Physics-Informed Neural Nets for Control of Dynamical Systems

Eric Aislan Antonelo[a], Eduardo Camponogara[a], Laio Oriel Seman[a], Jean Panaioti Jordanou[a], Eduardo Rehbein de Souza[a], Jomi Fred Hübner[a]

[a]*Department of Automation and Systems, Federal University of Santa Catarina, Cx.P. 476, Florianopolis, 88040-900, Santa Catarina, Brazil*

Abstract

Physics-informed neural networks (PINNs) incorporate established physical principles into the training of deep neural networks, ensuring that they adhere to the underlying physics of the process while reducing the need for labeled data, since the desired output is not a prerequisite for physics-informed training. For modeling systems described by Ordinary Differential Equations (ODEs), traditional PINNs typically take continuous time as an input variable and produce the solution to the corresponding ODE. However, in their original form, PINNs neither accommodate control inputs nor do they effectively simulate variable long-range intervals without experiencing a significant decline in prediction accuracy. In this context, this work introduces a novel framework known as "Physics-Informed Neural Nets for Control" (PINC). PINC presents an innovative PINN-based architecture tailored to control problems, capable of simulating longer-range time horizons that are not predetermined during training. This increased flexibility sets it apart from traditional PINNs. The variable simulation time is achieved by adding inputs to the PINC network that convey the initial condition and the control signal for a particular time interval. Simulating variable long-range intervals involves running the PINC net across a sequence of shorter intervals. In this autoregressive process, the network predictions are linked in a self-feedback mode, with the initial state (input) of the next interval set to the last predicted state (network output) of the previous interval. We showcase the effectiveness of our proposal in identifying and controlling three nonlinear dynamic systems: the Van der Pol oscillator, the four-tank system, and an electric submersible pump. Crucially, these experiments demonstrate that learning the dynamics of these systems can be achieved without relying on any sample collected from the actual process, and it offers faster inference speed compared to numerical simulations.

Keywords: Deep learning, Ordinary differential equations, Nonlinear model predictive control, Learning with physics laws.

1. Introduction

In the era of Industry 4.0, simulation and control of complex real-world systems in smart and efficient ways are becoming increasingly important. Thus, harnessing deep learning for smart automation and control of real plants is desirable and inevitable. In this context, deep neural networks can be employed as models in Model Predictive Control (MPC) [1]. MPC is a technique that has become standard for multivariate control in industry and academia [2]. Since its inception in the 1970s, MPC has been successfully applied in the oil and gas [3], aerospace [4] and process industries, as well as in robotics [5]. The main idea of MPC is to control a system by employing a prediction model: at every iteration of the control loop, an optimization problem is solved using a model of the plant in a receding horizon approach, whereby only the control signals for the current time are implemented while the remaining signals serve for prediction and to prevent poor long-term performance.

We consider two prominent cases for which practical application of MPC or even just efficient simulation of a dynamic system are challenging: (a) historical data of the real plant is sparse or insufficient to build a sufficiently accurate machine learning model; (b) the numerical simulation of a precise model given by Ordinary Differential Equations (ODEs) or Partial Differential Equations (PDEs) is too costly to be considered in a real-time application. However, a recently introduced approach for training deep neural networks using laws of physics, namely Physics-Informed Neural Networks (PINN) [6], is a promising approach to address both of the aforementioned challenges. For the first challenge (a), we assume that a priori knowledge built previously by experts or borrowed from the laws of nature is available. For (b), instead of relying on numerical solutions of differential equations, PINNs can ease the computational burden of solving ODEs or PDEs and, consequently, extend the application of MPC to more real-time scenarios.

With PINNs, the need for labeled data is significantly reduced since their training is not influenced only by labeled samples, but also by unlabeled input points evaluated using physics laws given in terms of PDEs or ODEs. In this approach, the laws of physics constrain the output of the deep network. The available differential equations describe the dynamics of a system or industrial plant, which are included in the learning problem's cost function as nonlinear differential operators on the network's output $\mathbf{y}$. This way, through learning, the output $\mathbf{y}$ will approach the solution of these ODE or PDE equations. For PINNs, the total loss function comprises a data loss term based on labeled samples, usually the commonly used regression residual, and a physics-informed loss term based on unlabeled collocation points. The latter brings a regularization of the network, favoring scenarios where labeled data is scarce, for instance. Effectively, PINNs allow solving complex PDEs or ODEs using deep learning, although alternative approaches exist, which provides a symbolic solution to a bilinear PDE [7, 8], or a data-driven Fourier neural operator for simulating PDEs at different resolutions than the ones used in training [9]. Since the PINN's proposal [6], many extensions have been published. Zhu et al. [10] used PINNs for high-dimensional surrogate modeling and uncertainty quantification without labeled data; Sirignano and Spiliopoulos [11] presented a deep learning algorithm for solving PDEs; Meng et al. [12] proposed a parareal PINN for time-dependent PDEs; Yang et al. [13] employed a Bayesian PINN for solving PDEs with noisy data; Pang and Karniadakis [14] discussed the differences between Gaussian processes versus neural networks on physics-informed learning machines; Stinis [15] used PINNs to enforce constraints on time series prediction; Dwivedi and Srinivasan [16] proposed to extend Extreme Learning Machines as physics-informed networks for rapidly learning solutions to PDEs; Xiang et al. [17] proposed an improved PINN with a self-adaptive loss function to weight the loss terms relating to data and physics dynamically; Nazari et al. [18] introduced a PINN architecture for modeling water flows in a river channel; Olivares et al. [19] applied convolutional PINNs for WiFi signal propagation simulation.

A standard PINN has a continuous-time t as input, and the system's state variables $\mathbf{y}$ as output. To the best of our knowledge, there is no PINN architecture in continuous time that allows optimal control techniques such as Multiple Shooting (MS) [20] and Model-based Predictive Control (MPC) to be readily applied. Previous work used neural networks such as Echo State Networks and Long Short-Term Memory (LSTM) networks as models of the plant or process to be controlled [21]. However, these networks are trained exclusively on labeled data collected from the considered system and, thus, are not sample efficient to the extent PINNs are, as the latter can benefit from prior knowledge of the system's physics laws. In this sense, the challenge is to make PINNs compliant to control applications so that they can serve as models of a plant or process in MPC. In their original form, PINNs neither support control inputs nor do they effectively simulate variable long-range intervals without experiencing a significant degradation in their predictions.

With those limitations in mind, our work presents a new framework called *Physics-Informed Neural Nets for Control* (PINC), which proposes a novel PINN-based architecture that is amenable to control problems. The main features of this framework are:

(i) our PINN-based architecture, called hereafter PINC net, is augmented with extra inputs, such as the initial state of the system and control input, in addition to the continuous-time t. This augmentation draws inspiration from the multiple shooting and collocation methods [20], which are numerical methods for solving boundary value problems in ODEs, which find a solution by splitting the time horizon into several shorter intervals (shooting intervals). In our proposal, a single network learns the ODE solution conditioned on the initial state and the given control input over the shorter interval.

(ii) this innovation allows for enhancing the simulation capabilities of conventional PINNs, which can not correctly sustain a simulation beyond the time interval that was fixed during network training. This degradation of the network prediction is related to the maximum value allowed for t, which is fixed at training time. However, our proposed PINC network can simulate for variable longer-range time interval, without significant deterioration of network prediction. This long-range simulation is achieved by chaining the network prediction in a self-feedback mode by setting the initial state (input) of the next interval k to the last predicted state (network output) of the previous interval $k-1$.

(iii) the proposed structure of PINC makes physics-informed networks in continuous time amenable to MPC applications, which is the first work in the literature to tackle this as far as the authors know.

(iv) finally, the real-time requirements for simulating differential equations, in particular for MPC applications, are better satisfied with PINC than with traditional numerical simulation methods, since the inference of an already trained PINC network can replace a numerical solution method at each timestep of the prediction horizon in MPC.

In the sequel, Section 2 presents some related works. Section 3 introduces PINNs, MPC, and the proposed PINC framework. We present our proposal in the context of controlling three nonlinear dynamic systems, the Van der Pol oscillator, the four-tank system and an Electric Submersible Pump employed in oil wells, in Section 4. These experiments demonstrate that the PINC network can effectively learn to model nonlinear dynamical systems with multiple inputs and outputs. Furthermore, it does so faster than numerical methods, all without relying on any sample collected from the actual process. As we will clarify in the following sections, only knowledge of the underlying physics laws is required. Section 5 concludes this work.

2. Related Works

2.1. Neural networks and MPC

Neural networks have been used as models in MPC tasks or as controllers that operate dynamic systems. Previous works have trained neural networks to imitate MPC strategies using the usual mean squared error cost functions [22, 23]. In [24], the control law is represented by a neural network approximator, trained offline to minimize a control-related cost function directly, without the need to calculate a model predictive controller during training.

In the vein of Recurrent Neural Networks (RNN), works such as [21] and [25] utilized Echo State Networks as dynamical models for the MPC. Jordanou et al. [21] used a Trajectory linearization approach [26] by derivating the input-output sensitivities along the nonlinear free response over the prediction horizon to calculate a forced response [2]. In [25], the whole ESN is approximated into a state space system to compute the control action. The same reduction approach is proposed by Terzi et al. [27], however, employing LSTMs instead of ESN.

Another example is the classical Approximate Predictive Control [28], where a feedforward neural network implements dynamics through the application of delayed outputs as inputs (an external dynamics model [29]). It obtains an ARX (Auto Regressive with eXogenous input) model from the network through derivation, and performs GPC (Generalized Predictive Control) calculations per time step [2]. A neural network as the approximation to an MPC is considered in [30], in the same vein as Akesson and Toivonen in [24].

2.2. Long-range simulation with PINNs

In [12], *parareal* PINNs are proposed for long-time integration of time-dependent PDEs. They decompose a long-time problem into several short-time independent problems supervised by a fast coarse-grained solver, which provides approximate solution predictions at discrete times. Several smaller, fine PINNs are trained in parallel with the help of the supervision given by the fast solver. Each PINN solves the problem for a particular time interval independently. Notice that their approach does not include the possibility of control inputs and, thus, can not be readily used for control applications. On the other hand, while the initial goal of our proposal is to extend PINNs for control, we also benefit from being able to simulate a PINN for ODEs for a long time interval (we expect that our architecture can be extended for PDEs, too).

2.3. PINNs for control

Since the first appearance of this work [31], some architectures based on PINNs for control have been introduced. In [32], PINNs are supposedly used to control chaos in van der Pol oscillating circuits. However, the circuits employed in their paper have no control input, and neither does their method output a control signal to be applied. Their PINN architecture still has only time t as input and has an unusual loss function, which includes the loss for the data points and the reference to be followed. Surprisingly, we have not found any physics law in the network training, making their network an ordinary one. Besides, no control input can be derived from it to control a plant or dynamical system.

In [33], a model-based Reinforcement Learning (RL) algorithm for the first time, employed physical laws to learn the state transition dynamics of an agent's environment. The model corresponds to an encoder-decoder recurrent network architecture that learns the state transition function by minimizing the violation of conservation laws. The actual samples (state-action data pairs and corresponding rewards) from the environment are used to train the agent and the transition model simultaneously. In turn, the latter can generate samples in an alternative replay buffer that ultimately improves sample efficiency in the RL update and reduces real-world interaction. As the transition function is part of a Markov Decision Process (MDP) formulation, it represents a discrete evolution of the environment dynamics. For this reason, the physical loss function is built on the laws of the system in their discretized form instead of the continuous form as proposed in our work. While they require training a recurrent network, our work is based on feedforward networks as time appears explicitly in the input here.

In [34], the authors employed PINNs to learn a control-oriented thermal model of a building. As in [33], they assumed that the model is a discrete transition function in an MDP that predicts the next state, given the current state and action. In that way, control actions could be input into the model. Their physical loss also had to be discretized, unlike our work. Although their proposal is control-oriented, they did not show actual control experiments with the trained PINNs, as we do in Section 4.

3. Methods

3.1. Physics-informed Neural Networks (PINNs)

In [6], physics-informed neural networks are introduced, where deep neural networks are trained in a supervised way to respect any physical law described by partial differential equations (PDEs). The PINN approach automatically allows one to find data-driven solutions for PDEs or ODEs. In this paper, we consider nonlinear ODEs of the following general form:

$$\partial_t \mathbf{y} + \mathcal{N}[\mathbf{y}] = 0, \quad t \in [0, T] \tag{1}$$

where $\mathcal{N}[\cdot]$ is a nonlinear differential operator and $\mathbf{y}$ represents the state of the dynamic system (the latent ODE solution).

We define $\mathcal{F}(\mathbf{y})$ to be equivalent to the left-hand side of Equation (1):

$$\mathcal{F}(\mathbf{y}) := \partial_t \mathbf{y} + \mathcal{N}[\mathbf{y}] \tag{2}$$

Here, $\mathbf{y}$ also represents the output of a multilayer neural network (hence the notation $\mathbf{y}$ instead of $\mathbf{x}$) which has the continuous time t as input: $\mathbf{y} = f_{\mathbf{w}}(t)$, where $f_{\mathbf{w}}$ represents the mapping function obtained by a deep network parameterized by adaptive weights $\mathbf{w}$. This formulation implies that a neural network must learn to compute the solution of a given ODE.

Assuming an autonomous system for this formulation, a given neural network $\mathbf{y}(t)$ is trained using optimizers such as ADAM [35] or L-BFGS [36] to minimize a mean squared error (MSE) cost function:

$$\text{MSE} = \text{MSE}_y + \text{MSE}_{\mathcal{F}}, \tag{3}$$

where

$$\text{MSE}_y = \frac{1}{N_y} \sum_{i=1}^{N_y} \frac{1}{N_t} \sum_{j=1}^{N_t} |y_i(t^j) - \widehat{y}_i^j|^2, \tag{4a}$$

$$\text{MSE}_{\mathcal{F}} = \frac{1}{N_y} \sum_{i=1}^{N_y} \frac{1}{N_{\mathcal{F}}} \sum_{k=1}^{N_{\mathcal{F}}} |\mathcal{F}(y_i(t^k))|^2, \tag{4b}$$

where: N_t, $N_{\mathcal{F}}$, and N_y correspond to the number of training data samples, the number of collocation points, and the number of outputs of the neural network, respectively; $y_i(\cdot)$ is the i-th output of the network; $\widehat{y}_i^j$ represents the desired i-th output for $y_i(\cdot)$, considering the j-th data pair $(t^j, \widehat{y}_i^j)$. The first loss term MSE_y corresponds to the usual cost function for regression [37] based on collected training data $\{(t^j, \widehat{y}_i^j)\}_{j=1}^{N_t}$, which usually provides the boundary (initial or terminal) conditions of ODEs when solving these equations.

The second loss term $\text{MSE}_{\mathcal{F}}$ penalizes the misadjusted behavior of $\mathbf{y}(t)$, measured by $\mathcal{F}(\mathbf{y})$ in Equation (2), whereby $\mathcal{F}(\mathbf{y})$ imposes the physical structure of the solution at a finite set of randomly sampled collocation points $\{t^k\}_{k=1}^{N_{\mathcal{F}}}$. Experiments show that the training data size N_t required for learning a certain dynamical behavior is drastically reduced due to the a priori information assimilated from $\text{MSE}_{\mathcal{F}}$. Since $\mathcal{F}(\mathbf{y}) = 0$ represents the differential equation of the physical system, the term $\text{MSE}_{\mathcal{F}}$ is a measure of how well the PINN adheres to the solution of the physical model. This physics-informed approach provides a framework that unifies a previously available theoretical, possibly approximate model and measured data from processes, which can correct imprecisions in the theoretical model or provide sample efficiency in process modeling.

