Implementation of Fast Predictive Controllers on FPGA Platforms based on Parallel Lipschitz Interpolation

J.M. Nadales, J.M Manzano, A. Barriga, D. Limon

Abstract— **The implementation of nonlinear model predictive controllers for systems operating at high frequencies constitutes a significant challenge, mainly because of the complexity and time consumption of the optimization problem involved. An alternative that has been proposed is the employment of data-driven techniques to offline learn the control law, and then to implement it on a target embedded platform. Following this trend, in this paper we propose the implementation of predictive controllers on FPGA platforms making use of a parallel version of the machine learning technique known as Lipschitz interpolation. By doing this, computation time can be enormously accelerated. The results are compared to those obtained when the sequential algorithm runs on standard CPU platforms, and when the system is controlled by solving the optimization problem online, in terms of the error made and computing time. This method is validated in a case study where the nonlinear model predictive controller is employed to control a self-balancing two-wheel robot.**

I. Introduction

The fast development of computing systems and data management techniques is favouring the application of data-driven techniques to many fields. An area where machine learning techniques, deep learning techniques, and artificial intelligence methods in general are gaining popularity is automatic control. One of the branches of the control field, where many examples can be found in recent literature, is predictive control [1]. This trend has been consolidated during past years because of the necessity of implementing controllers on embedded platforms for real-time system operation, while satisfying constraints on its signals [2]. In the case of predictive controllers, the use of data-driven techniques is significantly useful, because they can be employed to avoid having to solve the -often complex- optimization problem involved, thus accelerating considerably the generation of new control actions.

Frequently, these data-driven methods require the evaluation of massive amounts of data. The processing of these data can be exceedingly time-consuming, making these methods not applicable to systems operating at high frequencies, in where real-time decision making is necessary. In that respect, one of the main bottlenecks is the employment of inherently sequential processing platforms, which process data in a recursive manner.

In this paper, we propose a fast implementation of predictive controllers on re-configurable embedded platforms, making use of a parallel version of the learning method known as Lipschitz interpolation (LI) [3]. New paradigms such as *Industry 4.0* require the implementation of real-time control systems on low-cost and low-energy embedded platforms. Often, these systems cannot compete with traditional general purpose CPU-based systems in terms of computing power, and thus new methodologies (mainly data-based algorithms) must be developed in order to implement advanced control methods for embedded applications. This is the case of predictive controllers, where the model of the system is called thousands of times to compute a single control action.

An important task is to select the appropriate platform and circuit topology. In this sense, re-configurable platforms as field-programmable gate arrays (FPGAs) offer the possibility to carry out the implementation of highly flexible parallel circuits relatively easy to be parallelized, thus increasing the applicability of the designed circuit. Several machine learning techniques which seem to be gaining popularity in the community are not suitable for being computed in a parallel scheme (see for example Gaussian processes [4]). In contrast, artificial neural networks present properties that make them prone to be parallelized [5]. In line with this, in this paper, we propose the employment of LI because of its nice properties to be easily implemented on FPGAs. Among others, some of the main reasons why this method is chosen are:

- It is easily parallelizable.
- It is based on simple operations.
- It depends on a single hyperparameter.

Lipschitz interpolation is a supervised machine learning technique used for the regression of unknown Lipschitz functions, from which a data set of observations is available. The objective is then to learn the ground truth function, being able to estimate its value for unseen queries.

The rest of the paper is organized as follows. In Section II, the mathematical fundamentals of the parallel LI algorithm are explained in detail. In Section III, the FPGA-based parallel system architecture that performs the interpolation algorithm is presented. In Section IV, the way the embedded controller is designed to optimize the FPGA resources is presented and an example of design where a two-wheel robot is controlled is shown. Section V concludes the article.

This work was supported by the Agencia Estatal de Investigación (AEI)-Spain under Grant PID2019-106212RB-C41/AEI/10.13039/501100011033 and by Ministerio de Economía y Competitividad of Spain under project DPI2016-76493-C3-1-R.

