2021 American Control Conference (ACC)
New Orleans, USA, May 25-28, 2021

Deep Learning-based Approximate Nonlinear Model Predictive Control with Offset-free Tracking for Embedded Applications

Kimberly J. Chan†, Joel A. Paulson†, and Ali Mesbah

Abstract— **The implementation of nonlinear model predictive control (NMPC) in applications with fast dynamics remains an open challenge due to the need to solve a potentially non-convex optimization problem in real-time. The offline approximation of NMPC laws using deep learning has emerged as a powerful framework for overcoming these challenges in terms of speed and resource requirements. Deep neural networks (DNNs) are particularly attractive for embedded applications due to their small memory footprint. This work introduces a strategy for achieving offset-free tracking despite the presence of error in DNN-based approximate NMPC. The proposed approach involves a correction factor defined via a small-scale target tracking optimization problem, which is easier to approximate than the tracking NMPC law itself. As such, the overall control strategy is amenable to efficient implementations on low-cost embedded hardware. The effectiveness of the proposed offset-free DNN-based NMPC is demonstrated on a benchmark problem in which the control strategy is deployed onto a field programmable gate array (FPGA) architecture that is verified using hardware-in-the-loop simulations.**

I. Introduction

Model predictive control (MPC) is one of the most popular methods for the control of constrained multivariable systems [1]. Although MPC has found widespread use in industrial process systems [2], it has been gaining traction in various emerging applications due to its versatility and inherent robustness properties (e.g., autonomous vehicles [3], biomedical systems [4], and humanoid robots [5]). However, the transition to such emerging applications, characterized by their large-scale, highly nonlinear, and/or fast dynamics, brings about additional computational challenges – due to the fact that the underlying optimization problem defining the MPC law must be solvable in real-time. This problem is further exacerbated when the controller must be deployed on resource-constrained embedded systems.

Over the past few decades, significant work has been done in the development of fast MPC methods, which can be broadly categorized in two paradigms. The first set of approaches involve the development of fast solvers and customized implementations, e.g., variations of Nesterov's fast gradient method [6]–[8] and the alternating direction method of multipliers (ADMM) [9], [10]. Some of these algorithms have been implemented in software tools that can automatically generate C-code for direct implementation on embedded platforms [11], [12]. The second set of approaches, often referred to as *explicit* MPC, look to compute an offline solution to the MPC for all feasible values of the state [13]–[15]. Explicit MPC leverages the fact that, for linear time-invariant systems with quadratic objective functions, the MPC problem can be formulated as a parametric quadratic program whose solution is a piecewise affine function defined over polytopic regions [13]. However, not only is explicit MPC restricted in terms of the possible model, cost, and constraint functions, the number of polytopes that define the solution grows (in the worst-case) exponentially with the number of constraints [15]. As such, explicit MPC is traditionally limited to small systems with relatively small prediction horizons.

This work focuses on nonlinear MPC (NMPC). Although there has been significant progress in optimization algorithms for NMPC [16]–[18], several additional challenges related to numerical robustness at low computational accuracy and the avoidance of infeasibility arise in embedded applications of NMPC. Moreover, low-code complexity for portability and a small memory footprint are key aspects needed for efficient implementations on embedded hardware. Software tools such as ACADO [19] and GRAMPC [20] attempt to address these issues by generating highly efficient code, but the need to store and perform operations based on relatively large-scale matrices may still be challenging.

A promising approach to overcome these challenges is the offline approximation of NMPC laws using deep learning, as opposed to alternative function approximation methods such as the polynomial methods in [21], [22]. In particular, recent theoretical results have shown that deep neural networks (DNNs) – with several hidden layers – can efficiently represent non-smooth control laws that arise in a variety of MPC formulations [23]–[28]. However, due to the error introduced by the DNN-based NMPC approximation, it cannot be ensured that the closed-loop system converges to the desired steady-state condition. In [29], an approach was proposed for modifying DNNs (with rectifier linear unit activation functions) to guarantee convergence to a particular equilibrium point for linear systems. To the best of our knowledge, a similar modification has not yet been developed for nonlinear systems over a range of target conditions. Thus, the first contribution of this work is to introduce a novel correction factor that removes offset whenever the system converges to an equilibrium. The proposed correction factor is related to a steady-state target tracking optimization

K. J. Chan and A. Mesbah are with the Department of Chemical and Biomolecular Engineering at the University of California, Berkeley, CA 94720, USA. {kchan45,mesbah}@berkeley.edu

J. A. Paulson is with the Department of Chemical and Biomolecular Engineering at The Ohio State University, Columbus, OH 43210, USA. paulson.82@osu.edu

This work was supported by the US National Science Foundation under Grant 1912772.

†K. J. Chan and J. A. Paulson contributed equally to this work.

problem that is of a much smaller scale than that derived from NMPC.