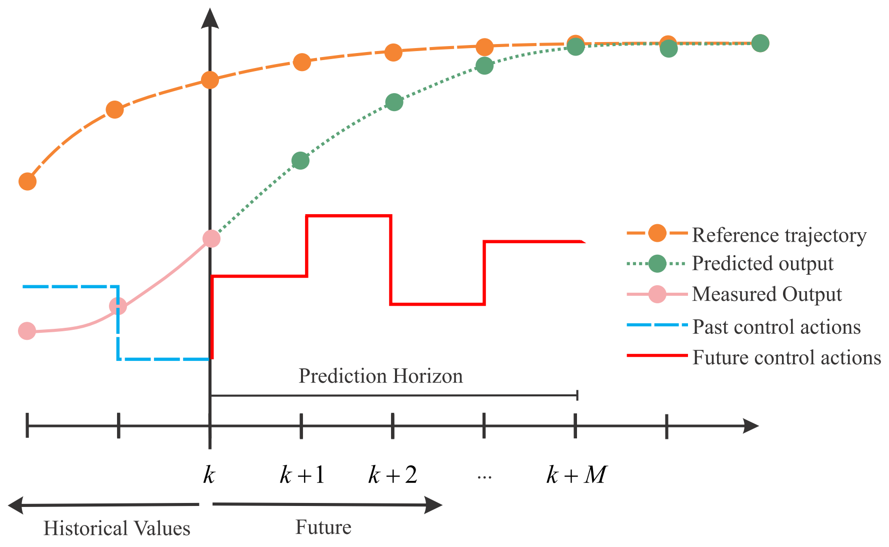


Figure 1: Representation of the output prediction at a time instant t_k, where the proposed actions generate a predicted behavior that reduces the distance between the value predicted by the model and a reference trajectory.

3.2. Nonlinear Model Predictive Control

Model Predictive Control (MPC) has evolved considerably over the last two decades, significantly impacting industrial process control. This impact can be attributed to its flexibility in formulating the process control problem in the time domain, suitable for SISO (Single-Input Single-Output) and MIMO (Multiple-Input Multiple-Output) systems. Soft and hard constraints can be imposed on the formulation of the control law through optimization problems while minimizing an objective function over a prediction horizon [38].

MPC is not a specific control strategy but rather a denomination of a vast set of control methods developed considering some standard ideas and predictions [38]. Figure 1 shows a representation of the output prediction at a time instant, where the proposed actions generate a predicted behavior that reduces the distance between the value predicted by the model and a reference trajectory.

The MPC strategy uses a discrete mathematical model based on the process of interest. A predicted output is calculated in a prediction horizon by comparing the mathematical model to the process's output. To propose control actions, the MPC strategy uses an iterative optimization process, considering the mathematical model of interest and the involved constraints to which it is subject. Based on objectives and constraints, the optimization problem comprises mathematical expressions established in the controller's design phase, taking many forms. Usually, quadratic functions penalize the error in the reference tracking.

According to Camacho and Bordons [2], there are several ways to classify these controllers, taking into account characteristics such as model linearity, treatment of uncertainties, and how the optimization problem is solved. This work focuses on the lack of model linearity, particularly in Nonlinear Model Predictive Control (NMPC) [1]. The discrete NMPC formulation is given by:

$$J = \sum_{j=N_1}^{N_2} \left\| \mathbf{x}[k+j] - \mathbf{x}^{\text{ref}}[k+j] \right\|_{\mathbf{Q}}^2 + \sum_{i=0}^{N_u-1} \|\Delta\mathbf{u}[k+i]\|_{\mathbf{R}}^2 \tag{5a}$$

while being subject to:

$$\mathbf{x}[k+j+1] = \mathbf{f}(\mathbf{x}[k+j], \mathbf{u}[k+j]), \quad \forall j = 0, \ldots, N_2 - 1 \tag{5b}$$

$$\mathbf{u}[k+j] = \mathbf{u}[k-1] + \sum_{i=0}^{j} \Delta\mathbf{u}[k+i], \quad \forall j = 0, \ldots, (N_u - 1) \tag{5c}$$

$$\mathbf{u}[k+j] = \mathbf{u}[k+N_u-1], \quad \forall j = N_u, \ldots, N_2 - 1 \tag{5d}$$

$$\mathbf{h}(\mathbf{x}[k+j], \mathbf{u}[k+j]) \leq 0, \quad \forall j = N_1, \ldots, N_2 \tag{5e}$$

$$\mathbf{g}(\mathbf{x}[k+j], \mathbf{u}[k+j]) = 0, \quad \forall j = N_1, \ldots, N_2 \tag{5f}$$

where k represents the time step at which the MPC problem is solved, $\mathbf{x}[k]$ is the recurrent state of the dynamic system, which, for simplification purposes, is also the output (i.e., $\mathbf{x} = \mathbf{y}$), $\mathbf{x}^{\text{ref}}$ is the set-point signal over the prediction horizon (i.e., reference), being defined by the first penalized instant $k + N_1$ and the last instant $k + N_2$. The cost function J is the penalization on the quadratic error between the model output $\mathbf{x}$ and the reference $\mathbf{x}^{\text{ref}}$ along the horizon, and the penalization on the control increment $\Delta\mathbf{u}$. The penalizations are weighted by the diagonal matrices $\mathbf{Q}$ and $\mathbf{R}$, respectively. Eqn. (5b) is the constraint imposed by the considered state-equation model with $\mathbf{x}$ as the state, and equations (5e) and (5f) refer to inequality and equality constraints imposed by functions $\mathbf{h}$ and $\mathbf{g}$, respectively. Eqn. (5c) defines the relation between the control action $\mathbf{u}$ and the control increments, which are aggregated into the control action from time k up until time $k + N_u - 1$ (end of the control horizon). Eqn. (5d) states that the control actions $\mathbf{u}$ from time $k + N_u$ until $k + N_2 - 1$ (end of the prediction horizon) are fixed at the last action of the control horizon.

The optimization problem is defined by equations from (5a) to (5f), resulting in a Nonlinear Programming (NLP) Problem, which can be solved using well-established methods like Sequential Quadratic Programming (SQP) [39] and the Interior-Point (IP) method, available in commercial [40] and non-commercial solvers [41]. The NLP is solved at each time step k, and typical approaches only apply the first control increment into the system [2].

3.3. Physics-Informed Neural Nets-based Control (PINC)

Unlike PINNs that assume fixed inputs and conditions, the proposed PINC framework operates with variable initial conditions and control inputs that can change over the complete simulation, making it suitable for model predictive control tasks. The network is augmented with two, possibly multidimensional inputs: control action $\mathbf{u}$ and initial state $\mathbf{y}(0)$, as illustrated in Figure 2. The output of the network is given by:

$$\mathbf{y}(t) = f_{\mathbf{w}}(t, \mathbf{y}(0), \mathbf{u}), \quad t \in [0, T] \tag{6}$$

where $f_{\mathbf{w}}$ represents the mapping given by a deep network parameterized by weights $\mathbf{w}$. This work assumes the control input as a constant value for the time interval $t \in [0, T]$. Thus,

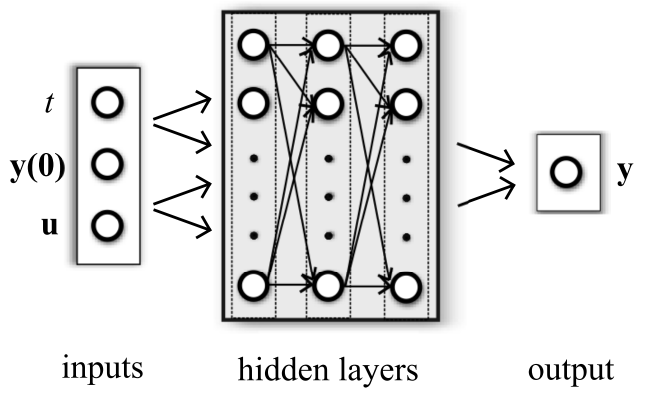


Figure 2: The PINC network has initial state $\mathbf{y}(0)$ of the dynamic system and control input $\mathbf{u}$ as inputs, in addition to continuous time scalar t. Both $\mathbf{y}(0)$ and $\mathbf{u}$ can be multidimensional. The output $\mathbf{y}(t)$ corresponds to the state of the dynamic system as a function of $t \in [0, T]$, and initial conditions given by $\mathbf{y}(0)$ and $\mathbf{u}$. The deep network is fully connected even though not all connections are shown, All hidden units have hyperbolic tangent activation function.

the new formulation provides a conditioned response $\mathbf{y}(t)$ on $\mathbf{u}$ and $\mathbf{y}(0)$ during this interval of T seconds.

Predictions by traditional PINNs tend to degrade rapidly for time intervals longer than the one used in training. Thus, PINNs behave as expected as long as the time input t is in the range the network was trained for (e.g., $t \in [0, T]$). The proposed PINC framework allows PINNs to extrapolate the simulation time for arbitrarily long periods. To simulate longer intervals, when $t > T$, the total simulation time is divided by T, splitting it into M intermediate time intervals of T seconds each (Fig. 3). Thus, the total simulation time corresponds to MT s. We call this shorter period of T seconds as the *inner continuous time interval* of the problem, in which a solution of an ODE is obtained given some initial condition $\mathbf{y}(0)$ (which models the current system state) and control input $\mathbf{u}$, which is kept fixed for $t \in [0, T]$. If the network was trained with only one initial condition and one control action as training inputs, it would only learn to simulate this single scenario. As the proposed PINC net will be trained with random multiple initial conditions and control actions (Section 3.3.3), it will learn to predict any state $\mathbf{y}(t)$ in the interval $t \in [0, T]$ for any combination of initial condition and control action applied in that interval. In other words, this network output $\mathbf{y}(t)$ represents the ODE solution conditioned on control action $\mathbf{u}$ and initial condition $\mathbf{y}(0)$. In the next section, the solution to the complete interval of MT s is built by chaining the predictions in an autoregressive way. This means that each of the M intermediate inner simulation intervals is solved by the same PINC net, i.e., a single network generates the predictions for each inner interval.

3.3.1. Autoregressive Simulation: Combining the Intermediate Solutions

As noted earlier, the complete interval comprises M intermediate continuous time intervals of T seconds each, as shown in Figure 3. By changing notation, we refer to $\mathbf{y}[k]$ as the states inferred by the network in discrete time k, seen at the top of the figure as black dots. Between two black dots, e.g., $\mathbf{y}[k]$ and $\mathbf{y}[k+1]$, the dashed trajectory connecting them represents the continuous output $\mathbf{y}(t)$[1] of the PINC net for $t \in [0, T]$. The corresponding inputs of the network during this interval also appear in the lower part of the figure. Both the initial condition and control action inputs are fixed in this inner interval, between steps k and $k+1$, and its intermediate solution is given by Equation (6). The time input t is the only variable input in this inner interval.

The autoregressive part in PINC is represented by the red arrow in the figure and takes place by setting the initial state of the next interval to the last predicted state of the previous interval. Since t is an input to the network, the state at $t = T$ can be directly inferred by a single forward network propagation:

$$\mathbf{y}[k] = f_{\mathbf{w}}(T, \mathbf{y}[k-1], \mathbf{u}[k]), \tag{7}$$

where the initial state at interval k is set to the last predicted state $\mathbf{y}(T)$ of the previous time interval $k-1$, i.e., $\mathbf{y}[k-1]$ (red arrow in Figure 3); and the control input $\mathbf{u}[k]$ indicates the action applied in the inner continuous time interval between steps $k-1$ and k. This control action can change from one step k to the next, enabling PINNs for control applications. The above equation can be run recursively until the M intervals are complete, which is equivalent to simulating the respective ODE for the total simulation time regardless of the fixed interval T used in training, and with variable control actions between discrete timesteps. It is worthwhile to notice that the autoregressive property of PINC is realized only in simulation mode, after training.

3.3.2. PINC Simulation in MPC

The previously presented recursive simulation can produce predictions for MPC applications in a predictive autoregressive way. In this self-loop mode, network predictions $\mathbf{y}$ are fed back as initial condition inputs $\mathbf{y}(0)$ at each discrete time step (Fig. 4a). As already described, this feedback enables the simulation of the complete time interval of MTs. In the context of MPC, M represents the prediction horizon. However, the initial state of the complete interval must be defined somehow. In MPC, this value comes from the measurements of the plant (Fig. 4b), which initiates the autoregressive simulation for a prediction horizon. Thus, every simulation starts with the PINC connected to the plant (Fig. 4b) and runs until the end of the prediction horizon in a self-loop mode (Fig. 4a).

Within one iteration of MPC, the PINC net is used for a specific prediction horizon without feedback from the process. This means that the network prediction $\mathbf{y}[k-1]$ and not the real state $\widehat{\mathbf{y}}[k-1]$ is fed back as input to the same network in the next timestep k of the prediction horizon (Fig. 4a). This is because the true state is unknown in the prediction horizon of MPC.

A sampling period T_s must be chosen in discrete time control applications. The setting of T_s usually depends on the particular dynamics of the process modeled. Here, T is equal to the sampling period T_s. In addition, using Equation (7), we can encapsulate the PINC

[1]We omit the discrete index k of the inner time interval from $\mathbf{y}(t)$ for the sake of simplicity.

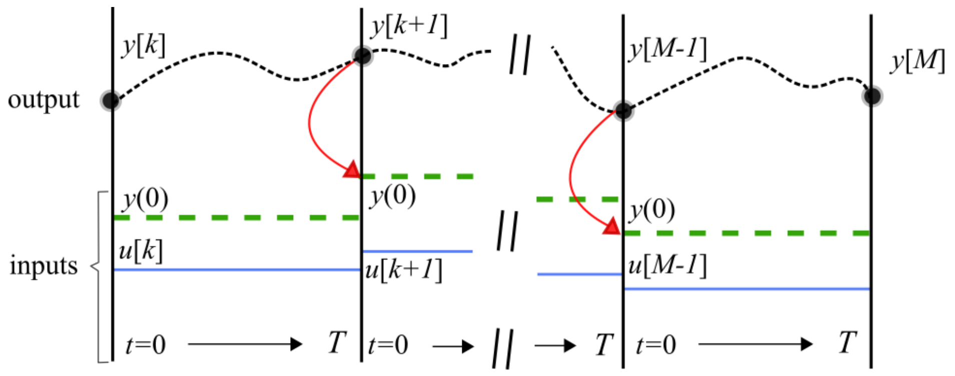


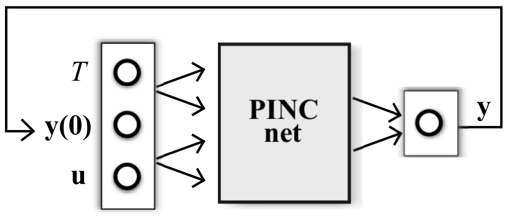
Figure 3: Moving horizon prediction of PINC network in self-loop mode (Fig. 4a). The top dashed black curve corresponds to a predicted trajectory $\mathbf{y}$ of a hypothetical dynamic system in continuous time. The states $\mathbf{y}[k]$ are snapshots of the system in discrete time k positioned at the equidistant vertical lines. Between two vertical lines (during the *inner continuous interval* between steps k and $k+1$), the PINC net learns the solution of an ODE with $t \in [0, T]$, conditioned on a fixed control input $\mathbf{u}[k]$ (blue solid line) and initial state $\mathbf{y}(0)$ (green thick dashed line). Control action $\mathbf{u}[k]$ is changed at the vertical lines and kept fixed for T seconds, and the initial state $\mathbf{y}(0)$ in the interval between steps k and $k+1$ is updated to the last state of the previous interval $k-1$ (indicated by the red curved arrow). The PINC net can directly predict the states at the vertical lines without inferring intermediate states $t < T$ as numerical simulation does. Here, we assume that $T = T_s$ and, thus, the number of discrete timesteps M is equal to the length of the prediction horizon in MPC.

prediction function so that it is only a function of the control action $\mathbf{u}[k]$ and previous prediction $\mathbf{y}[k-1]$, making T implicit:

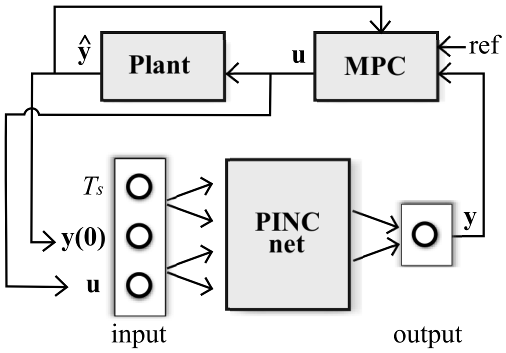
$$\begin{aligned}\mathbf{y}[k] &= \widehat{f}_{\mathbf{w}}(\mathbf{y}[k-1], \mathbf{u}[k]) \\ &= f_{\mathbf{w}}(T, \mathbf{y}[k-1], \mathbf{u}[k])\end{aligned} \tag{8}$$

We call $\widehat{f}_{\mathbf{w}}$ the control interface for the PINC framework. Thus, the Jacobian matrix $\frac{\partial \widehat{f}_{\mathbf{w}}}{\partial \mathbf{u}}$ can be computed and provided to solvers used in MPC, possibly employing automatic differentiation. This control interface provides the prediction of the states of the dynamic system at the vertical lines in Fig. 3, that is, at every T_s seconds, the state $\mathbf{y}[k]$ is predicted in a single forward network propagation operation, for $k = 1, ..., M$ (prediction horizon). This prediction interface differs from numerical integration methods that need to integrate over the continuous inner interval [42].