J.M. Nadales and D. Limon are with the Department of Automatic Control and System Engineering, University of Seville. J.M. Manzano is with the Department of Engineering at Universidad Loyola Andalucía. A. Barriga is with the Deparment of Electronics and Electromagnetism, University of Seville. Emails: nadales@us.es, jmanzano@uloyola.es, barriga@us.es, dlm@us.es

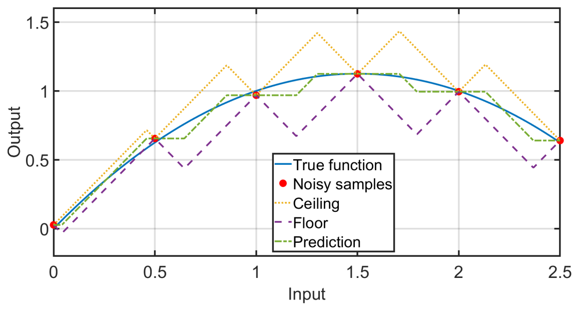

Fig. 1: Lipschitz interpolation.

II. Parallel Lipschitz Interpolation

This section briefly describes the standard Lipschitz interpolation algorithm, and introduces the details needed for its parallelization.

Let $f : \mathcal{W} \subset \mathbb{R}^{n_w} \to \mathcal{Y} \subset \mathbb{R}^{n_y}$ be the function to be learned by the algorithm. Suppose a set of $N_\mathcal{D}$ noisy samples is known, grouped in

$$\mathcal{D} = \{(\hat{f}(w_i), w_i) \mid w_i \in \mathcal{W}, \hat{f}(\cdot) \in \mathcal{Y}, i = 1, ..., N_\mathcal{D}\}, \quad (1)$$

where $\hat{f}(\cdot)$ are noisy observations of $f(\cdot)$. The sets $\mathcal{W}$ and $\mathcal{Y}$ are assumed to be compact. The noise affecting the output is assumed to belong to a compact set $\mathcal{E} \subset \mathbb{R}^{n_y}$. It is also assumed that the function $f(\cdot)$ is Lipschitz continuous, i.e., $\forall w_1, w_2 \in \mathcal{W}$, there exists a Lipschitz constant L such that

$$\|f(w_1) - f(w_2)\| \leq L\|w_1 - w_2\|. \quad (2)$$

With this, the output prediction of f for a new query point (input) $q \in \mathbb{W}$ is given by [6]:

$$\begin{aligned} \hat{\mathfrak{f}}_j(q; L, \mathcal{D}) &= \frac{1}{2} \min_{i=1,\ldots,N_\mathcal{D}} (\hat{f}_{i,j} + L\|q - w_i\|) \\ &+ \frac{1}{2} \max_{i=1,\ldots,N_\mathcal{D}} (\hat{f}_{i,j} - L\|q - w_i\|) \\ &= \frac{1}{2} \min_{i=1,\ldots,N_\mathcal{D}} \mathfrak{u}_i + \frac{1}{2} \max_{i=1,\ldots,N_\mathcal{D}} \mathfrak{l}_i, \end{aligned} \quad (3)$$

where $\hat{\mathfrak{f}}_j$ is the j-th component of $\hat{\mathfrak{f}}$, $\hat{f}_{i,j}$ the j-th component of the observed output value for the i-th sample point in $\mathcal{D}$ and w_i its corresponding input. The terms $\mathfrak{u}$ and $\mathfrak{l}$ are referred to as the *ceiling* and *floor* terms, respectively, and the space between them is called the *enclosure*. Any norm type can be employed in the previous expression. The infinity norm will be employed in this paper. The inference procedure is illustrated in Fig. 1.

The true Lipschitz constant L in unknown. Many works propose inference methods to estimate the value of L, based on data [7]. In this work, we assume that the Lipschitz constant has been previously calculated, as in [8].

One of the most interesting aspects of this method is that if the ground truth function is Lipschitz continuous and if the Lipschitz constant is known, the prediction error is bounded, and decreases with the amount of data in the training set $\mathcal{D}$ [7].

The algorithm consists in obtaining the *ceiling* and *floor* functions for every point in $\mathcal{D}$, computing the minimum *ceiling* and maximum *floor* terms (particularized at every new query point q), and then calculating the mean between them. Given a new input q, the calculation of each $\mathfrak{u}_i$ and $\mathfrak{l}_i$ is independent for each point i. This is the reason why this method is suitable to be parallelized, as it is possible to compute simultaneously as many *ceiling* and *floor* terms.

