Physics-informed Neural Networks-based Model Predictive Control for Multi-link Manipulators [⋆]

Jonas Nicodemus [*] Jonas Kneifl [**] Jörg Fehr [**] Benjamin Unger [*]

[*] *Stuttgart Center for Simulation Science (SC SimTech), University of Stuttgart, Universitätsstr. 32, 70569 Stuttgart, Germany (e-mail: {jonas.nicodemus,benjamin.unger}@simtech.uni-stuttgart.de)*
[**] *Institute of Engineering and Computational Mechanics, University of Stuttgart, Pfaffenwaldring 9, 70569 Stuttgart, Germany (e-mail: {joerg.fehr,jonas.kneifl}@itm.uni-stuttgart.de).*

Abstract: We discuss *nonlinear model predictive control* (NMPC) for multi-body dynamics via physics-informed machine learning methods. *Physics-informed neural networks* (PINNs) are a promising tool to approximate (partial) differential equations. PINNs are not suited for control tasks in their original form since they are not designed to handle variable control actions or variable initial values. We thus present the idea of enhancing PINNs by adding control actions and initial conditions as additional network inputs. The high-dimensional input space is subsequently reduced via a sampling strategy and a zero-hold assumption. This strategy enables the controller design based on a PINN as an approximation of the underlying system dynamics. The additional benefit is that the sensitivities are easily computed via automatic differentiation, thus leading to efficient gradient-based algorithms. Finally, we present our results using our PINN-based MPC to solve a tracking problem for a complex mechanical system, a multi-link manipulator.

Keywords: Physics-informed Machine Learning, Model Predictive Control, Surrogate Model, Mechanical System, Real-time Control

1. INTRODUCTION

Model predictive control (MPC) is a flexible and intuitive control scheme that lets us impose constraints and helps to operate complex systems optimally. The major challenge for MPC is the repetitive solution of an optimal control problem. Even with today's computing power, efficient model representation to be real-time capable remains the bottleneck. This issue is especially prominent in systems with a fast dynamic, like robotic manipulators, where operation speed relates to increased productivity.

In this work, we study a tracking problem for a multi-link manipulator, see section 2 for further details, with an a-priori unknown tracking trajectory. In this scenario, standard linearization strategies for the nonlinear dynamics, which are used to speed up the computation, cannot be implemented without further challenges. Instead, a repetitive numerical evaluation of the underlying nonlinear dynamics is required. Since the time required by standard time integration schemes may pose a critical constraint during the solution of the optimal control problem, we propose replacing the time-integration with a *machine learning* (ML) approach. Since standard ML techniques typically require an extensive training data set and cannot compete with state-of-the-art time-integrators (Otness et al., 2021), we propose a physics-informed approach, see Karniadakis et al. (2021) for a recent overview, thus exploiting the underlying known physical law during the training process. More precisely, we replace the nonlinear dynamics with a *physics-informed neural network* (PINN), initially introduced by Raissi et al. (2019). In addition, we exploit the efficient computation of partial derivatives of the *deep neural network* (DNN) via automatic differentiation, see for instance Baydin et al. (2018).

Our main results are the following:

(1) Following ideas presented by Antonelo et al. (2021), we detail in section 3.3 how to replace the nonlinear dynamics with a PINN approximation in the context of *nonlinear model predictive control* (NMPC). Due to the efficient computation of the partial derivative of the PINN with respect to the controller, the optimal control problem can be solved efficiently with a gradient-based method, without any adjoint computations (for gradient-based methods) or acceleration strategies (for nonlinear programming methods).
(2) In section 4 we apply the strategy to a PowerCube serial robot (cf. Fehr et al. (2020)) and demonstrate that replacing the numerical time-integration of the nonlinear dynamics with a PINN speeds up the computation time while retaining a sufficient accuracy within the NMPC framework.

⋆ The authors acknowledge funding from the DFG under Germany's Excellence Strategy – EXC 2075 – 390740016 and are thankful for support by the Stuttgart Center for Simulation Science (SimTech).

The use of ML techniques in MPC is not new. For instance, in Åkesson and Toivonen (2006) and Hertneck et al. (2018) the controller itself is replaced by a neural network. In the context of a-priori unknown tracking trajectories, such an approach is either not feasible or requires learning in a high-dimensional space. Instead, similarly as in this work, the nonlinear dynamics are replaced by a recurrent neural network in Wu et al. (2021) or a PINN in Arnold and King (2021); Antonelo et al. (2021). However, Arnold and King (2021) use a different strategy for dealing with variable control inputs, and compared to Antonelo et al. (2021), we test our PINN-based MPC on a mechanical system that features faster dynamics than their examples.

Let us briefly outline the structure of the paper. First, we postulate the precise problem setting, including the PowerCube serial robot as sample application in section 2. A short review of NMPC and PINNs in sections 3.1 and 3.2, respectively, is followed by the PINN-based NMPC framework in section 3.3. We demonstrate the efficiency of the framework with a numerical example of multi-link manipulator in section 4.

