Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 56-2 (2023) 6298–6303

Learning-based NMPC on SoC platforms for real-time applications using parallel Lipschitz interpolation[⋆]

J. M. Nadales[*] A. D. Carnerero[*] C. Moreno-Blazquez[*] R. M. Haes-Ellis[*] D. Limon[*]

[*] *Department of Automation and Systems Engineering, University of Seville, Seville, Spain (e-mail: nadales,acarnerero,dlm@us.es)*

Abstract: One of the main problems associated with advanced control strategies is their implementation on embedded and industrial platforms, especially when the target application requires real-time operation. Frequently, the dynamics of the system are totally or partially unknown, and data-driven methods are needed to learn an approximate model of the plant to control. On many occasions, these learning techniques use non-differentiable functions that cannot be handled by most traditional low-level gradient-based optimization methods. In addition, many data-driven techniques require the online processing of a vast amount of data, which may be exceedingly time-consuming for most real-time applications. To solve these two problems at once, we propose a low-cost solution based on a system on a chip (SoC) platform featuring an embedded microprocessor (MP) and a field programmable gate array (FPGA) to implement nonlinear model predictive control strategies. The model employed to make predictions about the future evolution of the system is learnt by means of a data-driven learning method know as parallel Lipschitz interpolation (LI) and implemented in the FPGA part. On the other hand, the optimization problem associated with the model predictive control strategy is solved by software in the MP using an adapted version of the particle swarm optimization method.

Keywords: Learning-based Control, Nonlinear Model Predictive Control, Lipschitz Interpolation, Particle Swarm Optimization, SoC platforms, HW/SW Design.

1. INTRODUCTION

Despite its ever-growing popularity among the research community, many people are still reluctant to apply advanced control laws in industry, as in the case of nonlinear model predictive control (NMPC) laws (see Goodwin et al. (2012)). In this sense, there are two hurdles to overcome. The first one is the difficulty of implementing these advanced control laws on embedded and industrial platforms and the associated costs. The second one is to make the controller robust to model uncertainties, external disturbances, and measurement noise.

The first problem can be tackled by two different perspectives. The first one is to develop new and efficient algorithms adapted to the requirements of the target system where the control law will be implemented. See, for instance, the work developed in Krupa et al. (2020), where a tool for the automated code generation of standardized IEC 61131-3 PLC programming languages is employed to solve the MPC optimization problem. The alternative is to design new system architectures that respond to the needs of the algorithm to be implemented, as it is done in Knagge et al. (2009), where a novel application specific integrated circuit (ASIC) is proposed for the implementation of online MPC control laws.

The second problem is that the controller must respond to occurrences such as changes in the model of the system to control, and has traditionally been addressed by developing adaptive control laws, as it is done in Valluru and Patwardhan (2019), where an MPC scheme based on Bayesian and a frequent real time optimizer (RTO) is proposed.

To solve all these problems associated with the implementation of NMPC control strategies, in this work the following approach is proposed. The —in principle unknown— dynamics of the system to control are learnt by means of a data-driven learning technique known as Lipschitz interpolation (LI) (see Beliakov (2006)). Lipschitz interpolation is a learning methodology that allows the regression of unknown functions given they are Lipschitz continuous using data obtained through performing tests on the system. As detailed in Nadales et al. (2022), some of the reasons why LI is employed among all available learning methodologies are the following:

- The algorithm is simple, it only requires basic operations, and it can be parallelized, reasons why it can easily be implemented in field programmable

[⋆] This work was supported by the Agencia Estatal de Investigación (AEI)- Spain under Grant PID2019-106212RB-C41/AEI/10.13039/501100011033, by Ministerio de Ciencia e Innovación under Proyect PDC2021-121120-C21, and by Fondo Europeo de Desarrollo Regional FEDER under Programa Interreg V-A España-Portugal (POCTEP) 2014-2020.

gate array (FPGA) platforms using low-level design methodologies.
- Bounds on the error committed from data representation can be found by design, something interesting for robustness purposes.

The main problem associated with LI is that this family of learning methodologies usually returns non-differentiable outputs, and because of that, conventional optimization techniques based on gradient methods cannot be employed. To overcome this difficulty and solve the NMPC problem, in this work we propose the implementation of a solver based on an adapted version of the algorithm known as particle swarm optimization (PSO) (see Poli et al. (2007)). These kinds of genetic algorithms are usually used to solve non-convex or non-differentiable problems and have also been addressed in the field of MPC-like schemes (see Carnerero et al. (2021)).