Additionally, as it will be detailed in the following sections, avoiding the multiplication by L in (3) simplifies the design of the system. This multiplication can be avoided by a simple storage policy of the data set, storing $\tilde{f} = \hat{f}/L$ instead of $\hat{f}$, and taking into account that the obtained result must be multiplied by L. Following this, the new parallel algorithm can be expressed as

$$\begin{aligned} \tilde{\mathfrak{f}}_j(q; \bar{\mathcal{D}}) = &= \frac{1}{2} \min_{i=1,\ldots,N_\mathcal{D}} \left(\tilde{f}_{i,j} + \|q - w_i\|_\infty\right) \\ &+ \frac{1}{2} \max_{i=1,\ldots,N_\mathcal{D}} \left(\tilde{f}_{i,j} - \|q - w_i\|_\infty\right) \\ &= \frac{1}{2} \left(\min_{i=1,\ldots,N_\mathcal{D}} \overset{\rightrightarrows}{\mathfrak{u}_i} + \max_{i=1,\ldots,N_\mathcal{D}} \overset{\rightrightarrows}{\mathfrak{l}_i} \right), \end{aligned} \quad (4)$$

where $\tilde{f}_{i,j} = \hat{f}_{i,j}/L$, $\bar{\mathcal{D}}$ is equal to $\mathcal{D}$ but with every output divided by L, and the notation $\overset{\rightrightarrows}{\mathfrak{u}_i}$ and $\overset{\rightrightarrows}{\mathfrak{l}_i}$ is employed to express that all *ceiling* and *floor* terms are calculated using the parallel strategy proposed in this work.

III. Architecture

The FPGA architecture to implement the LI algorithm is shown in Fig. 2. At a conceptual level, the system can be divided into three different parts: (i) a cluster of block random access memories (BRAMs), where training data are stored, (ii) an arithmetical circuit where the processing of the *ceiling* and *floor* terms is performed, and (iii) a sequential finite state machine (FSM)–synchronized with a clock signal–that controls all BRAMS in the system. The only input of the system is the new query point q. Likewise, the only output is the estimated value $\tilde{\mathfrak{f}}$.

To perform the calculation of the *ceiling* and *floor* functions associated to each data point i stored in the BRAMs ($\mathfrak{u}_i$ and $\mathfrak{l}_i$, respectively), a total of K enclosure calculation arithmetic units (ECAUs) are allocated in parallel. The total number of ECAUs that can be placed depends on the resource availability of the selected platform. The total number of BRAMs feeding the ECAUs is also equal to K. Because a total number of $N_\mathcal{D}$ data must be processed, each BRAM stores a total of $n = \left\lceil \frac{N_\mathcal{D}}{K} \right\rceil$. Inside each ECAU, different arithmetical operations are employed to calculate the *ceiling* and *floor* terms, such as the difference between the query point and the stored input data, the calculation of the infinity norm, and the addition and subtraction with the stored output data. The employment of the infinity norm is justified because it simplifies the internal structure of these blocks. After performing the calculus of all *ceiling* and *floor terms*, a tree of comparators finds the maximum among all *floor*

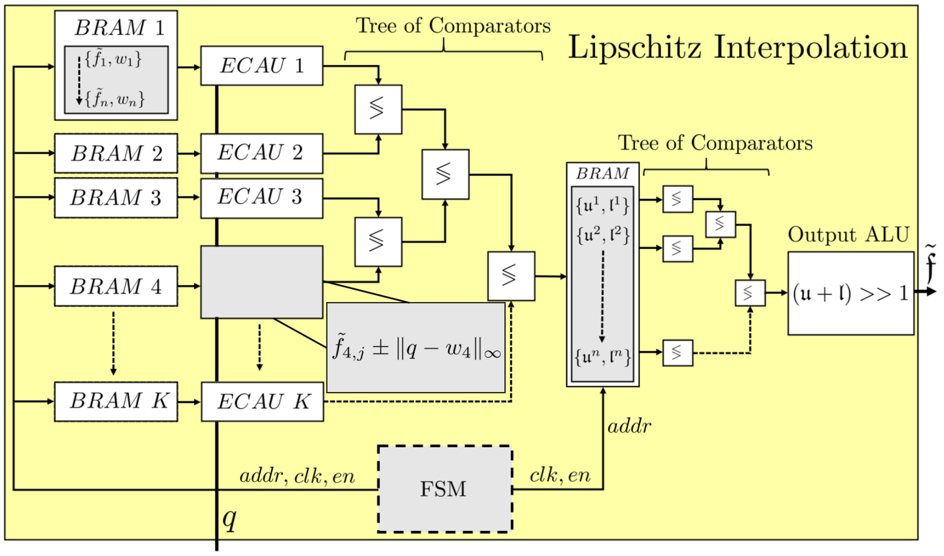

Fig. 2: Parallel Lipschitz interpolation module global architecture.

terms and the minimum among all *ceiling* terms. The number of comparison blocks required is upper-bounded by K.

