol{u}}(t) = [\hat{u}_1(t), \hat{u}_2(t)]^\top$, representing all four tank levels and the two valve actuators. The resulting network contains 2310 trainable parameters and is optimized using the same two-phase Adam-based training procedure described previously.

To further improve physical consistency, the loss function (17) was augmented with an additional term that incorporates equilibrium-based tracking for the upper tanks. Given the set-points $(y_{1,\mathrm{sp}}, y_{2,\mathrm{sp}})$, the corresponding steady-state levels $(x^*_{3,\mathrm{ss}}, x^*_{4,\mathrm{ss}})$ were computed from the system equilibrium. The additional term, denoted as $\mathcal{L}_{x_{\mathrm{trk}}}$, penalizes deviations of $\hat{x}_3(t)$ and $\hat{x}_4(t)$ from these equilibrium values. The same tracking weight $w_{y_{\mathrm{trk}}}$ was applied uniformly across all four tracking terms to ensure balanced learning between controlled and non-controlled states. In the MIMO configuration, the output-violation term $\mathcal{L}_{y_{\mathrm{b}}}$ has been omitted from (17) because bounds are enforced only on the non-controlled states x_3 and x_4 in this case.

The time discretization scheme mirrors that of the SISO configuration, employing three step sizes (0.01 s, 0.1 s, and 1.0 s) distributed non-uniformly across the prediction horizon, resulting in $N_{\mathrm{col}} = 1110$. The prediction horizon was extended to $T = 30$ s to account for the slower coupled dynamics of the multivariable system, whose open-loop response settles in approximately 20.6 s. The tracking interval was kept at $T_{\mathrm{s}} = 1.0$ s, yielding $N_{\mathrm{trk}} = 31$. To demonstrate the scalability of our approach, the PINN-MPC in the MIMO study was tuned using the same hyperparameter selection strategy as in the SISO case study. The final set of hyperparameters is summarized in Table 6.

4.2.3. Training and validation results

The evolution of the total loss over epochs for the MIMO PINN-MPC is shown in Fig. 9. The two-phase training exhibits a pattern similar to that of the SISO case, characterized by a rapid decrease in loss during the first phase, followed by a second rapid decrease and subsequent stabilization in the second phase as the constraints become satisfied. The overall convergence remains stable, though with slightly larger fluctuations due to the higher dimensionality and stronger coupling among the four tank states. These variations reflect the increased

Table 6
Training Hyperparameters in the MIMO Case Study.

Loss weights	Training settings
$w_{\text{ode}} = 1069.00$	$T = 30\,\text{s}$
$w_{\text{IC}} = 2.44$	$T_s = 1.0\,\text{s}$
$w_{y_{\text{trk}}} = 434.92,\ w_{u_{\text{trk}}} = 887.24$	$N_{\text{col}} = 1110,\ N_{\text{trk}} = 31$
$w_{\Delta u} = 55.08$	$bs = 100$
$w_{x_{\text{b}}} = 21.55$	$N_{\text{eps}} = 10000$
$w_{u_{\text{b}}} = 330.47$	$lr_1 = 4.45 \times 10^{-3},\ lr_2 = 1.51 \times 10^{-3}$

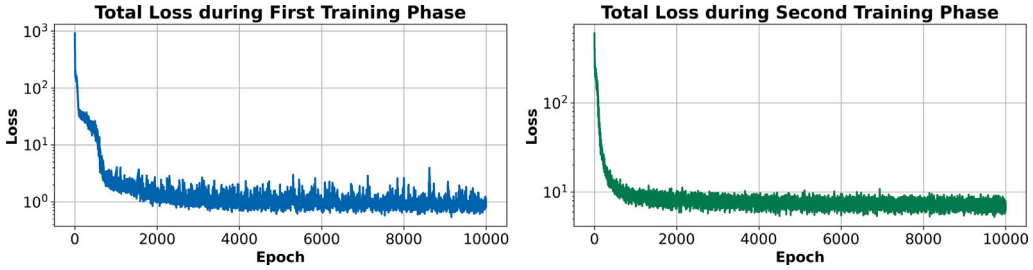

Fig. 9. Evolution of loss over epochs for both training phases in the MIMO case study.

Table 7
Loss terms at the end of phase 1 and phase 2 in the MIMO case study.