To implement the controller, in this paper we make use of a system on chip (SoC) embedded platform composed of an embedded microprocessor (MP) and FPGA-based programmable logic. Using this type of platform, the NMPC problem is implemented and solved on the soft part of the SoC (the MP) using the PSO algorithm and the model of the system is implemented as an external hard peripheral on the FPGA using the parallel version of the LI algorithm proposed in Nadales et al. (2022). By doing this, we accelerate the time it takes to make predictions required by the PSO solver. This allows real-time implementation of the Algorithm.

The rest of the paper is structured as follows. In Section 2, the mathematical fundamentals of parallel LI (PLI) are briefly described. In Section 3, the modified version of the PSO algorithm for solving NMPC control problems is presented. In Section 4 the HW/SW partition for the proposed implementation of the PSO solver using PLI is detailed. In Section 5, the proposed methodology is employed to control a self-balancing helix-bar system. Finally, the main conclusions of the this work are summarized in Section 6.

2. LIPSCHITZ INTERPOLATION

In this work we suppose that the dynamics of the system to be controlled are unknown a priori. It is assumed that the only available information about system is a set of historical data containing a collection of measured input-output pairs. It is also assumed that this set of data can be employed to describe a model of the system by means of a nonlinear autoregressive exogenous (NARX) model of the plant (see Leontaritis and Billings (1985)). Let $y(k) \in \mathbb{R}^{n_y}$ be the measured output of the system, and $u(k) \in \mathbb{R}^{n_u}$ the control input. Both inputs and outputs are subject to hard constraints, $y(k) \in \mathcal{Y}$, $u(k) \in \mathcal{U}$. The objective is to find a predictor of the future system evolution of the form

$$y(k+1) = f(x(k), u(k), e(k)), \tag{1}$$

where $x(k) = (y(k), y(k-1), ..., y(k-n_a), u(k), u(k-1), ..., u(k-n_b)) \in \mathcal{X} := \mathcal{Y}^{n_a+1} \times \mathcal{U}^{n_b+1}$ for some memory horizon lengths, n_a and n_b, whose value can be estimated through some cross-validation method. The term $e(k)$ accounts for the process noise and is assumed to be confined in a compact set $\mathcal{E} \in \mathbb{R}^{n_y}$. For notation simplicity and compactness, the inputs of f are aggregated into a single vector $w = (x, u) \in \mathcal{W}$, which is referred to as regressor.

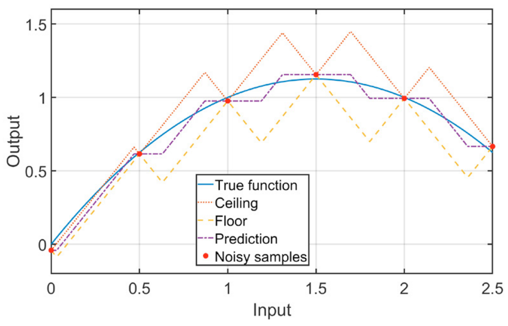

Fig. 1. Lipschitz interpolation over $f(w) = \frac{-w^2}{2} + \frac{3w}{2}$ with $N_{\mathcal{D}} = 6$ and $L = 1.5$

.

To estimate f, a parallel version of the learning-based algorithm known as LI is employed (see Beliakov (2006)). This parallel version of the algorithm is first proposed in Nadales et al. (2022), where the algorithm is employed to learn nonlinear model predictive control laws for systems operating in real time. The algorithm can be summarized as follows: consider the dynamics of the system are given by an unknown function $f : \mathcal{W} \subset \mathbb{R}^{n_u} \to \mathcal{Y} \subset \mathbb{R}^{n_y}$, where n_w is the number of inputs and n_y the number of outputs. After performing several tests of the system, a set of $N_{\mathcal{D}}$ (possibly) noisy input/output pairs is obtained, expressed as

$$\mathcal{D} = \{(\tilde{f}(w_i), w_i) \mid i = 1, ..., N_{\mathcal{D}}\}, \tag{2}$$

where $w \in \mathcal{W}$ and $\tilde{f}(\cdot) \in \mathcal{Y} \oplus \mathcal{E}$ are the inputs of the system and noisy observation of the real output $f(\cdot) \in \mathcal{Y}$, respectively, and where it is assumed that the noise is additive and confined in a compact set $\mathcal{E} \in \mathbb{R}^{n_y}$. It is also assumed that the function f is Lipschitz continuous with Lipschitz constant $L \in \mathbb{R}^{n_y}$, i.e., for each component of the output $j = 1, ..., n_y$,

$$\|f_j(w) - f_j(w^*)\| \leq L_j \|w - w^*\| \; \forall w, w^* \in \mathcal{W}. \tag{3}$$