In case the number ECAUs that can be placed in parallel (K) is lower than the total number of data to be processed ($N_{\mathcal{D}}$), a sequential strategy is followed. The outputs of the comparison stage are stored in another BRAM, whose depth n is equal to the number of iterations required to process all data. The results of processing all batches of data are compared again so that the minimum *ceiling* and maximum *floor* terms among all partial results are obtained. The minimum *ceiling* and maximum *floor* terms are finally transmitted to an output ALU, where the average among them is calculated.

The sequential procedure detailed above requires the implementation of a sequential finite state machine. This system is designed as an asynchronous Moore state machine whose outputs are the address ($addr$) signals of all BRAMs. This machine is sensitive to the clock signal, clk. Every time a positive edge of this signal is detected, the address signal is incremented. This is repeated until all data in the BRAMs are processed. The *write* and *enable* signals are accordingly controlled.

The time it takes to calculate a single output is $n\theta$, where θ is the clock cycle period. This time increases with N_D, in case $K < N_{\mathcal{D}}$, and is equal to θ in case $n = 1$, i.e., if $K \geq N_{\mathcal{D}}$. In general, $n\theta$ is various orders of magnitude lower than the time it takes to compute a single action when the standard LI algorithm is employed running on a CPU, which grows linearly with $N_{\mathcal{D}}$ [9].

IV. Embedded Controller Design

In this section, we describe the process of designing an embedded controller based on parallel LI following an strategy based on resource optimization, on FPGA platforms, illustrating the contribution with a case study.

A. Parallel Lipschitz interpolation controller

The problem addressed in this paper consists in the design of a control law for real-time operation of a discrete-time nonlinear plant, whose set of manipulable inputs is denoted $u \in \mathbb{R}^{n_u}$, and whose states are denoted $x \in \mathbb{R}^{n_x}$.

Moreover, it is our objective to inherit the well-known properties of model predictive control laws, namely, the ability to satisfy hard constraints during the operation (i.e. $u \in \mathcal{U}$ and $x \in \mathcal{X}$), while driving the system to a given reference equilibrium point (x_s, u_s) in an efficient way, i.e., minimizing costs.

However, model predictive controllers require real-time solving of an optimization problem, which is not always feasible at high frequencies. These are the cases considered in this paper: those in which the applicability of an MPC is not possible due to computation time requirements. Instead, we propose to compute the control law online based on a parallel Lipschitz interpolation algorithm, making use of a historical data set obtained from an existing control law. The data set necessary to learn the control law is generated offline by solving the optimization problem. This can be done, for instance, by selecting a collection of suitable states of the plant and calculating numerically the output of the control law. A practical method consists in selecting a collection of initial states and simulating the closed-loop trajectories, obtaining a set of states and control inputs form these trajectories. The data set should be analyzed, filtered and enhanced in an iterative procedure until the desired level of accuracy is obtained.

Therefore, the objective ground truth function f described in Section II is composed of inputs $w(k)$ equal to states of the system –$x(k)$– and outputs $f(w(k))$ equal to control actions –$u(k)$– such that the parallel LI algorithm is implemented as

$$u(k) = \kappa_{\text{LI}}(x(k)) = \tilde{\mathfrak{f}}(x(k); \bar{\mathcal{D}}). \tag{5}$$

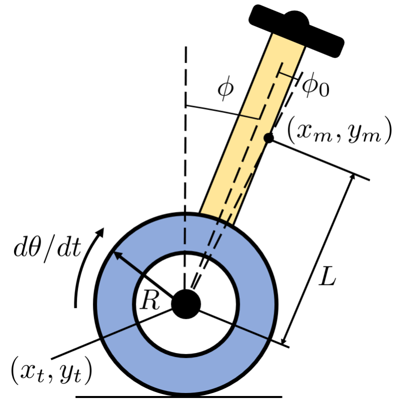

Fig. 3: Scheme of the two-wheel robot, reproduced from [10]. The length of the robot L should not be confused with the Lipschitz constant previously employed.

