Efficient FPGA parallelization of Lipschitz interpolation for real-time decision making

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

***Abstract*—One of the main open challenges in the field of learning based control is the design of computing architectures able to process data in an efficient way. This is of particular importance when time constraints must be met, as for instance in real-time decision making systems operating at high frequencies or when a vast amount of data must be processed. In this respect, FPGA-based parallel processing architectures have been hailed as a potential solution to this problem. In this paper, a low-level design methodology for the implementation on FPGA platforms of Lipschitz interpolation algorithms is presented. The proposed design procedure exploits the potential parallelism of the Lipschitz interpolation algorithm and allows the user to optimize the area and energy resources of the resulting implementation. Besides, the proposed design allows to know in advance a tight bound of the error committed by the FPGA due to the representation format. Therefore, the resulting implementation is a highly parallelized and a fast architecture with an optimal use of the resources and consumption and with a fixed numerical error bound. These facts flawlessly suit the desirable specifications of learning-based control devices. As an illustrative case study, the proposed algorithm and architecture have been used to learn a nonlinear model predictive control law applied to self-balance a two-wheel robot. The results show how computational times are several orders of magnitude reduced by employing the proposed parallel architecture, rather than sequentially running the algorithm on an embedded ARM-CPU based platform.**

***Index Terms*—Data-Driven Control, FPGA Implementation, Machine Learning Algorithms, Parallel Embedded Systems, Real-Time Decision Making, Real-Time Control.**

I. INTRODUCTION

DURING the past few years, many are the research fields in which data-driven and learning methods are finding application, such as medical diagnosis [1], face recognition [2] or social media [3], among others. Particularly, one of the areas in which learning methods have experienced a significant growth is the field of automatic control and real-time decision making [4], [5].

At the same time, advances in other fields and the appearance of new paradigms, such as *industry 4.0* [6], *smart cities* [7], or the so-called *internet of things* [8], have also favored the fast development of data-driven methods. They all have as one of their core principle the availability of massive quantities of data and their further processing, what has spurred the development of storage systems [9] and embedded processing platforms [10] specially designed for these purposes.

Given the importance of data in present-day society, not only the implementation of new data-based methods is crucial, but also the development of computing platforms that give support to the designed algorithms. The cost of manufacturing application-specific integrated circuits (ASICs), specifically designed for the implementation of data-driven algorithms, may be prohibitively high when the system is not intended for large-scale production, and CPU-based platforms cannot compete with field-programmable gate array (FPGA) platforms in terms of flexibility and execution speed [11]. An alternative could be the employment of graphics processing unit (GPU) devices [12], but the latency [13] associated with memory access and communication with the CPU may be excessively large for real-time operation and the use of this type of devices is sometimes only justified when the set of data to be processed is sufficiently large.

For this reason, the use of reconfigurable architectures and programmable logic devices as FPGA platforms has emerged as one of the preferred options when it comes to developing embedded hardware systems to run machine learning algorithms in real time. Works relating to this topic are abundantly found in current literature as, for instance, the implementation of convolutional neural networks [14], classification algorithms [15], algorithms based on fuzzy logic [16], and many other different families of machine learning methods.

Lipschitz interpolation (LI) is a learning and predicting methodology for the regression of unknown Lipschitz functions, for which a data set of observations is available [17]. This method ensures a bounded estimation error and the prediction algorithm is based on the available data and a single hyperparameter, so its tuning process is considerably simpler than other learning techniques. LI algorithms have been used to learn nonlinear dynamical systems, sometimes being referred to as *nonlinear set membership* [18] or *kinky inference* [19]. Recently, efficient, robust and safe predictive control laws have been proposed using LI-based nonlinear models [20], [21]. However, the time required to compute the prediction depends linearly on the cardinality of the data set [20], which may hinder its application to fast real-time systems when the application requires a large amount of data.

This issue has motivated this work, in which a parallel architecture and a low-level design methodology for FPGA platforms is proposed to efficiently implement Lipschitz interpolation methods in real-time. This real-time implementation exploits the following nice properties of the LI algorithm:

This work was supported by the Agencia Estatal de Investigación (AEI)-Spain under Grant PID2019-106212RB-C41/AEI/10.13039/501100011033 and by the Junta de Andalucia and the Fondos Europeos para el Desarrollo Regional (FEDER) under Grant P20_00546.