As detailed in Calliess (2014), the resulting prediction of f, qiven a new input $q \in \mathcal{W}$ is calculated as

$$\mathfrak{f}_j(q; L_j, \mathcal{D}) = \frac{1}{2} \min_{i=1,...,N_{\mathcal{D}}} (\hat{f}_{i,j} + L_j \|q - w_i\|) \tag{4}$$

$$+ \frac{1}{2} \max_{i=1,...,N_{\mathcal{D}}} (\hat{f}_{i,j} - L_j \|q - w_i\|) \tag{5}$$

$$= \frac{1}{2} \min_{i=1,...,N_{\mathcal{D}}} \mathfrak{u}_i + \frac{1}{2} \max_{i=1,...,N_{\mathcal{D}}} \mathfrak{l}_i \tag{6}$$

$$= \frac{1}{2}(\mathfrak{u} + \mathfrak{l}), \tag{7}$$

where $\mathfrak{f}_j$ is the j-th component of the prediction, $\hat{f}_{i,j}$ the j-th component of the value of the observed map for the i-th data point in $\mathcal{D}$ and w_i is its corresponding input. The terms $\mathfrak{u}$ and $\mathfrak{l}$ are called the *ceiling* and *floor* functions, respectively. Note that any norm defined in the input space can be employed in the previous expressions (see Calliess (2014)). As detailed in Nadales et al. (2022), given a new input q, and the set of stored data $\mathcal{D}$, all ceiling and floor terms, $\mathfrak{u}_i$ and $\mathfrak{l}_i$ can be calculated at the same time, and then a tree of compactors can be employed to find the minimum and maximum values among them, respectively. Finally, these values $\mathfrak{u}$ and $\mathfrak{l}$ can be added together and divided by two to obtain the final predicted output. The Algorithm is represented in Figure 1.

Remark 1. The true value of the Lipschitz constant employed in the previous expressions is unknown a priori. Detailed information about how this value can be estimated using data can be found in Calliess et al. (2020).

Note that, in the previously described algorithm, every time a new input q is received all data points in the training data set $\mathcal{D}$ are processed to obtain the desired prediction. It could easily happen that the amount of area resources of the FPGA platform where the algorithm is intended to be implemented are not sufficient to process all data in parallel. A possible solution is to establish a sequential procedure, but computational times could be exceedingly high. Because of this, in this work we adapt the projected version of the LI algorithm as proposed in Manzano et al. (2019). In this version of the LI algorithm, the input space $\mathcal{W}$ is divided into several subspaces $\mathcal{W}_k$, such that $\mathcal{W} = \cup_{k=1}^{K} \mathcal{W}_k$ and their intersection is null, i.e., $\mathcal{W}_i \cap \mathcal{W}_j = \oslash \; \forall i \neq j$. To make this partition, the function $\psi : \mathcal{W} \rightarrow \mathcal{W}_k$ is defined. Then, every time a new input $q \in \mathcal{W}$ is received, only the data points belonging to the corresponding space division are employed to make the prediction, i.e.,

$$\mathfrak{f}^*_j(q; L, \mathcal{D}) = \mathfrak{f}_j(q; L, \mathcal{D}_k), \tag{8}$$

where

$$\mathcal{D}_k = \{(\hat{f}_{i,j}, w_i) \mid w_i \in \mathcal{W}_k\} \tag{9}$$

is the subset of data to be processed for some particular query point q.

3. PSO FOR NMPC PROBLEMS

In this section, we present an adapted version of the PSO algorithm to solve NMPC control problems where the LI algorithm is employed to model the system to control.

3.1 Basic PSO algorithm

The PSO algorithm was first introduced in Kennedy and Eberhart (1995). In this paper, the version of the algorithm detailed in Poli et al. (2007) will be used.

Consider the optimization problem

$$\min_{v} \quad J(v) \quad s.t. \quad \underline{v} \leq v \leq \overline{v}, \tag{10}$$

where $v \in \mathbb{R}^n$ is the vector of decision variables, $\underline{v}$ and $\overline{v}$ are vectors defining the lower and upper bounds on v and $J : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function. Note that we do not assume convexity or differentiability on this function.

