2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Learning Iterative Solvers for Accurate and Fast Nonlinear Model Predictive Control via Unsupervised Training

Lukas Lüken and Sergio Lucia

Abstract— **Model predictive control (MPC) is a powerful control method for handling complex nonlinear systems that are subject to constraints. However, the real-time application of this approach can be severely limited by the need to solve constrained nonlinear optimization problems at each sampling time. To this end, this work introduces a novel learning-based iterative solver that provides highly accurate predictions, optimality certification, and fast evaluation of the MPC solution at each sampling time. To learn this iterative solver, we propose an unsupervised training algorithm that builds on the Karush-Kuhn-Tucker optimality conditions, modified by a Fischer-Burmeister formulation, and eliminates the need for prior sampling of exact optimizer solutions. By exploiting efficient vector-Jacobian and Jacobian-vector products via automatic differentiation, the proposed training algorithm can be efficiently executed.**

We demonstrate the potential of the proposed learning-based iterative solver on the example of nonlinear model predictive control of a nonlinear double integrator. We show its advantages when compared to exact optimizer solutions and with an imitation learning-based approach that directly obtains a data-based approximation of the MPC control law.

I. INTRODUCTION

Due to its ability to deal with potentially large-scale nonlinear systems, the integration of constraints and the consideration of advanced control objectives, model predictive control (MPC) is a very powerful control method [1], [2]. At each time step, an optimal control problem based on a prediction model of the system is solved and the calculated optimal next control action is then applied to the system. Despite its advantages, an important challenge of MPC is that it might be difficult to implement in practice due to its computational complexity. In particular, when dealing with large-scale nonlinear systems or very strict real-time and hardware requirements such as in embedded applications with very small sampling times, the time to solve the optimization problem might become prohibitively large [3], [4], [5].

Explicit MPC, in which the MPC control law is explicitly precomputed [6], can be used to mitigate this challenge but is typically only suitable for small linear systems. Instead of exactly representing the control law, an approximation, e.g. via neural networks can be used [7], [8], [9], [10]. This idea dates back already in the 90s with a work by [11]. However, with the widespread increase in machine learning capabilities, it has continued to gain interest in recent years [8]. Due to the fast evaluation of neural network-based function approximations compared to nonlinear optimization algorithms, a significant reduction in computation time of up to multiple orders of magnitude can be achieved, e.g. when applied to large-scale nonlinear systems [3] or highly-nonlinear robust economic MPC problems [4]. Furthermore, these controllers can be deployed with minimal memory requirements, as the evaluation of the neural networks involves simple function evaluations and powerful frameworks for embedded deployment already exist [12]. An overview on recent advances can be found in [5].

Despite these promising results, appproximate MPC based on neural networks has some important drawbacks. For instance, sampling the training data can be very costly, as the number of data points required for an accurate approximation generally increases with the complexity of the optimization problem. Furthermore, obtaining highly-accurate approximations for larger systems is very challenging. This means that guaranteeing constraint satisfaction, especially in the nonlinear case, becomes difficult. Several approaches exist for dealing with constraint satisfaction, such as applying probabilistic or deterministic validation [13], [14], [15], performing forward simulations with the approximate MPC policy and fall back to safe solutions in case of constraint violations [16], deriving bounds on the neural network prediction error [17], [18] or projecting the neural network prediction onto a safe set [19]. However, these approaches are often not applicable to nonlinear MPC or lead to conservative behavior by requiring additional back-off parameters or conservative fallback strategies. Moreover, the optimality of approximate MPC predictions is not guaranteed in general, and especially when safety-enhancing measures are used which increase the conservatism of the controller.

We propose in this paper a learning-based iterative solver that allows for highly accurate predictions, certification of optimality based on the predicted solution, fast evaluation, and an unsupervised training algorithm that does not require sampling of optimizer solutions. That is, instead of directly approximating the MPC solution, we aim at learning a problem-specific iterative solver. To this end, we use a neural network that takes as input a measure of the optimality of a current solution iterate and parameters of the optimization problem, and iteratively predict solver steps that update the iterates. Additionally, we suggest a loss function for training that is based on KKT conditions adapted with a Fischer-Burmeister formulation for unsupervised training. Furthermore, we present an efficient training algorithm that utilises vector-Jacobian and Jacobian-vector products (VJP and JVP) to eliminate the requirement for computationally

Lukas Lüken and Sergio Lucia are with the Chair of Process Automation Systems, TU Dortmund University, 44227 Dortmund, Germany {lukas.lueken, sergio.lucia}@tu-dortmund.de

expensive calculations and inversions of Jacobians.

Some works have recently attempted to learn parameters for specific optimization algorithms [20], or update rules of unconstrained optimization algorithms for machine learning tasks [21]. However, to the best of the author's knowledge, this is the first work to present a learning-based iterative solver for nonlinear constrained model predictive control problems that achieves very tight accuracy and is trained without the need for prior computation of optimal solutions. This significantly alleviates the main challenges of approximate MPC based on neural network approximations.

This paper is structured as follows. The background on MPC, approximate MPC and constrained nonlinear optimization is presented in Section II. The proposed learning-based iterative solver is presented in Section III. The promising results of the proposed algorithm are illustrated with an NMPC example in Section IV and the paper is concluded in Section V.

II. BACKGROUND

A. Nonlinear Model Predictive Control

In the following, we consider a nonlinear discrete-time dynamic system at time step k with states $\boldsymbol{x}_k$, control actions $\boldsymbol{u}_k$, dynamics f as well as initial conditions $\boldsymbol{x}_{\text{init}}$:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}}. \tag{1}$$

For the given system f, the NMPC algorithm seeks to optimize an objective while also adhering to constraints over a given time horizon N. This control task is formalized in the following nonlinear program (NLP):

$$\begin{aligned}
\min_{\boldsymbol{x}_{[0:N]}, \boldsymbol{u}_{[0:N-1]}} \quad & V_f\left(\boldsymbol{x}_N\right) + \sum_{k=0}^{N-1} \ell\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) && \text{(2a)}\\
\text{s.t.} \quad & \boldsymbol{x}_{k+1} = f\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right), \quad \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} && \text{(2b)}\\
& \boldsymbol{x}_L \leq \boldsymbol{x}_k \leq \boldsymbol{x}_U && \text{(2c)}\\
& \boldsymbol{u}_L \leq \boldsymbol{u}_k \leq \boldsymbol{u}_U && \text{(2d)}\\
& g\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) \leq 0 && \text{(2e)}\\
& h\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) = 0 && \text{(2f)}\\
& k \in \{0, \ldots, N-1 \mid N \in \mathbb{N}, N \geq 1\}. && \text{(2g)}
\end{aligned}$$

The decision variables are the state trajectory $\boldsymbol{x}_{[0:N]}$ and control trajectory $\boldsymbol{u}_{[0:N-1]}$. The objective of the NMPC is formulated as the cost function with the terminal cost of the states at time point N as V_f and the sum over all stage costs ℓ, considering all states and control actions up to time point $N-1$. Nonlinear constraints in form of equality and inequality constraints are described by h and g. Upper and lower bounds on the states and control actions are defined as $\boldsymbol{x}_U$, $\boldsymbol{u}_U$ and $\boldsymbol{x}_L$, $\boldsymbol{u}_L$. The discrete-time system model $f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is represented as an equality constraint.