The second contribution is to show how a separate DNN can be used to build a high accuracy approximation of the target tracker, so that the overall strategy remains amenable to efficient embedded implementations. We also discuss the key considerations for optimizing performance of the embedded controller. To demonstrate the feasibility of the method, we deploy the proposed control strategy onto a field programmable gate array (FPGA) architecture using a high level synthesis (HLS) approach. The DNN-based MPC strategy with and without the correction term is applied to a benchmark case study. Hardware-in-the-loop simulations are performed to illustrate feasibility of the approach for embedded applications wherein significantly improved performance is observed due to the added correction factor.

Notation. The set of real numbers, non-negative integers, and positive integers are denoted by $\mathbb{R}$, $\mathbb{N}$, and $\mathbb{N}_+$, respectively. Given $a, b \in \mathbb{N}$ such that $a < b$, we let $\mathbb{N}_a^b = \{a, a+1, \dots, b\}$ denote the sequence of integers from a to b. The i^{th} element of a vector is denoted by $[x]_i$. Given two column vectors a and b, we let $(a, b) = [a^\top, b^\top]$ denote vector concatenation. The composition of two functions f and g is denoted by $f \circ g(\cdot) = f(g(\cdot))$.

II. Problem Statement

Consider the discrete-time nonlinear system

$$\begin{aligned} x^+ &= f(x, u), \\ y &= h(x), \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^{n_x}$ is the current state, x^+ is the successor state, $u \in \mathbb{R}^{n_u}$ is the control input, and $y \in \mathbb{R}^{n_y}$ is the measured output. The system is constrained at each time $k \in \mathbb{N}$ by

$$x(k) \in X, \quad u(k) \in U, \tag{2}$$

where $x(k)$ and $u(k)$ are the state of the system and control input at sampling time k, respectively, and X and U are compact sets in $\mathbb{R}^{n_x}$ and $\mathbb{R}^{n_u}$, respectively. We assume state feedback. In addition, let $\phi(j, x, \mathbf{u})$ denote the solution to (1) at time j given the initial condition is x and the input sequence is $\mathbf{u} = \{u(0), u(1), \dots\}$.

In the case of tracking control problems for constrained nonlinear systems, if the target operating condition is changed, the controller may fail to track the new reference. A widely used approach to NMPC for tracking involves the addition of the steady state and input as decision variables [30]. In this approach, the cost function is formulated as

$$\begin{aligned} V_N(\mathbf{u}, x_s, u_s, y_s; x, r) = \sum_{i=0}^{N-1} \ell(x(i) - x_s, u(i) - u_s) \\ + V_f(x(N) - x_s, y_s) + V_O(r - y_s), \end{aligned} \tag{3}$$

where x_s, u_s, and y_s denote the steady states, inputs, and outputs, respectively, which must satisfy $x_s = f(x_s, u_s)$ and $y_s = h(x_s)$; r is the reference value for the outputs; $\ell : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}_{\geq 0}$ is the positive definite stage cost; $V_f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}_{\geq 0}$ is the positive definite terminal cost; and $V_O : \mathbb{R}^{n_y} \to \mathbb{R}_{\geq 0}$ is the positive definite offset cost. Here, the offset cost is included to penalize the deviation between the artificial steady-state output y_s and the reference r. For feasible reference values and under technical conditions discussed in [30], the offset cost ensures offset-free tracking for asymptotically constant references, i.e., $y(k) \to r$; otherwise, this term ensures convergence to a minimally penalized steady-state output. The tracking NMPC law can then be defined in terms of the solution to the following optimization problem $\mathbb{P}_N(x, r)$

$$\min_{\mathbf{u}, \theta} \quad V_N(\mathbf{u}, \theta; x, r), \tag{4a}$$

$$\text{s.t.} \quad x(i) = \phi(i, x, \mathbf{u}), \ \forall i \in \mathbb{N}_0^N, \tag{4b}$$

$$(x(i), u(i)) \in X \times U, \ \forall i \in \mathbb{N}_0^{N-1}, \tag{4c}$$

$$x_s = g_x(\theta), \ u_s = g_u(\theta), \ y_s = g_y(\theta), \tag{4d}$$

$$(x(N), \theta) \in \Gamma, \tag{4e}$$

where $\theta \in \mathbb{R}^{n_\theta}$ is a parameter that defines the steady-state triplet (x_s, u_s, y_s); $g_x(\theta)$, $g_u(\theta)$, and $g_y(\theta)$ are functions that relate θ to the steady states, inputs, and outputs, respectively; and $\Gamma \subset \mathbb{R}^{n_x + n_\theta}$ denotes an augmented terminal constraint set. This parameter $\theta = y_s$ is typically taken to be the steady outputs. Based on the receding-horizon implementation of NMPC, the control law is given implicitly as

$$\kappa_N(x, r) = u^\star(0; x, r). \tag{5}$$

Although we focus on state feedback in this work, more general output feedback problems can be accommodated by replacing the state x with a corresponding state estimate $\hat{x}$. An output feedback strategy is also able to eliminate offset due to underlying sources of plant-model mismatch through proper choice of a disturbance model and estimator (e.g., [31]). However, for embedded applications, the chosen state estimator must also be embedded. This could require additional implementation effort and would take up hardware resources. Thus, output feedback is not considered here.