Eq.	Term	Phase 1		Phase 2	
		Raw (pre-weighted)	Weighted	Raw (pre-weighted)	Weighted
(7)	ODE Residuals ($\mathcal{L}_{\text{ode}}$)	8.56×10^{-5}	9.15×10^{-2}	2.65×10^{-4}	2.83×10^{-1}
(8)	Initial Conditions ($\mathcal{L}_{\text{IC}}$)	2.89×10^{-1}	7.06×10^{-1}	3.17×10^{-1}	7.73×10^{-1}
(9)	Output Tracking ($\mathcal{L}_{y_{\text{trk}}}$)	2.53×10^{-4}	1.10×10^{-1}	6.62×10^{-4}	2.88×10^{-1}
(10)	Input Tracking ($\mathcal{L}_{u_{\text{trk}}}$)	1.22×10^{-4}	1.08×10^{-1}	4.01×10^{-3}	3.56
(11)	Smoothness ($\mathcal{L}_{\Delta u}$)	–	–	4.19×10^{-2}	2.31
(16b)	State Bounds ($\mathcal{L}_{x_{\text{b}}}$)	–	–	2.42×10^{-6}	5.22×10^{-5}
(16c)	Output Bounds ($\mathcal{L}_{u_{\text{b}}}$)	–	–	0.00	0.00
Total loss (weighted)		1.02		7.22	

complexity of the multivariable dynamics but do not hinder the final convergence behavior.

Table 7 summarizes the final values of all loss components at the end of Phases 1 and 2, both in their weighted and unweighted forms. The highest weight is assigned to $\mathcal{L}_{\text{ode}}$, which ensures strict adherence to the system dynamics. At the end of Phase 2, the near-zero values of $\mathcal{L}_{x_{\text{b}}}$ and $\mathcal{L}_{u_{\text{b}}}$ confirm consistent constraint satisfaction across all trajectories. The smoothness penalty $\mathcal{L}_{\Delta u}$ exhibits a significant reduction, falling from 1.09×10^{1} at the start of Phase 2 to 4.19×10^{-2} by the end of training, effectively demonstrating the controller's adherence to smoothness constraints. Meanwhile, the principal losses - $\mathcal{L}_{\text{ode}}$, $\mathcal{L}_{\text{IC}}$, $\mathcal{L}_{y_{\text{trk}}}$, and $\mathcal{L}_{u_{\text{trk}}}$ - remain well balanced in magnitude, indicating stable multi-objective convergence across both training phases.

The generalization performance of the MIMO PINN-MPC was evaluated using two independent test suites following the same procedure as in the SISO study: one for set-point tracking and one for disturbance rejection, each comprising 5000 randomized episodes. Each episode was simulated for 30 s, and the steady-state offsets $\Delta \boldsymbol{y} = \boldsymbol{y}(t = 30\ \text{s}) - \boldsymbol{y}_{\text{sp}}$ were recorded for both controlled tanks. The mean, minimum, and maximum tracking errors for both cases are summarized in Table 8.

The results demonstrate that the PINN-MPC achieves accurate set-point tracking and disturbance rejection performance across both controlled outputs, exhibiting behavior similar to that observed in the SISO case. It maintains precise steady-state tracking with minimal offsets and smooth transient responses, even under varying disturbance conditions. Overall, the findings confirm that the proposed approach generalizes effectively to multivariable nonlinear systems, sustaining high tracking accuracy and disturbance rejection over a broad range of operating conditions.

Fig. 10 illustrates the closed-loop behavior of the MIMO PINN-MPC under simultaneous set-point and disturbance variations applied to both controlled tanks. The top panel shows the controlled outputs $y_1(t)$ (blue solid line) and $y_2(t)$ (red solid line) along with their respective set-points $y_{1,\text{sp}}(t)$ (black dashed line) and $y_{2,\text{sp}}(t)$ (gray dashed line). The second panel presents the upper-tank levels $x_3(t)$ (green solid line) and $x_4(t)$ (orange solid line) together with their maximum allowable limits (dotted lines), highlighting that the network maintains these internal states safely within their prescribed bounds throughout operation. The third panel displays the manipulated inputs $u_1(t)$ (purple solid line) and $u_2(t)$ (brown solid line), along with the upper constraint $u_{\max}$ (gray dotted line), while the bottom panel depicts the disturbance profiles $d_1(t)$ (yellow dashed line) and $d_2(t)$ (cyan dashed line). Across the full simulation, the controller achieves smooth, coordinated tracking of both reference signals while effectively compensating for coupling and disturbance effects. All states and inputs remain within their admissible bounds, demonstrating effective enforcement of the state and input constraints through the loss formulation.

Table 9 summarizes the performance of different training configurations for the MIMO PINN-MPC, using the same evaluation procedure as in the SISO case. Based on these results, the two-phase training schedule, which activates the constraint-violation terms only during the second phase (configuration (1)), once again yields the best closed-loop performance. In contrast, activating the constraint-related losses during both phases (configuration (2)) degrades performance, while activating the constraint terms only in the second phase but using identical learning rates across phases (configurations (3) and (4)) results in intermediate performance.

Table 8
Performance metrics for set-point tracking and disturbance rejection in the MIMO case study. Results are reported separately for the two controlled tanks.