Notation For a positive semidefinite matrix $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, we define the weighted (euclidean) seminorm

$$\|\cdot\|_{\boldsymbol{Q}}\colon \mathbb{R}^n \to \mathbb{R}_{\geq 0}, \qquad \boldsymbol{x} \mapsto \sqrt{\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x}}.$$

2. PROBLEM FORMULATION

On the time-interval $\mathbb{T} \subseteq \mathbb{R}$ we study control systems of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{1a}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{1b}$$

for some initial time $t_0 \in \mathbb{T}$ and $\boldsymbol{x}\colon \mathbb{T} \to \mathcal{X} \subseteq \mathbb{R}^n$, $\boldsymbol{x}_0 \in \mathbb{R}^n$, $\boldsymbol{u}\colon \mathbb{T} \to \mathcal{U} \subseteq \mathbb{R}^m$ the *state*, *initial value* and *control*, respectively. We assume that f is continuous and (locally) Lipschitz-continuous with respect to the state, such that the initial value problem (1) has a unique (weak) solution for each $\boldsymbol{u} \in L^\infty(\mathbb{T}, \mathcal{U})$, cf. Sonntag (1989, Thm. 54). The corresponding solution operator (also called *flow*) that maps the control and initial value to the solution at time $t \geq t_0$, is denoted with

$$\boldsymbol{\varphi}(t, \boldsymbol{u}, \boldsymbol{x}_0) = \boldsymbol{x}(t). \tag{2}$$

For our further presentation it will be beneficial to view the control system (1a) as the under-determined *differential-algebraic equation* (DAE)

$$0 = \boldsymbol{F}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t)) := \dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)). \tag{3}$$

Let us emphasize that, in principle, we could start with a descriptor system instead of the control system (1). However, for the sake of simplicity, we restrict ourselves to control systems of the form (1).

As an exemplary real-world application, we study a multi-link manipulator consisting of Schunk PowerCubes modules, as described in Fehr et al. (2020). Multi-link manipulators play an important role in the automation process of the manufacturing industry and are a heavily studied research topic in fields like control theory, see for instance Spong et al. (2020). The control problem of a multi-link manipulator is simple but still challenging. Thanks to previous works (Kargl, 2020), the dynamical model of the manipulator is already identified and simulation data is available for comparison. The schematic sketch of the here studied manipulator is shown in Fig. 1.

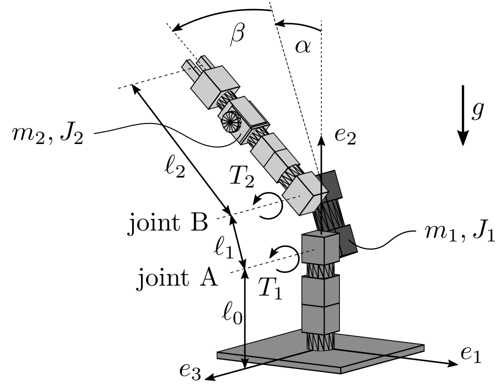


Fig. 1. Sketch of the PowerCube serial robot.

The resulting equations of motion can be read as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{k}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{B}\boldsymbol{u},$$

with the generalized coordinates $\boldsymbol{q} = [\alpha, \beta]^\top \in \mathbb{R}^2$, the nonsingular mass matrix $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{2\times 2}$, the vector of centrifugal, coriolis and gyroscopic forces $\boldsymbol{k}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^2$, the vector of applied forces $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^2$, the input matrix $\boldsymbol{B} \in \mathbb{R}^{2\times 2}$ and the input vector $\boldsymbol{u} \in \mathbb{R}^2$, which consists of the motor currents of the A and B revolute joint. Introducing $\boldsymbol{x} := [\boldsymbol{q}, \dot{\boldsymbol{q}}]^\top$ yields the first-order reformulation

$$\boldsymbol{0} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}(\boldsymbol{q}) \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}} \\ \ddot{\boldsymbol{q}} \end{bmatrix} - \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{k}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix} - \begin{bmatrix} 0 \\ \boldsymbol{B}\boldsymbol{u} \end{bmatrix},$$

which is in the form of (3), with $\boldsymbol{x} \in \mathbb{R}^4$.

3. METHODS

The practical implementation of a controller on a digital control unit typically requires a temporal discretization of the continuous-time control system (1). We follow Grüne and Pannek (2011) and introduce the time-grid

$$t_0 < t_1 < \ldots < t_N.$$

For simplicity we choose an equidistant time grid, i.e., $t_k = k\tau + t_0$ with constant sampling period $\tau > 0$. Introducing the shifted input $\boldsymbol{u}_k := \boldsymbol{u}(\cdot + k\tau)$ for $k = 0, 1, \ldots, N-1$ we thus obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{\varphi}(\tau, \boldsymbol{u}_k, \boldsymbol{x}_k) \tag{4}$$

with $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$ for $k = 0, 1, \ldots, N$. If we make an additional zero-hold assumption for the control input, i.e., $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u}_{k|(0,\tau)} \equiv 0$, then (4) resembles a discrete-time control system. In this case, by abuse of the notation, we use the symbol $\boldsymbol{u}_k$ both to denote the control function and its constant value on the interval $(0, \tau)$.