At each time step, the optimization problem (2) is solved considering the current initial state $\boldsymbol{x}_{\text{init}}$. The next control action $\boldsymbol{u}_0$ is then applied to the system (1). The following control law summarizes this procedure:

$$\boldsymbol{u}_0 = \Pi_{\text{MPC}}(\boldsymbol{x}_{\text{init}}). \tag{3}$$

B. Approximate Nonlinear Model Predictive Control via Imitation Learning

To circumvent the possibly prohibitive computation times of solving the optimization problem (2) during the online application of the NMPC, an approximation of the control law (3) can be used. By solving the optimization problem offline for various initial states, a dataset $\mathbb{D}^{\text{MPC}} = \{(\boldsymbol{x}_{\text{init}}, \boldsymbol{u}_0)^i | i = 1, \ldots, N_s\}$ of size N_s containing $(\boldsymbol{x}_{\text{init}}, \boldsymbol{u}_0)$ data pairs can be sampled. Function approximators such as neural networks (NN) can then be used to represent the control law in form of a model with parameters θ and predicted next control actions $\hat{\boldsymbol{u}}_0$:

$$\hat{\boldsymbol{u}}_0 = \Pi_{\text{approxMPC}}(\boldsymbol{x}_{\text{init}}; \theta). \tag{4}$$

Evaluating this approximate MPC can be up to orders of magnitude faster than solving the NLP directly, since a prediction with a neural network involves usually only a few explicit function calls [3], [4].

The neural network parameters θ of the approximate MPC are usually determined via imitation learning, i.e. by minimizing the distance between the optimal control action $\boldsymbol{u}_0$ and the predicted control action $\hat{\boldsymbol{u}}_0$ on the data $\mathbb{D}^{\text{MPC}}$ using standard neural network training methods such as stochastic gradient descent [5], [8].

C. Constrained Nonlinear Optimization

The NMPC problem (2) can be more generally represented as the following parametric nonlinear program (NLP):

$$\begin{aligned}
\min_{\boldsymbol{w}} \quad & q\left(\boldsymbol{w}; \boldsymbol{p}\right) && \text{(5a)}\\
\text{s.t.} \quad & g_i\left(\boldsymbol{w}; \boldsymbol{p}\right) \leq 0, \quad i = 1, \ldots, n_g && \text{(5b)}\\
& h_j\left(\boldsymbol{w}; \boldsymbol{p}\right) = 0, \quad j = 1, \ldots, n_h. && \text{(5c)}
\end{aligned}$$

Here, the decision variables are depicted as $\boldsymbol{w}$ and the parameters as $\boldsymbol{p}$. The objective function is represented by q, the n_g nonlinear inequality constraints as g_i and the n_h nonlinear equality constraints as h_i.

The primal-dual solution to this problem is described by $\boldsymbol{z}^* = (\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \boldsymbol{z}^*(\boldsymbol{p})$, with $\boldsymbol{w}^*$ describing the optimal solution of the decision variable, and $\boldsymbol{\lambda}^*$ and $\boldsymbol{\nu}^*$ representing the optimal solution of the Lagrange multipliers associated with the inequality constraints g and equality constraints h respectively. The primal-dual solution $\boldsymbol{z}^*(\boldsymbol{p})$ is dependent on the parameters $\boldsymbol{p}$ of the NLP.

A solution $\boldsymbol{z}^*(\boldsymbol{p})$ to the NLP has to satisfy the first-order necessary conditions of optimality, the so-called Karush-Kuhn-Tucker (KKT) conditions [22]. The KKT conditions are based on the Lagrangian $\mathcal{L}$ of the NLP, which is defined as

$$\mathcal{L}\left(\boldsymbol{z}; \boldsymbol{p}\right) = q\left(\boldsymbol{w}; \boldsymbol{p}\right) + \boldsymbol{\lambda}^\top \boldsymbol{g}\left(\boldsymbol{w}; \boldsymbol{p}\right) + \boldsymbol{\nu}^\top \boldsymbol{h}\left(\boldsymbol{w}; \boldsymbol{p}\right). \tag{6}$$

The KKT conditions can then be formulated as [22]:

$$F_{\mathrm{KKT}}(\boldsymbol{z}^*, \boldsymbol{p}) = \begin{cases} \nabla_{\boldsymbol{w}} \mathcal{L}\left(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*; \boldsymbol{p}\right) = 0 & \text{(7a)} \\ \boldsymbol{h}\left(\boldsymbol{w}^*; \boldsymbol{p}\right) = 0 & \text{(7b)} \\ \boldsymbol{g}\left(\boldsymbol{w}^*; \boldsymbol{p}\right) \leq 0 & \text{(7c)} \\ \boldsymbol{\lambda}^* \geq 0 & \text{(7d)} \\ \boldsymbol{\lambda}^* \odot \boldsymbol{g}\left(\boldsymbol{w}^*; \boldsymbol{p}\right) = 0. & \text{(7e)} \end{cases}$$

The $\odot$ symbol describes the element-wise product (Hadamard-Product). The KKT conditions are described by the stationarity condition (7a), primal feasibility (7b and 7c), dual feasibility (7d) and complementary slackness (7e).

These conditions are also crucial for solving the NLP, as many iterative solvers are based on calculating steps that minimize some form of residuals of these conditions [23], [24], [25]. These iterative solvers are usually based on the following update rule:

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \alpha_k \cdot \Delta \boldsymbol{z}_k. \tag{8}$$

Starting with a given iterate of the primal-dual solution $\boldsymbol{z}_k$ at iteration k, the solution is updated with step $\Delta \boldsymbol{z}_k$ and step size α_k. A large variety of methods exist to calculate the step $\Delta \boldsymbol{z}_k$ and the step size α_k [22]. Well suited for NLPs because of their fast convergence near an optimal solutions are algorithms based on Newton's method. In general, the optimality conditions, summarized as F_{KKT}, are transformed into a nonlinear system of equations and Newton's method is applied to calculate the next step $\Delta \boldsymbol{z}_k$. However, due to the inequality constraints (7c) and (7d) and the non-smoothness of the complementary slackness (7e) a formulation as a nonlinear system of equation is not straightforward [22]. To this end, interior-point methods [23] or active-set methods [22] can be used.