Metric	Set-point tracking		Disturbance rejection	
	Tank 1	Tank 2	Tank 1	Tank 2
Mean Tracking Offset (m)	2.08×10^{-2}	1.83×10^{-2}	2.02×10^{-2}	1.30×10^{-2}
Max Tracking Offset (m)	4.85×10^{-2}	3.95×10^{-2}	6.70×10^{-2}	5.33×10^{-2}
Min Tracking Offset (m)	0.00	0.00	0.00	0.00

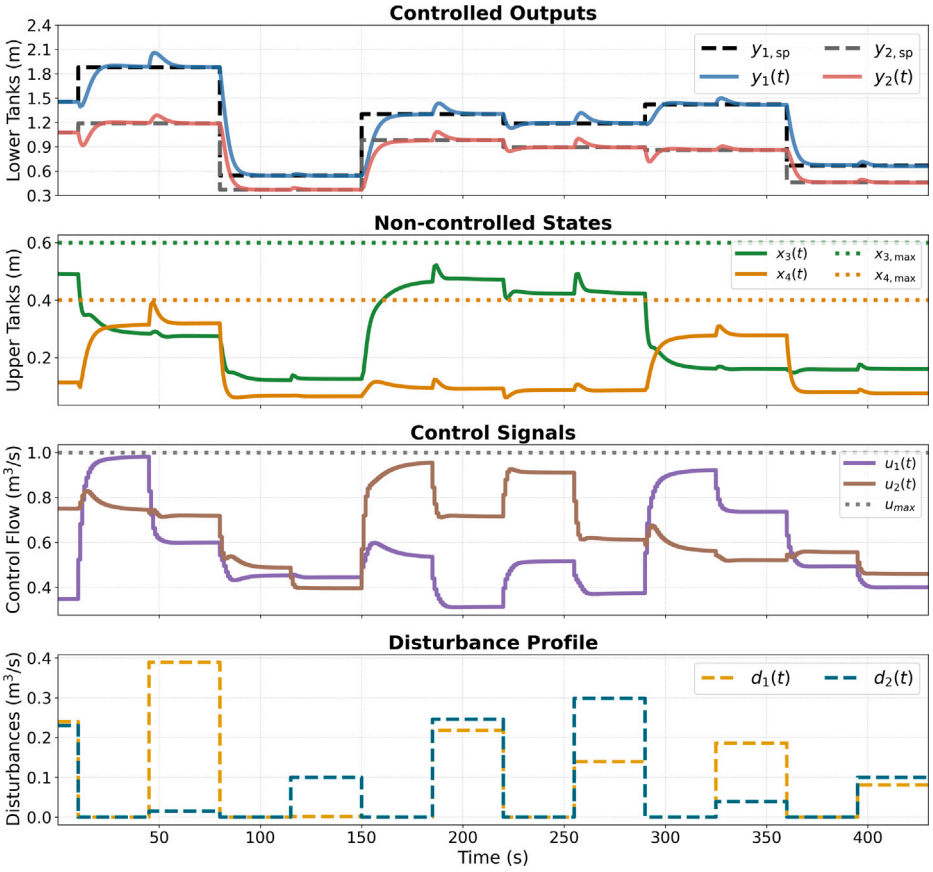

Fig. 10. Set-point tracking and disturbance rejection scenarios in the MIMO case study.

Table 9
Quantitative analysis of different training configurations in the MIMO case study. Each configuration is defined by the learning rates (lr_1, lr_2) and the activation schedule of the constraint terms in the loss function across the two-phase optimization schedule. Configuration (1) corresponds to the proposed approach.

Conf.	Learning rate		Constraint terms active		Set-point tracking offset (m)				Disturbance rejection offset (m)			
	lr_1	lr_2	Phase 1	Phase 2	Tank 1		Tank 2		Tank 1		Tank 2	
					Mean	Max	Mean	Max	Mean	Max	Mean	Max
(1)	4.45×10^{-3}	1.51×10^{-3}	No	Yes	2.08×10^{-2}	4.85×10^{-2}	1.83×10^{-2}	3.95×10^{-2}	2.02×10^{-2}	6.70×10^{-2}	1.30×10^{-2}	5.33×10^{-2}
(2)	4.45×10^{-3}	1.51×10^{-3}	Yes	Yes	6.73×10^{-2}	1.81×10^{-1}	4.34×10^{-2}	1.24×10^{-1}	3.89×10^{-2}	1.65×10^{-1}	2.36×10^{-2}	1.01×10^{-1}
(3)	4.45×10^{-3}	4.45×10^{-3}	No	Yes	1.85×10^{-2}	1.35×10^{-1}	1.43×10^{-2}	8.84×10^{-2}	2.47×10^{-2}	1.07×10^{-1}	1.92×10^{-2}	5.99×10^{-2}
(4)	1.51×10^{-3}	1.51×10^{-3}	No	Yes	4.24×10^{-2}	9.96×10^{-2}	3.41×10^{-2}	1.27×10^{-1}	3.38×10^{-2}	9.97×10^{-2}	2.55×10^{-2}	8.55×10^{-2}

4.3. Explicit PINN-MPC vs. Conventional NMPC runtime comparison

To further demonstrate the advantages of the proposed explicit PINN-MPC, we conducted a comparative study against a conventional NMPC implemented in CasADi [51] using IPOPT as the nonlinear solver. Both controllers share the same objective function and include a measured disturbance channel to ensure a fair comparison. The NMPC was tuned such that its control effort and tracking performance closely matched those of the PINN-MPC. The comparison experiments were performed on a 13th-Gen Intel® Core™ i5-13400 CPU.