3.1 Nonlinear Model Predictive Control

Suppose we have a control system in the form of (4), with the state measured at discrete time instants t_k as described above, then the tracking problem is to find suitable control inputs $\boldsymbol{u}_k$ such that $\boldsymbol{x}_k$ follows a given reference $\boldsymbol{x}_k^{\mathrm{ref}}$ as good as possible. This problem can be solved by applying NMPC.

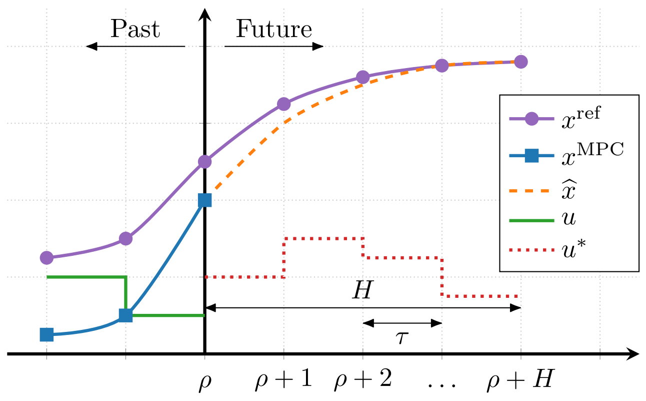


Fig. 2. Basic MPC scheme for the horizon width $H = 4$.

For this purpose, we introduce a cost function $\ell\colon \mathcal{X}\times\mathcal{X}\times\mathcal{U}\to\mathbb{R}_{\geq 0}$. In the quadratic case, this function may be chosen as

$$\ell(\boldsymbol{x}_k^{\mathrm{ref}}, \boldsymbol{x}_k, \boldsymbol{u}_k) = \|\boldsymbol{x}_k^{\mathrm{ref}} - \boldsymbol{x}_k\|_Q^2 + \|\boldsymbol{u}_k\|_R^2. \tag{5}$$

Following the NMPC scheme, visualized in Fig. 2, at each time instant $k=\rho$ the discrete-time optimal control problem for the moving horizon with horizon width H for the current time instant ρ is given by

$$\min_{\boldsymbol{u}_\rho,\dots,\boldsymbol{u}_{\rho+H-1}} \sum_{k=\rho}^{\rho+H-1} \ell(\boldsymbol{x}_k^{\mathrm{ref}}, \boldsymbol{x}_k, \boldsymbol{u}_k) \tag{6a}$$

$$\text{s.t.}\quad \boldsymbol{x}_{k+1} = \boldsymbol{\varphi}(\tau, \boldsymbol{u}_k, \boldsymbol{x}_k), \tag{6b}$$

$$\boldsymbol{u}_k\in\mathcal{U}, \boldsymbol{x}_k\in\mathcal{X}. \tag{6c}$$

Then, the optimal control input sequence $\boldsymbol{u}_k^*$ for $k=\rho,\dots,\rho+H-1$ for the current time instant ρ is obtained, by solving (6). From this sequence the first control input $\boldsymbol{u}_\rho^*$ is applied to the system until $t_{\rho+1}$, then the procedure repeats. The optimal control problem (6) is solved numerically either: (i) directly, via the Hamiltonian-Jacobi-Bellmann equation, (ii) via gradient-based methods, where the gradients are computed via the adjoint method, or as a (iii) nonlinear programming problem with methods such as sequential quadratic programming or the interior-point method, see for instance Nocedal and Wright (2006).

In many applications, $\boldsymbol{\varphi}$ is not available and a numerical approximation $\widehat{\boldsymbol{\varphi}}$ is used, for instance obtained via numerical time-integration. In the case of the tracking problem with a-priori known tracking trajectory, one can linearize the nonlinear system in an offline-step to speed up the computation in the online stage. This strategy is used for instance in Fehr et al. (2020) for the optimal control of the present robot manipulator. We emphasize that the reference trajectory needs to be known a-priori and the linearization introduces an additional source of error. We thus cannot apply this strategy and restrict ourselves to the fully nonlinear case.