In this kind of algorithms, a set of possible candidate solutions (called particles) are evaluated at each iteration and, depending on the performance of each particle, new candidate solutions are proposed. This new set of particles might be closer to the global optimum of the optimization problem. This process continues until some stopping criteria is satisfied. For simplicity reasons, we consider here a fixed number of iterations.

A certain number of particles $\boldsymbol{p}_i$, $i = 1, ..., K$, are randomly placed in the feasible search space of the function to be optimized. This search space, denoted as Ω, is defined by the box constraints in v. The candidate solution of the optimization problem for a certain particle $\boldsymbol{p}_i$ is denoted as $\boldsymbol{p^c_i} \in \mathbb{R}^n$. This candidate solution has an associated cost, denoted as $\boldsymbol{p^J_i} \in \mathbb{R}$. Also, each particle saves its previous best candidate solution, $\boldsymbol{p^b_i} \in \mathbb{R}^n$, and the best objective value reached so far by the particle, $\boldsymbol{p^{J_b}_i} \in \mathbb{R}$. Moreover, each particle $\boldsymbol{p}_i$ is aware of the best global candidate solution reached by any particle in the search space. This is denoted as $\boldsymbol{p^g} \in \mathbb{R}^n$ whereas the best global cost is denoted as $\boldsymbol{p^{J_g}} \in \mathbb{R}$. Finally, each particle has an associated inertia associated, usually called velocity. This is denoted as $\boldsymbol{p^v_i} \in \mathbb{R}^n$.

After the initialization step, the objective function is evaluated for every particle. Once this is done, the best previous candidate solutions and costs of each particle are replaced if better solutions were attained. In case that a new solution is better than the previous global solution, the best global particle is updated. Let $\boldsymbol{U}_n(a, b)$ be a $n-$dimensional uniform random distribution where each component can take values between a and b. Then, the velocity of each particle is updated as

$$\boldsymbol{p}^v_i \leftarrow \xi(\boldsymbol{p^v_i} + P_1 \odot (\boldsymbol{p^b_i} - \boldsymbol{p^c_i}) + P_2 \odot (\boldsymbol{p^g} - \boldsymbol{p^c_i})), \tag{11}$$

where $P_1 \sim \boldsymbol{U}_n(0, \phi_1)$ and $P_2 \sim \boldsymbol{U}_n(0, \phi_2)$, and ϕ_1, ϕ_2 and ξ are constants whose values are

$$\phi_1 = \phi_2 = 1.496, \; \phi = \phi_1 + \phi_2, \; \xi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}},$$

and where $\odot$ stands for component-wise multiplication. P_1 and P_2 are random variables employed to prevent the algorithm from getting stuck in a local minima. Finally, the new set of candidates is computed as

$$\boldsymbol{p^c_i} \leftarrow \boldsymbol{p^c_i} + \boldsymbol{p^v_i}. \tag{12}$$

After the new set of candidates is computed, we need to verify that they do not violate the box constraints imposed by equation (10). In case a certain new candidate fall outside the imposed limits, it will be projected to the nearest feasible value satisfying the constraints.

3.2 PSO for NMPC using LI

In this subsection, we adapt the PSO algorithm to implement NMPC controllers (see Grüne and Pannek (2017)) employing the LI algorithm to make predictiona about the evolution of the system. The stage cost $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ and the terminal cost $V_T : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ are defined as

$$l(x, u) := \frac{1}{2}\Big(\|x\|^2_Q + \|u\|^2_R\Big), \quad V_T(x) := \lambda\|x\|^2_P, \tag{13}$$

where $Q \in \mathbb{R}^{n_x \times n_x}$, $R \in \mathbb{R}^{n_u \times n_u}$, $P \in \mathbb{R}^{n_x \times n_x}$ and $\lambda \in \mathbb{R}$ are design parameters employed to tune the controller and ensure stability. Thus, the performance index to optimize is

$$V_{N_p}(\mathbf{x}, \mathbf{u}) = \sum_{j=0}^{N_p - 1} l(\hat{x}(j \mid k), u(k + j)) + V_T(\hat{x}(N_p \mid k)),$$

where $\mathbf{u} \in \mathbb{R}^{N_p \times n_u}$ is the set of control actions, $\mathbf{x} \in \mathbb{R}^{N_p \times n_x}$ is the set of predicted states, $\hat{x}(j \mid k) \in \mathbb{R}^{n_x}$ is the state $x(k + j)$ predicted at time k, $u(k + j) \in \mathbb{R}^{n_u}$ is the control action to be applied at time instant $k + j$, x_k is the initial state at time k and N_p is the prediction horizon.