Figs. 11 and 12 present representative closed-loop simulations for the SISO and MIMO systems, respectively. Each simulation spans a horizon of 50 s with a controller sampling period of $T_s = 1$ s, corresponding to 50 control updates per run. The solid lines denote the proposed PINN-MPC, while the dotted lines correspond to the standard NMPC configuration. In both systems, the trajectories nearly overlap, indicating that the two controllers achieve comparable control performance, constraint satisfaction, and control effort. These results establish a fair basis for comparing the runtime characteristics of the two methods under equivalent operating conditions.

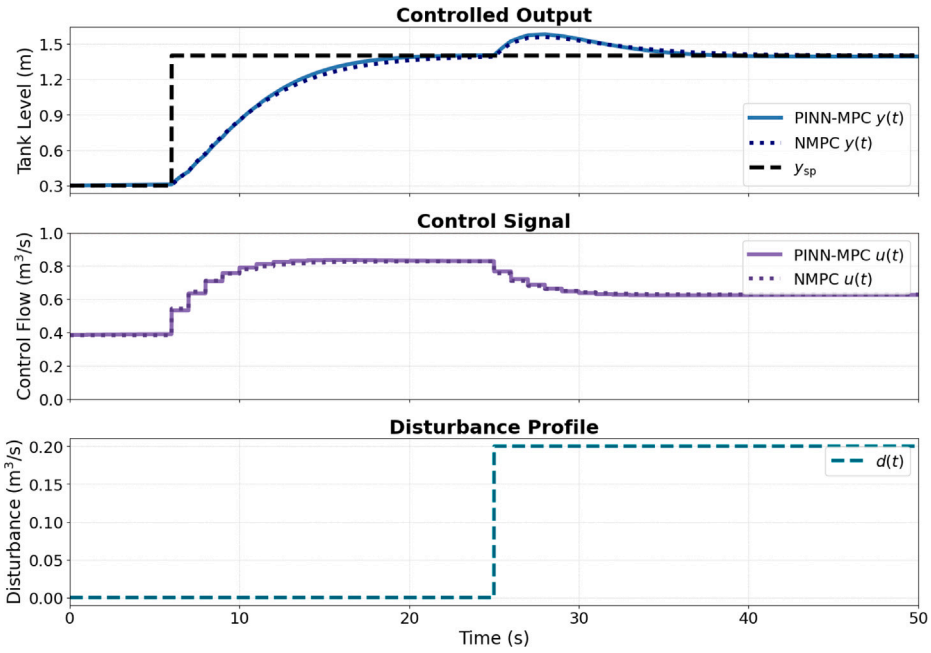

Fig. 11. PINN-MPC and NMPC responses in a closed-loop simulation involving set-point tracking and disturbance rejection in the SISO case study.