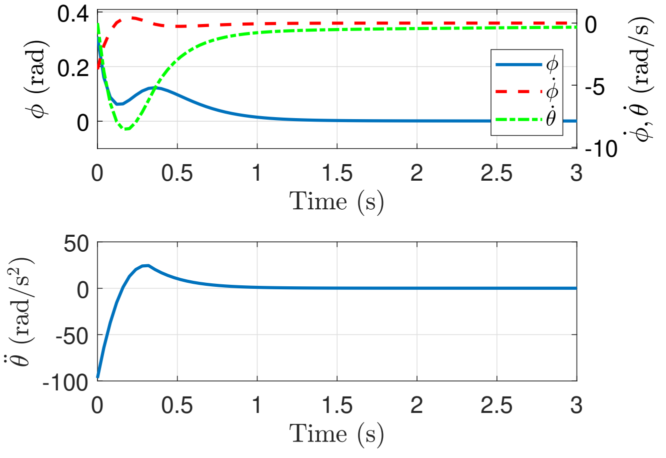

Fig. 4: Closed-loop simulation of the robot controlled by an MPC.

B. Case study

To illustrate the procedure, we propose a case study in which the controller is employed to self-balance a two-wheel robot. The platform where the controller is implemented and all tests here shown are performed is a Xilinx 7-series FPGA (XC7Z010-1CLG400C). The model of the system is extensively described in [10]. The state of the system is composed of the tilt angle ϕ (rad), its angular velocity $\dot\phi$ (rad s^{-1}) and the velocity of the angle between the wheel spin and the vertical axis $-\dot\theta$ (rad s^{-1}). The control action to be applied to the controlled system is the angular acceleration on the wheel $-\ \ddot\theta$ (rad s^{-2}). The sampling time is $40\,\mathrm{ms}$, which, as it will be shown later on, forbids the application of a linear MPC in real-time, and hence motivates the use of FPGA platforms to learn a control law.

C. Control design

The following steps describe the control system design:

1) Predictive control law: The first step is to design offline a model predictive controller [11] that stabilizes the system and meets performance requirements. The proposed MPC problem formulation is as follows

$$\begin{aligned} \min_{\mathbf{u}} \quad & \sum_{i=0}^{N-1} \|\hat{x}(i|k) - x_s\|_Q^2 + \|\hat{u}(i) - u_s\|_R^2 \\ & + \|\hat{x}(N|k) - x_s\|_P^2 && \text{(6a)} \\ \text{s.t.} \quad & \hat{x}(0|k) = x(k) && \text{(6b)} \\ & \hat{x}(i+1|k) = g(\hat{x}(i|k), \hat{u}(i)) && \text{(6c)} \\ & \hat{u}(i) \in \mathbb{U},\ i \in 0, \dots, N-1 && \text{(6d)} \\ & \hat{x}(N|k) = x_s, && \text{(6e)} \end{aligned}$$

where N is the prediction horizon, $\hat{x}(i \mid k)$ is the estimated state at time i given current state is k, x_s is the reference state to be tracked, $\mathbf{u} = \{\hat{u}_i \in \mathbb{U} \mid i \in 0, ..., N-1\}$ is the decision variable, $g(\cdot,\cdot)$ is the model of the system, matrices $Q_{\geq 0} \subset \mathbb{R}^{n_w \times n_w}$ and $R_{>0} \subset \mathbb{R}^{n_y \times n_y}$ are the tuning parameters of the problem and $P_{\geq 0} \subset \mathbb{R}^{n_w \times n_w}$ defines the terminal cost for stability purposes, which is obtained solving a LQR problem. The values to which matrices Q, R and P have been set are

$$Q = P = diag([10, 1, 10]), \qquad R = 0.1.$$

To obtain the training data set for the learning method, several closed-loop simulations of this predictive controller have been performed. In these simulations, the robot starts from a random initial state $x(0)$ and it is balanced in the vertical position (i.e., $x_s = [0\ 0\ 0]$). The system is exposed to random sensors' noise and impulsive disturbances on the state to excite the system. One of these simulations is shown in Fig. 4.

Each control action is calculated in approximately $70\,\mathrm{ms}$ in Matlab (using Matlab *fmincon* solver), running the simulation on an Intel®Core$^{\mathrm{TM}}$ i7-6700HQ CPU @ 2.60GHz 12GB RAM, while the time required by the system is $40\,\mathrm{ms}$. Its implementation in an embedded system is likely to take much more time.