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

- The prediction algorithm is suitable to be parallelized, thus allowing different batches of data to be concurrently processed.
- The calculation of the output only requires simple algebraic and comparison operations, so they can be implemented using basic native modules.
- The error committed derived from the data representation in the algorithm can be bounded. This is of special interest in those applications where the worst-case error must be delimited, e.g. for robustness purposes.

In this paper we propose a parallel architecture for the implementation of LI algorithm on FPGAs and a design methodology that allows the user to optimize the area and/or energy consumption while ensuring a given calculation error bound. Minimising area consumption allows to increase the total amount of data that can be processed at once, while reducing energy consumption gives the system greater autonomy. The proposed design allows the user to balance between both objectives depending on the embedded application target.

The rest of the paper is structured as follows. In Section II, the foundations of Lipschitz interpolation algorithms are summarized, and its parallel implementation is shown. In Section III, the FPGA-based parallel architecture that performs the interpolation algorithm is presented, while the data representation and the calculation error are studied in Section IV. The optimal design methodology is shown in Section V. In order to showcase the results of the paper, a case study where the designed architecture and proposed methodology are employed to implement a system that learns a nonlinear model predictive control law is shown in Section VI, accelerating the computation of the control action and allowing its real-time implementation. The paper ends with some conclusions in Section VII.

II. LIPSCHITZ INTERPOLATION ALGORITHM

A. Algorithm

This paper is devoted to implement a machine learning algorithm known as Lipschitz interpolation [17], which aims to learn an unknown function given a data set of input/output samples, under the assumption that such function is Lipschitz continuous. This is summarized in the following statement:

Setup: Consider a function $f : \mathcal{W} \subset \mathbb{R}^{n_w} \to \mathcal{Y} \subset \mathbb{R}^{n_y}$. From this f, a data set of $N_\mathcal{D}$ (possibly) noisy samples is known, grouped in:

$$\mathcal{D} = \{(\hat{f}(w_i), w_i), i = 1, \dots, N_\mathcal{D}\}, \tag{1}$$

where $\hat{f}(\cdot)$ stands for the noisy observation of $f(\cdot)$. The sets $\mathcal{W}, \mathcal{Y}$ are assumed to be compact, and the additive noise is assumed to be confined in a compact set $\mathcal{E} \subset \mathbb{R}^{n_y}$.

It is also assumed that f is Lipschitz continuous, i.e., $\forall w_1, w_2 \in \mathcal{W}$, for each output component $j = 1, \dots, n_y$,

$$\|f_j(w_1) - f_j(w_2)\| \leq L_j \|w_1 - w_2\|, \tag{2}$$

for a certain Lipschitz constant $L \in \mathbb{R}^{n_y}$

Then, the resulting prediction of f at a query q is computed as [19]:

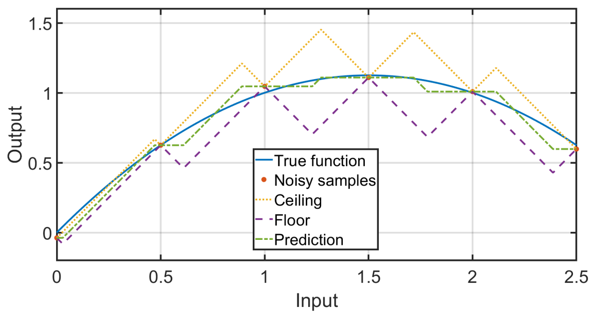


Fig. 1: Lipschitz interpolation over $f(w) = \frac{-w^2}{2} + \frac{3w}{2}$ with $N_\mathcal{D} = 6$ and $L = 1.5$.

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

where $\hat{\mathfrak{f}}_j$ is the j-th component of $\hat{\mathfrak{f}}$, $\hat{f}_{i,j}$ the j-th component of the 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, and the space between them is called the *enclosure*. Note that due to the equivalence among norms, any norm could be employed in the previous expression [19], although, for reasons that will be explained further on, the infinity norm will be taken in this paper. The interpolation method is illustrated in Figure 1.