III. Learning-Based Iterative Solver

A. A Neural Network-Based NLP Solver

We propose a learning-based iterative solver for calculating highly-accurate solutions of the NMPC problem (2) in combination with an efficient unsupervised training framework that does not require previously sampled solutions. Furthermore, this approach directly provides estimates of the primal-dual solution $\boldsymbol{z}^*$, allowing to directly certify the optimality of a prediction using the KKT conditions (7). To this end, we formulate an approximation of the computation of the steps $\Delta \boldsymbol{z}_k$ of an iterative solver as in (8).

We want to obtain a learning-based iterative solver (LIS) so that when applying it for N_{steps} steps on an initial guess $\boldsymbol{z}_0$ for given parameters $\boldsymbol{p}$ a solution $\hat{\boldsymbol{z}}$ is obtained that minimizes a modified residual of the optimality conditions F_{KKT}. The iterative nature of this approach is intended to mitigate the (almost) unavoidable approximation errors of an NN-based approach by iteratively reducing these residuals, thus, aiming at determining solutions $\hat{\boldsymbol{z}} = \boldsymbol{z}_{N_{steps}} \approx \boldsymbol{z}^*$ with very high accuracy. This approximate solver can then be described as follows:

$$\Delta \hat{\boldsymbol{z}}_k = \Pi_{\mathrm{LIS}}(\boldsymbol{z}_k, \boldsymbol{p}; \theta_{\mathrm{LIS}}). \tag{9}$$

Contrary to the approximate MPC scheme (Section II-B), we do not only consider to predict the next control action $\hat{\boldsymbol{u}}_0$ but the full primal-dual solution $\hat{\boldsymbol{z}}$ as this is required to formulate the full optimality conditions $F_{\mathrm{KKT}}(\boldsymbol{z}, \boldsymbol{p})$. This allows to directly determine the residual of the optimality conditions and thus the constraint satisfaction as well as the optimality of the solution. Another advantage of this approach is the unsupervised training algorithm (proposed in Subsection III-C), which does not require previously sampled solutions of the optimization problem (2) or the control law (3) as it directly considers the optimality conditions which are fully determined by the parameters $\boldsymbol{p}$ and the predicted primal-dual solution $\hat{\boldsymbol{z}}$.

For the control of system (1), the predicted next control action $\hat{\boldsymbol{u}}_0 \subset \hat{\boldsymbol{z}}$ can then be extracted from the full solution. In the case of high residuals, further steps of the iterative solver can thus be applied or a fallback strategy, such as in [16], can be used.

B. Modified KKT Conditions Based on Smoothed Fischer-Burmeister Equations

A modified residual of the optimality conditions is required for unsupervised learning as well as for the application of the learning-based iterative solver, such that the KKT conditions containing inequality expressions can be transformed into a nonlinear system of equations. For this purpose, we propose to use Fischer-Burmeister equations [24] with a smoothing parameter as proposed by [25]. An advantage of this approach is the ability to deal with infeasible iterates $\boldsymbol{z}_k$, which are difficult to avoid in a learning-based approach, especially during training. Furthermore, this approach can be inherently warm-started, so that we do not have to iteratively reduce the smoothing parameters, as would be the case with the barrier parameters of an interior-point method [25].

The inequality constraints (7c) and (7d) as well as the complementary slackness (7e) are substituted from the optimality conditions with the use of the Fischer-Burmeister function $\phi : \mathbb{R}^2 \to \mathbb{R}$. For the individual Lagrange multipliers λ_i and the corresponding inequality constraints g_i, this function is defined as follows:

$$\phi(\lambda_i, g_i) = \lambda_i - g_i - \sqrt{\lambda_i^2 + g_i^2 + \epsilon^2} = 0. \tag{10}$$

For $\epsilon = 0$, if this function is zero, the following holds:

$$\lambda_i \cdot g_i = 0, \quad \lambda_i \geq 0, \quad g_i \leq 0. \tag{11}$$

To deal with the non-smoothness of this function, the smoothing parameter ϵ is added [25], which relaxes the complementary slackness (7e), such that $\lambda_i \cdot g_i = -\frac{\epsilon^2}{2}$ holds. The vector-valued function Φ describes the element-wise application of the Fischer-Burmeister function ϕ to the vector of Lagrange multipliers $\boldsymbol{\lambda}$ and corresponding inequality constraints $\boldsymbol{g}$.

This leads to the following representation of the optimality conditions as a nonlinear system of equations:

$$F_{\Phi}(\boldsymbol{z}; \boldsymbol{p}) = \begin{pmatrix} \nabla_{\boldsymbol{w}} \mathcal{L}\left(\boldsymbol{z}; \boldsymbol{p}\right) \\ \boldsymbol{h}\left(\boldsymbol{w}; \boldsymbol{p}\right) \\ \Phi(\boldsymbol{\lambda}, \boldsymbol{g}\left(\boldsymbol{w}; \boldsymbol{p}\right)) \end{pmatrix} = 0. \tag{12}$$

C. Efficient Unsupervised Learning leveraging Automatic Differentiation

We propose an unsupervised training scheme that requires sampling only the input data $(\boldsymbol{z}_k, \boldsymbol{p})$, but not the output data $\Delta \boldsymbol{z}_k$, thereby avoiding the need to sample the optimizer solution as in approximate MPC. Furthermore, by leveraging Automatic Differentiation [26] especially in the form of Jacobian-vector products (JVP) and Vector-Jacobian products (VJP) we formulate the training in such a way that no Jacobians of the residuals F_Φ have to be calculated, leading to increased speed and memory efficiency of the proposed training algorithm.

To assess a predicted solver step $\Delta \hat{\boldsymbol{z}}_k$ based on the given iterate $\boldsymbol{z}_k$ and parameters $\boldsymbol{p}$, we formulate the following scalar metric for the predicted error of the linearized residual of the optimality conditions:

$$V_k(\boldsymbol{z}_k, \Delta \hat{\boldsymbol{z}}_k, \boldsymbol{p}) = \frac{1}{2} \| F_{\Phi,k} + \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta \hat{\boldsymbol{z}}_k \|_2^2. \tag{13}$$

The objective of the neural network training is to determine parameters θ_{LIS} of the learning-based iterative solver, which lead to steps $\Delta \hat{\boldsymbol{z}}_k$ minimizing the error measure V_k. This linearization has the advantage, that $F_{\Phi,k} = F_\Phi(\boldsymbol{z}_k, \boldsymbol{p})$ and $\nabla_{\boldsymbol{z}} F_{\Phi,k} = \nabla_{\boldsymbol{z}} F_\Phi(\boldsymbol{z}_k, \boldsymbol{p})$ are fully determined by the current iterate $\boldsymbol{z}_k$ and the parameters $\boldsymbol{p}$. In case of $V_k \to 0$, this is equivalent to calculating Newton steps, e.g. as presented by [24], [25].