Since the prediction is fed back as an input at every discrete timestep, errors accumulate in the long free run or when the MPC model (PINC net) is used in a future finite prediction horizon to solve a constrained optimization problem. In this case, the prediction $\mathbf{y}[k-1]$ is fed back as no readings from the real process at a future time are possible (Fig. 4a). This is not exclusive of this approach and is standard to recurrent neural networks or autoregressive approaches. However, because MPC works in a receding horizon control approach, at every timestep k of the control loop, the input $\mathbf{y}[k-1]$ representing the initial state is set to the system's actual state $\widehat{\mathbf{y}}[k-1]$ (Fig. 4b). Thus, the prediction horizon in MPC always starts



(a) PINC in self-loop or autoregressive mode



(b) PINC connected to the plant

Figure 4: Modes of operation of a trained PINC network during the prediction phase. (a) PINC net operates in self-loop or autoregressive mode, using its own output prediction as initial state $\mathbf{y}(0)$ of the next interval, after T seconds. This mode can be used for variable long-range simulation. In the control context, this operation mode is used within one iteration of MPC for trajectory generation until the prediction horizon of MPC completes (predicted output from Fig. 1). (b) Block diagram for PINC connected to the plant. One pass through the diagram arrows corresponds to one MPC iteration applying a control input $\mathbf{u}$ for T_s timesteps for both plant and PINC network. Note that the initial state of the PINC net is set to the plant's actual output. In practice, in MPC, these two operation modes are executed alternately (optimization in the prediction horizon and application of control action).

from the true initial state $\widehat{\mathbf{y}}[k-1]$, that is, Equation (8) becomes

$$\mathbf{y}[k] = \widehat{f}_{\mathbf{w}}(\widehat{\mathbf{y}}[k-1], \mathbf{u}[k]) \tag{9}$$

which counters error accumulation between consecutive control iterations.

3.3.3. Training

The training of PINC follows an offline approach similar to PINNs, with all training data generated before the training starts. Upon completion of the training process, the trained PINC can then be seamlessly employed as a model within Model Predictive Control (MPC).

The first loss term in Equation (3), which computes the prediction error with respect to the data points (initial conditions), can be generalized to the PINC net as:

$$\text{MSE}_y = \frac{1}{N_y} \sum_{i=1}^{N_y} \frac{1}{N_t} \sum_{j=1}^{N_t} |y_i(\mathbf{v}^j) - \widehat{y}_i^j|^2, \tag{10}$$

where: the pair $(\mathbf{v}^j, \widehat{\mathbf{y}}^j)$ corresponds to the j-th training example; $\mathbf{v}^j = (t, \mathbf{y}(0), \mathbf{u})^j$ is the whole input to the network (i.e., time, initial state, and control input); and $\widehat{\mathbf{y}}^j$ is the target output at time t for initial condition $\mathbf{y}^j(0)$ and control input $\mathbf{u}^j$. Usually, this dataset comes from measured data. However, in this work, we will show that if we assume that the given ODE is an exact representation of the process, it is enough for this dataset to contain only the initial conditions of the modeled ODE. For instance, one such training data pair is $((0, 0.4, 0.6), 0.4)$, which means that at $t = 0$ the initial state is 0.4, the control input is 0.6, and the desired output is equal to the initial state (0.4). As all training data pairs represent initial conditions, $t = 0$ for all points, whereas $\mathbf{y}(0)$ and $\mathbf{u}$ are randomly sampled from intervals defined according to the modeled dynamic system. This means that MSE_y represents the mean squared error for all randomly sampled initial conditions of the considered ODE and control inputs. Notice also that the input $\mathbf{y}^j(0)$ is always equal to the desired output $\widehat{\mathbf{y}}^j$ in the training set, since the time input is always $t = 0$. In this way, the network must learn from labeled data to *reproduce* the initial state $\mathbf{y}^j(0)$ into the network output $\mathbf{y}(\mathbf{v}^j)$ at $t = 0$. In practice, the assumption above allows training using randomly sampled data to solve the ODE without requiring measured process data.

The second loss term in Equation (3), which is the physics-informed loss on the collocation points (unlabeled points), is rewritten as:

$$\text{MSE}_{\mathcal{F}} = \frac{1}{N_y}\sum_{i=1}^{N_y}\frac{1}{N_{\mathcal{F}}}\sum_{k=1}^{N_{\mathcal{F}}}|\mathcal{F}(y_i(\mathbf{v}^k))|^2, \tag{11}$$

where $\mathbf{v}^k$ corresponds to the k-th collocation point $(t, \mathbf{y}(0), \mathbf{u})^k$, where now all three types of inputs (and not only the last two), i.e., time, initial condition, and control input, are randomly sampled from their respective particular intervals. Specifically, the interval for t is $[0, T]$, where T is the *inner continuous interval* of the PINC framework.

Basically, this formulation means that the PINC net is trained with *data* points that lie on the boundary of simulations, i.e., only initial states of ODEs appear in the loss function in Eq. (10). Practically, this does not require collecting data from ODE simulators. On the other hand, the collocation points in $\text{MSE}_{\mathcal{F}}$ serve to regularize the PINC net to satisfy the behavior defined by $\mathcal{F}$. Thus, in the training process, the PINC net is only directly informed with an initial state in Eq. (10), while its physics-informed cost loss in Eq. (11) must enforce the structure of the differential equation into its output $\mathbf{y}(\cdot)$ for the remaining *inner continuous interval* of T seconds (e.g., $t \in (0, T]$).

Fig. 5 illustrates how the datasets for both data loss and physics-informed loss are generated. While the white dots, the initial conditions, are used to minimize the residual regression MSE_y, the red dots are the collocation points used to minimize the physics-informed residual $\text{MSE}_{\mathcal{F}}$. For the former, the target output of PINC is the initial condition itself $\mathbf{y}(0)$. For the latter, the target is not available. The figure also shows the different pairs of initial conditions $\mathbf{y}(0)$ and control actions $\mathbf{u}$ that are randomly sampled in the interval of interest of the application, forming the labeled training data points (white dots).

The total loss can be generalized to $\text{MSE} = \text{MSE}_y + \lambda \cdot \text{MSE}_{\mathcal{F}}$, where λ represents a rescaling factor so that both terms are approximately in the same scale. Once the PINC net structure, datasets and the losses are defined, the training process starts with the ADAM optimizer [35] for K_1 epochs and subsequently continues with the L-BFGS optimizer [36] for K_2 iterations in order to adapt the network's weights $\mathbf{w}$ towards the minimization of MSE. Notice that automatic differentiation is employed for the physics-informed term $\text{MSE}_{\mathcal{F}}$ in Eq. (11), using deep learning frameworks such as *Tensorflow*.

Training the PINC does not explicitly account for the autoregressive property, which happens only in simulation mode. That is, there is no error backpropagation-through-time

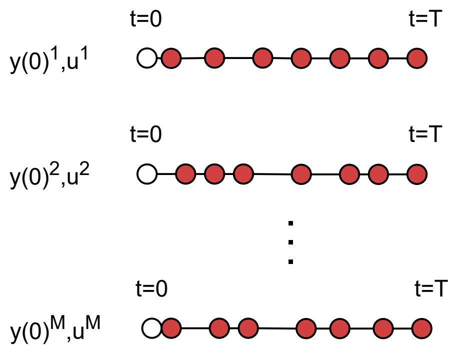


Figure 5: Illustration of a dataset generated for training a PINC net. There are M randomly generated pairs of initial conditions $\mathbf{y}(0)$ and control actions $\mathbf{u}$. For each one of them, a number of collocation points is generated in the interval $t \in [0, T]$, represented by red filled dots. All labeled data points, given by white dots, correspond to initial conditions $t = 0$, where each one is associated with a randomly generated pair of $\mathbf{y}(0)$ and $\mathbf{u}$ as well. In practice, the $N_{\mathcal{F}}$ randomly drawn collocation points used for computing $\text{MSE}_{\mathcal{F}}$ are independent of the N_t data points used to compute MSE_y.

at the instant of feedback. Actually, no prediction feedback takes place during training since the inputs to the network can be randomly sampled, immensely simplifying the training process. By sampling enough data and collocation points and running the training properly (e.g., long enough), the autoregressive simulation will happen naturally by chaining the predictions in a self-loop mode in the test stage.

3.3.4. NMPC

After training, the PINC net is used as a model in nonlinear MPC, whose algorithm appears in Section 3.2. Thus, the control interface function $\widehat{f}_{\mathbf{w}}$ in Equation (8) replaces Equation (5b) in the MPC formulation, redefining the notation of a dynamic system's state by the prediction given by the PINC network, i.e., $\mathbf{x}[k] = \mathbf{y}[k]$. After these substitutions, we arrive at a Multiple Shooting (MS)-inspired formulation for the NMPC problem under the PINC framework:

$$J = \sum_{j=N_1}^{N_2} \left\| \mathbf{y}[k+j] - \mathbf{y}^{\text{ref}}[k+j] \right\|_{\mathbf{Q}}^2 + \sum_{i=0}^{N_u-1} \|\mathbf{\Delta u}[k+i]\|_{\mathbf{R}}^2 \tag{12a}$$

while being subject to:

$$\mathbf{y}[k+j+1] = \widehat{\mathbf{f}}_{\mathbf{w}}(\mathbf{y}[k+j], \mathbf{u}[k+j]), \quad \forall j = 0, \ldots, N_2 - 1 \tag{12b}$$

$$\mathbf{u}[k+j] = \mathbf{u}[k-1] + \sum_{i=0}^{j} \mathbf{\Delta u}[k+i], \quad \forall j = 0, \ldots, (N_u - 1) \tag{12c}$$

$$\mathbf{u}[k+j] = \mathbf{u}[k+N_u-1], \quad \forall j = N_u, \ldots, N_2 - 1 \tag{12d}$$

$$\mathbf{h}(\mathbf{y}[k+j], \mathbf{u}[k+j]) \leq 0, \quad \forall j = N_1, \ldots, N_2 \tag{12e}$$

$$\mathbf{g}(\mathbf{y}[k+j], \mathbf{u}[k+j]) = 0, \quad \forall j = N_1, \ldots, N_2 \tag{12f}$$

3.4. Metrics

The PINC net prediction performance is evaluated on a validation set in self-loop mode (Figures 3 and 4a). In particular, the generalization MSE is computed only at the discrete time steps (vertical lines in Fig. 3):

$$\text{MSE}_{gen} = \frac{1}{N_y}\sum_{i=1}^{N_y}\frac{1}{N}\sum_{k=1}^{N}\big(y_i[k]-\widehat{y}_i[k]\big)^2, \tag{13}$$

where: $\mathbf{y}[k]$ is the prediction of the PINC net given by Equation (8) and $\widehat{\mathbf{y}}[k]$ is obtained with Runge-Kutta (RK) simulation of the true model of the plant; N is the length of the vector $\mathbf{y}$; and the PINC net and the RK model receive the same control input signal $\mathbf{u}[k]$.

The control performance is measured by employing the Integral of Absolute Error (IAE) on a simulation of C iterations:

$$\text{IAE} = \frac{1}{N_y}\sum_{i=1}^{N_y}\sum_{k=1}^{C}\big|y_i^{\text{ref}}[k]-y_i[k]\big| \tag{14}$$

and the Root Mean Squared Error (RMSE):

$$\text{RMSE} = \frac{1}{N_y}\sum_{i=1}^{N_y}\sqrt{\frac{1}{C}\sum_{k=1}^{C}\big(y_i^{\text{ref}}[k]-y_i[k]\big)^2} \tag{15}$$

where $y_i^{\text{ref}}[k]$ is the reference value of $y_i[k]$ at timestep k.

The IAE is ideal for comparing simulation runs with the same reference signal, as the sum of absolute errors is very sensitive to changes in control performance [43]. Meanwhile, the RMSE can capture the average error behavior of the controller.

3.5. PINC Algorithms

In this section, an overview of the proposal is presented with the help of high-level algorithms. Algorithm 1's objective is training the PINC network. It uses data points and collocation points (generated as described in Section 3.3.3) to minimize Eq. (10) + Eq. (11), first with ADAM optimizer and then with L-BFGS optimizer.

The values for K_1 and K_2 are not critical for the training process in this work. ADAM is used for a few initial iterations to speed up training, avoiding local minima, and the more stable L-BFGS is used afterward until convergence. ADAM is run with a learning rate that is halved every 100 epochs, starting at 10^{-3}.

Algorithm 2 employs MPC with PINC using the minimization process (NMPC) described in Section 3.3.4 for each timestep k out of C iterations (i.e., the total length of the reference signal) to yield a control action $\mathbf{u}[k]$ to be applied to the plant.

Algorithm 1: PINC Training Algorithm

```
input: $K_1$, $K_2$, $\mathcal{F}(\cdot)$, $\{(\mathbf{v}^j, \widehat{\mathbf{y}}^j) : j = 1, \ldots, N_t\}$, $\{\mathbf{v}^k : k = 1, \ldots, N_{\mathcal{F}}\}$;
initialize PINC weights $\mathbf{w}$ with Xavier normal distribution;
// Train with ADAM
for $K_1$ epochs do
    Compute the gradients of Eq. (10) + Eq. (11) (with $\mathcal{F}(\cdot)$) with respect to $\mathbf{w}$
      using the data points $\{(\mathbf{v}^j, \widehat{\mathbf{y}}^j) : j = 1, \ldots, N_t\}$ and collocation points
      $\{\mathbf{v}^k : k = 1, \ldots, N_{\mathcal{F}}\}$;
    Update $\mathbf{w}$ with ADAM optimizer and the obtained gradients;
// Train with L-BFGS
for $K_2$ iterations do
    Compute the gradients of Eq. (10) + Eq. (11) (with $\mathcal{F}(\cdot)$) with respect to $\mathbf{w}$
      using the data points $\{(\mathbf{v}^j, \widehat{\mathbf{y}}^j) : j = 1, \ldots, N_t\}$ and collocation points
      $\{\mathbf{v}^k : k = 1, \ldots, N_{\mathcal{F}}\}$;
    Update $\mathbf{w}$ with L-BFGS optimizer and the obtained gradients;
    Save network $\mathbf{w}$ with the best performance seen so far on a validation set using
      Eq. (13);
output: network $\mathbf{w}$ with lowest validation error;
```

Algorithm 2: MPC with PINC Algorithm

```
input: $\mathbf{Q}$, $\mathbf{R}$, $N_u$, $N_1$, $N_2$, $\mathbf{y}^{\mathrm{ref}}$, $\widehat{\mathbf{f}}_{\mathbf{w}}$, $\mathbf{x}[0]$
// Use the trained PINC to perform the control procedure
for $k := 0, 1, 2, \ldots, C$ do
    Set initial state $\mathbf{y}[k]$ to the plant's current state $\mathbf{x}[k]$;
    Minimize (12a) s.t. (12b)—(12f), with respect to control $\mathbf{u}$ for reference $\mathbf{y}^{\mathtt{ref}}$ at
      timestep $k$, using the trained network $\widehat{\mathbf{f}}_{\mathbf{w}}$ as predictive model, control horizon
      $N_u$, prediction horizon $M = N_2 - N_1 + 1$, and weight matrices $\mathbf{Q}$ and $\mathbf{R}$;
    Apply $\mathbf{u}[k]$ to the plant, obtaining the next states $\mathbf{x}[k+1]$;
output: control action $\mathbf{u}$
```

4. Experiments

This section presents experiments regarding applying PINC to the modeling and control of the Van der Pol Oscillator, the four-tank system, and an ESP-lifted oil well. The first two dynamical systems are often considered for nonlinear analysis in the literature, and the latter is used in real-world applications.