3.2 Physics-Informed Neural Networks

If various solution trajectories of (1) are available, then DNNs can be used to approximate the solution map (2) via a supervised learning task. In more detail, a (data-based) loss-function based on the available solution trajectories is minimized with respect to so-called *weights* $\boldsymbol{\omega}\in\mathbb{R}^p$ such that the DNN $\widehat{\boldsymbol{\varphi}}$ approximates $\boldsymbol{\varphi}$, i.e.,

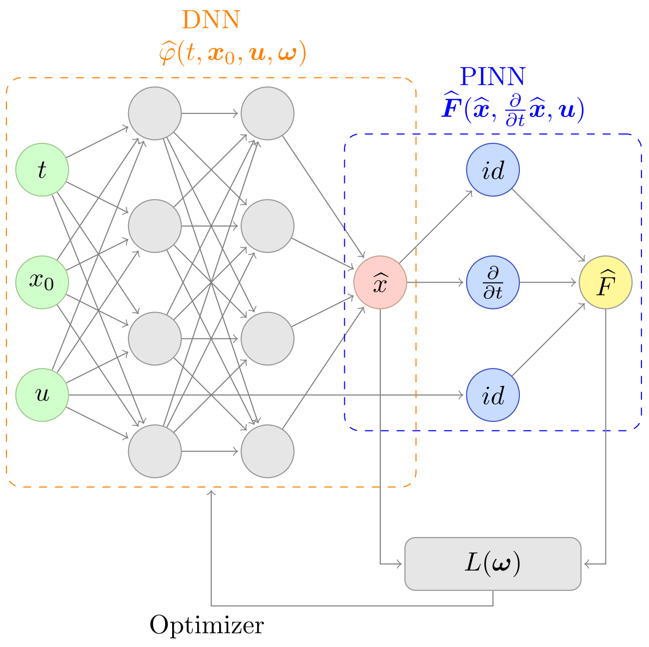

DNN
$\widehat{\varphi}(t, \boldsymbol{x}_0, \boldsymbol{u}, \boldsymbol{\omega})$
PINN
$\widehat{\boldsymbol{F}}(\widehat{\boldsymbol{x}}, \frac{\partial}{\partial t}\widehat{\boldsymbol{x}}, \boldsymbol{u})$
t
x_0
u
$\widehat{x}$
id
$\frac{\partial}{\partial t}$
id
$\widehat{F}$
$L(\boldsymbol{\omega})$
Optimizer

Fig. 3. PINN architecture.

$$\widehat{\boldsymbol{x}}(t) := \widehat{\boldsymbol{\varphi}}(t, \boldsymbol{x}_0, \boldsymbol{u}, \boldsymbol{\omega}) \approx \boldsymbol{\varphi}(t, \boldsymbol{x}_0, \boldsymbol{u}),$$

for all admissible $(t, \boldsymbol{x}_0, \boldsymbol{u})$. One strategy to achieve this, is to use existing data

$$\boldsymbol{x}_i := \boldsymbol{\varphi}(t_i, \boldsymbol{x}_{0,i}, \boldsymbol{u}_i), \qquad i = 1,\dots,N_{\mathrm{data}}$$

and minimize the mean-squared error loss-function

$$L_{\mathrm{data}}(\boldsymbol{\omega}) := \frac{1}{N_{\mathrm{data}}}\sum_{i=1}^{N_{\mathrm{data}}} \|\widehat{\boldsymbol{\varphi}}(t_i, \boldsymbol{x}_{0,i}, \boldsymbol{u}_i, \boldsymbol{\omega}) - \boldsymbol{x}_i\|^2.$$

The idea of PINNs, introduced in Raissi et al. (2019), is to add the differential equation to the loss function, as shown in Fig. 3, to robustify the network further and allow for a good approximation even in a data-poor regime. The actual PINN results from inserting the approximated flow $\widehat{\boldsymbol{\varphi}}$ and its derivative $\frac{\partial}{\partial t}\widehat{\boldsymbol{\varphi}}$, which can be computed via automatic differentiation, into the differential equation (3), yielding the residual

$$\widehat{\boldsymbol{F}}(t, \boldsymbol{x}_0, \boldsymbol{u}, \boldsymbol{\omega}) := \boldsymbol{F}(\widehat{\boldsymbol{\varphi}}(t, \boldsymbol{x}_0, \boldsymbol{u}, \boldsymbol{\omega}), \tfrac{\partial}{\partial t}\widehat{\boldsymbol{\varphi}}(t, \boldsymbol{x}_0, \boldsymbol{u}, \boldsymbol{\omega}), \boldsymbol{u}).$$

Note that this approach requires sufficiently smooth activation functions in the DNN, for instance the hyperbolic tangent. The loss function corresponding to the differential equation is then given by

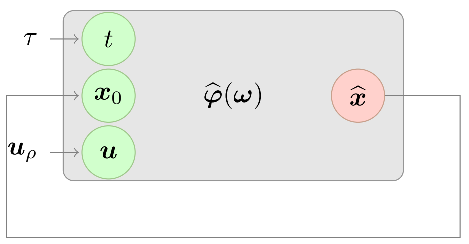
$$L_{\mathrm{phys}}(\boldsymbol{\omega}) := \frac{1}{N_{\mathrm{phys}}}\sum_{i=1}^{N_{\mathrm{phys}}} \|\widehat{\boldsymbol{F}}(t_i, \boldsymbol{x}_{0,i}, \boldsymbol{u}_i, \boldsymbol{\omega})\|^2,$$

where the residual is evaluated on a finite set of so-called *collocation points* $(t_i, \boldsymbol{x}_{0,i}, \boldsymbol{u}_i)$. Let us emphasize that the collocation points can be chosen arbitrarily without the need to solve the control system (1).