In this work, we aim to embed the control law (5) on resource-limited hardware for systems with fast sampling times on the order of milliseconds. To avoid the need to solve a non-convex optimization of the form (4) online, we look to use deep learning methods to build an efficient approximation of the NMPC law offline. The deep learning-based NMPC method is presented in the next section, along with a novel strategy for asymptotically eliminating offset due to controller approximation errors.

III. Deep Learning-based Tracking NMPC

A. Approximate NMPC using deep neural networks

This subsection recalls the principles of approximating the solution to the parametric nonlinear programming problem $\mathbb{P}_N(x, r)$ in terms of a cheap-to-evaluate explicit function. First, one must specify a feasible (or working) region in the augmented space $\mathcal{Z} \subset \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$. The feasible space $\mathcal{Z}$ is then sampled to generate N_s randomly chosen initial states and references, i.e., $\{(x_i, r_i)\}_{i=1}^{N_s}$. For every sample, problem $\mathbb{P}_N(x_i, r_i)$ is solved to obtain the corresponding optimal

input $u_i^\star = \kappa_N(x_i, r_i)$. Using data $\mathcal{D} = \{(x_i, r_i, u_i^\star)\}_{i=1}^{N_s}$, any parametrized function $\kappa_{\text{approx}}(x, r; \lambda)$, where $\lambda \in \mathbb{R}^{n_\lambda}$ are unknown parameters, can be trained by minimizing some loss function such as the mean squared error

$$\hat{\lambda} = \arg\min_{\lambda} \ \frac{1}{N_s} \sum_{i=1}^{N_s} \|u_i^\star - \kappa_{\text{approx}}(x_i, r_i; \lambda)\|^2. \quad (6)$$

Ideally, the data set $\mathcal{D}$ should be chosen to be representative for the intended control application. That is, data $\mathcal{D}$ should cover the span of situations expected to be observed during online operation. We briefly discuss two of the most common sampling strategies, which we refer to as *open-loop* and *closed-loop* training. In open-loop training, one specifies an explicit set such as $\mathcal{Z} = X \times R$, where R represents the set of possible reference values. Although this is a simple strategy, it can result in large spaces that include highly unlikely (or maybe even nonphysical) situations. An alternative is to define the set $\mathcal{Z}$ implicitly in terms of a tube of closed-loop trajectories. In this case, the references can be treated as additional states via $z = (x, r)$. Given some autonomous reference system $r^+ = f_r(r)$, we can arrive at the closed-loop system

$$z^+ = f_z(z) = \begin{bmatrix} f(x, \kappa_N(x, r)) \\ f_r(r) \end{bmatrix}. \quad (7)$$

The working region of the controller is then defined using the tube of trajectories generated by this system from some initial set $Z_0 \subset \mathbb{R}^{n_x+n_y}$ and number of time steps $T \in \mathbb{N}_+$

$$\mathcal{Z} = \textstyle\bigcup_{k=0}^{T-1} Z(k), \quad (8)$$

where $Z(k)$ is the reachable set of states at time $k \in \mathbb{N}_0^{T-1}$. The reachable set is defined recursively as follows

$$Z(k+1) = \{f_z(z) : z \in Z(k)\}, \quad Z(0) = Z_0. \quad (9)$$

Random samples can be generated in the set (8) by performing closed-loop simulations under a randomly drawn initial condition $z_i \in Z_0$; notice that the solution to $\mathbb{P}_N(x, r)$ obtained at every step of the simulation is added to $\mathcal{D}$.

Deep neural networks (DNNs) have recently become a popular choice for the functional form $\kappa_{\text{approx}}(z; \lambda)$.[1] For L hidden layers and H nodes per layer, a DNN is given by

$$\kappa_{\text{approx}}(z; \lambda) = \alpha_{L+1} \circ \beta_L \circ \alpha_L \circ \cdots \circ \beta_1 \circ \alpha_1(z). \quad (10)$$

The hidden layers are made up of affine transformations of the output of the previous layer

$$\alpha_l(\xi_{l-1}) = W_l \xi_{l-1} + b_l, \quad (11)$$

where $\xi_{l-1} \in \mathbb{R}^H$ for $l = 2, \ldots, L+1$ and $\xi_0 = z$. The functions β_l for $l = 1, \ldots, L$ represent nonlinear activation functions (e.g., rectified linear units (ReLU), sigmoid, hyperbolic tangent), which are critical for ensuring (10) is a universal function approximator [32]. The parameter vector $\lambda = \{W_1, b_1, \ldots, W_{L+1}, b_{L+1}\}$ is composed of all weights W_l and biases b_l in the network, with dimensions

$$W_l \in \begin{cases} \mathbb{R}^{H \times (n_x+n_y)} & \text{if } l = 1, \\ \mathbb{R}^{H \times H} & \text{if } l = 2, \ldots, L, \\ \mathbb{R}^{n_u \times H} & \text{if } l = L+1, \end{cases} \quad (12)$$

and

$$b_l \in \begin{cases} \mathbb{R}^H & \text{if } l = 1, \ldots, L, \\ \mathbb{R}^{n_u} & \text{if } l = L+1. \end{cases} \quad (13)$$

Once the network architecture is trained according to (6), the approximate DNN-based NMPC law $\kappa_{\text{approx}}(x, r; \hat{\lambda})$ can be used online to cheaply evaluate the optimal control input.