While the Jacobian of the residual $\nabla_{\boldsymbol{z}} F_{\Phi,k} \in \mathbb{R}^{n_z \times n_z}$ can be computed using Automatic Differentiation tools, due to the scaling of the size of this expression $(n_z \times n_z)$ with respect to the number of decision variables and constraints, it can become computationally expensive to compute as well as store. We circumvent this by using vector-valued Jacobian vector products (JVP) and vector Jacobian products (VJP) instead as described in the following.

For a batch of N_s input data $\mathbb{D}^{\mathrm{LIS}} = \{(\boldsymbol{z}_k, \boldsymbol{p})^i | i = 1, \dots, N_s\}$, which can be sampled e.g. randomly or based on previous solver steps as in Section III-E, we map the loss function of the neural network training as follows:

$$L_{\mathrm{LIS}} = \frac{1}{N_s} \sum_{i=1}^{N_s} \ln(V_k^i). \tag{14}$$

Here, we sum the log-scaled error metrics V_k^i over N_s data points. Without the log-scaling, the influence of training inputs with high values of V_k^i would be significantly greater than the influence of training inputs with low values of V_k^i, especially if they differ by several orders of magnitude. The training of the learning-based iterative solver is thus described by:

$$\min_{\theta_{\mathrm{LIS}}} L_{\mathrm{LIS}}(\mathbb{D}^{\mathrm{LIS}}; \theta_{\mathrm{LIS}}). \tag{15}$$

This training loss can then be minimized by using standard methods such as stochastic gradient descent or more sophisticated first-order methods such as AdamW [27].

For this, the gradients of the loss function L_{LIS} with respect to the parameters of the learning-based iterative solver θ_{LIS} are required. This expression is composed as follows:

$$\frac{\partial L_{\mathrm{LIS}}}{\partial \theta_{\mathrm{LIS}}} = \frac{1}{N_s} \sum_{i=1}^{N_s} \frac{1}{V_k^i} \cdot \frac{\partial V_k^i}{\partial \Delta \hat{\boldsymbol{z}}_k} \cdot \frac{\partial \Delta \hat{\boldsymbol{z}}_k}{\partial \Delta \tilde{\boldsymbol{z}}_k} \cdot \frac{\partial \Delta \tilde{\boldsymbol{z}}_k}{\partial \theta_{\mathrm{LIS}}}. \tag{16}$$

Here, the expression $\frac{\partial \Delta \hat{\boldsymbol{z}}_k}{\partial \Delta \tilde{\boldsymbol{z}}_k}$ represents the scaling of the neural network output $\Delta \tilde{\boldsymbol{z}}_k$ to the predicted step $\Delta \hat{\boldsymbol{z}}_k$. The scaling procedure is described in Section III-D. The derivative of the neural network output with respect to the parameters is depicted as $\frac{\partial \Delta \tilde{\boldsymbol{z}}_k}{\partial \theta_{\mathrm{LIS}}}$. With the expression $\frac{\partial V_k^i}{\partial \Delta \hat{\boldsymbol{z}}_k}$ known, this derivative becomes a vector-Jacobian product (VJP) and can be computed efficiently via standard back-propagation frameworks such as PyTorch [28].

The gradient of V_k with respect to $\Delta \hat{\boldsymbol{z}}_k$ is determined as follows:

$$\frac{\partial V_k}{\partial \Delta \hat{\boldsymbol{z}}_k} = \left(F_{\Phi,k} + \nabla_{\boldsymbol{z}} F_{\Phi,k} \cdot \Delta \hat{\boldsymbol{z}}_k \right)^\top \nabla_{\boldsymbol{z}} F_{\Phi,k} \in \mathbb{R}^{n_z}. \tag{17}$$

For a given step $\Delta \hat{\boldsymbol{z}}_k$ we can now define the intermediate variable A:

$$A = \nabla_{\boldsymbol{z}} F_{\Phi,k} \cdot \Delta \hat{\boldsymbol{z}}_k \in \mathbb{R}^{n_z}. \tag{18}$$

This expression is a Jacobian-vector Product (JVP) of the Jacobian of the residual $\nabla_{\boldsymbol{z}} F_{\Phi,k}$ and the given step $\Delta \hat{\boldsymbol{z}}_k$ and as such is much more efficient to compute with standard Automatic Differentiation (AD) tools such as CasADi [29] than the Jacobian itself [26]. Furthermore, we define a vector B which adds the residual vector and the JVP:

$$B = F_{\Phi,k} + A \in \mathbb{R}^{n_z}. \tag{19}$$

Using this expression, the gradient $\frac{\partial V_k}{\partial \Delta \hat{\boldsymbol{z}}_k}$ can be described as a vector-Jacobian product (VJP) of vector B and Jacobian $\nabla_{\boldsymbol{z}} F_{\Phi,k}$:

$$\frac{\partial V_k}{\partial \Delta \hat{\boldsymbol{z}}_k} = B^\top \nabla_{\boldsymbol{z}} F_{\Phi,k} \in \mathbb{R}^{n_z}. \tag{20}$$

In this way, the entire gradient $\frac{\partial L_{\mathrm{LIS}}}{\partial \theta_{\mathrm{LIS}}}$ can be described without the need to determine complete Jacobians of the residuals $F_{\Phi,k}$.

D. Neural Network Scaling for Highly Accurate Solutions

In order for the learning-based iterative solver (9) to provide solutions of high accuracy, an appropriate scaling of the inputs and outputs of the underlying neural network is essential. Steps $\Delta \hat{\boldsymbol{z}}_k$ minimizing the error metric (13) can deviate over several orders of magnitude depending on how close the iterate $\boldsymbol{z}_k$ is to the optimal solution. Also, small changes in $\boldsymbol{z}_k$ or $\boldsymbol{p}$ can result in changes in the residuals of the optimality conditions $F_\Phi(\hat{\boldsymbol{z}}, \boldsymbol{p})$ and $F_{\mathrm{KKT}}(\hat{\boldsymbol{z}}, \boldsymbol{p})$ of several orders of magnitude. Since approximation errors are unavoidable for the neural network underlying the approximate solver, a classical linear scaling of the outputs, such as with min-max or standard scaling, is not suitable because it would lead to high deviations especially in the range of very accurate $\boldsymbol{z}_k$. To address these problems, we present the following scaling approaches.