4.1. Van der Pol Oscillator

4.1.1. Model

The Van der Pol oscillator [44] is an ODE initially discovered by Balthazar Van der Pol that had the original purpose of modeling triode oscillations in electric circuits. Since then, the oscillator ODE has been used for other purposes, such as seismology and biological neuron modeling [44], and as a standard proof-of-concept dynamical system for optimal control applications [45]. The equations that govern the Van der Pol Oscillator are as follows:

$$\dot{x}_1 = x_2 \tag{16a}$$

$$\dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1 + u \tag{16b}$$

where $\mu = 1$ is referred to as the damping parameter, which affects how much the system will oscillate, $\mathbf{x} = (x_1, x_2)$ is the system state, and u is an exogenous control action.

The open-loop Van der Pol oscillator has an equilibrium point at $\bar{\mathbf{x}} = (u, 0)$. The equilibrium is stable for a constant $u \in (-\sqrt{3}, -1)$ or $u \in (1, \sqrt{3})$. The oscillator also has a limit cycle that can be perceived in polar coordinates [44]. In our experiments, we consider $x_1, x_2 \in [-3, 3]$ and $u \in [-1, 1]$ for not being locally stable and containing the origin.

4.1.2. PINC Analysis

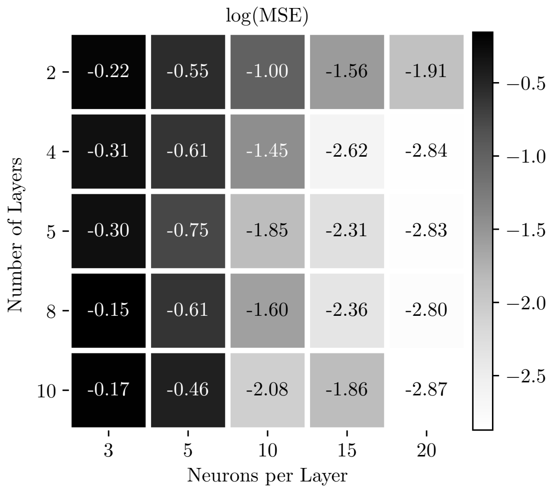
To find the most suitable configuration for the PINC net to control a dynamical system, we propose first running grid search experiments over hyperparameters, such as the network complexity and the number of data points (N_t) and collocation points (N_f).

Here, the sampling time is chosen according to the particular dynamics of the Van der Pol oscillator: $T = T_s = 0.5$ s. In our experiments, $N_t = 1,000$ and $N_f = 100,000$ provided sufficient points to train a PINC net.

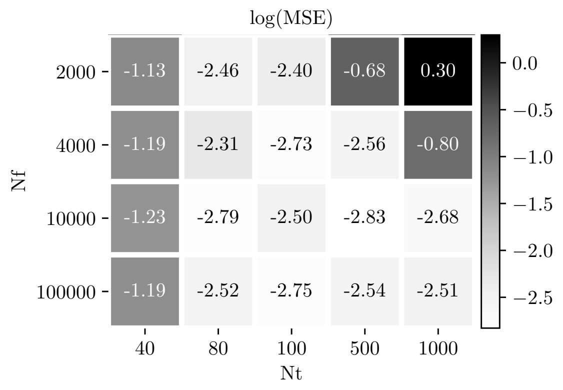
For training the PINC net, ADAM is used to optimize the loss function for $K_1 = 500$ epochs, and afterward, L-BFGS is used for $K_2 = 2,000$ iterations to enhance the stability of the training process. Note that this K_2 does not exhaust the training so that it may be increased before the final deployment of the PINC net. The parameter λ is set empirically so that MSE_y and $\text{MSE}_\mathcal{F}$ are not in disparate scales. The validation dataset is composed of 1810 points obtained using a randomly generated control action u (e.g., Fig. 10), which is equivalent to $905s$ of simulation, since $T_s = 0.5s$. The validation or generalization error considers the self-loop mode of PINC to compute Eq. (13).

The first experiment analyzes the network complexity (Fig. 6a) and shows the validation MSE using Eq. (13) averaged over ten different random initializations of the network weights.

In general, as the network grows deeper and with more neurons per layer, the performance increases. Besides, layers with 3 or 5 neurons cannot model the required task. Note that these errors would decrease even further if the training had continued for more epochs (*correcting* the lower performance of the net of 10 layers with 15 neurons each, for instance). Although the network of 10 layers with 20 neurons each achieves the best performance, we choose a configuration of 4 layers with 20 neurons for the following experiments, which also achieved excellent performance but with less computational overhead.



(a) Effect of the number of layers and neurons per layer



(b) Effect of the number of collocation points N_f and data points N_t

Figure 6: Analysis of the PINC net for the Van der Pol Oscillator. The network training time is fixed to a constant number of iterations. The MSE validation error is computed according to Equation (13). (a) The $\log_{10}$ of the MSE error as a function of network complexity averaged over ten different simulations. A deep network of 10 layers with 20 neurons each achieved the best generalization error ($10^{-2.87}$). Note that the performance increases as the network becomes deep and neurons are added to each layer. Besides, layers with 3 or 5 neurons are not sufficient to achieve a satisfactory model. (b) The effect of the number of collocation points N_f and data points N_t on generalization performance, averaged over five randomly initialized networks. These experiments show that 40 data points are insufficient and the proportion N_f/N_t should be significantly higher than 4 (hence the dark cells in the upper-right corner of the plot).

In Fig. 6b, the proportion between data points and collocation points is investigated. Each error cell in the plot corresponds to the average of 5 experiments with randomly generated networks. Clearly, 40 data points are insufficient, and the proportion N_f/N_t should be considerably higher than 4 (hence the dark cells in the upper-right corner of the plot).

4.1.3. Long-range Simulation

In order to showcase the capacity of long-range simulation of the proposed approach compared with the conventional PINN, we trained traditional PINNs that have the same complexity as the PINC net, i.e., 4 layers of 20 neurons each, but that have only one input t

as usual for PINNs. A new PINN is trained for each interval considered $T \in \{0.5, 1, 2, 5, 10\}$ seconds. Note that these PINNs do not allow for arbitrary initial conditions after training, as PINC nets do. Furthermore, once the interval value T is chosen before training for conventional PINNs, further simulation beyond T_s rapidly deteriorates, as we shall see.

The PINNs were trained with the ADAM optimization algorithm for $K_1 = 500$ epochs initially with a learning rate of 0.0035, and then other $K_1 = 700 \times T$ epochs with a learning rate of 0.001, and finally for $K_2 = 1,000 \times T$ iterations of the L-FGBS optimization method. In addition, the number of collocation points also increased with the value of T, $N_f = 5,000 \times T$. Thus, the longer the interval T, the longer the training and the higher the number of collocation points employed. On the other hand, only one PINC network was trained, following the configuration from the previous section, but for longer, as indicated in Fig. 9.

In Fig. 7, the results are shown, which compare the trajectories of the single PINC net that works for any considered interval T (e.g., shorter or longer than 10s) with the ones from the PINN networks. The two rows in the plot correspond to the two states of the oscillator. Each subplot involves training a new PINN from scratch for a specific $T \in \{0.5, 1, 2, 5, 10\}$ s, except for the PINC net, which is trained only once. Besides, the training of each PINN considers a fixed control input $u = 0.54$ along the run, with fixed initial conditions $x_1 = -2.14$ and $x_2 = 0.25$, both randomly chosen. Unlike PINC, conventional PINNs must be retrained from scratch if a different initial condition or control input is required.

In the plot of Fig. 7, the dots in the predicted PINC trajectories, in blue and pink colors, mark the moments at which the final predicted states at $T = 0.5$ s are fed back as new initial conditions and input to the network, corresponding to the vertical lines in Fig. 3. Although PINC is trained with a fixed $T = 0.5$ s, its chained (self-loop) prediction enables long-range simulation for an arbitrary simulation time T without fixing it beforehand as with traditional PINNs, whose trajectory is shown by the dashed gray lines in the plots of Fig. 7. Note that the target true trajectory of the dynamical system, drawn in a solid black line, is entirely superimposed by the predicted PINC trajectory. In addition, observe that only the PINN trained specifically with $T = 10$ s can simulate without degradation until 10 s, and not beyond that, for the given fixed initial condition and control input. That is, the simulation from 0 s to 10 s of any PINN trained with $T < 10$ s shows that extrapolation beyond the T s fixed at training time is unfeasible.

Fig. 8 presents the RMSE error for these experiments, making clear the high prediction error obtained by the conventional PINN compared to the proposed PINC approach when the T used for PINN training is lower than $10s$. At $T = 10$ s, PINN has a slightly lower error than PINC, likely because of the small accumulation of prediction errors during self-loop simulation for PINC.

The control input u was fixed here to compare the conventional and new approaches. However, PINC can have a variable u along the simulation, yielding an additional advantage for allowing control applications, as showcased in the next section.

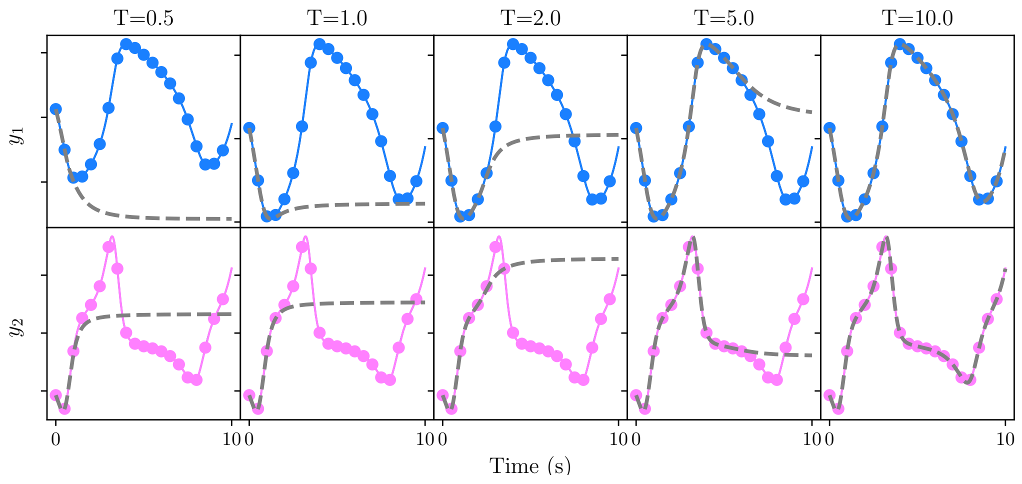


Figure 7: Comparison between conventional PINN (dashed grey line) and proposed PINC (solid blue and pink lines) for long-range simulation of the Van Der Pol oscillator with fixed control input u and fixed initial condition $\mathbf{x} = (x_1, x_2)$ along the simulation. The target trajectories for states x_1 and x_2 are plotted in black solid lines, which are completely superimposed by the PINC predictions y_1 and y_2. From left to right, the PINN nets are trained with fixed $T \in \{0.5, 1, 2, 5, 10\}$s, while the PINC net is trained only once with $T = 0.5$ s even though it can run for arbitrary longer simulation times not fixed beforehand. The conventional PINN can not extrapolate beyond its training time interval T. For instance, with $T = 2$ s, the simulation of the resulting PINN from 0 s to 10 s fails to follow the system trajectory after 2 s.

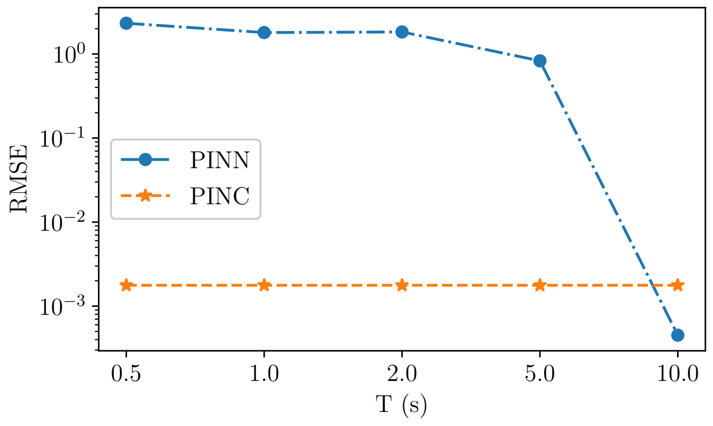


Figure 8: Performance comparison in terms of RMSE between the target trajectory of the Van der Pol oscillator and the predicted trajectory for the PINN and PINC networks from Fig. 7 for a simulation of $10s$. The horizontal axis corresponds to the fixed T used for training the PINN network. These experiments show that the conventional PINN yields a high prediction error in comparison with the proposed PINC approach when the training time T of the PINN is lower than 10 s. With a training time $T = 10$ s, the PINN achieves a slightly lower error than PINC, arguably because of the small accumulation of prediction errors during self-loop simulation for PINC.

4.1.4. PINC Control

The PINC net with 4 hidden layers and 20 neurons each is chosen for the Van der Pol oscillator. Besides, we continue setting $N_t = 1,000$, $N_f = 100,000$, and $K_1 = 500$, but the training is extended with $K_2 = 20,000$ allows the MSE to settle in an asymptotic curve (Fig. 9). For comparison, a vertical black dashed line is plotted in Fig. 9, indicating the moment training would have stopped for earlier experiments from Fig. 6. Thus, further training allows improving validation error (according to Equation (13)) at least one order of magnitude. Note that the validation error continues to decrease as training follows, arguably due to the regularization effect of $\text{MSE}_{\mathcal{F}}$ in the loss function.

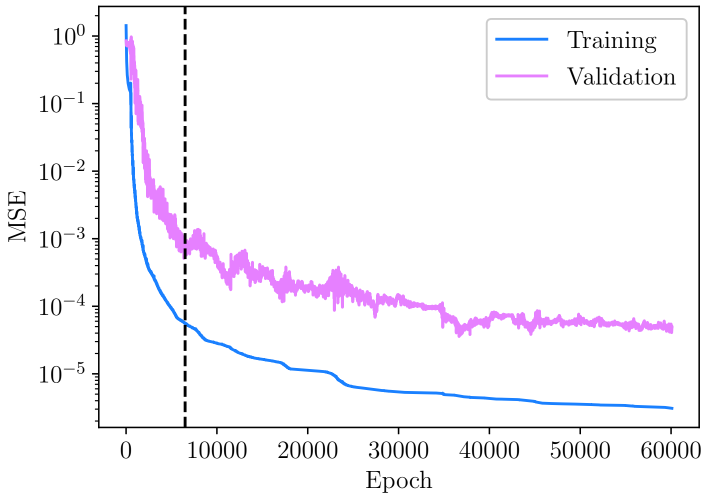


Figure 9: MSE evolution during training of the final PINC net. Previous experiments from Section 4.1.2 stopped training at the vertical dashed line. The validation dataset consists of 1810 points or 90 s of simulation since $T = T_s = 0.5$ s. The validation MSE is noisier because it is computed on a much smaller dataset and in self-loop mode using Eq. (13).