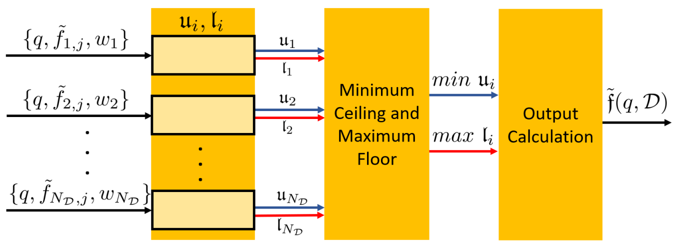
Remark 1: Note that the true Lipschitz constant (the smallest L_j that satisfies (2)) may be unknown a priori. Several works propose inference methods that obtain an estimation of the Lipschitz constant based on the available data [22], [23]. Since this is not within the scope of this paper, knowledge of the true Lipschitz constant is assumed, as in [18], [24].

Remark 2: One of the main advantages of this machine learning method is that if the ground truth function is Lipschitz continuous and its Lipschitz constant is either known or estimated (as in [23]), the method provides a bounded prediction error, which decreases as the density of the data set increases.

B. Parallelization of the algorithm

The core expression of the prediction algorithm to be implemented is given by (3). The calculation consists in obtaining the *ceiling* and *floor* functions for every data point in $\mathcal{D}$, computing the minimum *ceiling* and maximum *floor* terms (particularized at the query q) and then calculating the estimation as the average of both values. This method is represented in Figure 1.

Therefore, it is possible to compute simultaneously as many *ceiling* and *floor* terms as possible, since each of them is independent of the others for a given q, allowing the parallelization of the algorithm. This strategy, represented in Figure 2, splits the algorithm in three serial blocks: in the first block the calculation of all *ceiling* and *floor* terms is carried



$\mathfrak{u}_i, \mathfrak{l}_i$
$\{q, \tilde{f}_{1,j}, w_1\}$
$\{q, \tilde{f}_{2,j}, w_2\}$
$\{q, \tilde{f}_{N_\mathcal{D},j}, w_{N_\mathcal{D}}\}$
$\mathfrak{u}_1$
$\mathfrak{l}_1$
$\mathfrak{u}_2$
$\mathfrak{l}_2$
$\mathfrak{u}_{N_\mathcal{D}}$
$\mathfrak{l}_{N_\mathcal{D}}$
Minimum Ceiling and Maximum Floor
$min\ \mathfrak{u}_i$
$max\ \mathfrak{l}_i$
Output Calculation
$\tilde{\mathfrak{f}}(q, \mathcal{D})$

Fig. 2: Parallel Lipschitz interpolation.

out in parallel. In the second block, the maximum of the floors and the minimum of the ceilings are calculated, and these are finally averaged on the last block to obtain the prediction.

Additionally, as it will be justified in Section VI, avoiding multiplications in the algorithm simplifies the area and energy optimization problem as well as the calculation of data word-length. In this paper we propose that the multiplication by L_j, which is the only one in the algorithm, to be avoided by a simple storing policy of the data set. In effect, storing $\tilde{f}_j = \hat{f}_j / L_j$ (instead of $\hat{f}_j$), and bearing in mind that the result (now denoted $\tilde{\mathfrak{f}}_j$) has to be multiplied by L_j to get the prediction. Thus, the algorithm can be rewritten as

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

where $\tilde{f}_{i,j} = \hat{f}_{i,j} / L_j$, $\mathfrak{u}$ and $\mathfrak{l}$ redefine the *ceiling* and *floor* functions, respectively; and $\bar{\mathcal{D}}$ is equal to $\mathcal{D}$ but with every output divided by L_j.

In the following section, the FPGA-based parallel architecture is proposed, tailored to the algorithm presented above.

III. Parallel Architecture

The implementation of the parallel algorithm in the FPGA requires to define an appropriate architecture of the overall system. The proposed global parallel architecture is shown in Figure 3 and it is conceptually composed of three different parts: (i) a cluster of block random access memories (BRAMs) where data are stored (green shadowed); (ii) a processing subsystem in charge of performing the arithmetic calculations of the ceiling and floor terms and the final prediction (as well as storing partial results if more than one iteration is needed)(red shadowed); and (iii) a sequential finite state machine (FSM) synchronized with the clock signal, which governs the BRAMs in the system (blue shadowed). The only entries of the system are the clock signal clk that synchronizes data uploads, and the new query point q. The corresponding output of the system is the estimated value of the function $\tilde{\mathfrak{f}}(q)$.