For the parameters of the optimization problem $\boldsymbol{p}$, linear or problem-specific scaling methods can be applied, as in approximate MPC. Instead of the iterates $\boldsymbol{z}_k$ as direct inputs of the neural network, we use a normalized vector of residuals $\tilde{F}_{\Phi,k}$, as well as a log-scaled 2-norm of these residuals $\overline{\|F_{\Phi,k}\|_2}$, which are defined as follows:

$$\tilde{F}_{\Phi,k} = \frac{F_{\Phi,k}}{\|F_{\Phi,k}\|_2}, \tag{21}$$

$$\overline{\|F_{\Phi,k}\|_2} = \ln(\|F_{\Phi,k}\|_2). \tag{22}$$

The motivation behind the normalized residual vector (21) is that the entries of the residual can span several orders of magnitude. Using the log-scaled 2-norm of this residual (22), we extend the inputs with the information of the magnitude of the residual. Overall, we can now summarize the inputs of the NN as follows:

$$\tau_k(\boldsymbol{z}_k, \boldsymbol{p}) = \left(\boldsymbol{p}, \tilde{F}_{\Phi,k}, \overline{\|F_{\Phi,k}\|_2}\right), \tau_k \in \mathbb{R}^{n_p+n_z+1}. \tag{23}$$

Based on these inputs, we formulate the neural network Ψ_{LIS} with parameters θ_{LIS}, which predicts a scaled step $\Delta\tilde{\boldsymbol{z}}_k$, as follows:

$$\Delta\tilde{\boldsymbol{z}}_k = \Psi_{\mathrm{LIS}}(\tau_k(\boldsymbol{z}_k, \boldsymbol{p}); \theta_{\mathrm{LIS}}). \tag{24}$$

Using the 2-norm of the residual vector $\|F_{\Phi,k}\|_2$ and a variable scalar scaling factor γ_k, we thus compute the unscaled step of the approximate solver:

$$\Delta\hat{\boldsymbol{z}}_k = \gamma_k \cdot \|F_{\Phi,k}\|_2 \cdot \Delta\tilde{\boldsymbol{z}}_k. \tag{25}$$

This scaling causes the magnitude of the predicted steps $\|\Delta\hat{\boldsymbol{z}}_k\|_2$ to evolve as a function of the magnitude of the residual $\|F_{\Phi,k}\|_2$, allowing for more accurate predictions in case of iterates $\boldsymbol{z}_k$ with low errors on the optimality criteria.

The purpose of the variable scaling factor γ_k is to determine an optimal step size based on a predicted step $\Delta\tilde{\boldsymbol{z}}_k$.

First, we describe the problem to determine the variable scaling factor γ_k for a given iteration point $\boldsymbol{z}_k$ and a predicted step $\hat{\boldsymbol{z}}_k$ as follows:

$$\min_{\gamma_k} \|F_{\Phi,k} + \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k \gamma_k\|_2^2. \tag{26}$$

By solving this optimization problem, a γ_k that minimizes the norm of the linearized predicted residual F_Φ for a given step $\Delta\hat{\boldsymbol{z}}_k$ is determined. To this end, we establish the first-order necessary conditions of optimality:

$$0 \overset{!}{=} \left(F_{\Phi,k} + \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k \gamma_k\right)^\top \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k \tag{27}$$

$$\Leftrightarrow F_{\Phi,k}^\top \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k + \gamma_k \left(\nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k\right)^\top \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k \tag{28}$$

$$\Leftrightarrow \gamma_k = -\frac{F_{\Phi,k}^\top \nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k}{\left(\nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k\right)^\top \left(\nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k\right)}. \tag{29}$$

Using the JVP $A = (\nabla_{\boldsymbol{z}} F_{\Phi,k} \Delta\hat{\boldsymbol{z}}_k)$ as described earlier in (18), this expression can be simplified as follows:

$$\gamma_k = -\frac{F_{\Phi,k} A}{A^\top A}. \tag{30}$$

As those expressions have to be calculated for the gradient of the loss function anyways, calculating γ_k comes at negligible cost. We project γ_k to a fixed interval so that the scaling factors do not differ too much: $\gamma_k \in [\underline{\gamma}, \overline{\gamma}]$, e.g. $[0.01, 2.0]$.

The approximation of the solver step (9) can thus be summarized as follows:

$$\begin{aligned}\Delta\hat{\boldsymbol{z}}_k &= \Pi_{\mathrm{LIS}}(\boldsymbol{z}_k, \boldsymbol{p}; \theta_{\mathrm{LIS}}) \\ &= \Psi_{\mathrm{LIS}}(\tau_k(\boldsymbol{z}_k, \boldsymbol{p}); \theta_{\mathrm{LIS}}) \cdot \gamma_k \cdot \|F_{\Phi,k}\|_2.\end{aligned} \tag{31}$$

The iterative update rule (8) considers not only the step $\boldsymbol{z}_k$ to be determined, but also a step size α_k. Since we already determine a variable scaling factor γ_k, which can be interpreted as a step size, we set $\alpha_k = 1$.

E. Multi-Step Self-Sampling for Unsupervised Learning

In this work we use a multi-step ahead sampling strategy to sample initial guesses for the primal-dual solution $\boldsymbol{z}_k$ for given parameters $\boldsymbol{p}$ based on predicted steps $\Delta\hat{\boldsymbol{z}}_k$ of previous training steps. This approach aims at finding initial guesses for the approximate solver, which are broadly distributed across from points with high residual norms $\|F_{\Phi,k}\|_2$ to data points with low residual norms. This sampling strategy is used to determine the inputs of the neural network during training and no sampling of optimal solutions, as in approximate MPC, is required.

For an initial batch of data $\{\boldsymbol{z}_0^i, \boldsymbol{p}^i\}_{i=1}^{N_b}$, which was determined e.g. by uniform random sampling, with batch size N_b the learning-based iterative solver is evaluated (forward pass) and thus a batch of steps $\{\hat{\boldsymbol{z}}_k\}_{i=1}^{N_b}$ is determined. Following this forward pass, the training loss (14) as well as the gradient (16) are evaluated and the parameters θ_{LIS} of the neural network are updated (backward pass). Afterwards, for each data point $i = 1, \ldots, N_b$ the next iterate $\boldsymbol{z}_{k+1}$ is calculated. This procedure is repeated for N_{steps} inside a training epoch. For N_{epochs}, this loop is repeated with a new batch of initial data $\{\boldsymbol{z}_0^i, \boldsymbol{p}^i\}_{i=1}^{N_b}$. This procedure is summarized in Alg. 1.