We randomly generate a control input u for 10 s to view the PINC prediction after training. In Fig. 10, the predicted trajectory is given for such a control input. With our method, we can directly infer each circle in the trajectory using Equation (8) every $T = 0.5$ s. The trajectory between two consecutive circles can be predicted by varying the input t of the network and keeping the other inputs $\mathbf{y}(0)$ and u fixed. The prediction matches the target trajectory very well, as the latter is also plotted but superimposed by the former.

The resulting control from PINC can be seen in Fig. 11 in a simulation of $60s$, where MPC was employed to find the optimal value of the control input, considering a prediction and control horizon of $5T$ (or $2.5s$). The control parameters are given as follows: $N_1 = 1$, $N_2 = 5$, $N_u = N_2$, $\mathbf{Q} = 10\mathbf{I}$, and $\mathbf{R} = \mathbf{I}$. Here, the optimization in MPC to find a control input at the current timestep uses the PINC network's predicted trajectory for future timesteps, i.e., for the prediction horizon of $2.5s$. This procedure is repeated for all 120 points of the plotted trajectory.

The controlled plant consists of the Van der Pol oscillator, whose states are obtained

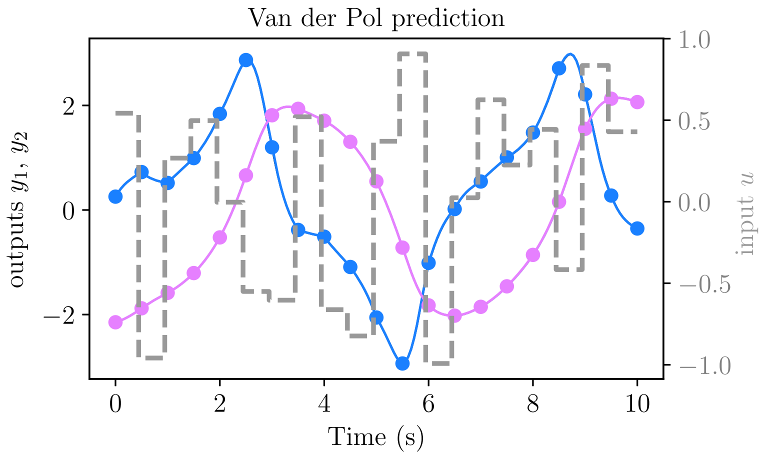


Figure 10: PINC net prediction for the Van der Pol oscillator on test data. The gray dashed line gives the randomly generated input u, while the predictions for the oscillator states x_1 and x_2 correspond to the solid blue and pink lines, respectively. The target trajectory, from RK method, is also plotted in black but is not visible as the prediction completely superimposes the former. Each dot in the predicted trajectory corresponds to the vertical lines in Fig. 3 when the control action and initial state change (after $T_s = T = 0.5$ s).

by an RK integrator. Table 1 presents the control performance for a $60s$ simulation, which also shows the result when the original ODE model serves as the predictive model in NMPC instead of the PINC net. In this case, the classic, fourth-order Runge-Kutta method (RK4) is employed as a numerical solution to compute the system states for NMPC. This means that practically other approximations to the plant/system are not likely to improve the ODE model itself, thus justifying our comparison to the baseline NMPC.

Remarkably, PINC practically achieves the same result as the ODE/RK approach regarding RMSE and IAE, while being slightly faster on average when executed with 10 repetitions on the same desktop computer. Notice that, for each reference r (dashed black signal), the controller drives the oscillator toward the steady-state $(x_1, x_2) = (u, 0)$ with $u = r$.

4.2. Four Tanks

4.2.1. Model

The four-tank system is a widely used benchmark for multivariate control systems [46], for being a nonlinear and multivariate system with some degree of coupling between variables. By setting its parameters to a given combination of values, it is possible to induce the system to have non-minimum phase transmission zeros, which are an additional difficulty for PID controllers [46].

As Figure 12 shows, the four-tank system comprises four tanks, denoted by the index $i \in \{1, 2, 3, 4\}$, and two pumps $j \in \{1, 2\}$ supplying each tank with water. Each tank has a cylindric form with a basis area of A_i, and an orifice of area a_i at the basis center. Tank 1 (2) is located right below tank 3 (4) so that the flow ω_i from the tank above goes directly to the tank below. Both pumps are linear actuators controlled by the voltage u_j with coefficient

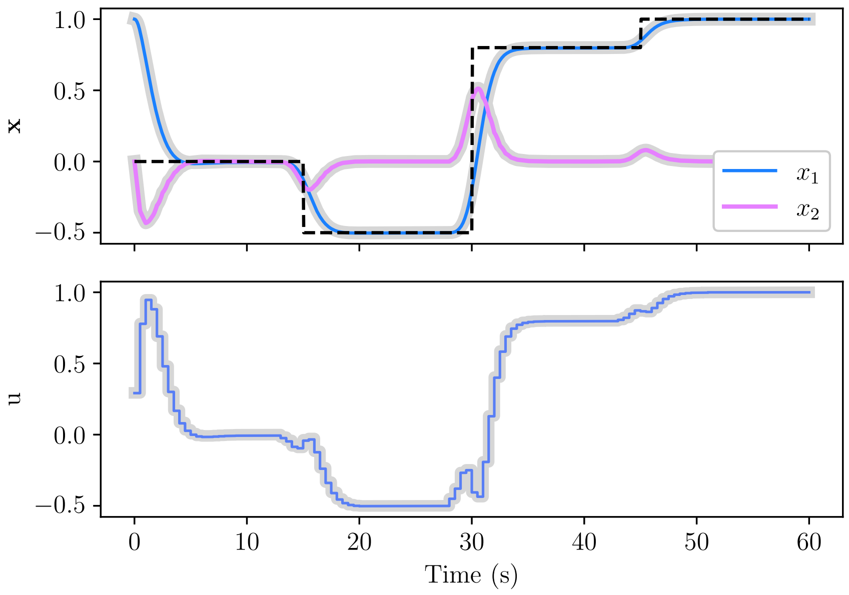


Figure 11: Control of the Van der Pol oscillator for 60s with PINC and with NMPC using the ODE/RK model. A dashed black step signal gives the reference trajectory for x_1, while for x_2 the reference is kept at zero (not shown). The controlled variables are the states x_1 and x_2 given by blue and pink lines, respectively. These latter colors are used to show the control by PINC, while the baseline (ODE/RK model) is given by a thick gray line. The control input u is the manipulated variable in the lower plot, found by MPC. For each reference r (dashed black signal), the controller drives the oscillator toward the steady-state $(x_1, x_2) = (u, 0)$ with $u = r$. The control RMSE (IAE) errors for PINC and RK models were 0.1506 (123) and 0.1507 (121), respectively. See text for more details.

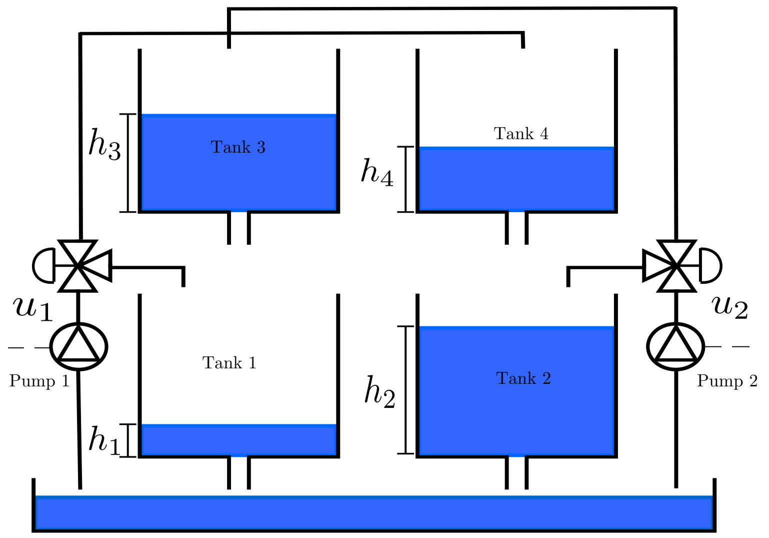


Figure 12: Schematic representation of the four-tank system, from [47].

k_j, converting the voltage into the pump flow. Pump 1 (2) is associated with a directional valve that distributes the resulting flow into tanks 1 and 4 (2 and 3), which is the coupling source in this system. The directional valves have an opening $\gamma_j \in (0, 1)$, the fraction they distribute to the bottom tanks. Adjusting γ_j is one of the main factors in regulating the control problems associated with the system [46]. The state variables h_i denote the water levels in the tanks. The following equations govern the four-tank system, which are derived from mass balance:

$$\dot{h}_1 = \frac{\gamma_1 k_1 u_1 + \omega_3 - \omega_1}{A_1} \tag{17a}$$

$$\dot{h}_2 = \frac{\gamma_2 k_2 u_2 + \omega_4 - \omega_2}{A_2} \tag{17b}$$

$$\dot{h}_3 = \frac{(1 - \gamma_2) k_2 u_2 - \omega_3}{A_3} \tag{17c}$$

$$\dot{h}_4 = \frac{(1 - \gamma_1) k_1 u_1 - \omega_4}{A_4} \tag{17d}$$

where the flow in each tank orifice ω_i is described by the Bernoulli orifice equation, adding the sole nonlinearity of the system:

$$\omega_i = a_i \sqrt{2 g h_i} \tag{18}$$

with g as the acceleration of gravity. The parameters used for this application are the same as the ones stipulated for the non-minimum phase experiment in [46].

4.2.2. PINC Control

We have followed a similar approach to the first control problem concerning finding a suitable configuration for network complexity and the proportion between data and collocation points. We observed that 5 is the minimum number of layers to obtain sufficient prediction performance for the four-tank system since it is a more complex plant, with multiple inputs and multiple outputs (MIMO) operating at different timescales. The following experiments consider a PINC net with 5 layers of 20 neurons each. Besides, we continue setting $N_t = 1,000$, $N_f = 100,000$, $K_1 = 500$, and $K_2 = 20,000$. The sampling period is $T = T_s = 10$ s. The control parameters are once again given by $N_1 = 1$, $N_2 = 5$, $N_u = N_2$, $\mathbf{Q} = 10\mathbf{I}$, and $\mathbf{R} = \mathbf{I}$.

After training the PINC net, the prediction on test data, with new randomly generated control actions (not shown), is presented in Fig. 13. The deviation in prediction at longer ranges, as seen in the first plot for h_1 and h_2, is expected since the network works in self-loop mode, feeding its prediction of the last state back as input for the initial state (Fig. 4a), every $T = 10$ s. Thus, the error accumulates in this chaining procedure. However, MPC uses this trajectory only up to 50 s, equivalent to a prediction horizon of 5 steps, indicated by the vertical dashed line in the figure, and the following optimization procedure in MPC resets the initial state to the actual value as obtained by sensors of the physical process (Fig. 4b).

PINC's control employs prediction and control horizons, both of 5 steps ($50s$ in simulation time) for the four-tank system. Besides, both h_3 and h_4 tank levels are constrained to the

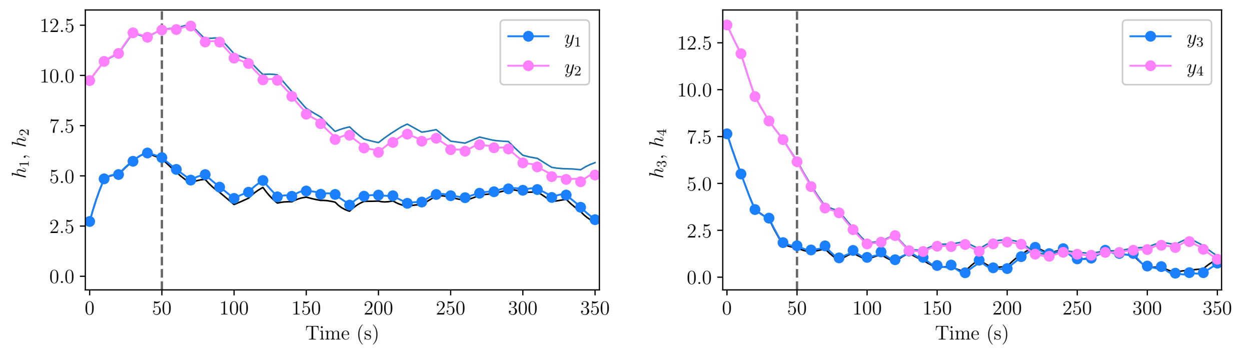


Figure 13: PINC net prediction in self-loop mode for the four-tank system on test data, with randomly generated control input signals similar to Fig. 10. The predictions for the level of the tanks h_1 and h_2 correspond to the solid blue and pink lines, respectively. The target trajectories are plotted in dark, solid lines without dots from the RK method. At each dot in the predicted trajectory, the PINC net receives new inputs for control action and initial state (every $T_s = T = 10$ s), as explained in Fig. 4a. The vertical dashed line indicates the prediction horizon used for MPC. Notice that PINC net predictions are highly accurate for the prediction horizon (50 s) as they match the state trajectory obtained with the RK method.

interval $[0.6, 5.5]$cm. Fig. 14 shows the results. The top two plots present the controlled and constrained tank levels, while the bottom plot depicts the control action found by the MPC. The plots on the right-hand side show a close-up during the initial 160s of the simulation. The control was successful despite the constraints imposed on h_3 and h_4 (which were respected) and some minor errors in the steady-state regime, which can be countered by adding the calculation of a correction factor through filtering the error between the measurement and the network prediction, as done in [3] for a recurrent network. In Fig. 15, we use the same simulation setup, focusing on the timesteps between 500s and 1300s, to compare with the response (in yellow color) of the control using the plant reference model as a predictive model in MPC. This ODE/RK-based model is the reference model that represents the plant itself, which justifies the negligible steady-state regime error observed in the figure. It can be noticed that the PINC simulation is very close to the nominal MPC given by the ODE/RK model. This comparison suffices as another NMPC would employ an approximation of the ODE/RK model as a predictive model.

Table 1 shows the control performance regarding RMSE and IAE. Although IAE seems to differ more between PINC and ODE/RK, RMSE errors for both methods are almost equivalent. The reported simulation times in this paper considered the execution on a Mac, with 3.1 GHz 6-Core Intel Core i5 processor and 32 GB 2667 MHz DDR4. The average time spent for the complete control simulation using PINC, repeated 10 times, $10.85s$, is 23.3% inferior to using the ODE of the four tanks as a model for MPC ($14.15s$), which is remarkable given that the PINC has an architecture of just 5 hidden layers with 20 neurons each.