B. Hyperparameter selection

In practice, there are several *hyperparameters* that appear in training of DNNs, which must be specified before $\hat{\lambda}$ can be computed. Generally speaking, the hyperparameters can be related to the structure of the network, as well as the training algorithm itself. The network structure parameters include the number of nodes H, number of layers L, choice of activation functions $\{\beta_l\}_{l=1}^L$, and the various connections allowed between the layers. The training-related hyperparameters depend on the choice of the solver; typically some variant of stochastic gradient descent (SGD) is utilized that involves parameters related to the batch size (number of samples used to approximate the gradient at each iteration) and learning rate (how quickly the step size decays as number of iterations increase).

Two of the most common approaches for selecting hyperparameters are grid and random search. Grid-based methods are applicable when only a relatively small and finite number of hyperparameter combinations are of interest, implying training (6) can be repeated for all possible combinations. The main difference between random and grid search is that, in the former, the samples are randomly generated from some user-selected distribution. A more sophisticated alternative is to formulate hyperparameter selection as an optimization problem that can be directly tackled with derivative-free optimization methods. Bayesian optimization [33] is a particularly attractive approach since it can handle a mixture of continuous and integer variables and noisy objective evaluations, as well as naturally tradeoff between exploration of the search space and exploitation of the best result obtained at the current iteration (see, e.g., [34]).

An important distinction here is that the hyperparameter optimization problem involves constraints related to resource limitations of the embedded hardware. Thus, the goal is not to obtain the most accurate DNN, but to ensure the final control law satisfies resource utilization and sampling time constraints. These issues are discussed further in Section IV.

C. Offset elimination using steady-state correction factor

Let $\kappa_{\text{DNN}}(x, r) = \kappa_{\text{approx}}(x, r; \hat{\lambda})$ denote the trained DNN control law. Since only a finite amount of data is considered

[1] With slight abuse of notation, we replace (x, r) with augmented state z as the input to the control law in some places for simplicity of exposition.

for training, along with other factors related to local training and fixed DNN structure, the approximate control law introduces some level of error

$$\|\kappa_N(x,r) - \kappa_{\text{DNN}}(x,r)\| \leq \epsilon_{\text{approx}},\ \forall (x,r) \in \mathcal{Z}. \quad (14)$$

Therefore, additional modifications are needed to ensure convergence to the desired target condition in the face of this error. This is particularly important for setpoint tracking problems wherein we expect the system to operate near steady-state for the majority of time.

The basis for the proposed correction factor is the following target tracking optimization problem $\mathbb{P}_s(r)$

$$\min_{x_s,u_s,y_s} \quad \ell_s(y_s,u_s) + V_O(r-y_s), \quad (15a)$$
$$\text{s.t.} \quad x_s = f(x_s,u_s), \quad (15b)$$
$$y_s = h(x_s), \quad (15c)$$
$$(x_s,u_s) \in X \times U, \quad (15d)$$

where $\ell_s : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the steady-state cost function that is assumed to guarantee that $\mathbb{P}_s(r)$ has a unique solution for every $r \in \mathbb{R}^{n_y}$. We let $(x_s^\star(r), u_s^\star(r), y_s^\star(r))$ denote the solution to the tracking optimization (15). The proposed offset-free DNN-based NMPC law then takes the form

$$\kappa_{\text{OF}}(x,r) = \kappa_{\text{DNN}}(x,r) + (u_s^\star(r) - \kappa_{\text{DNN}}(x_s^\star(r),r)), \quad (16)$$

whose properties are summarized next.

Theorem 1: Assume $X \times U$ contains at least one feasible steady state and the offset cost $V_O(\varepsilon)$ satisfies the properties of an exact penalty function [35] for (15), wherein $V_O(\varepsilon)$ contains some penalty weight ρ that is greater than the dual norm of the optimal Lagrange multiplier vector corresponding to the optimal solution to (15) without softened constraint (15c). Then, the offset-free DNN-based NMPC law (16) exists for any $r \in \mathbb{R}^{n_y}$ and guarantees that a closed-loop equilibrium point (x_∞, u_∞) exists such that $r = h(x_\infty)$ for any feasible setpoint r.