IV. ILLUSTRATIVE EXAMPLE

A. NMPC of a Nonlinear Double Integrator

In order to evaluate the effectiveness of the proposed learning-based iterative solver over exact optimizer solutions and the alternative approximate MPC, we consider the NMPC of a simple nonlinear double integrator. This an adaptation of the example presented by [30]. The system with two states $\boldsymbol{x} \in \mathbb{R}^2$ and one control action $u \in \mathbb{R}^1$ is described by the following discrete-time nonlinear dynamical model:

$$\boldsymbol{x}_{k+1} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix}0.5\\ 1\end{bmatrix} u_k + \begin{bmatrix}0.025\\ 0.025\end{bmatrix} \boldsymbol{x}_k^\intercal \boldsymbol{x}_k \tag{32a}$$

$$= f(\boldsymbol{x}_k, u_k). \tag{32b}$$

We formulate an NMPC problem with constraints on the states and control actions as $\|\boldsymbol{x}_k\|_\infty \leq 10$ and $|u_k|_\infty \leq 2$. We use a terminal cost $V_f(\boldsymbol{x}_N)$ and a stage cost $\ell(\boldsymbol{x}_k, u_k, u_{k-1})$, which are defined as

$$V_f(\boldsymbol{x}_N) = \boldsymbol{x}_N^\intercal \begin{pmatrix}0.8 & 0\\ 0 & 0.8\end{pmatrix} \boldsymbol{x}_N, \tag{33}$$

Algorithm 1 Unsupervised Training w. Multi-Step Sampling

Input: N_{steps}, N_{epochs}, N_b, θ_{LIS}

for $j = 0$ to $N_{\text{epochs}} - 1$ **do**
  Sample Batch: $\{\boldsymbol{z}_0^i, \boldsymbol{p}^i\}_{i=1}^{N_b}$
  for $k = 0$ to $N_{\text{steps}} - 1$ **do**
    $\{F_{\Phi,k}^i\}_{i=1}^{N_b} \leftarrow$ Evaluate Error (12)
    $\{\tau_k\}_{i=1}^{N_b} \leftarrow$ Input Vector (23)
    Forward Pass:
    $\{\Delta\tilde{\boldsymbol{z}}_k^i\}_{i=1}^{N_b} \leftarrow$ Evaluate Neural Network (24)
    $\{\gamma_k^i\}_{i=1}^{N_b} \leftarrow$ Scaling Factor (30)
    $\{\Delta\hat{\boldsymbol{z}}_k^i\}_{i=1}^{N_b} \leftarrow$ Step Scaling (25)
    $L_{\text{LIS}} \leftarrow$ Calculate loss (14)
    Backward Pass:
    $\frac{\partial L_{\text{LIS}}}{\partial \theta_{\text{LIS}}} \leftarrow$ Backpropagation (16)
    $\theta_{\text{LIS}} \leftarrow$ Update Weights Via Gradient Descent
    Update Iterates:
    $\boldsymbol{z}_k^i \leftarrow \boldsymbol{z}_k^i + \Delta\hat{\boldsymbol{z}}_k^i, \quad \forall i \in \{1, \dots, N_b\}$
  end for
end for

$$\ell\left(\boldsymbol{x}_k, u_k, u_{k-1}\right) = \boldsymbol{x}_k^\intercal \begin{pmatrix} 0.8 & 0 \\ 0 & 0.8 \end{pmatrix} \boldsymbol{x}_k \tag{34}$$

$$+0.1u_k^2 + 10^{-4}(u_k - u_{k-1})^2. \tag{35}$$

We then obtain the following optimization problem:

$$\min_{\boldsymbol{x}_{[0:N]}, u_{[0:N-1]}} \quad V_f\left(\boldsymbol{x}_N\right) + \sum_{k=0}^{N-1} \ell\left(\boldsymbol{x}_k, u_k, u_{k-1}\right) \tag{36a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = f\left(\boldsymbol{x}_k, u_k\right) \tag{36b}$$

$$\boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} \tag{36c}$$

$$\|\boldsymbol{x}_k\|_\infty \leq 10, \quad k = 1, \dots, N-1 \tag{36d}$$

$$|u_k|_\infty \leq 2, \quad k = 0, \dots, N-1. \tag{36e}$$

The control objective is to guide the system from a displacement $\boldsymbol{x}_{\text{init}}$ to the origin. The parameters of this optimization problem are the current initial states $\boldsymbol{x}_{\text{init}}$ and the previous control action u_{-1}, which are given by the control action u_0 of the previous iteration step during the closed-loop evaluation of the NMPC. This parameter vector $\boldsymbol{p} = (\boldsymbol{x}_{\text{init}}, u_{-1}) \in \mathbb{R}^{n_p}$ has dimension $n_p = 3$. The decision variables of the optimization problem can be summarized as $\boldsymbol{w} = \left(\boldsymbol{x}_{[0:N]}, u_{[0:N-1]}\right) \in \mathbb{R}^{(N+1)\cdot n_x + N \cdot n_u}$. For a prediction horizon of $N = 10$, it follows that $n_w = (N+1) \cdot n_x + N \cdot n_u = 32$. The number of Lagrange multipliers for the equality and inequality constraints is $n_\nu = (N+1) \cdot n_x = 22$ and $n_\lambda = (N-1) \cdot 2n_x + N \cdot 2n_u = 56$. The total dimension of the primal-dual solution $\boldsymbol{z} \in \mathbb{R}^{n_z}$ follows as $n_z = n_w + n_\nu + n_\lambda = 110$.

B. Setup

We compare now exact solutions of the optimization problem against solutions of an approximate NMPC and the proposed learning-based iterative solver. The code to reproduce the results is openly available.[1]

a) Training of Approximate MPC: First, we sample a training and a test dataset with optimal solutions for problem (36), $\mathbb{D}^{\text{Train}} = \{[(\boldsymbol{x}_0, u_{-1}), u_0]^i | i = 1, \dots, N_{\text{Train}}\}$ and $\mathbb{D}^{\text{Test}} = \{[(\boldsymbol{x}_0, u_{-1}), u_0]^i | i = 1, \dots, N_{\text{test}}\}$. During sampling, the parameters are uniformly random sampled, with the range determined by the bounds of the states and control actions of the NMPC (36).

We set up the nonlinear optimization problem using the open-source toolbox do-mpc [31], which is based on CasADi [29] as backend for Automatic Differentiation and as an interface to the optimizer. To solve the NLPs, we use the interior-point solver Ipopt [23] with a tolerance of 10^{-8}. We filter out parameters for which no feasible solution exists and sample until the desired amount of data points is obtained. The training and test datasets sizes are $N_{\text{Train}} = 10000$ and $N_{\text{Test}} = 1500$.