To ensure that both the data and the differential equation are approximated sufficiently well, we minimize the DNN with respect to the combined loss function.

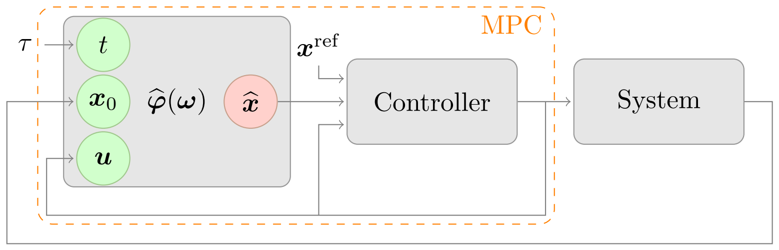
$$L(\boldsymbol{\omega}) = L_{\mathrm{data}}(\boldsymbol{\omega}) + L_{\mathrm{phys}}(\boldsymbol{\omega}).$$

In practical applications, it may be important to introduce a weighting of the two components of the loss function. For notational convenience, we assume that such a scaling is already implicitly encoded in $\widehat{\boldsymbol{F}}$.



τ
t
$\boldsymbol{x}_0$
$\boldsymbol{u}_\rho$
$\boldsymbol{u}$
$\widehat{\varphi}(\boldsymbol{\omega})$
$\widehat{\boldsymbol{x}}$

(a) PINN in self-loop prediction.



MPC
τ
t
$\boldsymbol{x}_0$
$\boldsymbol{u}$
$\widehat{\varphi}(\boldsymbol{\omega})$
$\widehat{\boldsymbol{x}}$
$\boldsymbol{x}^{\text{ref}}$
Controller
System

(b) PINN-based MPC in closed-loop.

Fig. 4. On the left PINN in self-loop prediction and on the right PINN-based MPC connected to a system.

3.3 PINN-based MPC

In principle, we could implement the PINN approach as presented in the previous section. Nevertheless, the specific control application results in two major challenges: First, to obtain accurate approximations, we have to sample the infinite-dimensional input space $L^\infty(\mathbb{T}, \mathcal{U})$. Second, while the PINN approximation is inherently smooth, the solution of the control problem (1) is typically not differentiable with respect to time. Although the universal approximation theorem guarantees arbitrarily accurate approximations of continuous functions, this normally comes with the cost of a large number of neurons.

To remedy these issues within the NMPC framework, we make the following observation: If $\boldsymbol{f}$ is sufficiently smooth and the zero-hold assumption is applied, then also $\boldsymbol{x}_{|(k\tau,(k+1)\tau)}$ is smooth. Moreover, within the interval $(k\tau, (k+1)\tau)$, the zero-hold assumption reduces the infinite-dimensional input space to the finite-dimension set $\mathcal{U} \subseteq \mathbb{R}^m$. We immediately arrive at the following strategy. Given an initial value $\boldsymbol{x}_k$ and values $\boldsymbol{u}_k$ for the control, train a PINN that is able to predict $\boldsymbol{x}_{k+1}$. Note that in order to exploit the underlying dynamics in the training process, we additionally need an explicit dependency on the time variable. We conclude that the zero-hold assumption allows us to reduce the sampling space from $\mathbb{T} \times \mathcal{X} \times L^\infty(\mathbb{T}, \mathcal{U})$ to $[0, \tau] \times \mathcal{X} \times \mathcal{U}$, thus rendering the learning task feasible.

Two remarks are in order: First, the initial value is included in the sampling space, thus deviating from the original PINN approach presented by Raissi et al. (2019), where the networks are trained for a fixed initial value. We emphasize that this is not due to our particular approach but inherent to the MPC framework. Second, our network is not restricted to predictions over the interval $[0, \tau]$. To compute predictions for $t > \tau$, we simply take the PINN approximation $\widehat{\varphi}(\tau, \boldsymbol{x}_0, \boldsymbol{u}_0)$ at time $t = \tau$ as new initial value for the time interval $[\tau, 2\tau]$ and repeat this process iteratively. This approach, which we refer to as self-loop prediction, is illustrated in Fig. 4a. To corresponding MPC scheme where the nonlinear dynamics are replaced with the PINN approximation is presented in Fig. 4b.