Proof: First, $\mathbb{P}_s(r)$ is feasible for all $r \in \mathbb{R}^{n_y}$ when $(x_s,u_s) \in X \times U$ for at least one point $x_s = f(x_s,u_s)$. This is due to the fact that we have softened constraint $r = h(x_s)$ through the inclusion of y_s, i.e., $r = h(x_s) + \varepsilon$ for any $\varepsilon = r - y_s \in \mathbb{R}$. This verifies the first claim. When V_O is an exact penalty function for this constraint, a feasible r implies $x_s^\star(r) = f(x_s^\star(r), u_s^\star(r))$ and $r = h(x_s^\star(r))$. We can now show that the closed-loop system $x^+ = f(x, \kappa_{\text{OF}}(x,r))$ preserves this equilibrium point since $\kappa_{\text{OF}}(x_s^\star(r),r) = \kappa_{\text{DNN}}(x_s^\star(r),r) + (u_s^\star(r) - \kappa_{\text{DNN}}(x_s^\star(r),r)) = u_s^\star(r)$. □

Theorem 1 shows that an offset-free equilibrium point exists for the proposed control law (16); however, it does not prove that the controller makes the system converge to this equilibrium. In fact, it is quite likely that a poorly trained DNN may lead to closed-loop instability. Therefore, it is still necessary to perform some level of stability analysis and/or performance validation before deploying the DNN-based NMPC in practice. For example, see [36], [37] and the references therein for scenario-based methods for performing this type of validation.

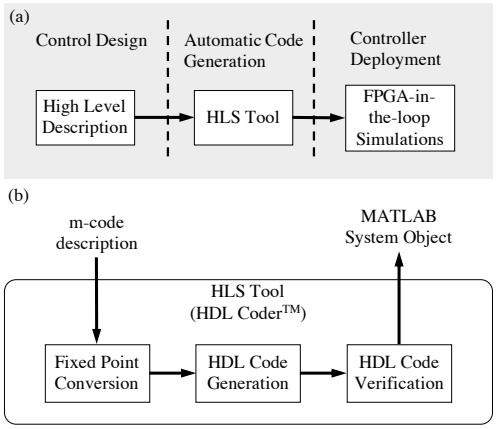

Fig. 1: Design workflow for FPGA Implementation: (a) global overview, (b) HDL Coder™ workflow.

It is interesting to note that, whenever the approximation error $\epsilon_{\text{approx}} = 0$, (16) reduces to $\kappa_{\text{OF}}(x,r) = \kappa_{\text{DNN}}(x,r) = \kappa_N(x,r)$ since the DNN matches the steady-state condition imposed in (4) and the proposed correction term goes to zero. This provides insight into the correction term that effectively corrects for bias around the equilibrium condition. That is, the closer the DNN approximates the NMPC law, the more likely the closed-loop system to be attracted near the equilibrium manifold; and these (fairly small) errors can be overcome as no other equilibrium with the same steady-state input exists. A more formal characterization of the properties that ensure convergence will be the focus of future work.

IV. Embedded FPGA Implementation

In this section, we describe embedded implementation of the offset-free DNN-based NMPC using an FPGA architecture. To this end, we use the HLS design workflow (Fig. 1a), which is a two-step approach that offers an optimized but simple design method for controller deployment on the FPGA. The first step involves creating a suitable high-level description (e.g., C/C++, MATLAB m-files) of the control problem that can be used in a second step by the HLS tool. The output of the HLS tool should be synthesizable hardware description language (HDL) code, which can be used to program the device. There are several HLS tools (Xilinx Vivado HLS, MATLAB HDL Coder™) that can convert a high-level description of an algorithm to a low-level HDL. Recent works have shown the viablility of such tools in simplifying the design process [38].

Despite the simplification in design workflow, several considerations must be made throughout the design process to successfully implement the control scheme. Given a particular target device, there are constraints on the available resources, such as memory, look-up tables (LUTs), flip flops (FFs), or DSP slices. In addition, given the measurement sampling time, there is a constraint on the maximum allowable runtime of the implemented algorithm. Based on these limits, Fig. 2 summarizes the various considerations during each step of the HLS workflow. Because of the limitations

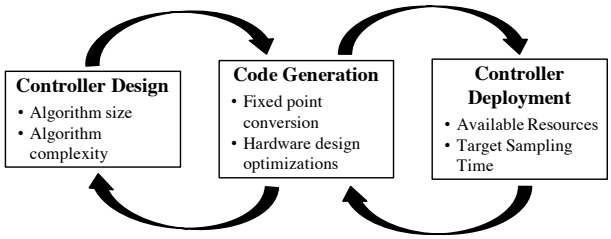

Fig. 2: Design variables to consider at each step of the HLS design workflow.

and the black-box nature of automatic code generation, systematic optimization of these parameters is nontrivial and, consequently, the overall design workflow is an iterative process.