The approximate MPC is formulated as defined in (4), with the parameter vector $\boldsymbol{p} = (\boldsymbol{x}_{\text{init}}, u_{-1})$ as input and the predicted next control action $\hat{u}_0$ as output. A feedforward NN with 6 hidden layers and 100 neurons per layer with ReLU activation function is used. Thus, in total, the NN has $2.02 \cdot 10^5$ parameters. The large size of the NN is justified as previous work by [8] has shown that the expressiveness of a NN for approximate MPC increases significantly with the depth of the networks. However, the use of a shallow NN architecture, as used for the learning-based iterative solver, leads to poor performance. We scale the inputs and outputs of the NN with a min-max scaler considering the bounds of the MPC. The NNs are implemented using PyTorch [28]. For training the approximate MPC, we use a mean squared error (MSE) loss function. Furthermore, we apply AdamW [27] as optimizer for the NN training and set a batch size of $N_{\text{batch}} = 1024$. The training is performed for $N_{\text{epochs,total}} = 4000$ epochs with an initial learning rate of 10^{-2}, which is reduced by a factor of 10 after every $N_{\text{epochs}} = 1000$ epochs (learning rate scheduling).

b) Training of the Learning-based Iterative Solver: The learning-based iterative solver is formulated as defined in (31). Based on a parameter vector $\boldsymbol{p}$ and a primal-dual iterate $\boldsymbol{z}_k$, the input vector τ is determined as in (23). Therefore, the input dimension of the NN of the approximate solver is $n_\tau = n_z + n_p + 1 = 114$ and the dimension of the output is $n_z = 110$. Contrary to the approximate MPC, we use a shallow feedforward NN consisting of a hidden layer with 2000 neurons and ReLU activation function, resulting in a total number of NN parameters θ_{AS} of $4.50 \cdot 10^5$. In this example, the approximation quality of the shallow NN is already sufficient and does not improve much with depth, while the training times are lower compared to deeper architectures.

The unsupervised training as described in Algorithm 1 is performed with a learning rate of 10^{-3} for $N_{\text{epochs}} = 100$ and $N_{\text{steps}} = 200$ with a smoothing parameter $\epsilon = 10^{-6}$

[1] https://github.com/lukaslueken/2023_Learning_based_Iterative_Solver

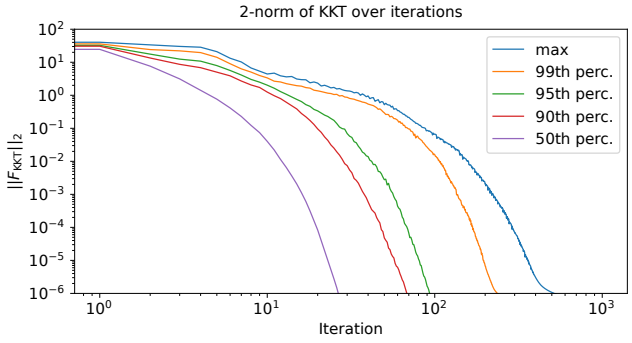

Fig. 1. Application of the approximate solver to $N_{\text{test}} = 1500$ test data points ($\boldsymbol{p}$) with random initial guesses $\boldsymbol{z}_0$. The log-scaled x-axis shows the number of iterations and the log-scaled y-axis shows the error on the original KKT conditions measured in the 2-norm. Visualized are different trajectories with the corresponding 50, 90, 95, 95 percentage points as well as the maximum error.

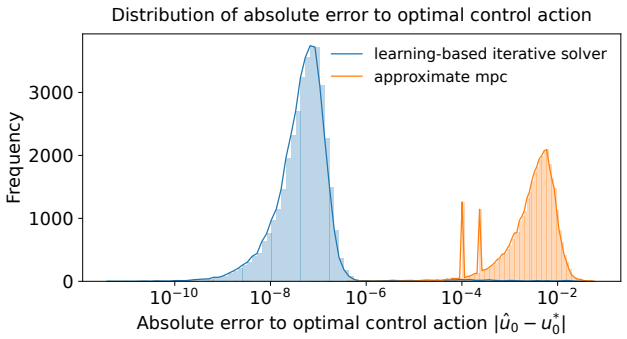

Fig. 2. Comparison of the prediction accuracy of the next control actions of the learning-based iterative solver and the approximate MPC compared to exact solutions, determined via Ipopt. The maximum number of iterations of the learning-based iterative solver is limited to $N_{\text{iter,max}} = 40$ and the iteration is stopped when a tolerance on the error $\|F_\Phi\|_2 = 10^{-6}$ has been reached.

and a batch size of $N_{\text{batch}} = 1024$. The parameters $\boldsymbol{p}$ are uniformly random sampled in the beginning of each training epoch. The initial guesses of the primal-dual solution $\boldsymbol{z}_0$ are sampled from the normal distribution $\mathcal{N}(0, 1)$. We note that this initial guess is relatively poor and more advanced initialization strategies can be used if necessary. The training is performed on an AMD Ryzen Threadripper 1920X CPU.

C. Convergence Behavior of Learning-based Iterative Solver

We look first at the convergence behavior of the proposed learning-based iterative solver. We apply the learning-based iterative solver to the parameters $\boldsymbol{p}$ of the test dataset $\mathbb{D}^{\text{Test}}$ with uniformly randomly sampled initial guesses $\boldsymbol{z}_0 \sim \mathcal{N}(0, 1)$. A maximum number of solver iterations of $N = 1000$ is considered. For evaluation, we consider the 2-norm of the KKT conditions (7). The results are shown in Fig. 1.

After less than 30 iterations, the learning-based iterative solver has achieved an error below 10^{-6} for more than 50% of the runs. This error is obtained in over 95% of cases after fewer than 100 iterations. By about 500 iterations, all $N_{\text{test}} = 1500$ runs have reached the tolerance of 10^{-6}.

We can thus state that the approximate solver led to convergence in all cases of the test data set and that this happened in more than 50% of the cases even after a small number of steps. While the median number of iterations is low, the maximum number of iterations is larger. However, this test data is conducted in an *open-loop* manner, meaning the predicted next control action is not implemented to the system to establish new subsequent initial states. Consequently, the data is widely distributed over the entire feasible space, including data points that are *close to infeasible*.

D. Comparison to Approximate NMPC

We examine now the closed-loop application of the learning-based iterative solver for the NMPC of the system (32). To prevent the system from moving to a stationary state, we add an additive white noise $d_k \sim \mathcal{N}(0, 0.3)$: $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, u_k) + d_k$. Based on the initial values for $\boldsymbol{x}_{\text{init}}$ and u_{-1}, which we take from the test dataset $\mathbb{D}^{\text{Test}}$, we simulate the system (32) with the predicted control actions of the learning-based iterative solver for $N_{\text{closed-loop}} = 25$ time steps forward. We have set the maximum number of solver iterations as $N_{\text{iter,max}} = 40$, based on our observation that the entire learning-based iterative solver can be executed faster than Ipopt on average. This is despite our currently non-optimized implementation, which includes slow connections to CasADi and PyTorch via Python code. In addition, the solver iterations stop early when a tolerance on the norm $\|F_\Phi\|_2 = 10^{-6}$ has been reached.