4. RESULTS

We test our methodology introduced in section 3 with the multi-link manipulator robot model discussed in section 2. Our approach is implemented in TensorFlow [1]. The network topology, including all relevant hyperparameters, is discussed in section 4.1. To ensure that the PINN approximation is sufficiently accurate, we first present in section 4.2 the prediction for a given input sequence. There the PINN operates in self-loop prediction mode as described in section 3.3. The tracking problem for the multi-link manipulator is then solved via the PINN-based MPC in closed-loop (cf. section 4.3). In the following, we use the symbols $\widehat{\alpha}$, $\widehat{\beta}$, α_{MPC}, and β_{MPC} to denote the PINN predictions for the angles α and β from Fig. 1, and the resulting angles from the MPC scheme, respectively.

To ensure reproducibility of the conducted experiments, the code for the numerical examples is publicly available under the `doi:10.5281/zenodo.5520662`. All simulations are performed on an Apple M1 chip.

4.1 Network topology and training

Our network consists of 4 layers, each hidden layer is activated by the hyperbolic tangent and contains 64 neurons. For the data-based loss function L_{data}, we use $N_{\text{data}} = 100$ data points, which only contain a subset of possible initial values, thus not a single solution of the nonlinear system (1) is required, see also Raissi et al. (2019).

Our numerical experiments have shown that it is reasonable to train the network for a slightly larger sampling period than it will be evaluated later on, i.e., we train the network for the time interval $[0, \tilde{\tau}]$ with $\tilde{\tau} := 0.25$ s > 0.2 s $=: \tau$, and use

$$\mathcal{X} = [-\pi, \pi]^2 \times [-2.5, 2.5]^2, \qquad \mathcal{U} = [-0.5, 0.5]^2.$$

For the physics-informed loss function L_{phys}, we sample $[0, \tilde{\tau}] \times \mathcal{X} \times \mathcal{U} \subseteq \mathbb{R}^7$ with $M_{\text{phys}} = 20000$ collocation points, generated via Latin Hypercube sampling (McKay et al., 1979). All hyper parameter are chosen as a result of a grid search. Let us emphasize that despite the small state dimension ($n = 4$), we have to sample a 7-dimensional space, rendering this a challenging problem from an approximation perspective.

The network training, i.e., the minimization of the loss function, is performed via L-BFGS, a quasi-Newton, full-batch gradient-based optimization algorithm (Liu and Nocedal, 1989) with 800000 iterations (referred to as epochs in the literature). To optimize the the training results, we choose to generate new data and collocation after the first 400000 epochs.

4.2 PINN in self-loop prediction

To ensure a sufficiently good prediction capability of the PINN approximation, we first run the PINN in self-loop

[1] https://www.tensorflow.org/

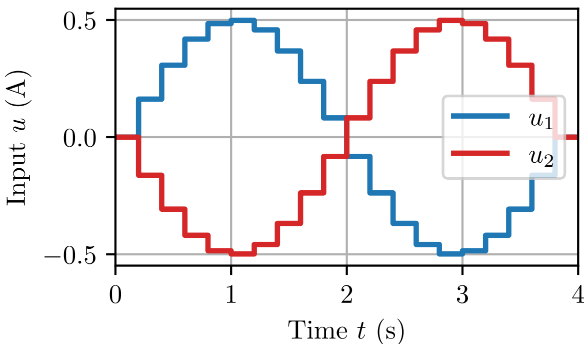


Fig. 5. Testing input sequence u for the open-loop application.

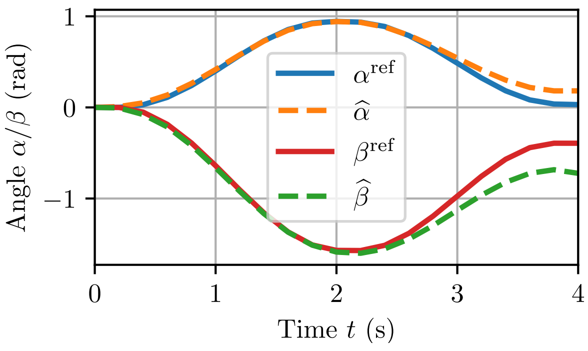


Fig. 6. Open-loop simulation result from PINN operated in self-loop prediction mode.

mode (cf. Fig. 4a) with the testing input sequence presented in Fig. 5. The sampling period is set to $\tau = 0.2$ s, thus this experiment results in 20 loop iterations. The corresponding open-loop prediction is presented in Fig. 6. We observe that the discrepancy between the PINN prediction and the nominal dynamics is relatively small for $t \leq 2$ s. For $t \geq 2$ s, the accumulation of the approximation errors due to the self-loop mode is noticeable and increases further over time. In the MPC context, the solution of the optimal control problem is robust regarding to this error accumulation for larger time horizons (cf. Schaller (2021)), such that we expect the PINN approximation quality for the dynamics is sufficient within the MPC framework.