Algorithm 1 PSO Algorithm for NMPC using LI

```
1: Initialize $\boldsymbol{p_i}$, $\forall i = 1, ..., K$
2: while the stopping criteria is not satisfied do
3:     for $i = 1 : K$ do
4:         Compute the trajectory $t_i(x_k, \boldsymbol{p_i^c})$ using LI.
5:         Evaluate $V_{N_p}(t_i, \boldsymbol{p_i^c})$.
6:         Update $\boldsymbol{p_i^{J_b}}$ and $\boldsymbol{p_i^b}$ if necessary.
7:     end for
8:     Find the particle with the best cost.
9:     Update $\boldsymbol{p^{J_g}}$ and $\boldsymbol{p^g}$ if necessary.
10:    for $i = 1 : K$ do
11:        Compute the velocity as in equation (11).
12:        Compute the new candidate as in equation (12).
13:        Project the candidate solution onto $\Omega$.
14:    end for
15: end while
16: Choose $\mathbf{u}$ as the candidate solution of the particle that
    obtained the best cost, that is $\boldsymbol{p_i^g}$.
```

Then, the problem to solve at each sampling time k can be written as

$$\min_{\mathbf{x},\mathbf{u}} \; V_{N_p}(\mathbf{x}, \mathbf{u}) \tag{14a}$$

$$\text{s.t } \hat{x}(0 \mid k) = x(k) \tag{14b}$$

$$\hat{x}(j+1 \mid k) = f_{LI}(\hat{x}(j \mid k), u(k+j), \mathcal{D}), \tag{14c}$$

$$\underline{u} \leq u(k+j) \leq \overline{u}, \tag{14d}$$

where the function $f_{LI} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_u}$ is the predictor of the system using LI and $\underline{u}$ and $\overline{u}$ are lower and upper bounds on the input respectively. Note that constraint (14b) sets the initial state, constraint (14c) defines the predictor of the state evolution using the LI algorithm and constraint (14d) defines box constraints on the control action, respectively. Also, note that it is assumed that the origin is an equilibrium point of the system.

In the adapted version of the PSO algorithm for soling NMPC problems, the dimension of the particles will be given by the number of control actions and the prediction horizon of the optimization problem problem, i.e., $\boldsymbol{p_i^c}, \boldsymbol{p_i^b}, \boldsymbol{p_i^g}, \boldsymbol{p_i^v} \in \mathbb{R}^{N_p \times n_u}$. Thus, in order to be able to compute the velocities as in equation (11), we assume that $P_1 \sim \boldsymbol{U}_{N_p \times n_u}(0, \phi_1)$ and $P_2 \sim \boldsymbol{U}_{N_p \times n_u}(0, \phi_2)$, where $\boldsymbol{U}_{n_1 \times n_2}(a, b)$ corresponds to a $(n_1 \times n_2)$-dimensional uniform random distribution where each component takes values between a and b. Also, denote $t_i(x(k), \boldsymbol{p_i^c}) \in \mathbb{R}^{N_p \times n_x}$ as the trajectory obtained by calling the LI predictor iteratively with an initial state $x(k)$, that is, $t_i(x_k, \boldsymbol{p_i^c}) = [\hat{x}(0 \mid k) \; \hat{x}(1 \mid k) \; \ldots \; \hat{x}(N_p \mid k)]$. Then, it is easy to see that the total cost $V_{N_p}(\cdot)$ for each particle can be computed as long as we have t_i and $\boldsymbol{p_i^c}$. Besides that, the algorithm does not need further modifications. The adapted PSO procedure for solving NMPC problems using LI is summarized in Algorithm 1.