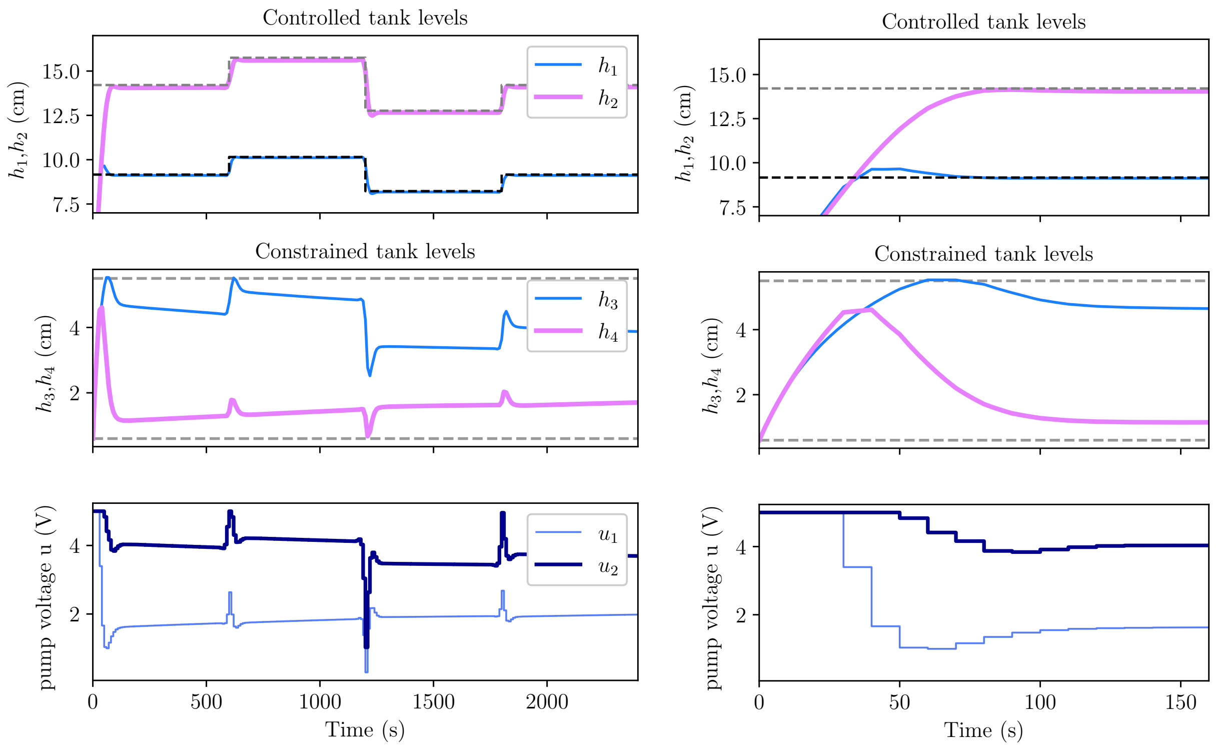


Figure 14: Control of the four-tank system with PINC. The controlled variables are the tank levels h_1 and h_2 given by blue and pink lines, respectively, whereas the reference trajectory for h_1 (h_2) corresponds to the dashed black (gray) step signal. The control inputs $\mathbf{u}$ are the manipulated voltages shown in the lower plot, found by MPC. Dashed gray horizontal lines represent the lower and upper limits for h_3 and h_4. Left: simulation amounting to 2400s. Right: close-up on the first 160s. The initial conditions for h_1 and h_2 are $(2, 2)$, which is the minimum of the allowed interval $[2, 20]$. See text for more details.

Table 1: Results for two benchmark control experiments.

	Van der Pol (4 layers of 20 units)			Four tanks (5 layers of 20 units)		
	RMSE	IAE	time (s)	RMSE	IAE	time (s)
PINC	0.15	123.6	$\mathbf{3.32 \pm 0.15}$	0.811	876	$\mathbf{10.85 \pm 0.14}$
ODE / RK	0.15	**122**	3.41 ± 0.04	**0.807**	**544**	14.15 ± 0.13

4.2.3. Sensitivity to Perturbations

As introduced in this work, the PINC approach does not have an inherent method to deal with modeling errors and completely reject disturbances, making these points valid for future research. Nonetheless, the control algorithm can implement error correction filtering [2] and Kalman filters [48] for robustness to model mismatch and to counter disturbances.

While works such as [49] and [50] are focused on proving stability theoretically through the use of Lyapunov functions, we showcase the PINC robustness experimentally since our focus is more on applications. To test the robustness of the proposed formulation to pa-

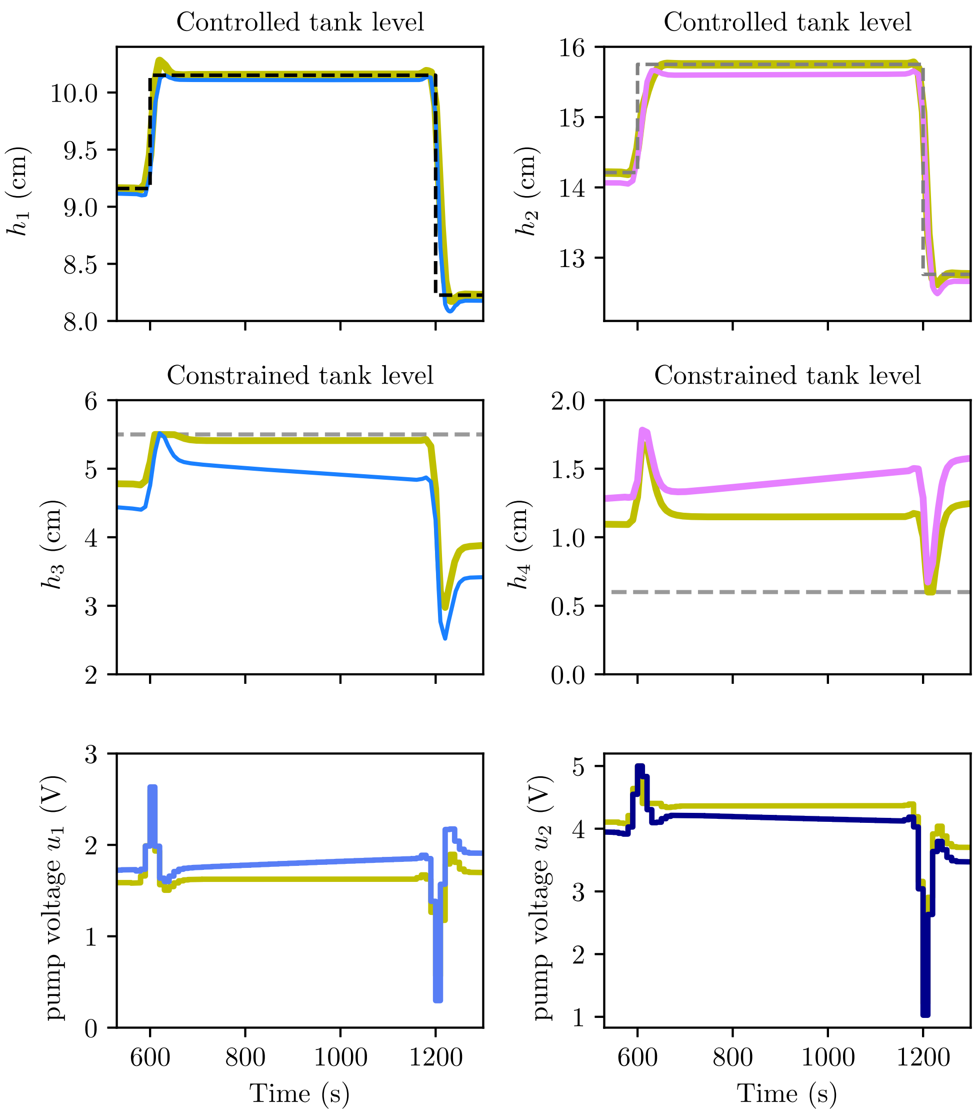


Figure 15: Control of the four-tank system with PINC and the RK/ODE model for timesteps 500s to 1300s from Fig. 14. All the yellow lines correspond to the simulation where the ODE model itself is the predictive model (using a numerical solution at each timestep in the prediction horizon). In contrast, the remaining lines refer to the simulation with PINC net as a predictive model. Though slightly different, the control signals produced by the NMPC with PINC and RK/ODE follow the same patterns. Also, the simulation of the state variables with PINC MPC is very close to the nominal MPC simulation given by the ODE/RK model.

rameter mismatch, we performed a sensitivity analysis for the four-tank scenarios previously presented. The analysis assumed random deviations in the values of the k_1 and k_2 parameters (see Eq. 17). The perturbed values $\widetilde{k}_1$ and $\widetilde{k}_2$ are sampled from uniform probability distributions $U_{5\%}(a,b) = [0.95x, 1.05x]$, with x being the nominal value for training the PINN.

Altogether, 151 simulations were carried out by injecting the deviations in the system. The results are shown in the first column of Fig. 16 for two different networks, one with 5 hidden layers of 20 neurons each and another with 8 hidden layers of 20 neurons each. Despite the random variation of the parameters k_1 and k_2, one can see that the IAE of the system has variations within a tolerance range considered adequate. Fig. 17 shows the control of the four tanks when the plant controlled has the maximum deviation of 5% in k_1 and k_2 parameters, showcasing that the perturbation only slightly bias the trajectories.

A second experiment consisted of perturbing the initial condition with a uniform distribution. The perturbed initial conditions for h_1 and h_2 are sampled from $U_{5\%}(a, b) = [0.95x, 1.05x]$, with x being the nominal value 9 for both states. Fig. 16 shows these results in the second column. The peak of the histogram approximately coincides with the IAE obtained by the unperturbed model of the plant. Thus, other initial conditions can imply relatively lower or higher IAE. Notice that the bottom plots show instances with lower IAE, evidencing the higher accuracy of a deeper network, with 8 hidden layers, in this particular situation.

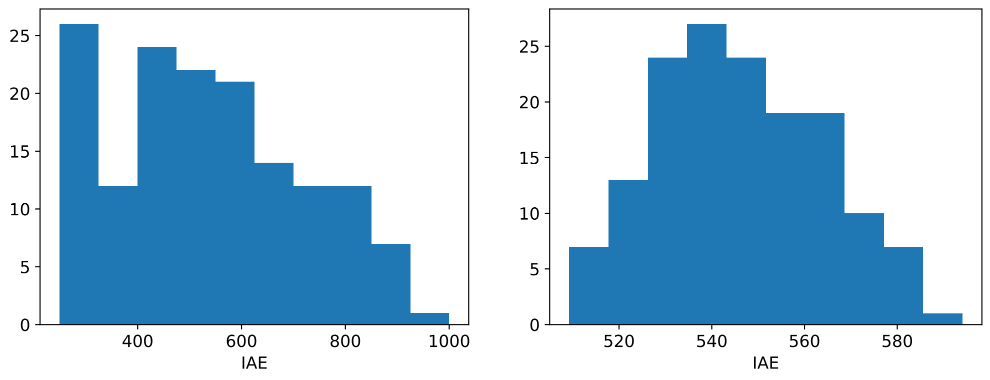
In summary, the sensitivity experiments imply that the system's performance does not degrade regarding IAE. Since the PINC control strategy has no integrators, a small steady-state error that depends on model match is expected. Because the model mismatch increases as the parameters k_1 and k_2 are driven away from their nominal values, the steady-state error is expected to be higher but still within an acceptable range of IAEs.

4.3. Electric Submersible Pump

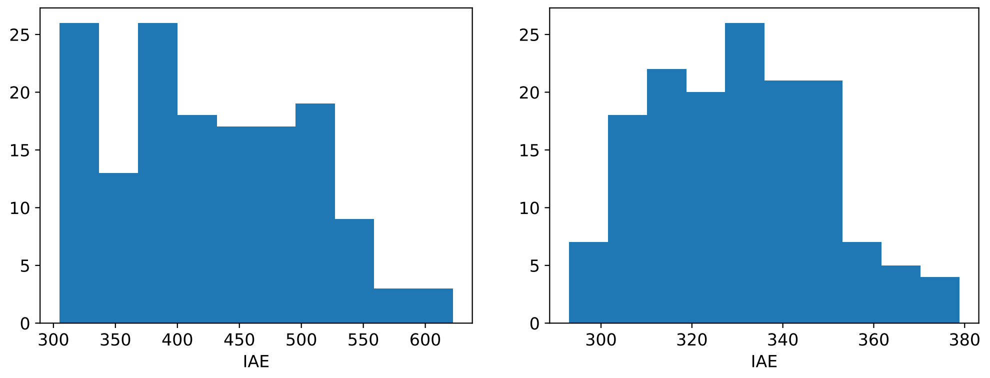
4.3.1. Model

An Electric Submersible Pump (ESP) is a type of pump installed in oil wells to enable or increase production, typically if the reservoir pressure is not sufficiently high to sustain the flow from the bottomhole to the top side. Figure 18 depicts the schematic of an ESP. The ESP has some advantages that make it a popular choice. The pump can produce high liquid volumes with relatively high efficiency and low maintenance. Furthermore, it is well suited for use in various locations, from urban environments to offshore installations and in deviated wells. However, the installation of an ESP requires a reliable power source and the presence of gas and materials like sand can compromise its operation.

The mathematical model for ESPs considered here is based on the model developed by Statoil (now Equinor) in [51] with additional equations from [52] for viscosity modeling. The system model consists of an ESP and a production choke valve. The principles of ESP operation are relatively simple. The pressure gradient imposed by the difference between the reservoir pressure p_r and the well bottomhole pressure p_{bh}, $(p_r - p_{bh})$, induces the inflow q_r of a mixture of fluids (oil, water, and possibly gas) from the reservoir into the well. It reaches the ESP pump, which increases the pressure gradient by regulating the pump frequency f, thereby lifting the production to the top side. The pressure upstream of the production choke is the wellhead pressure p_{wh}, regulated by the choke opening z to ensure the pressure balance with the fixed manifold pressure p_m. An operator can control the ESP frequency f and the production choke opening z to reach a desired production target, typically achieved by tracking a reference for the bottomhole pressure informed by the optimization system.



(a) **5 layers** of 20 neurons each (left: k_1 and k_2 perturbed; right: initial condition perturbed)



(b) **8 layers** of 20 neurons each (left: k_1 and k_2 perturbed; right: initial condition perturbed)

Figure 16: Sensitivity to modeling errors (left) and to initial conditions (right) for the MPC of the four-tank system with PINC net as the model. For each plot, 151 runs of the MPC algorithm using a trained PINC net of 5 layers (top plots) and 8 layers (bottom plots) are executed. The resulting IAEs between the references (as in Fig. 14) and the controlled signals h_1 and h_2 are computed and shown in a histogram. The initial conditions are $h_1 = h_2 = 9$, which is the middle point of the allowed interval.

The model assumes constant fluid properties to keep the controller design relatively simple [51].

The ESP dynamic model considers reservoir inflow, production pipe, ESP, and production choke. Despite neglecting effects related to gas production and viscosity variation, the model can represent well dynamics quite accurately [51]. The system has three states: bottomhole pressure p_{bh}, wellhead pressure p_{wh}, and average flow rate q. The differential equations are as follows:

$$\dot{p}_{bh} = \frac{\beta_1}{V_1}(q_r - q) \tag{19a}$$

$$\dot{p}_{wh} = \frac{\beta_2}{V_2}(q - q_c) \tag{19b}$$

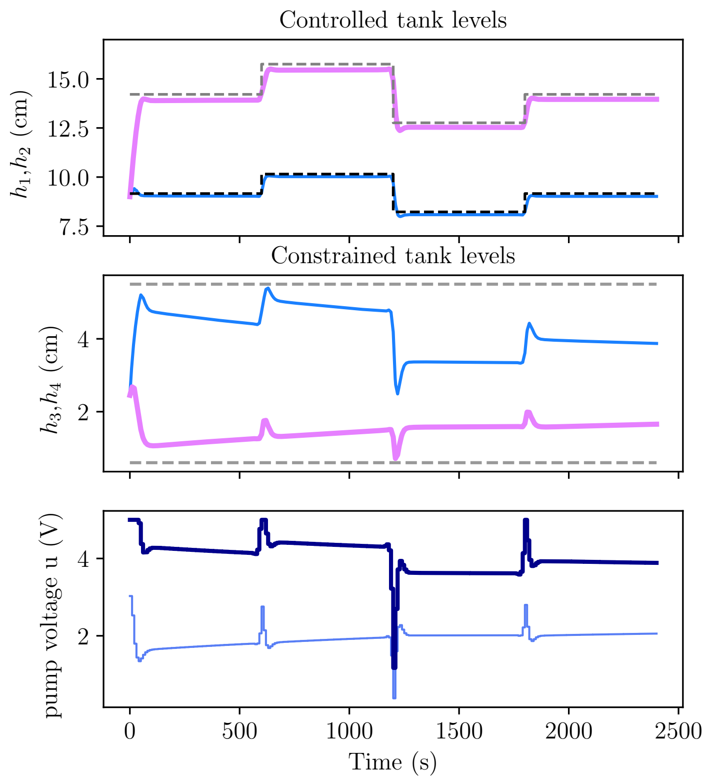


Figure 17: Control of the four-tank system with PINC as in Fig. 14, but with maximum perturbation of 5% in k_1 and k_2 parameters of the model being controlled. Initial conditions as in Fig. 16, that is, $h_1 = h_2 = 9$. Control results were adequate even though the IAE was 1022, which is in the tail of the histogram from Fig. 16(a) left plot.

$$\dot{q} = \frac{1}{M}(p_{bh} - p_{wh} - \rho g h_w - \Delta p_f + \Delta P_p) \tag{19c}$$

where Δp_f is the pressure loss due to friction, ΔP_p is pressure gain from the ESP, h_w is the total vertical length of the well, ρ is the density of the produced fluid, and g is the gravitational acceleration constant. The differential equations come with a set of constraints known as algebraic equations involving variables and parameters. Hence, the ESP model is a Differential Algebraic Equation (DAE) system. According with the DAE system, the influence of the pump frequency f on the ESP pressure gain ΔP_p is a highly nonlinear function. Similarly, the flow q_c through the choke is governed by a nonlinear function involving the bottomhole pressure, manifold pressure, and choke opening. To keep the presentation brief, a complete description of the ESP model's variables, parameters, and algebraic equations is found in Appendix A.