2) Learning the control law: It is hence necessary to learn the control law on the FPGA, using parallel Lipschitz interpolation. Thus, we aim to learn the mapping between the robot's state and the control action to be applied as following:

$$\hat{u}(k) = \ddot\theta = \kappa_{\mathrm{MPC}}(x(k)). \tag{7}$$

In this case, the number of inputs of the function to be learnt is $n_w = 3$, i.e., the number of states. The number of outputs is $n_y = 1$, i.e., the number of manipulable inputs. The collected data set is composed of $N_{\mathcal{D}} = 14000$ points. This set is employed to learn the control law using Lipschitz interpolation. The Lipschitz constant is estimated as in [7], yielding $L = 4.67$. Then, the control to be applied to the system as a function of the estimated state of the system is

$$\hat{u}(k) = L\tilde{\mathfrak{f}}(\hat{x}(k); \bar{\mathcal{D}}). \tag{8}$$

A closed-loop simulation of the system controlled applying the control law implemented using LI is shown in Fig. 5. It can be seen how the system is stabilized to the vertical

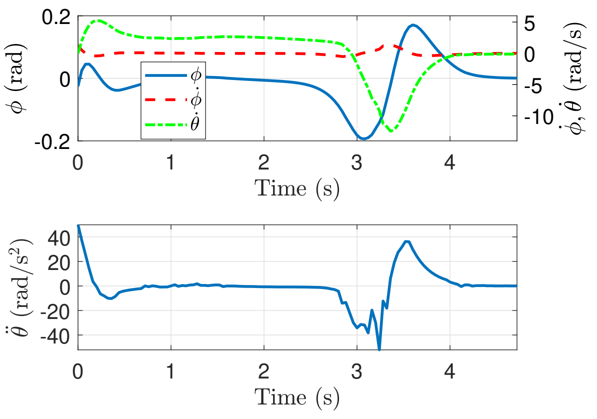

Fig. 5: Closed-loop simulation of the robot controlled by the LI controller, sequentially running on the CPU.

position. The time it takes to compute the control action is around $0.3\,\mathrm{ms}$. Note that this approach will suffice to implement a controller on the real robot, but that will not be the case for other plants in which larger data sets are needed, or with more inputs or states.

An analysis on how the computational time increases with the cardinality of $\mathcal{D}$, with the dimensionality of $\mathcal{U}$ or $\mathcal{X}$ for standard LI algorithms may be found in [9]. In addition, note that this case study considers a simple mechanical robot. For example, in the case of AC/DC power converters, a typical switching frequency could be close to $150\,\mathrm{kHz}$ [12], i.e., each control action must be generated every $6.66\,\mu\mathrm{s}$. Hence, the control actions must be computed various orders of magnitude faster than with the current approach. It is in these cases where the parallel LI presented in this paper finds application, as it is shown next.

D. System configuration

To represent data, fixed-point representation is employed because it optimizes area and energy consumption. The number of bits devoted to the integer part –ι– must be selected so that overflow is avoided, while the number of bits of the fractional part –φ– must be selected in such a way that the representation error is acceptable and bounded. Additionally, it must be considered that the larger the number of bits employed to represent data, the more area resources that are consumed, and thus fewer data can be processed in parallel because the number of ECAUs that can be implemented in parallel –K– is reduced. To reach a compromise, the following optimization problem is proposed

$$\underset{\varphi,\iota}{\text{minimize}} \quad C_A(\varphi,\iota)+\alpha C_E(\varphi,\iota) \tag{9a}$$
$$\text{s.t.} \quad \rho(\varphi)\le\rho_{\max} \tag{9b}$$
$$\iota\ge\chi(\underline{\nu},\overline{\nu}), \tag{9c}$$

where $C_A(\cdot,\cdot):\mathbb{R}^2_{\ge 0}\to\mathbb{R}_{\ge 0}$ and $C_E(\cdot,\cdot):\mathbb{R}^2_{\ge 0}\to\mathbb{R}_{\ge 0}$ are area and energy consumption functions, φ and ι are the number of bits employed to represent the fractional and integer part, respectively, $\rho(\varphi)$ is the maximum representation error committed, ρ_{max} is the maximum error we are willing to allow and $\chi(\underline{\nu},\overline{\nu})$ the minimum number of bits of the integer part that guarantees no overflow appears as a function of the absolute minimum and maximum values any signal in the circuit can take $\underline{\nu}$ and $\overline{\nu}$, respectively. Before solving this optimization problem, we need to define area and energy consumption functions and find analytical expressions for constraints (9b) and (9c).