4. HW/SW PARTITION

In this work we propose to employ SoC platforms (see Hübner and Becker (2010)). These systems can typically be divided into two different parts: a processing system (PS), such as an embedded MP, and a programmable logic device (PL), such as a FPGA platform. The PS is the place where the operative system, the main code of the application, and the different software interfaces are

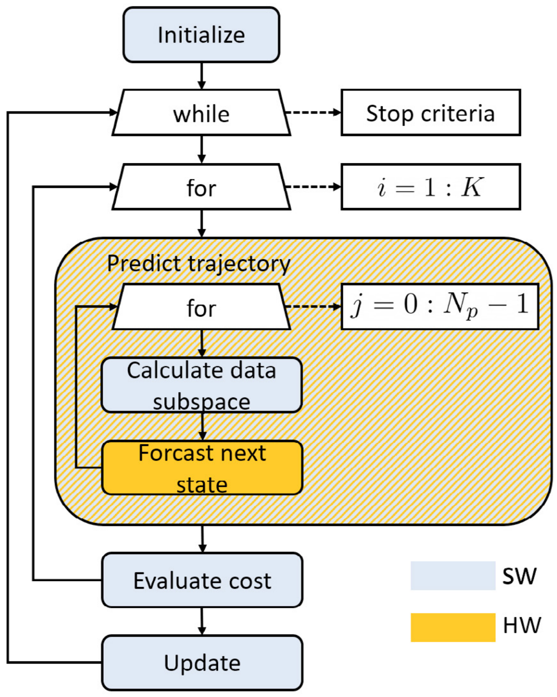

Fig. 2. HW/SW partition of Algorithm 1.

implemented. On the other hand, the PL, also known as the hard part, is ideal for implementing high speed logic, arithmetic and data flow circuits, and algorithms prone to be implemented in a parallel manner, thus reducing computational times.

To implement the proposed control strategy, the HW/SW co-design methodology has been adopted (see Schaumont (2012)). This design methodology proposes to simultaneously design the different parts of the main application to be implemented in both the PS and the PL part. The first step is to partition the main application and decide what is going to be implemented using software on the PS and what is going to be implemented using hardware on the PL. A common approach is to build the main application on the PS and employ the PL to implement different modules to accelerate code execution, for example, by parallelizing parts of the application that would otherwise have to be implemented sequentially on the PS part. These modules are called peripherals and serve as co-processors to the main processing unit.

The proposed HW/SW partition for implementing Algorithm 1 is represented in Figure 2. The boxes shaded in blue represent those tasks that will be implemented using software on the PS while the boxes shaded in orange represent those tasks that will be implemented using hardware on the PL. As it can be appreciated, everything is implemented using software on the PS except for the model of the system. For each particle $i = 1, ..., K$, its corresponding future trajectory is recursively forecast calling the LI algorithm implemented in the PL. Before that, the associated subspace in the data set is calculated for each particle. This information, together with the new input data q, is sent as input to the LI algorithm, which returns the predicted next state of the system.

Once the HW/SW partition has been decided, each of the different parts of the main application are concurrently implemented and tested. For more details about the implementation of the parallel version of the LI algorithm on the PL part, the reader is referred to Nadales et al. (2022) where an extensive and detailed description of how to efficiently implement parallel LI algorithms on FPGA platforms is provided.

5. CASE STUDY

To illustrate the performance of the proposed methodology, this has been applied to a real-time control problem: the real-time implementation of a constrained nonlinear model predictive control law for a helix-bar system.

5.1 Hardware architecture

The proposed design has been experimentally implemented and tested on a Xilinx Zynq-7000 SoC platform embedded on a Zybo z7 board which integrates a dual-core ARM Cortex-A9 processor with a Xilinx 7-series FPGA. For details about the hardware implementation of the proposed architecture the reader is referred Crockett et al. (2014) where a complete guide for the implementation of HW/SW systems on the Zynq platform is provided. The LI algorithm for modeling the system's behavior is implemented in the PL as an AXI peripheral connected to the PS through an AMBA AXI on-chip communication protocol (see Gaikwad and Patil (2018)). The rest of the tasks detailed in Algorithm 1 are implemented in C code on one of the two ARM cores of the ARM processor. The other ARM core is employed to implement the model of the system to perform processor in the loop (PIL) simulations.

5.2 Model of the plant

The proposed control design methodology is applied to the self-balancing helix-bar system. The objective is to control the angle that the bar forms with the vertical using the force provided by the air current generated by the helix located at the extreme of the bar. Note that a spring is placed connecting the bar to the wall to provide rigidity to the system. The state of the system is defined by the angle that the bar forms with the vertical, $\theta[rad]$, and its angular velocity, $\dot{\theta}[rad/s]$. The state varies with respect to the control input, which is the force $F[N]$ generated by the helix located at the extreme of the bar, according to the following set of ordinary differential equations (ODEs), which will be employed to simulate the behavior of the system but are supposed to be unknown:

$$\begin{aligned}\dot{x}_1(t) &= x_2(t),\\ \dot{x}_2(t) &= -\frac{1}{I}\left[\left(\frac{g\cdot M\cdot L}{2} + X_1\cdot g\cdot M_1\right)sx_1(t)\right.\\ &\quad \left. + (X_2^2\cdot K)s^2x_1(t) + Bx_2(t) - X_1u(t)\right],\end{aligned} \tag{15}$$

where s denotes the sine function, $\theta(t)$ is represented by state $x_1(t)$, $\dot{\theta}(t)$ by state $x_2(t)$, and the control action $F(t)$ by $u(t)$. In the model, $g[m/s^2]$ represents the force of gravity , $M[kg]$ the mass of the bar, $M_1[kg]$ the mass of the helix and its engine, $L[m]$ the length of the bar, $X_1[m]$ the distance between the helix and the point of the supporting spring connecting the bar and to the wall, $X_2[m]$ the distance between this same point and the wall along the length of the bar, $K[N/m]$ is the characteristic constant of the spring, $B[kg\cdot m/s]$ is the friction constant of the air, and $I[kg\cdot m^2]$ the moment of inertia of the whole system. The values these parameters take are detailed in Table 1.

Table 1. Parameters of the helix-bar system.

g	$9.8\ m/s^2$	X_2	$0.5\ m$
L	$1\ m$	K	$2.82\ N/m$
M	$0.05\ kg$	B	$0.1\ kg\cdot m/s$
M_1	$0.1\ kg$	I	$0.08\ kg\cdot m^2$
X_1	$1\ m$		

5.3 Learning-based model

The first step before implementing the model of the plant using the parallel LI algorithm is to obtain the training data set. To do that, several simulations on the real model of the system have been carried out in which the system is excited with different chirp signals ranging from 0.03 Hz to 0.3 Hz for a random initial state. From these simulations, and after a filtering process, a data set consisting of 16384 uniformly distributed input-output pairs is obtained. These input-output pairs will be employed to build the training data set.

Using this set of data, and after a cross-validation procedure, different estimations of f_{LI} for different values of n_a and n_b are obtained. Following the methodology proposed in Calliess et al. (2020), the values of the Lipschitz constant L and the memory horizons n_a and n_b have been calculated and set to $L = 1.935$, $n_a = 2$, $n_b = 1$ for a sampling period $T_m = 0.0342\ s$. The predictor of the system for these values of n_a and n_b takes the form

$$y(k+1) = f_{LI}(y(k), y(k-1), y(k-2), u(k), u(k-1)), \tag{16}$$

where $y(k)$ is the real state of the plant, which is assumed to be accessible, and the control action $u(k)$ is the force generated by the helix. Using this information, and the set of input-output pairs, the training data set $\mathcal{D}$ is obtained. Because not enough area resources are available to process all data at once, a partition of the training data set is carried out as detailed in Manzano et al. (2019). Doing this, the training data set is partitioned into 64 subsets each of them containing a total of 256 data points, and the sequential procedure detailed in Nadales et al. (2022) is followed to process all partitions of the data set. To represent data, fixed-point representation format has been employed following the procedure detailed in Nadales et al. (2022) to find the optimal representation format. After implementing the system and performing a time analysis, a clock signal of 10 ns has been selected, which provides a positive worst negative slack (WNS) (see Chadha and Bhasker (2009)).

5.4 NMPC using PSO

To solve the optimization problem, Algorithm 1 is implemented in C code in one of the ARM cores part of the ARM

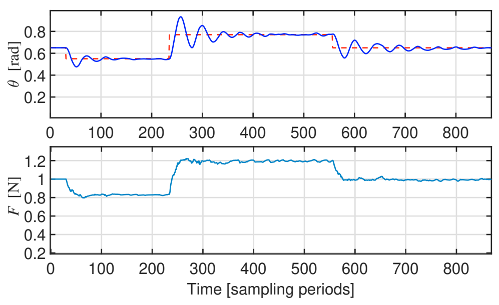

Fig. 3. Simulation results of the system controlled by the proposed HW/SW controller using PSO and parallel LI.

processor. The total number of particles has been set to 90. Instead of a stopping criteria, a total of 50 iterations has been fixed for each iteration of the optimization algorithm. The prediction horizon is set to $N_p = 5$, and the values of the weighting matrices that ponder the performance index are set to $Q = \text{diag}[100, 10]$, $R = 50$. Since the model of the plant is unknown, the value of the terminal cost matrix P has been obtained using the set of input-ouput pairs following the guidelines detailed in Berberich et al. (2021) and has been set to $P = [14.7782, 0.1344; 0.1344, 5.011]$. The sampling time is set to $T_m = 0.0342$.