4.3.2. PINC for ESP

The inner continuous interval for PINC is $T = \frac{1}{11}$ s. The ranges for the randomly generated control signals and initial conditions are as follows: $f \in [35, 65]$ Hz and $z \in [0.1, 1]$;

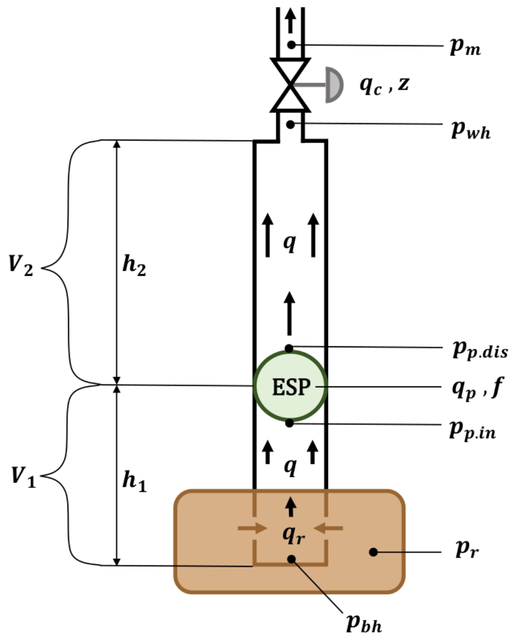


Figure 18: Schematic of an electric submersible pump. The pressure gradient imposed by the difference between the reservoir pressure p_r and the well bottomhole pressure p_{bh}, $(p_r - p_{bh})$, induces the inflow q_r of fluids from the reservoir into the well. The ESP lifts the fluids to the topside by adding energy regulated by the pump frequency f, which generates a pressure gain. The wellhead pressure p_{wh} corresponds to the pressure upstream of the production choke, regulated by the choke opening z to ensure pressure balance with the fixed manifold pressure p_m.

and $p_{bh} \in [70 \times 10^5, 75 \times 10^5]$ bar, $p_{wh} \in [25 \times 10^5, 35 \times 10^5]$ bar, $q \in \left[\frac{30}{3600}, \frac{50}{3600}\right]$ m^3/h. We performed a Bayesian search using Optuna [53] over three hyperparameters: the number of collocation points $N_f \in [20000, 40000]$, the number of layers (from 3 to 6), and the number of neurons per layer (from 20 to 50). The following experiments consider a PINC net with 4 layers of 49 neurons each, whose training employed the settings $N_t = 10,000$, $N_f = 32,631$, $\lambda = 1$, $K_1 = 10,000$, and $K_2 = 20,000$. Besides, the inputs t, $\mathbf{y}(0)$ and $\mathbf{u}$ to the network were normalized before performing signal propagation through the network and the residuals $\mathcal{F}(\cdot)$ employed denormalized output predictions $\mathbf{y}$ as well as inputs (t, $\mathbf{y}(0)$ and $\mathbf{u}$).

The PINC prediction for the ESP system on the test input can be seen in Fig. 19 for 2.64 s, where the control signal corresponding to the production choke opening z, starting at 0.6, is changed four times to the arbitrary values 1, 0.5, 0.73, 0.61, and 0.88, respectively. The pump frequency was kept fixed at $f = 57$ Hz for this particular simulation.

The average prediction time of the trained PINC over 50 simulations was 0.025 s with standard deviation of 0.007 s, while the RK numerical simulation took 0.262 s on average with standard deviation of 0.02 s (Table 2, column "Prediction"). Thus, our PINC can be at least one order of magnitude faster the RK method, without considering further optimizations of the PINC inference time (e.g., using BLAS libraries).

4.3.3. PINC Control

We address the control problem of following a reference signal for the bottom-hole pres-

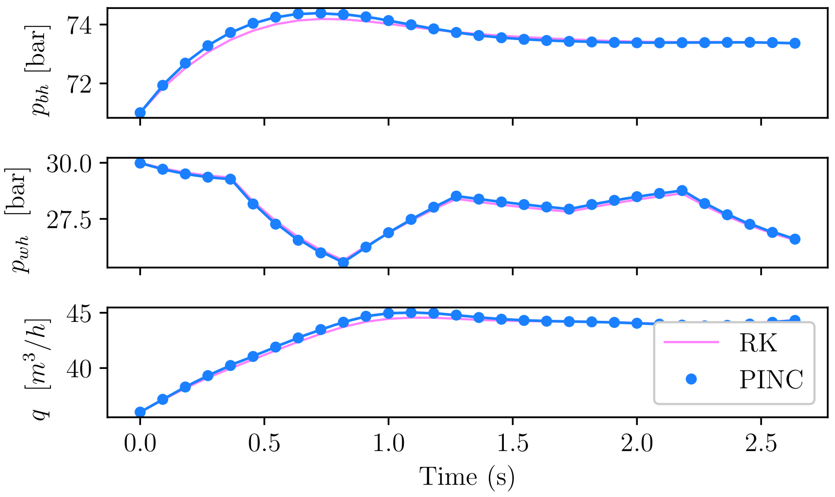


Figure 19: Long-range autoregressive PINC prediction for the ESP system' states: bottom-hole pressure, wellhead pressure of the well, and the production flow (from top to bottom). The production choke opening z (not shown) is manipulated to change five times to arbitrary values every $4T$ s (0.45 s) during this simulation, while the pump frequency f is kept constant at 57 Hz. The prediction is autoregressive every T s or at every blue dot. For the points in the continuous interval between any two consecutive blue dots, the PINC prediction for the states is done just by changing the time input t and setting the initial condition as the terminal condition of the previous interval.

Table 2: Results from ESP simulation and control experiments.

	Prediction		Control		
	RMSE	time (s)	RMSE (bar)	IAE (bar)	time (s)
PINC	0.1569	$\mathbf{0.025 \pm 0.007}$	2.44 ± 1.49	198.1 ± 134	36 ± 8.5
RK	–	0.262 ± 0.02	–	–	–

sure of the ESP-lifted well, considering both the pump frequency $f \in [35, 65]$ Hz and the production choke opening $z \in [0.1, 1]$ as manipulated variables for the controller. The ranges depicted are the operational ranges considered as constraints by the NMPC. Since there are two degrees of freedom, even if the reference signal for the bottom-hole pressure is followed, the closed-loop system may behave erratically for other involved variables. Therefore, for the NMPC of the ESP, considering Problem (12), we add a penalization on the wellhead pressure p_{wh} increment in the cost function. We also add rate limiting constraints for $|\Delta f| \leq 6$ Hz and $|\Delta z| \leq 0.2$ on the manipulated variables.

We configured the controller to have a sampling time $T_s = 3/12$ s [2], a control horizon and a prediction horizon of size 7 ($N_1 = 1$, $N_2 = 7$, $N_u = N_2$) , and $\mathbf{Q} = 1$ and $\mathbf{R} = \mathbf{I}$. The reference signal $p_{bh,ref}$ is a random stair signal that has a value between 65 and 85 bar. The

[2]We did not have to retrain the network, since we can perform the self-loop 3 times with a PINC trained with $T = 1/11$ s when $T_s = 3/12$ s.

simulation runs for 200 time steps (50 s), with the signal changing at a minimum period of 10 time steps.

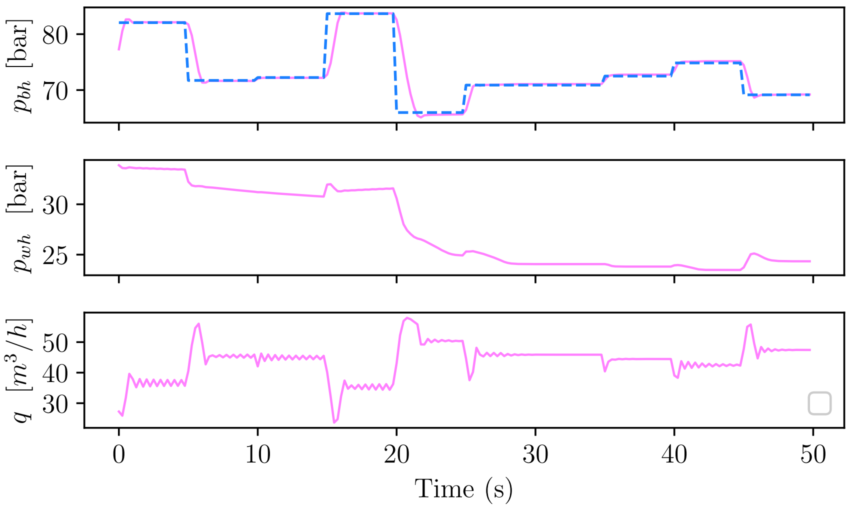


Figure 20: Result of the NMPC for the ESP system with the PINC as the prediction model. The reference signal (dashed line) was chosen at random within the interval $p_{bh,ref} \in [65, 85]$ bar. The first plot shows the bottom-hole pressure (p_{bh}), the second plot depicts the wellhead pressure of the well (p_{wh}), and the third plot showcases the well production flow (q).

Figure 20 depicts the state variables of the ESP, namely p_{bh}, p_{wh}, and q along the simulation, with p_{bh} being plotted alongside its associated reference signal. The topmost subplot shows that the closed-loop system can smoothly follow the reference signal, no matter how large the difference between the current and previous value. Notice that the wellhead pressure (second plot) is not quite constant due to the lack of a tracking reference and corresponding error penalization in the cost function. Nevertheless, the wellhead pressure is sufficiently smooth due to the increment penalization introduced in the cost function. The behavior of the production flow q is a direct consequence of the settings of the two previous variables (see Eq. 19), showing a behavior that converges to a constant value. The initial conditions for this experiment were $f = 45\,\text{Hz}$, $z = 0.7$, $p_{bh} = 75\,\text{bar}$, $p_{wh} = 35\,\text{bar}$, and $q = 40\,\text{m}^3/\text{h}$. Figure 21 gives plots for both manipulated variables, f and z, along the same simulation, showing that the controller manipulates the pump frequency more often than the choke opening.

Table 2 shows results for 10 repetitions of 200 timesteps of the ESP control experiment, with the aforementioned settings and same initial conditions, but considering randomly generated reference signals. The average and standard deviation of the IAE and the RMSE are computed for the p_{bh} tracking error, as well as the average execution time for the whole control experiment.

Although the experiments have a high variance, the simulations are within the expected range of performance since the mean value of the IAE is close to 200, which is the number of simulated time steps . This good performance is corroborated by the RMSE, which achieved

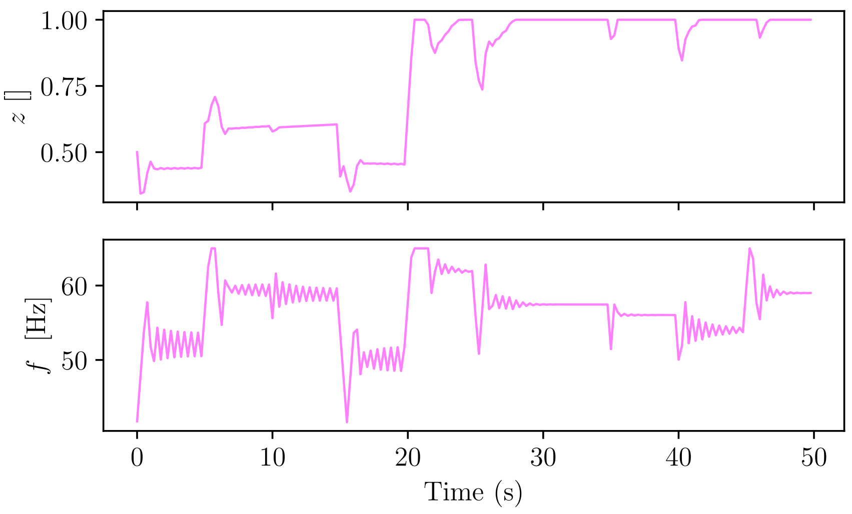


Figure 21: Plot of the choke opening z (first subplot) and pump frequency f (second subplot) during the simulation of the control.

a low mean error. The execution time is also reasonable for an NMPC that computes predictions using a static neural network.

As with the other examples, we tried employing the NMPC directly on the DAE model of the ESP using the Runge-Kutta solution method. However, the NMPC was not able to converge. The inability of the NMPC to converge might be attributed to the fact that the ESP model has very complicated nonlinearities (see Appendix A) that are challenging for numerical optimization algorithms, such as the square root, with a derivative approaching infinity as its argument draws close to zero, and the max function, which is non-smooth.

4.4. Discussion

PINC solves a family of ODE for variable initial condition and control inputs, becoming "time independent." To our knowledge, this was the first work to propose the architecture for control tasks, particularly for NMPC applications. Using PINC in NMPC instead of solving ODEs and DAEs in the optimization process brings some advantages. PINC is a smooth and differentiable function of the inputs, simplifying the numerical computations that otherwise would be needed to solve a DAE system, for instance, which may involve solving implicit algebraic equations and complex functions such as square roots, fractional exponents, and ratios. This was shown in the previous section, where the control of the ESP was made feasible with PINC, while the corresponding RK model used in MPC was unfeasible because of the resulting ill-conditioned formulation of the optimization problem with the DAE as a model.

One of the main obstacles of having a fully effective simulation from a PINC network is the long training time of such networks. Nonetheless, this is a common issue in most proposals dealing with deep learning, especially PINNs. Preliminary work in identifying more complex plants (e.g., in the oil and gas industry) shows that skip connections [54, 55] can help the training of deep networks by helping to backpropagate the gradient to the deepest layers during training, further improving the precision of the final trained model. It is worth noting that, after training a deep PINC model, it can predict directly any state in the range $[0, T]$ without requiring integration with intermediate points as in numerical simulation methods.

Besides, we noticed that a precise optimization algorithm towards the end of the training (e.g., L-BFGS) is essential in obtaining an accurate model. Furthermore, challenges to learning PINNs can arise from discontinuities in the ODE equations that model the plant, such as the presence of the max operator. Moreover, the random initialization of the weights of neural networks may cause different results and render invalid arguments to functions such as the square root if present in the ODE equations. Notice that some fixes or workarounds can be applied in these cases.

5. Conclusion

We have proposed a new framework that makes Physics-Informed Neural Networks (PINNs) amenable to control methods, such as MPC, opening a wide range of application opportunities. This Physics-Informed Neural Nets-based Control (PINC) approach allows a PINN to work for arbitrary longer-range time intervals that are not fixed beforehand at training stage, doing so without severe prediction degradation. In practice, PINC extends PINNs to work with MPC applications. In control applications, this framework (a) provides a way to identify a system by integrating collected data from a plant with a priori expert knowledge in the form of ordinary differential equations; (b) can simulate differential equations faster than numerical solution methods, making PINNs more appealing to real-time control applications. This speed advantage will probably increase as more complex applications are tackled, as already verified when modeling PDEs with conventional PINNs. PINC-based applications have also reported an increase in speed of at least one order of magnitude [56], which is similar to the reduction in prediction time we reported in the ESP experiment. Additional optimization of the network inference code using BLAS libraries could further reduce the network's inference time. Although only initial conditions were used as actual training data, additional sparse data will likely make the training of PINC nets much faster.