First, a suitable high-level description of the DNN-based NMPC law must be created. Ensuring the satisfaction of resource utilization constraint depends on the size of the DNN-based control law in terms of storage amount and number of operations performed (e.g., additions, multiplications) and the complexity of evaluating the control law (i.e., higher complexity tends towards higher resource utilization).

Considerations of hardware design within the automatic code generation step mainly include conversion to *fixed-point numerics* [9] and *hardware resource optimization* [38]:

1) Fixed-point numerics in hardware design is beneficial for both the speed and resource considerations at the cost of accuracy. Floating-point arithmetic takes advantage of a dynamic representation of values, allowing a wide range of numeric representation. However, floating-point arithmetic incurs high resource utilization and computational delays. Fixed-point arithmetic mitigates this issue at the cost of incurring quantization, overflow, and arithmetic errors [9], which must be considered and optimized accordingly. Within the HLS design workflow, the HLS tool may perform the conversion automatically, allowing the user to tune the conversion by specifying integer and fraction lengths for the fixed-point numbers. Alternatively, it may be necessary for the user to perform the conversion before inputting into the HLS tool. Optimally choosing fixed-point conversion involves balancing the error induced by lower bit representation and preprogrammed hardware specifications (e.g., certain LUTs may optimally perform calculations using 18-bit numerical representations).

2) Hardware resource optimization explores methods to optimize the physical execution of the control law for hardware design. With FPGA devices, the configurability of the logic blocks allows for parallelization techniques such as loop unrolling, pipelining, and inlining, which can provide a more efficient or faster implementation of the control law at the expense of high resource utilization [38]. The challenge, however, is to find the proper combination of these techniques to satisfy both resource utilization and sampling time constraints subject to the designed controller. Within the HLS design workflow, these optimizations are often included as options for the user in the HLS tool, but the level of specificity varies on the HLS tool used. Whether or not such options in the code generation can ultimately be placed on the device-of-choice depends on the designed controller.

Note that we mainly focus on modifying the hyperparameters of the proposed control law $\kappa_{\text{OF}}(x, r)$, rather than exploring manually selected variations in code generation that could be explored in future works. In particular, in this work we selected the combination of nodes and hidden layers that minimize the approximation error subject to the resource limitations of the hardware.

V. Case Study

The performance of the offset-free DNN-based NMPC is demonstrated on a benchmark problem using hardware-in-the-loop simulations. We use the benchmark problem in [39], which considers plant-model mismatch. The nonlinear plant model used in the simulations is described by

$$\begin{aligned}[x]_1^+ &= 0.95[x]_1 - 0.25[x]_1[x]_2 + [x]_2 \\ [x]_2^+ &= 0.7[x]_2 + u,\end{aligned} \tag{17}$$

which is a single-input single-output (SISO) bilinear system. The control goal is to track multiple possible reference values for the first state $[x]_1$ over time. The embedded device is a programmable logic (PL) side (or FPGA side) of a ZYBO Z7 development board by Digilent. Specifications of this hardware are summarized in Table I. This is a low-memory device, suggesting that embedding a nonlinear programming algorithm that solves (4) using an iterative approach would be difficult, especially for relatively large prediction horizons.

To embed the proposed control law (16), we must deal with two elements: the target tracker $\mathbb{P}_s(r)$ and $\kappa_{\text{DNN}}(x, r)$. Since $\mathbb{P}_s(r)$ is inherently a small-scale optimization, it can be embedded directly on the available hardware; however, this requires a specialized (non-trivial) workflow. To greatly simplify this procedure, we note that a second DNN can be trained to approximate $\mathbb{P}_s(r)$. Specifically, the correction term in (16) requires $x_s^\star(r)$ and $u_s^\star(r)$ to be computed from the target tracker at every sampling time. Thus, using the same approach outlined in Section III-A, a DNN can be trained that maps the "input" reference r to the "output" $(x_s^\star(r), u_s^\star(r))$. Since we are interested in achieving negligible offset, we intentionally train the target tracker DNN to have $< 10^{-4}$ mean squared error; this was consistently achievable with networks having only 2 nodes and 4 layers using only 200 random reference samples $\{r_i \in [-1, 1]\}_{i=1}^{200}$.

By running validation tests, we observed that the DNN approximation error of $\mathbb{P}_s(r)$ was negligible for the range of steady-state values of interest. This DNN is assumed fixed and will take up some limited portion of the available resources of the FPGA device. We then use the remaining resources to train $\kappa_{\text{DNN}}(x, r)$ with the smallest possible error. A total of $N_s = 200$ samples were computed by solving (4) in MATLAB using the CasADi toolbox [40]. The training process was performed for a range of hyperparameter values, i.e., $(H, L) \in \{4, 5, 6, 7\} \times \{2, 3, 4\}$. For each hyperparameter, the local optimization (6) was repeated 10 times to mitigate the effects of stochasticity during training. The

TABLE I: Specifications of the FPGA device.