5.5 Simulation results

Using the model of the system implemented in the second ARM core part of the MP, closed-loop processor in the loop simulations of the system controlled by the proposed HW/SW controller have been carried out. The results of the closed-loop system for random initial conditions and under the presence of bounded sensor noise are shown in Figure 3. As it can be seen, the implemented control law stabilizes the system toward the desired set point.

6. CONCLUSIONS

A novel HW/SW design methodology for the implementation of nonlinear model predictive controllers on SoC platforms was presented. The optimization problem is solved using an adapted version of the particle swarm optimization algorithm implemented in software on the processing system. The model of the system is implemented in the FPGA using a parallel version of the algorithm known as Lipschitz interpolation. This allows real-time implementation given the time needed to make prediction when the standard sequential version of the algorithm is employed. Simulation results show how the proposed controller is able to stabilize the system toward the desired set point.

REFERENCES

Beliakov, G. (2006). Interpolation of lipschitz functions. *Journal of computational and applied mathematics*, 196(1), 20–44.

Berberich, J., Köhler, J., Müller, M.A., and Allgöwer, F. (2021). On the design of terminal ingredients for data-driven MPC. *IFAC-PapersOnLine*, 54(6), 257–263.

Calliess, J.P. (2014). *Conservative decision-making and inference in uncertain dynamical systems.* Ph.D. thesis, University of Oxford.

Calliess, J.P., Roberts, S.J., Rasmussen, C.E., and Maciejowski, J. (2020). Lazily adapted constant kinky inference for nonparametric regression and model-reference adaptive control. *Automatica*, 122, 109216.

Carnerero, A.D., Ramirez, D.R., Alamo, T., and Limon, D. (2021). Probabilistically certified management of data centers using predictive control. *IEEE Transactions on Automation Science and Engineering.*

Chadha, R. and Bhasker, J. (2009). *Static Timing Analysis for Nanometer Designs: A Practical Approach.* Springer.

Crockett, L.H., Elliot, R.A., Enderwitz, M.A., and Stewart, R.W. (2014). *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC.* Strathclyde Academic Media.

Gaikwad, N. and Patil, V.N. (2018). Verification of AMBA AXI on-chip communication protocol. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1–5. IEEE.

Goodwin, G.C., Cea, M.G., Seron, M.M., Ferris, D., Middleton, R.H., and Campos, B. (2012). Opportunities and challenges in the application of nonlinear MPC to industrial problems. *IFAC Proceedings Volumes*, 45(17), 39–49.

Grüne, L. and Pannek, J. (2017). Nonlinear model predictive control. In *Nonlinear Model Predictive Control*, 45–69. Springer.

Hübner, M. and Becker, J. (2010). *Multiprocessor system-on-chip: hardware design and tool integration.* Springer Science & Business Media.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, 1942–1948. IEEE.

Knagge, G., Wills, A., Mills, A., and Ninness, B. (2009). ASIC and FPGA implementation strategies for model predictive control. In *2009 European Control Conference (ECC)*, 144–149. IEEE.

Krupa, P., Limon, D., and Alamo, T. (2020). Implementation of model predictive control in programmable logic controllers. *IEEE Transactions on Control Systems Technology.*

Leontaritis, I. and Billings, S.A. (1985). Input-output parametric models for non-linear systems part i: deterministic non-linear systems. *International journal of control*, 41(2), 303–328.

Manzano, J.M., Limon, D., de la Peña, D.M., and Calliess, J. (2019). Output feedback mpc based on smoothed projected kinky inference. *IET Control Theory & Applications*, 13(6), 795–805.

Nadales, J., Manzano, J., Barriga, A., and Limon, D. (2022). Efficient FPGA parallelization of lipschitz interpolation for real-time decision-making. *IEEE Transactions on Control Systems Technology.*

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1), 33–57.

Schaumont, P.R. (2012). *A practical introduction to hardware/software codesign.* Springer Science & Business Media.

Valluru, J. and Patwardhan, S.C. (2019). An integrated frequent RTO and adaptive nonlinear mpc scheme based on simultaneous bayesian state and parameter estimation. *Industrial & Engineering Chemistry Research*, 58(18), 7561–7578.