4.3 PINN in closed-loop

We now apply the PINN-based MPC discussed in section 3.3 to a tracking problem for the multi-link manipulator (cf. section 2). For this purpose, we use the testing trajectory shown in Fig. 7 and its corresponding reference states $\boldsymbol{x}^{\text{ref}}$, taken from Fehr et al. (2020). The trajectory is motivated by an obstacle avoidance and includes motion reversals in both joints to investigate the control performance on friction phenomena caused by the harmonic drive gears that underly highly nonlinear friction effects.

The plant system is simulated by the Runge–Kutta–Fehlberg method (RK45), the MPC sampling period is set to $\tau = 0.2$ s and a horizon width of $H = 5$ is used. These parameters are taken from Fehr et al. (2020). The MPC cost function (5) is parameterized by the diagonal matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. We chose the matrices as

$$\boldsymbol{Q} := \begin{bmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{bmatrix}, \qquad \boldsymbol{R} := \begin{bmatrix} 1\cdot 10^{-6} & 0 \\ 0 & 1\cdot 10^{-6} \end{bmatrix},$$

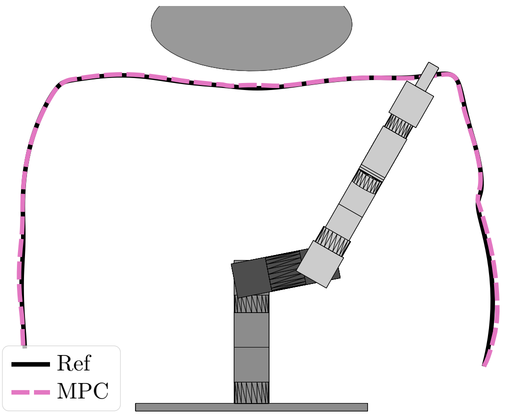


Fig. 7. Reference trajectory and resulting PINN-based MPC trajectory for the tracking problem of the mulit-link manipulator displayed at time $t = 8$ s.

which means we only control the positions of the angels.

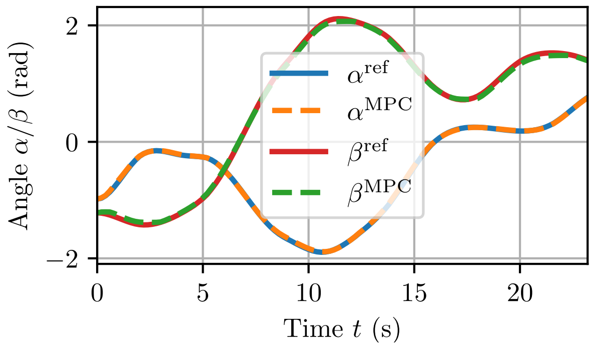


Fig. 8. Solution of the PINN-based MPC for the tracking problem of the multi-link manipulator.

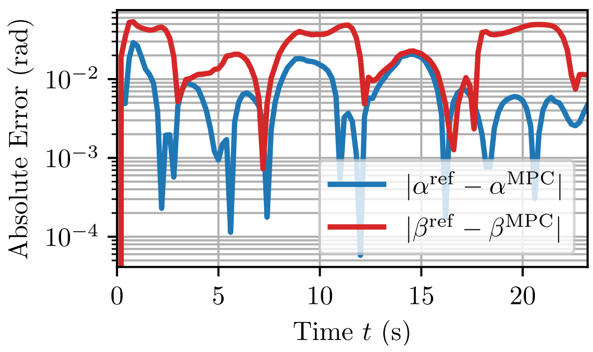


Fig. 9. Absolute error over time of the solution of PINN-based MPC for the tracking problem of the multi-link manipulator.

Figs. 8 and 9 illustrate that the NMPC scheme with PINN approximation for the nonlinear dynamics follows the reference trajectory closely with a mean absolute error of $7.53 \cdot 10^{-3}$ rad and $2.56 \cdot 10^{-2}$ rad for α and β, respectively. The computed control input is presented in Fig. 10. The solution of the NMPC problem is computed in average in $3.65 \cdot 10^{-2}$ s, which is clearly below the sampling period.

Let us emphasize that already the computation of $\boldsymbol{x}_{k+1}$ from $\boldsymbol{x}_k$ is more expensive with a classical time-integration scheme such as RK45 or the explicit Euler method, than with the PINN approximation, see Table 1.

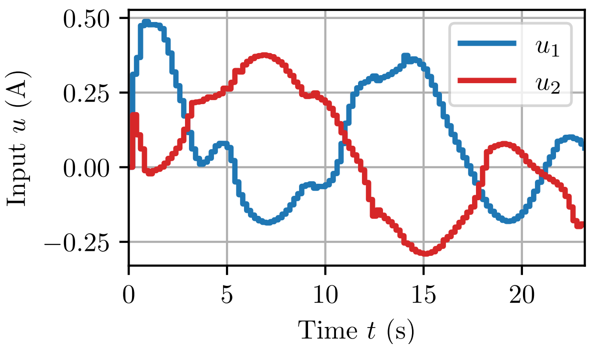


Fig. 10. PINN-based MPC optimal u.