Specification	Value
Device Name	Z-7020
Part Number	XC7Z020
Logic Cells	85K
Total Block RAM	4.9Mb
(# of 36Kb Blocks)	(140)
DSP Slices	220
Look-up Tables (LUTs)	53,200
Flip Flops (FF)	106,400

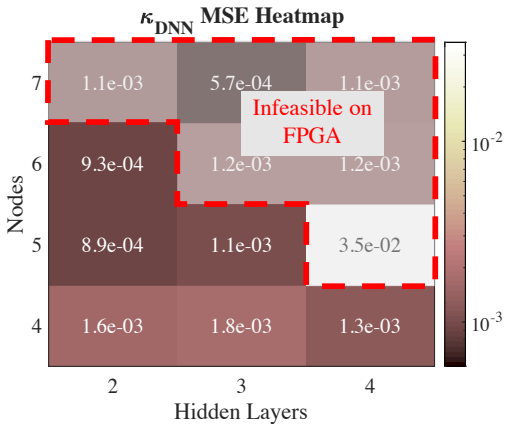

Fig. 3: Heatmap of the MSE of κ_{DNN} for varying number of layers and nodes. The combinations of hyperparameters that could not be placed on the FPGA (infeasible on FPGA) are indicated.

DNN with the lowest validation mean square error (MSE) is reported in Fig. 3. The trend observed in Fig. 3 demonstrates the limitation of the hardware resources available on the FPGA device. Larger DNNs introduce a greater number of physical resource utilization, and, in this case, the number of DSP slices on the FPGA device limited the feasiblity of placing the devised control strategy.

To identify which of these DNN architectures can be feasibly implemented on the FPGA, we attempted to embed the control law $\kappa_{\text{OF}}(x, r)$ onto the device following the procedure in Section IV. In particular, we use the HDL Coder toolbox from Mathworks, which provides an automatic conversion to fixed point and HDL code generation and verification (see Fig. 1b for the HDL Coder-specific workflow). An HDL Coder-friendly m-code evaluation of the complete control scheme is created based on the trained DNNs and used as input to the HDL Coder Workflow Advisor. A fixed-point version of the evaluation was generated using 24-bit word length, with deviations from the floating point representation on the order of 10^{-6}. HDL code in VHDL was then automatically generated and verified within the HDL Coder Workflow Advisor. The MATLAB System object™ generated during FPGA-in-the-loop verification was finally used to perform closed loop simulations.

The resulting FPGA-in-the-loop simulations for two step changes in the desired reference signal are shown in Fig.

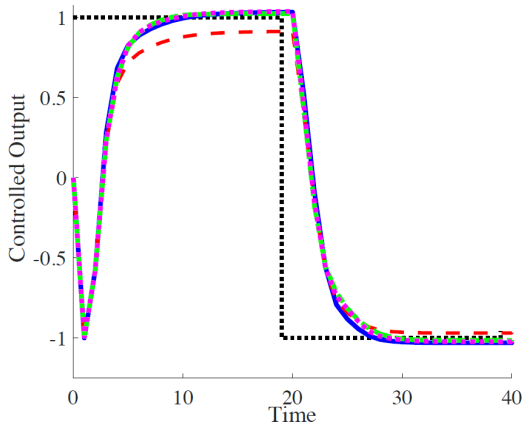

Fig. 4: Closed-loop reference tracking simulations: proposed control law κ_{OF} on CPU (solid blue); κ_{DNN} without correction on CPU (dashed red); exact NMPC on CPU κ_N (solid green); and κ_{OF} on FPGA (dotted magenta).

4. For comparison purposes, we also show results for the NMPC law κ_N, κ_{DNN} without the offset correction, and κ_{OF} with an exact correction term (i.e., $\mathbb{P}_s(r)$ solved online), all of which are run directly in MATLAB using the laptop CPU. We find that the reference tracking of the approximation κ_{OF} (both on the CPU and FPGA) very closely matches the results of the exact NMPC, which indicates that the tracker and NMPC approximation errors are small enough to achieve good performance in practice. Additionally, we observe that, by removing the correction term in κ_{DNN}, a noticeable offset occurs for both reference values, even though the transient response is quite similar. This highlights the importance of the proposed correction term, especially for applications wherein the system is expected to spend the majority of its time near a constant reference signal.

VI. Conclusions and Future Work

This paper proposed a novel correction factor to ensure the elimination of offset in DNN-based approximate NMPC laws. We demonstrated the effectiveness of an embedded DNN-based tracking NMPC controller on a FPGA device using hardware-in-the-loop simulations. Future work will involve creating a systematic framework for embedding deep learning-based NMPC algorithms on resource-limited devices and exploring emerging system-on-chip architectures.

References

[1] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[2] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, pp. 733–764, 2003.

[3] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Saint Paul, Minnesota), pp. 279–284, 2012.

[4] D. Gidon, D. B. Graves, and A. Mesbah, "Effective dose delivery in atmospheric pressure plasma jets for plasma medicine: A model predictive control approach," *Plasma Sources Science and Technology*, vol. 26, no. 8, p. 085005, 2017.