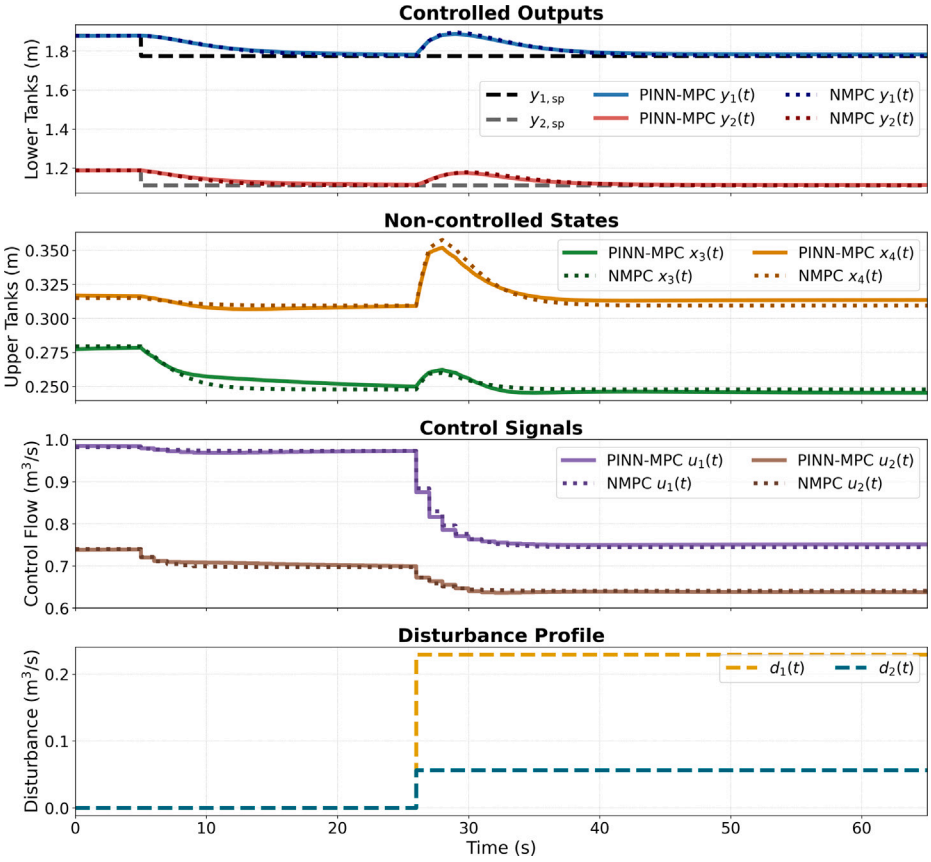

Fig. 12. PINN-MPC and NMPC responses in a closed-loop simulation involving set-point tracking and disturbance rejection in the MIMO case study.

Table 10 reports the computational performance evaluated over 1000 randomized episodes, each spanning a 50 s simulation horizon with varying set-points, initial conditions, and disturbances. As summarized in the table, the explicit PINN-MPC is substantially faster than NMPC — about 416× in the SISO case and roughly 2650× in the MIMO case. Moreover, the computational cost of NMPC grows by about 10.2× from SISO to MIMO, whereas PINN-MPC increases by only 1.61×, underscoring its superior scalability. These results confirm that the explicit PINN-MPC formulation achieves real-time inference with computation times several orders of magnitude lower than those of NMPC.

Table 10
Runtime statistics of PINN-MPC and NMPC over 1000 randomized episodes for both SISO and MIMO case studies.

Metric	SISO (Single water tank)		MIMO (Quadruple water tank)	
	PINN-MPC	NMPC	PINN-MPC	NMPC
Mean time per call (ms)	0.061 ± 0.009	25.390 ± 4.624	0.098 ± 0.003	259.517 ± 5.473
Max time per call (ms)	0.176 ± 0.110	54.608 ± 30.419	0.141 ± 0.027	591.663 ± 116.135
Total per episode (ms)	3.035 ± 0.430	1269.521 ± 231.224	4.887 ± 0.142	12975.829 ± 273.662

Notably, while NMPC runtime grows rapidly with system dimensionality, PINN-MPC remains nearly unaffected by problem size, highlighting its suitability for real-time deployment in nonlinear multivariable systems.

5. Conclusions

This paper presented a novel PINN-MPC framework that learns to synthesize optimal control trajectories by embedding the system dynamics and control objectives directly into the training process. In contrast to conventional MPC schemes that require computationally intensive online optimization at every control step, the proposed PINN-MPC framework performs all optimization offline during training. Once trained, a single feedforward neural network generates optimal state and control trajectories instantaneously, eliminating the need for online solvers and enabling fast deployment.

By augmenting the classical PINN structure with control-oriented loss terms, including set-point tracking, control smoothness, and soft constraint enforcement, the proposed method effectively bridges the gap between model learning and control synthesis. The network is trained using a two-phase optimization strategy: in the first phase, it learns to capture the system dynamics and tracking behavior, while in the second phase, the solution is refined to satisfy the imposed system constraints.

The proposed controller was first validated on a nonlinear single-tank water level system and subsequently extended to a more complex quadruple-tank configuration. The latter case study showcased that the same training framework and network architecture can scale effectively to higher-dimensional, coupled dynamics while preserving stability, tracking accuracy, and constraint satisfaction. In both systems, the PINN-MPC exhibited effective reference tracking and rejection of measured disturbances across thousands of randomized test scenarios. Performance remained robust even under measurement noise, with only minimal degradation observed.

A direct comparison with an online NMPC implemented in CasADi further highlighted the computational benefits of the explicit PINN-MPC formulation. While the NMPC's runtime increased substantially with system dimensionality, the PINN-MPC maintained sub-millisecond inference times and comparable control performance, confirming its scalability and suitability for implementation in nonlinear control applications.

The PINN-MPC framework is introduced as a promising new paradigm in optimal control design, with significant potential for further development and extension. Addressing robustness to modeling errors and performance under unmeasured disturbances constitute important directions for future research.

CRediT authorship contribution statement

Argyri Kardamaki: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Teo Protoulis:** Writing – original draft, Methodology, Investigation, Formal analysis. **Alex Alexandridis:** Writing – review & editing, Validation, Supervision, Investigation. **Haralambos Sarimveis:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Conceptualization.