Table 1. Execution times for different Runge-Kutta methods compared with the PINN, with $\tau = 0.2$ s and $h = 0.02$ s for Euler and RK4.

	PINN	Euler	RK4	RK45
Mean (s)	4.14e-04	5.93e-04	2.35e-03	8.62e-03
Median (s)	4.12e-04	5.90e-04	2.34e-03	1.77e-03

The main reason for the numerical integrators' performance is that these methods cannot use the sampling period as time-step, but need a smaller step-size h. For instance, the explicit Euler method is unstable for this system with step-size $h = \tau$. Instead, we use $h = 0.02\,\text{s}$ for the explicit Euler method, which in turn results in a slower execution time. Two additional remarks are in order: First, the evaluation of the PINN is implemented in TensorFlow. If the network with final weights is directly implemented using state-of-the-art BLAS libraries, we expect an additional speedup. Second, the computation of the gradient of the nonlinear dynamics concerning the control can be performed in parallel to the evaluation of the PINN (in contrast to adjoint-based methods), giving an additional speedup within the NMPC framework.

5. CONCLUSION

We studied the applicability of a physics-informed machine learning approach, namely physics-informed neural networks (PINNs) (Raissi et al., 2019), in the context of nonlinear model predictive control (NMPC) for a multi-link robot manipulator. Following ideas presented by Antonelo et al. (2021), we discussed how PINNs could be used to replace the nonlinear dynamics with an efficient-to-evaluate surrogate model. Due to automatic differentiation, the PINN approximation offers, in addition, a cheap and easy-to-implement computation of the partial derivative of the state with respect to the control input, thus paving the way for gradient-descent methods to solve the NMPC problem without the need of further acceleration strategies. In our numerical example for a tracking problem for a multi-link manipulator, we have shown that the PINN-based approximation of the nonlinear dynamics outperforms classical time-integrators in terms of computational time while at the same time being accurate enough to solve the tracking problem.

REFERENCES

Antonelo, E.A., Camponogara, E., Seman, L.O., de Souza, E.R., Jordanou, J.P., and Hubner, J.F. (2021). Physics-informed neural nets-based control. *arXiv preprint arXiv:2104.02556*.

Arnold, F. and King, R. (2021). State–space modeling for control based on physics-informed neural networks. *Engineering Applications of Artificial Intelligence*, 101, 104195. doi:10.1016/j.engappai.2021.104195.

Baydin, A.G., Pearlmutter, B.A., Radul, A.A., and Siskind, J.M. (2018). Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18.

Fehr, J., Schmid, P., Schneider, G., and Eberhard, P. (2020). Modeling, simulation, and vision-/MPC-based control of a PowerCube serial robot. *Applied Sciences*, 10(20), 7270. doi:10.3390/app10207270.

Grüne, L. and Pannek, J. (2011). *Nonlinear model predictive control*. Springer, London. doi:10.1007/978-3-319-46024-6.

Hertneck, M., Köhler, J., Trimpe, S., and Allgöwer, F. (2018). Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3). doi:10.1109/LCSYS.2018.2843682.

Kargl, A. (2020). DMD Methoden zur Identifikation von Reibkraftmodellen für einen Schunk PowerCube-Roboter. Bachelor thesis BSC-120, Institute of Engineering and Computational Mechanics, University Stuttgart.

Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3, 422–440. doi:10.1038/s42254-021-00314-5.

Liu, D.C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1), 503–528. doi:10.1007/BF01589116.

McKay, M.D., Beckman, R.J., and Conover, W.J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1), 55–61.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York.

Otness, K., Gjoka, A., Bruna, J., Panozzo, D., Peherstorfer, B., Schneider, T., and Zorin, D. (2021). An extensible benchmark suite for learning to simulate physical systems. *ArXiv e-print 2108.07799*.

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

Åkesson, B.M. and Toivonen, H, T. (2006). A neural network model predictive controller. *Journal of Process Control*, 16(9), 937–946. doi:10.1016/j.jprocont.2006.06.001.

Schaller, M. (2021). *Sensitivity Analysis and Goal Oriented Error Estimation for Model Predictive Control*. Ph.D. thesis, University of Bayreuth.

Sonntag, E.D. (1989). *Mathematical Control Theory*. Springer, New York, 2 edition. doi:10.1007/978-1-4612-0577-7.

Spong, M.W., Hutchinson, S., and Vidyasagar, M. (2020). *Robot modeling and control*. John Wiley & Sons.

Wu, Z., Rincon, D., Gu, Q., and Christofides, P.D. (2021). Statistical machine learning in model predictive control of nonlinear processes. *Mathematics*, 9(16), 1912. doi: 10.3390/math9161912.