[5] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, (Atlanta), pp. 292–299, 2013.

[6] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, pp. 1391–1403, 2011.

[7] P. Giselsson, M. D. Doan, T. Keviczky, B. De Schutter, and A. Rantzer, "Accelerated gradient methods and dual decomposition in distributed model predictive control," *Automatica*, vol. 49, pp. 829–833, 2013.

[8] M. Kögel and R. Findeisen, "A fast gradient method for embedded linear predictive control," *IFAC Proceedings Volumes*, vol. 44, pp. 1362–1367, 2011.

[9] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[10] M. S. Darup, G. Book, and P. Giselsson, "Towards real-time ADMM for linear MPC," in *Proceedings of the European Control Conference*, pp. 4276–4282, IEEE, 2019.

[11] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, pp. 1–27, 2012.

[12] P. Zometa, M. Kögel, and R. Findeisen, "μAO-MPC: A free code generation tool for embedded real-time linear model predictive control," in *Proceedings of American Control Conference*, (Washington, DC), pp. 5320–5325, 2013.

[13] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, pp. 3–20, 2002.

[14] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," *Automatica*, vol. 39, pp. 489–497, 2003.

[15] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear Model Predictive Control*, pp. 345–369, Springer, 2009.

[16] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range," *Automatica*, vol. 47, pp. 2279–2285, 2011.

[17] F. Debrouwere, M. Vukov, R. Quirynen, M. Diehl, and J. Swevers, "Experimental validation of combined nonlinear optimal control and estimation of an overhead crane," *IFAC Proceedings Volumes*, vol. 47, pp. 9617–9622, 2014.

[18] T. Albin, D. Ritter, D. Abel, N. Liberda, R. Quirynen, and M. Diehl, "Nonlinear MPC for a two-stage turbocharged gasoline engine airpath," in *Proceedings of the IEEE Conference on Decision and Control*, (Osaka, Japan), pp. 849–856, 2015.

[19] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit – An open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, pp. 298–312, 2011.

[20] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (GRAMPC)," *Optimization and Engineering*, vol. 20, pp. 769–809, 2019.

[21] M. Kvasnica, J. Löfberg, and M. Fikar, "Stabilizing polynomial approximation of explicit MPC," *Automatica*, vol. 47, no. 10, pp. 2292–2297, 2011.

[22] Y. Oishi, "Simplified approaches to polynomial design of model predictive controllers," in *Proceedings of the IEEE International Conference on Control Applications*, (Hyderabad, India), pp. 960–965, 2013.

[23] 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 *Proceedings of the American Control Conference*, (Milwaukee), pp. 1520–1527, 2018.

[24] 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, 2020.

[25] A. D. Bonzanini, J. A. Paulson, D. B. Graves, and A. Mesbah, "Toward safe dose delivery in plasma medicine using projected neural network-based fast approximate NMPC," in *Proceedings of the IFAC World Congress*, (Berlin), pp. 5353–5359, 2020.

[26] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an approximate model predictive controller with guarantees," *IEEE Control Systems Letters*, vol. 2, pp. 543–548, 2018.

[27] 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, pp. 719–724, 2020.

[28] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos, and A. Mesbah, "Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks," *Computers & Chemical Engineering*, vol. 145, p. 107174, 2021.

[29] B. Karg and S. Lucia, "Stability and feasibility of neural network-based controllers via output range analysis," *arXiv preprint arXiv:2004.00521*, 2020.

[30] D. Limon, A. Ferramosca, I. Alvarado, T. Alamo, and E. F. Camacho, *MPC for Tracking of Constrained Nonlinear Systems*, pp. 315–323. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[31] G. Pannocchia and J. B. Rawlings, "Disturbance models for offset-free model-predictive control," *AIChE Journal*, vol. 49, no. 2, pp. 426–437, 2003.

[32] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, pp. 930–945, 1993.

[33] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

[34] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, pp. 148–175, 2015.

[35] E. C. Kerrigan and J. M. Maciejowski, "Soft constraints and exact penalty functions in model predictive control," in *Proceedings of the UKACC International Conference*, (Cambridge), 2000.

[36] J. A. Paulson and A. Mesbah, "Data-driven scenario optimization for automated controller tuning with probabilistic performance guarantees," *IEEE Control Systems Letters*, vol. 5, no. 4, pp. 1477–1482, 2020.

[37] D. Krishnamoorthy, A. Mesbah, and J. A. Paulson, "An adaptive correction scheme for offset-free asymptotic performance in deep learning-based economic MPC," in *Proceedings of the 11th IFAC Symposium on Advanced Control of Chemical Processes*, (Venice, Italy), 2021 (accepted).

[38] S. Lucia, D. Navarro, Ó. Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, 2017.

[39] M. Morari and U. Maeder, "Nonlinear offset-free model predictive control," *Automatica*, vol. 48, no. 9, pp. 2059–2067, 2012.

[40] 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, 2019.