We compare the predicted next control action of our presented approach, as well as the prediction of the approximate MPC (4) with the exact solutions of the optimization problem, which we determine with Ipopt [23] and an optimizer tolerance of 10^{-10}. To this end, we determine the distance to the optimal solution $|\hat{u}_0 - u_0^*|$ for each prediction of the next control action.

The distribution of the prediction errors of the individual 25 closed-loop iterations for $N_{\text{test}} = 1500$ test data points is shown in Fig. 2. We see that despite the limitation of the maximum number of iterations of the learning-based iterative solver, the maximum error of the learned solver is lower than that of the approximate MPC. In addition, the distribution of prediction errors is significantly shifted, so that the learning-based iterative solver has average prediction errors that are several orders of magnitude lower.

Although the neural network of the learned solver has to be evaluated more times than the neural network of the approximate MPC (which is evaluated only once), the solver achieves the required tolerance in less than 18 iterations in this application in 90% of cases. We conclude that the proposed learning-based iterative solver is a very promising alternative to approximate MPC for the calculation of solutions of high accuracy.

V. CONCLUSIONS

We present a novel approach to learn a problem-specific iterative nonlinear optimization solver using neural networks.

The proposed approach achieves prediction errors several orders of magnitude lower than comparable neural network-based approximate MPC approaches. To achieve this, we present an efficient training algorithm that leverages the predicted KKT conditions in an unsupervised loss function, omitting the need for prior sampling of optimal solutions. Furthermore, by predicting complete primal-dual solutions, the optimality can be certified directly.

In an illustrative case study, considering the MPC of a nonlinear double integrator, we showcase the effectiveness of this approach to efficiently compute solutions of high accuracy. Further research will consider improved strategies for determining the initial guesses of the approximate solver. In addition, theoretical convergence properties of the proposed learning-based iterative solver will also be investigated as well as the application to larger nonlinear problems.

REFERENCES

[1] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Madison, Wisconsin: Nob Hill Publishing, 2017.

[2] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*, ser. Communications and Control Engineering. Cham: Springer International Publishing, 2017.

[3] P. Kumar, J. B. Rawlings, and S. J. Wright, "Industrial, large-scale model predictive control with structured neural networks," *Computers & Chemical Engineering*, vol. 150, p. 107291, July 2021.

[4] B. Karg and S. Lucia, "Reinforced approximate robust nonlinear model predictive control," in *2021 23rd International Conference on Process Control (PC)*, June 2021, pp. 149–156.

[5] C. Gonzalez, H. Asadi, L. Kooijman, and C. P. Lim, "Neural networks for fast optimisation in model predictive control: A review," *arXiv preprint arXiv:2309.02668*, 2023.

[6] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.

[7] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari, "Approximating Explicit Model Predictive Control Using Constrained Neural Networks," in *2018 Annual American Control Conference (ACC)*, June 2018, pp. 1520–1527.

[8] B. Karg and S. Lucia, "Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, Sept. 2020.

[9] A. Mesbah, K. P. Wabersich, A. P. Schoellig, M. N. Zeilinger, S. Lucia, T. A. Badgwell, and J. A. Paulson, "Fusion of machine learning and MPC under uncertainty: What advances are on the horizon?" in *2022 American Control Conference (ACC)*, 2022, pp. 342–357.

[10] L. Lüken, D. Brandner, and S. Lucia, "Sobolev Training for Data-efficient Approximate Nonlinear MPC," in *IFAC World Congress 2023*, 2023.

[11] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, Oct. 1995.

[12] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and Ó. Lucía, "Deep Learning-Based Model Predictive Control for Resonant Power Converters," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 409–420, Jan. 2021.

[13] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an Approximate Model Predictive Controller With Guarantees," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 543–548, July 2018.

[14] B. Karg, T. Alamo, and S. Lucia, "Probabilistic performance validation of deep learning-based robust NMPC controllers," *International Journal of Robust and Nonlinear Control*, vol. 31, no. 18, pp. 8855–8876, 2021.

[15] B. Karg and S. Lucia, "Stability and feasibility of neural network-based controllers via output range analysis," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 4947–4954.

[16] H. Hose, J. Köhler, M. N. Zeilinger, and S. Trimpe, "Approximate non-linear model predictive control with safety-augmented neural networks," *arXiv preprint arXiv:2304.09575*, 2023.

[17] F. Fabiani and P. J. Goulart, "Reliably-Stabilizing Piecewise-Affine Neural Network Controllers," *IEEE Transactions on Automatic Control*, vol. 68, no. 9, pp. 5201–5215, Sept. 2023.

[18] D. Teichrib and M. S. Darup, "Error bounds for maxout neural network approximations of model predictive control," in *IFAC World Congress 2023*, 2023.

[19] J. A. Paulson and A. Mesbah, "Approximate Closed-Loop Robust Model Predictive Control With Guaranteed Stability and Constraint Satisfaction," *IEEE Control Systems Letters*, vol. 4, no. 3, pp. 719–724, July 2020.

[20] J. Ichnowski, P. Jain, B. Stellato, G. Banjac, M. Luo, F. Borrelli, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Accelerating quadratic optimization with reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 21 043–21 055.

[21] T. Chen, X. Chen, W. Chen, Z. Wang, H. Heaton, J. Liu, and W. Yin, "Learning to optimize: A primer and a benchmark," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 8562–8620, 2022.

[22] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research. New York: Springer, 2006.

[23] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

[24] A. Fischer, "A special newton-type optimization method," *Optimization*, vol. 24, no. 3-4, pp. 269–284, Jan. 1992.

[25] D. Liao-McPherson, M. Huang, and I. Kolmanovsky, "A Regularized and Smoothed Fischer–Burmeister Method for Quadratic Programming With Applications to Model Predictive Control," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2937–2944, July 2019.

[26] A. Griewank and A. Walther, *Evaluating Derivatives*, ser. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 2008.

[27] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *NeurIPS*, 2019, p. 12.

[29] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, Mar. 2019.

[30] M. Lazar, D. Muñoz De La Peña, W. Heemels, and T. Alamo, "On input-to-state stability of min–max nonlinear model predictive control," *Systems & Control Letters*, vol. 57, no. 1, pp. 39–48, Jan. 2008.

[31] F. Fiedler, B. Karg, L. Lüken, D. Brandner, M. Heinlein, F. Brabender, and S. Lucia, "do-mpc: Towards FAIR nonlinear and robust model predictive control," *Control Engineering Practice*, vol. 140, p. 105676, Nov. 2023.