In future work, we intend to extend the framework to systems described by Differential-Algebraic Equations (DAEs) and PDEs, and systems for which prior knowledge is uncertain (unknown parameters), as well as apply PINC to industrial control problems such as in the oil and gas industry, for which some prior knowledge of ODEs is known in addition to highly noisy or sparse data. We expect that the reduction in the computational burden in using PINC for control scenarios will be even more relevant compared to the numerical solution approach as the model becomes increasingly more complex or, in the case of models described by PDEs [57]. Finally, we envision that the application of system identification in an industrial setting will expand if we use complementary sources of information for training deep networks, that is, by using physical laws and historical sparse data, making feasible a wide range of previously unsolved applications in systems and control.

Acknowledgments

This work was funded in part by CNPq (Grant 308624/2021-1) and FAPESC (Grant 2021TR2265).

Statements

Data Availability Statement

All data used in this study can be reproduced by anyone using the numerical simulation of Ordinary Differential Equations provided in the paper.

Competing Interests

The authors have no competing interests to declare that are relevant to the content of this article.

Appendix A. ESP Algebraic Equations

The algebraic equations associated with the dynamic equations of the ESP model are presented below according to their properties.

- Flow equations:

$$q_r = PI(p_r - p_{bh}) \tag{A.1a}$$

$$q_c = C_c\, z\sqrt{p_{wh} - p_m} \tag{A.1b}$$

- Friction equations:

$$\Delta p_f = F_1 + F_2 \tag{A.2a}$$

$$F_i = 0.158\, \frac{\rho L_i q^2}{D_i A_i^2} \left(\frac{\mu}{\rho D_i q}\right)^{\frac{1}{4}} \tag{A.2b}$$

- ESP equations:

$$\Delta p_p = \rho g H \tag{A.3a}$$

$$H = C_H(\mu)\left(c_0 + c_1\left(\frac{q}{C_Q(\mu)}\,\frac{f_0}{f}\right) - c_2\left(\frac{q}{C_Q(\mu)}\,\frac{f_0}{f}\right)^2\left(\frac{f}{f_0}\right)^2\right) \tag{A.3b}$$

$$c_0 = 9.5970 \cdot 10^2 \tag{A.3c}$$

$$c_1 = 7.4959 \cdot 10^3 \tag{A.3d}$$

$$c_2 = 1.2454 \cdot 10^6 \tag{A.3e}$$

where $C_H(\mu)$ and $C_Q(\mu)$ are 4^{th} order polynomial functions on the viscosity μ with coefficients defined in [52].

Table A.3: ESP Model variables

Control inputs	
f	ESP frequency
z	Choke valve opening

ESP data	
p_m	Production manifold pressure
p_{wh}	Wellhead pressure
p_{bh}	Bottomhole pressure
$p_{p,in}$	ESP intakepressure
$p_{p,dis}$	ESP discharge pressure
p_r	Reservoir pressure

Parameters from fluid analysis and well tests	
q	Average liquid flow rate
q_r	Flow rate from reservoir into the well
q_c	Flow rate through production choke

The state and algebraic variables involved in the ESP model appear in Table A.3. The parameters used in this model are based on the parameters from [52]. Table A.4 presents the parameters which consist of fixed values such as well dimensions and ESP parameters, and parameters found from analysis of fluid such as bulk modulus β_i and density ρ [52]. Parameters such as the well productivity index PI, viscosity μ, and manifold pressure p_m are assumed constant.

References

[1] L. Grüne, J. Pannek, Nonlinear Model Predictive Control: Theory and Algorithms, Springer, 2011.

[2] E. F. Camacho, C. Bordons, Model Predictive Control, Springer Science & Business Media, 2013.

[3] J. P. Jordanou, E. Camponogara, E. A. Antonelo, M. A. S. Aguiar, Nonlinear model predictive control of an oil well with echo state networks, IFAC-PapersOnLine 51 (2018) 13–18. doi:10.1016/j.ifacol.2018.06.348.

[4] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, B. Açıkmeşe, Model predictive control in aerospace systems: Current state and opportunities, Journal of Guidance, Control, and Dynamics 40 (2017) 1541–1566. doi:10.2514/1.G002507.

Table A.4: ESP Model Parameters

Well dimensions and other known constants			
g	Gravitational acceleration constant	9.81	m/s^2
C_c	Choke valve constant	$2 \cdot 10^{-5}$	*
A_1	Cross-section area of pipe below ESP	0.008107	m^2
A_2	Cross-section area of pipe above ESP	0.008107	m^2
D_1	Pipe diameter below ESP	0.1016	m
D_2	Pipe diameter above ESP	0.1016	m
h_1	Height from reservoir to ESP	200	m
h_w	Total vertical distance in well	1000	m
L_1	Length from reservoir to ESP	500	m
L_2	Length from ESP to choke	1200	m
V_1	Pipe volume below ESP	4.054	m^3
V_2	Pipe volume above ESP	9.729	m^3

ESP data			
f_0	ESP characteristics reference freq.	60	Hz
I_{np}	ESP motor nameplate current	65	A
P_{np}	ESP motor nameplate power	$1.625 \cdot 10^5$	W

Parameters from fluid analysis and well tests			
β_1	Bulk modulus below ESP	$1.5 \cdot 10^9$	Pa
β_2	Bulk modulus below ESP	$1.5 \cdot 10^9$	Pa
M	Fluid inertia parameter	$1.992 \cdot 10^8$	kg/m^4
ρ	Density of produced fluid	950	kg/m^3
P_r	Reservoir pressure	$1.26 \cdot 10^7$	Pa

Parameters assumed to be constant			
PI	Well productivity index	$2.32 \cdot 10^{-9}$	$m^3/s/$Pa
μ	Viscosity of produced fluid	0.025	$Pa \cdot s$
P_m	Manifold pressure	20	Pa

[5] T. P. Nascimento, C. E. T. Dórea, L. M. G. Gonçalves, Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach, International Journal of Advanced Robotic Systems 15 (2018). doi:10.1177/1729881418760461.

[6] 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, Journal of Computational Physics 378 (2019) 686–707. doi:0.1016/j.jcp.2018.10.045.

[7] R.-F. Zhang, S. Bilige, Bilinear neural network method to obtain the exact analytical solutions of nonlinear partial differential equations and its application to p-gBKP equation, Nonlinear Dynamics 95 (2019) 3041–3048. doi:10.1007/s11071-018-04739-z.

[8] R.-F. Zhang, M.-C. Li, Bilinear residual network method for solving the exactly explicit solutions of nonlinear evolution equations, Nonlinear Dynamics 108 (2022) 521–531. doi:10.1007/s11071-022-07207-x.

[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).

[10] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, Journal of Computational Physics 394 (2019) 56–81. doi:10.1016/j.jcp.2019.05.024.

[11] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics 375 (2018) 1339–1364. doi:10.1016/j.jcp.2018.08.029.

[12] X. Meng, Z. Li, D. Zhang, G. E. Karniadakis, PPINN: Parareal physics-informed neural network for time-dependent PDEs, Computer Methods in Applied Mechanics and Engineering 370 (2020) 113250. doi:10.1016/j.cma.2020.113250.

[13] L. Yang, X. Meng, G. E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, Journal of Computational Physics 425 (2021) 109913. doi:10.1016/j.jcp.2020.109913.

[14] G. Pang, G. E. Karniadakis, Physics-informed learning machines for partial differential equations: Gaussian processes versus neural networks, Kevrekidis P., Cuevas-Maraver J., Saxena A. (eds) Emerging Frontiers in Nonlinear Science. Nonlinear Systems and Complexity 32 (2020) 323–343. doi:10.1007/978-3-030-44992-6_14.

[15] P. Stinis, Enforcing constraints for time series prediction in supervised, unsupervised and reinforcement learning, in: Proceedings of the AAAI 2020 Spring Symposium on

Combining Artificial Intelligence and Machine Learning with Physical Sciences, volume 2587, 2020. URL: http://ceur-ws.org/Vol-2587/article_5.pdf.

[16] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (PIELM)–a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118. doi:10.1016/j.neucom.2019.12.099.

[17] Z. Xiang, W. Peng, X. Liu, W. Yao, Self-adaptive loss balanced physics-informed neural networks, Neurocomputing 496 (2022) 11–34. doi:10.1016/j.neucom.2022.05.015.

[18] L. F. Nazari, E. Camponogara, L. O. Seman, Physics-informed neural networks for modeling water flows in a river channel, IEEE Transactions on Artificial Intelligence (2023). doi:10.1109/TAI.2022.3200028.

[19] E. Olivares, H. Ye, A. Herrero, B. A. Nia, Y. Ren, R. P. Donovan, et al., Applications of information channels to physics-informed neural networks for WiFi signal propagation simulation at the edge of the industrial internet of things, Neurocomputing 454 (2021) 405–416. doi:10.1016/j.neucom.2021.04.021.

[20] L. T. Biegler, Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes, SIAM, Philadelphia, 2010.

[21] J. P. Jordanou, E. A. Antonelo, E. Camponogara, Echo state networks for practical nonlinear model predictive control of unknown dynamic systems, IEEE Transactions on Neural Networks and Learning Systems 33 (2022) 2615–2629. doi:10.1109/TNNLS.2021.3136357.

[22] J. G. Ortega, E. Camacho, Mobile robot navigation in a partially structured static environment, using neural predictive control, Control Engineering Practice 4 (1996) 1669–1679. doi:0.1016/S0967-0661(96)00184-0.

[23] L. Cavagnari, L. Magni, R. Scattolini, Neural network implementation of nonlinear receding-horizon control, Neural Computing & Applications 8 (1999) 86–92. doi:10.1007/s005210050010.

[24] B. M. Åkesson, H. T. Toivonen, A neural network model predictive controller, Journal of Process Control 16 (2006) 937–946. doi:10.1016/j.jprocont.2006.06.001.

[25] Y. Pan, J. Wang, Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks, IEEE Transactions on Industrial Electronics 59 (2012) 3089–3101. doi:10.1109/TIE.2011.2169636.

[26] M. Ławryńczuk, Computationally Efficient Model Predictive Control Algorithms, Springer International Publishing, 2014.

[27] E. Terzi, T. Bonetti, D. Saccani, M. Farina, L. Fagiano, R. Scattolini, Learning-based predictive control of the cooling system of a large business centre, Control Engineering Practice 97 (2020) 104348. doi:10.1016/j.conengprac.2020.104348.

[28] J. Witt, H. Werner, Approximate model predictive control for nonlinear multivariable systems, in: T. Zheng (Ed.), Model Predictive Control, 2010, pp. 141–166. doi:10.5772/46955.

[29] O. Nelles, Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models, 1 ed., Springer, Berlin, 2001.

[30] M. Hertneck, J. Köhler, S. Trimpe, F. Allgöwer, Learning an approximate model predictive controller with guarantees, IEEE Control Systems Letters 2 (2018) 543–548. doi:10.1109/LCSYS.2018.2843682.

[31] E. A. Antonelo, E. Camponogara, L. O. Seman, E. R. de Souza, J. P. Jordanou, J. F. Hubner, Physics-informed neural nets for control of dynamical systems, 2021. URL: https://arxiv.org/abs/2104.02556. doi:10.48550/ARXIV.2104.02556.

[32] H. Zhai, T. Sands, Physics-informed deep operator control: Controlling chaos in Van der Pol oscillating circuits, arXiv preprint arXiv:2112.14707 (2021).

[33] X.-Y. Liu, J.-X. Wang, Physics-informed dyna-style model-based deep reinforcement learning for dynamic control, Proceedings of the Royal Society A 477 (2021) 20210618. doi:10.1098/rspa.2021.0618.

[34] G. Gokhale, B. Claessens, C. Develder, Physics informed neural networks for control oriented thermal modeling of buildings, Applied Energy 314 (2022) 118852. doi:10.1016/j.apenergy.2022.118852.

[35] D. P. Kingma, J. Ba, ADAM: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[36] G. Andrew, J. Gao, Scalable training of L1-regularized log-linear models, in: Proceedings of the 24th International Conference on Machine Learning, ICML'07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 33–40. doi:10.1145/1273496.1273501.

[37] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer, 2006.

[38] J. E. Normey-Rico, E. F. Camacho, Control of Dead-time Processes, Springer London, 2007.

[39] J. Nocedal, S. J. Wright, Numerical Optimization, second ed., Springer, New York, NY, USA, 2006.

[40] P. E. Gill, W. Murray, M. A. Saunders, SNOPT: An SQP algorithm for large-scale constrained optimization, SIAM Review 47 (2005) 99–131. doi:10.1137/S0036144504446096.

[41] A. Wächter, L. T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, Mathematical Programming 106 (2006) 25–57. doi:10.1007/s10107-004-0559-y.

[42] A. Iserles, A First Course in the Numerical Analysis of Differential Equations, Cambridge University Press, 1996.

[43] W. C. Schultz, V. C. Rideout, Control system performance measures: Past, present, and future, IRE Transactions on Automatic Control AC-6 (1961) 22–35. doi:10.1109/TAC.1961.6429306.

[44] H. Y. Hafeez, C. E. Ndikilar, S. Isyaku, Analytical study of the Van der Pol equation in the autonomous regime, Progress in Physics 11 (2015) 252–255.

[45] J. Andersson, J. Åkesson, M. Diehl, Dynamic optimization with CasADi, in: Proceedings of the IEEE Conference on Decision and Control, 2012, pp. 681–686. doi:10.1109/CDC.2012.6426534.

[46] K. Johansson, The quadruple-tank process: A multivariable laboratory process with an adjustable zero, IEEE Transactions on Control Systems Technology 8 (2000) 456–465. doi:10.1109/87.845876.

[47] A. S. M. Brandão, Controle Preditivo com Geração de Código: Um Estudo Comparativo, Master's thesis, Universidade Federal de Santa Catarina, 2018.

[48] R. G. Brown, P. Y. C. Hwang, Introduction to Random Signals and Applied Kalman Filtering, John Wiley & Sons, 1992.

[49] P. Cheng, M. Chen, V. Stojanovic, S. He, Asynchronous fault detection filtering for piecewise homogenous Markov jump linear systems via a dual hidden Markov model, Mechanical Systems and Signal Processing 151 (2021) 107353. doi:10.1016/j.ymssp.2020.107353.

[50] T. Wei, X. Li, V. Stojanovic, Input-to-state stability of impulsive reaction–diffusion neural networks with infinite distributed delays, Nonlinear Dynamics 103 (2021) 1733–1755. doi:10.1007/s11071-021-06208-6.

[51] A. Pavlov, D. Krishnamoorthy, K. Fjalestad, E. Aske, M. Fredriksen, Modelling and model predictive control of oil wells with electric submersible pumps, in: IEEE Conference on Control Applications (CCA), 2014, pp. 586–592. doi:10.1109/CCA.2014.6981403.

[52] B. J. Binder, A. Pavlov, T. A. Johansen, Estimation of flow rate and viscosity in a well with an electric submersible pump using moving horizon estimation, volume 48, IFAC-PapersOnLine, 2015, pp. 140–146. doi:10.1016/j.ifacol.2015.08.022.

[53] 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 & Data Mining, 2019, pp. 2623–2631. doi:10.1145/3292500.3330701.

[54] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, Deeply-supervised nets, in: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, PMLR, 2015, pp. 562–570.

[55] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

[56] J. Nicodemus, J. Kneifl, J. Fehr, B. Unger, Physics-informed neural networks-based model predictive control for multi-link manipulators, IFAC-PapersOnLine 55 (2022) 331–336. doi:10.1016/j.ifacol.2022.09.117.

[57] R.-F. Zhang, M.-C. Li, A. Cherraf, S. R. Vadyala, The interference wave and the bright and dark soliton for two integro-differential equation by using BNNM, Nonlinear Dynamics 111 (2019) 8637–8646. doi:10.1007/s11071-023-08257-5.