There are a total of K BRAM memories and each of them stores a total of $n = \lceil \bar{D}/K \rceil$ data points. Data coming from these memories are the inputs of a cluster of K *enclosure calculation arithmetic units* (ECAUs) where the calculation of the *ceiling* and *floor* terms is performed for the query point input, according to (4). In Figure 4, the internal structure of each ECAU is shown. Notice that thanks to choosing the infinity norm in the algorithm, only basic blocks are employed in the ECAU. The one norm could also be implemented using simple blocks, but the error committed may be larger than when using the infinity norm [19].

Once the parallel calculation of all *ceiling* and *floor* terms has been performed, the maximum of the *floors* and the minimum of the *ceilings* are computed in a block consisting of a tree of comparators, as shown in Figure 5. The total number of comparison blocks required to implement this stage is upper-bounded by K, and the total number of comparison stages by $\lceil \log_2 K \rceil$, which correspond to the work and step complexities of the algorithm, respectively [25]. Note that, in case there are enough area resources to process all data at once, the step complexity of the comparison stage determines the step complexity of the whole algorithm since the step complexity of the previous stage where all ceiling and floor terms are calculated is just 1.

A sequential batch strategy is proposed in case the number of data points that can be processed in parallel (K) is lower than the total number of data points $N_\mathcal{D}$ to be processed (i.e., if $n > 1$).

In this case, the pairs of outputs of the comparison stage (which are the minimum *ceiling* and maximum *floor* terms resulting from the processing of K data points) are stored in another BRAM (a multi-port memory in this case, so that all output data can be downloaded at once), whose depth n is equal to the number of iterations required to process all data.

As it is shown in Figure 3, once all the required iterations have been completed, the results of processing each batch of data are again compared so that the minimum *ceiling* and the maximum *floor* terms among all the partial results are found. These terms are finally provided to an output arithmetic-logic unit (ALU), where they are averaged (cf. eq. (4)) to obtain the estimated output.

This sequential procedure requires the implementation of a management system in the form of a finite state machine whose states correspond to the batch of data loaded from the BRAMs to the ECAUs. This system manages both the cluster of BRAMs where the estimation data is stored and the BRAM where partial results are saved, as shown in Figure 6. This module is designed as a synchronous Moore finite state machine [26] whose outputs are the address ($addr$) and enable (ena, enb) signals of all BRAMs. Every time a positive edge of the external clock signal is detected, the address signals (which initially points to the first position of the BRAMs) are increased, pointing to the following position. At the same time, the enable signals of all memories are conveniently controlled.

Defining τ as the clock cycle, the proposed architecture takes $n\tau$ units of time to compute a prediction. Note that the computation time is directly proportional to $N_\mathcal{D}$, and inversely proportional to the number of ECAUs, K. In any case, K and τ are typically such that the computation time $n\tau$ is significantly faster than the computation time of the sequential algorithm implemented in a CPU-based system (as it will be