Area and energy consumption functions depend on the selected platform and the resource allocation algorithm. With the right configuration, and avoiding the employment of multiplications (which are generally carried using digital signal processing (DSP) blocks) area and energy consumption functions are linear with respect to the number of bits employed to represent data (only lookup tables (LUTs) are consumed). Additionally, solving the problem for the whole system is equivalent to solving the problem considering area and energy consumption for a single ECAU and a single comparator. This is because the number of ECAUs and the number of comparators both grow linearly with the number of data processed in parallel. Note that the output ALU and the FSM have not been considered in this analysis because their contribution is not relevant.

The other step before solving the optimization problem is to find overflow and error constraints. In this paper, the core principles detailed in [13] have been applied to obtain these constraints. Alternatively, a less conservative analysis could have been performed via probabilistic simulation-based analysis [14]. Supposing all input signals in the system are scaled in the range [0,1]–and imposing that the maximum error committed must be $\rho_{max}=4\times10^{-4}$, it has been determined that the minimum number of bits devoted to the integer and fractional parts that ensure that no overflow appears and the maximum error is upper-bounded by ρ_{max} are $\iota=3$ and $\varphi=12$.

Because of the design, area and energy consumption functions are linear. Thus, the optimal values that minimize area and energy consumption are exactly those given by overflow and error constraints, i.e., $\varphi^*=12$ and $\iota^*=3$. Note that one extra bit must be added to take into account the sign of each value so the total number of bits employed to represent data is equal to 16. The final step in the design process is to configure the FPGA architecture employing the values $\iota=3$ and $\varphi=12$ and determine how many data can be processed in parallel, which result to be 256, i.e., $K=256$. A summary of the resources employed by each of the component is shown in Table I.

Component	BRAM Tiles (140)	LUTs (53200)	Utilization
BRAM	1	0	0.71 %
COMP	0	8160	15.33 %
ECAU	0	44800	84.21 %

TABLE I: Resource utilization.

On the basis of the above, the LI controller is implemented on the FPGA platform. Closed-loop simulation results of the system controlled by the parallel LI controller are shown

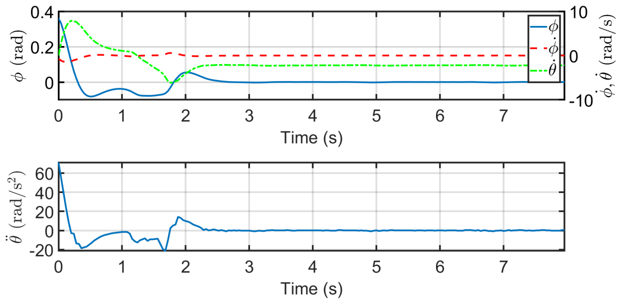

Fig. 6: Closed-loop simulation of the robot controlled by the parallel LI controller running on the FPGA.

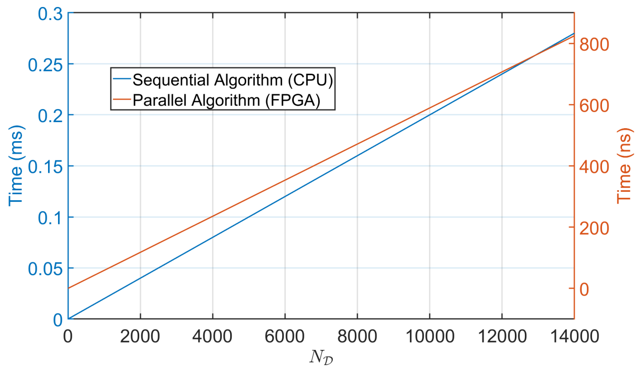

Fig. 7: Computation time w.r.t the size of the training data set.

in Fig. 6. The parallel LI controller stabilizes the system toward the vertical position. In Fig. 8, the error made for 2500 new entries due to parallelization is represented. The error between the standard LI control law and the proposed parallel version is of the order of 1×10^{-5}, which is less than ρ_{max}, and multiplied by L yields an error of at most 0.02.