Implementation and software environment

The PINN-MPC was implemented in Python `v3.11.1` [52] with PyTorch `v2.7.1` (CUDA `11.8` build) using single precision. All training experiments were conducted on a workstation equipped with an NVIDIA GeForce RTX 3060 Ti GPU and on AWS with NVIDIA A10G GPUs, ensuring consistent PyTorch–CUDA environments across platforms. For the implementation of the NMPC we used CasADi `v3.6.5` in combination with the IPOPT nonlinear programming solver [51]. The SISO and MIMO water-tank systems were simulated in PyTorch using an explicit Euler integrator with a time step of 0.01 s. All runtime comparison experiments between PINN-MPC and NMPC reported in Section 4.3 were performed on a 13th-Gen Intel® Core™ i5-13400 CPU.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the project "Network of Excellence for the Development, Dissemination and Application of Digital Transformation Technologies in the Greek Manufacturing Industry - GREECE4.0 (TAEDR-0535864)" within the framework of the National Recovery and Resilience Plan Greece 2.0, funded by the European Union – NextGenerationEU. AWS resources were provided by the National Infrastructures for Research and Technology GRNET and funded by the EU Recovery and Resiliency Facility.

Data availability

The Python code that generated the reported results is available on GitHub at: https://github.com/ntua-unit-of-control-and-informatics/pinn-mpc.

References

[1] S.J. Qin, T.A. Badgwell, A survey of industrial model predictive control technology, Control Eng. Pract. 11 (7) (2003) 733–764, http://dx.doi.org/10.1016/S0967-0661(02)00186-7.

[2] L. Li, Y. Lu, R. Wang, J. Chen, A three-dimensional dynamics control framework of vehicle lateral stability and rollover prevention via active braking with MPC, IEEE Trans. Ind. Electron. 64 (4) (2016) 3389–3401, http://dx.doi.org/10.1109/TIE.2016.2583400.

[3] M. Choi, S.B. Choi, Model predictive control for vehicle yaw stability with practical concerns, IEEE Trans. Veh. Technol. 63 (8) (2014) 3539–3548, http://dx.doi.org/10.1109/TVT.2014.2306733.

[4] S. Li, K. Li, R. Rajamani, J. Wang, Model predictive multi-objective vehicular adaptive cruise control, IEEE Trans. Control Syst. Technol. 19 (3) (2010) 556–566, http://dx.doi.org/10.1109/TCST.2010.2049203.

[5] S. Kouro, P. Cortés, R. Vargas, U. Ammann, J. Rodríguez, Model predictive control—A simple and powerful method to control power converters, IEEE Trans. Ind. Electron. 56 (6) (2008) 1826–1838, http://dx.doi.org/10.1109/TIE.2008.2008349.

[6] S. Vazquez, J.I. Leon, L.G. Franquelo, J. Rodriguez, H.A. Young, A. Marquez, P. Zanchetta, Model predictive control: A review of its applications in power electronics, IEEE Ind. Electron. Mag. 8 (1) (2014) 16–31, http://dx.doi.org/10.1109/MIE.2013.2290138.

[7] J. Richalet, A. Rault, J. Testud, J. Papon, Model predictive heuristic control: Applications to industrial processes, Autom. (Journal IFAC) 14 (5) (1978) 413–428, http://dx.doi.org/10.1016/0005-1098(78)90001-8.

[8] E.F. Camacho, C. Bordons, Introduction to model predictive control, in: Model Predictive Control, Springer, 2007, pp. 1–11, http://dx.doi.org/10.1007/978-0-85729-398-5_1.

[9] D. Mayne, Nonlinear model predictive control: Challenges and opportunities, in: Nonlinear Model Predictive Control, Springer, 2000, pp. 23–44, http://dx.doi.org/10.1007/978-3-0348-8407-5_2.

[10] A. Bemporad, Explicit model predictive control, in: Encyclopedia of Systems and Control, Springer, 2021, pp. 744–751, http://dx.doi.org/10.1007/978-3-030-44184-5_10.

[11] A. Bemporad, M. Morari, V. Dua, E.N. Pistikopoulos, The explicit linear quadratic regulator for constrained systems, Automatica 38 (1) (2002) 3–20, http://dx.doi.org/10.1016/S0005-1098(01)00174-1.

[12] E.N. Pistikopoulos, Perspectives in multiparametric programming and explicit model predictive control, AIChE J. (2009) http://dx.doi.org/10.1002/aic.11965.

[13] E.N. Pistikopoulos, N.A. Diangelakis, R. Oberdieck, Multi-Parametric Optimization and Control, John Wiley & Sons, 2020, http://dx.doi.org/10.1002/9781119265245.

[14] A. Alessio, A. Bemporad, A survey on explicit model predictive control, in: Nonlinear Model Predictive Control: Towards New Challenging Applications, Springer, 2009, pp. 345–369, http://dx.doi.org/10.1007/978-3-642-01094-1_29.

[15] I. Pappas, N.A. Diangelakis, E.N. Pistikopoulos, Multiparametric/explicit nonlinear model predictive control for quadratically constrained problems, J. Process Control 103 (2021) 55–66, http://dx.doi.org/10.1016/j.jprocont.2021.05.001.

[16] K. Wang, Z. Xu, K. Zhang, Y. Huang, J. Xu, Lattice piecewise affine approximation of explicit nonlinear model predictive control with application to trajectory tracking of mobile robot, IET Control Theory Appl. 18 (2) (2024) 149–159, http://dx.doi.org/10.1049/cth2.12553.

[17] S. Ganguly, D. Chatterjee, Explicit feedback synthesis for nonlinear robust model predictive control driven by quasi-interpolation, 2023, http://dx.doi.org/10.48550/arXiv.2306.03027, arXiv preprint arXiv:2306.03027.

[18] X. Li, M. Yan, X. Zhang, M. Han, A.W.-K. Law, X. Yin, Efficient data-driven predictive control of nonlinear systems: A review and perspectives, Digit. Chem. Eng. (2025) http://dx.doi.org/10.1016/j.dche.2025.100219.

[19] Y. Cao, R.B. Gopaluni, Deep neural network approximation of nonlinear model predictive control, IFAC-PapersOnLine 53 (2) (2020) 11319–11324, http://dx.doi.org/10.1016/j.ifacol.2020.12.538.

[20] W. Tang, P. Daoutidis, Data-driven control: Overview and perspectives, in: 2022 American Control Conference, ACC, IEEE, 2022, pp. 1048–1064, http://dx.doi.org/10.23919/ACC53348.2022.9867266.

[21] Y. Li, K. Hua, Y. Cao, Using stochastic programming to train neural network approximation of nonlinear MPC laws, Automatica 146 (2022) 110665, http://dx.doi.org/10.1016/j.automatica.2022.110665.

[22] Z. Wu, A. Tran, D. Rincon, P.D. Christofides, Machine learning-based predictive control of nonlinear processes. Part I: theory, AIChE J. 65 (11) (2019) e16729, http://dx.doi.org/10.1002/aic.16729.

[23] Z. Wu, A. Tran, D. Rincon, P.D. Christofides, Machine-learning-based predictive control of nonlinear processes. Part II: Computational implementation, AIChE J. 65 (11) (2019) e16734, http://dx.doi.org/10.1002/aic.16734.

[24] S. Yang, M.P. Wan, W. Chen, B.F. Ng, S. Dubey, Model predictive control with adaptive machine-learning-based model for building energy efficiency and comfort optimization, Appl. Energy 271 (2020) 115147, http://dx.doi.org/10.1016/j.apenergy.2020.115147.

[25] F. Mahmood, R. Govindan, A. Bermak, D. Yang, T. Al-Ansari, Data-driven robust model predictive control for greenhouse temperature control and energy utilisation assessment, Appl. Energy 343 (2023) 121190, http://dx.doi.org/10.1016/j.apenergy.2023.121190.

[26] M.S. Alhajeri, Y.M. Ren, F. Ou, F. Abdullah, P.D. Christofides, Model predictive control of nonlinear processes using transfer learning-based recurrent neural networks, Chem. Eng. Res. Des. 205 (2024) 1–12, http://dx.doi.org/10.1016/j.cherd.2024.03.019.

[27] Z. Huang, J. Liu, B. Huang, Model predictive control of agro-hydrological systems based on a two-layer neural network modeling framework, Internat. J. Adapt. Control Signal Process. 37 (6) (2023) 1536–1558, http://dx.doi.org/10.1002/acs.3586.

[28] S. Chen, K. Saulnier, N. Atanasov, D.D. Lee, V. Kumar, G.J. Pappas, M. Morari, Approximating explicit model predictive control using constrained neural networks, in: 2018 Annual American Control Conference, ACC, IEEE, 2018, pp. 1520–1527, http://dx.doi.org/10.23919/ACC.2018.8431275.

[29] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie, M. Klaučo, Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems, J. Process Control 116 (2022) 80–92, http://dx.doi.org/10.1016/j.jprocont.2022.06.001.

[30] P. Shah, S. Pahari, R. Bhavsar, J.S.-I. Kwon, Hybrid modeling of first-principles and machine learning: A step-by-step tutorial review for practical implementation, Comput. Chem. Eng. 194 (2025) 108926, http://dx.doi.org/10.1016/j.compchemeng.2024.108926.

[31] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations, 2017, http://dx.doi.org/10.48550/arXiv.1711.10561, arXiv preprint arXiv:1711.10561.

[32] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[33] K.T.C. Dae Yeong Lim, Development of PINN controller for fuel handling system of pressurized heavy water reactors, Stud. Inf. Control 29 (1) (2020) 25–34, http://dx.doi.org/10.24846/v29i1y202003.