After checking the worst negative slack (WNS) remains positive, a clock signal of $15\,\mathrm{ns}$ has been chosen. Thus, new *ceiling* and *floor* terms can be computed every $15\,\mathrm{ns}$. Because the number of data that can be processed in parallel is equal to $K = 256$, the total number of iterations required to process all data is equal to $n = 55$. Then, a new control action is generated every $825\,\mathrm{ns}$. Fig. 7 shows the time it takes to process a single new input as a function of the number of data to be processed $N_{\mathcal{D}}$. It can be seen how the parallel controller implemented on the FPGA is four orders of magnitude faster than the sequential algorithm running on CPUs, and six orders of magnitude faster than the model predictive controller.

V. Conclusions

A novel architecture has been presented for the learning and implementation of fast model predictive control laws on FPGA platforms making use of the data-driven technique known as Lipschitz Interpolation. An optimization problem in where area and energy consumption are minimized is

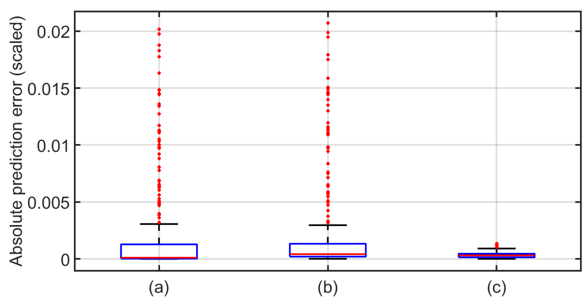

Fig. 8: Prediction error for 2500 queries. (a) MPC law vs standard LI (b) MPC law vs parallel prediction. (c) Standard LI vs parallel predictions.

proposed to efficiently select the optimal number of bits employed to represent data. Close-loop simulations were performed showing the performance of the system when it is controlled by the parallel LI algorithm. Open-loop simulations performed on the board show how the parallel algorithm running on the FPGA is four orders of magnitude faster than sequential algorithm running on a conventional CPU.

References

[1] L. Yang, J. Lu, Y. Xu, D. Li, and Y. Xi, "Constrained robust model predictive control embedded with a new data-driven technique," *IET Control Theory & Applications*, vol. 14, no. 16, pp. 2395–2405, 2020.

[2] 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.

[3] G. Beliakov, "Interpolation of Lipschitz functions," *Journal of computational and applied mathematics*, vol. 196, no. 1, pp. 20–44, 2006.

[4] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, pp. 63–71, Springer, 2003.

[5] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*, vol. 365. Springer, 2006.

[6] J.-P. Calliess, *Conservative decision-making and inference in uncertain dynamical systems*. PhD thesis, University of Oxford, 2014.

[7] J.-P. Calliess, S. J. Roberts, C. E. Rasmussen, and J. Maciejowski, "Lazily adapted constant kinky inference for nonparametric regression and model-reference adaptive control," *Automatica*, vol. 122, p. 109216, 2020.

[8] M. Canale, L. Fagiano, and M. Signorile, "Nonlinear model predictive control from data: a set membership approach," *International Journal of Robust and Nonlinear Control*, vol. 24, no. 1, pp. 123–139, 2014.

[9] J. M. Manzano, D. Limon, D. M. de la Peña, and J. Calliess, "Output feedback MPC based on smoothed projected kinky inference," *IET Control Theory & Applications*, vol. 13, no. 6, pp. 795–805, 2019.

[10] C. Gonzalez, I. Alvarado, and D. M. La Peña, "Low cost two-wheels self-balancing robot for control education," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9174–9179, 2017.

[11] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Nob Hill Pub., 2009.

[12] M. Alam, W. Eberle, D. S. Gautam, and C. Botting, "A soft-switching bridgeless AC-DC power factor correction converter," *IEEE Transactions on Power Electronics*, vol. 32, no. 10, pp. 7716–7726, 2016.

[13] C. F. Fang, R. A. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No. 03CH37486)*, pp. 275–282, IEEE, 2003.

[14] S. Omland, M. Hefter, K. Ritter, C. Brugger, C. De Schryver, N. Wehn, and A. Kostiuk, "Exploiting mixed-precision arithmetics in a multilevel monte carlo approach on FPGAs," in *FPGA Based Accelerators for Financial Applications*, pp. 191–220, Springer, 2015.