[34] J. Liu, P. Borja, C. Della Santina, Physics-informed neural networks to model and control robots: A theoretical and experimental investigation, Adv. Intell. Syst. 6 (5) (2024) 2300385, http://dx.doi.org/10.1002/aisy.202300385.

[35] R.G. Hart, O.S. Patil, E.J. Griffis, W.E. Dixon, Deep Lyapunov-based physics-informed neural networks (DeLb-PINN) for adaptive control design, in: 2023 62nd IEEE Conference on Decision and Control, CDC, IEEE, 2023, pp. 1511–1516, http://dx.doi.org/10.1109/CDC49753.2023.10383962.

[36] E.A. Antonelo, E. Camponogara, L.O. Seman, J.P. Jordanou, E.R. de Souza, J.F. Hübner, Physics-informed neural nets for control of dynamical systems, Neurocomputing 579 (2024) 127419, http://dx.doi.org/10.1016/j.neucom.2024.127419.

[37] R. Patel, S. Bhartiya, R.D. Gudi, Neural network-based model predictive control framework incorporating first-principles knowledge for process systems, Ind. Eng. Chem. Res. 64 (18) (2025) 9287–9302, http://dx.doi.org/10.1021/acs.iecr.4c04305.

[38] J.B. Rawlings, D.Q. Mayne, M. Diehl, et al., Model Predictive Control: Theory, Computation, and Design, vol. 2, Nob Hill Publishing Madison, WI, 2017.

[39] D.Q. Mayne, J.B. Rawlings, C.V. Rao, P.O. Scokaert, Constrained model predictive control: Stability and optimality, Automatica 36 (6) (2000) 789–814, http://dx.doi.org/10.1016/S0005-1098(99)00214-9.

[40] E.C. Kerrigan, J.M. Maciejowski, Soft constraints and exact penalty functions in model predictive control, in: Control 2000 Conference, Cambridge, 2000, pp. 2319–2327.

[41] C. Feller, C. Ebenbauer, Weight recentered barrier functions and smooth polytopic terminal set formulations for linear model predictive control, in: 2015 American Control Conference, ACC, IEEE, 2015, pp. 1647–1652, http://dx.doi.org/10.1109/ACC.2015.7170969.

[42] C. Feller, C. Ebenbauer, Relaxed logarithmic barrier function based model predictive control of linear systems, IEEE Trans. Autom. Control 62 (3) (2016) 1223–1238, http://dx.doi.org/10.1109/TAC.2016.2582040.

[43] C. Feller, C. Ebenbauer, A stabilizing iteration scheme for model predictive control based on relaxed barrier functions, Automatica 80 (2017) 328–339, http://dx.doi.org/10.1016/j.automatica.2017.02.001.

[44] M. Zanon, S. Gros, M. Diehl, A tracking MPC formulation that is locally equivalent to economic MPC, J. Process Control 45 (2016) 30–42, http://dx.doi.org/10.1016/j.jprocont.2016.06.006.

[45] N. Lanzetti, Y.Z. Lian, A. Cortinovis, L. Dominguez, M. Mercangöz, C. Jones, Recurrent neural network based MPC for process industries, in: 2019 18th European Control Conference, ECC, 2019, pp. 1005–1010, http://dx.doi.org/10.23919/ECC.2019.8795809.

[46] K. Seel, E.I. Grøtli, S. Moe, J.T. Gravdahl, K.Y. Pettersen, Neural network-based model predictive control with input-to-state stability, in: 2021 American Control Conference, ACC, 2021, pp. 3556–3563, http://dx.doi.org/10.23919/ACC50511.2021.9483190.

[47] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, http://dx.doi.org/10.48550/arXiv.1412.6980, arXiv preprint arXiv:1412.6980.

[48] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019, http://dx.doi.org/10.48550/arXiv.1907.10902.

[49] L. Pedroso, P. Batista, Reproducible low-cost flexible quadruple-tank process experimental setup for control educators, practitioners, and researchers, J. Process Control 118 (2022) 82–94, http://dx.doi.org/10.1016/j.jprocont.2022.08.010.

[50] A. Pachare, A. Thosar, Level control of quadruple tank system with feedback linearization, in: Smart Sensors Measurements and Instrumentation: Select Proceedings of CISCON 2020, Springer, 2021, pp. 455–469, http://dx.doi.org/10.1007/978-981-16-0336-5_38.

[51] J.A.E. Andersson, J. Gillis, G. Horn, J.B. Rawlings, M. Diehl, CasADi – A software framework for nonlinear optimization and optimal control, Math. Program. Comput. 11 (1) (2019) 1–36, http://dx.doi.org/10.1007/s12532-018-0139-4.

[52] G. Van Rossum, F.L. Drake Jr., Python Tutorial, Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.