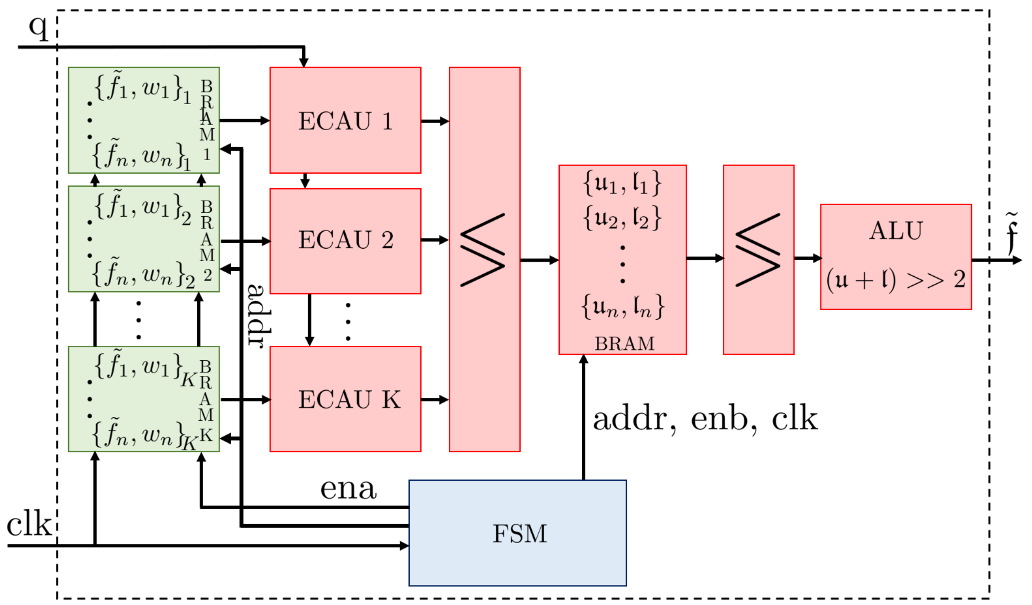


Fig. 3: Lipschitz interpolation module global architecture.

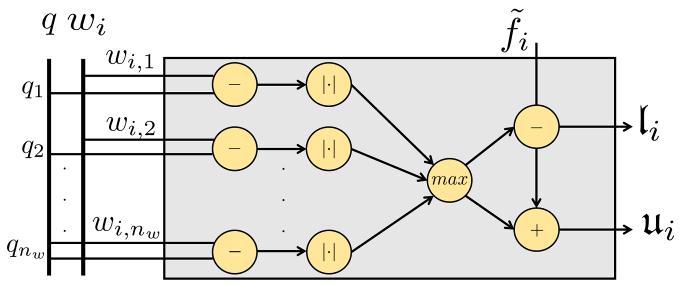


Fig. 4: Enclosure calculation arithmetic units (ECAUs).

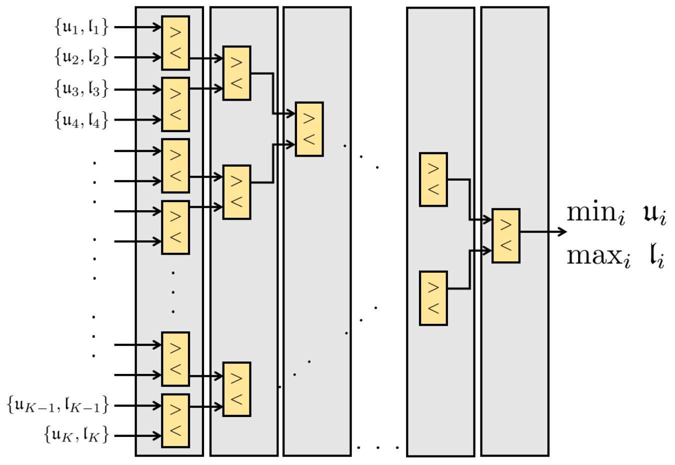

$\{\mathfrak{u}_1, \mathfrak{l}_1\}$
$\{\mathfrak{u}_2, \mathfrak{l}_2\}$
$\{\mathfrak{u}_3, \mathfrak{l}_3\}$
$\{\mathfrak{u}_4, \mathfrak{l}_4\}$
$\{\mathfrak{u}_{K-1}, \mathfrak{l}_{K-1}\}$
$\{\mathfrak{u}_K, \mathfrak{l}_K\}$
$\min_i$ $\mathfrak{u}_i$
$\max_i$ $\mathfrak{l}_i$

Fig. 5: Tree of comparators.

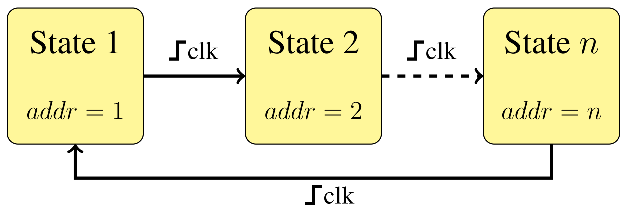


Fig. 6: Sequential control finite state machine.

illustrated in the case study, where the proposed implementation is four orders of magnitude faster than an ARM cortex A-based microcontroller).

Once we have seen the proposed architecture, the next step is to choose an appropriate data format, which is extensively described in the following section.

IV. Data Format

Each data point in $\bar{\mathcal{D}}$ consists of measured output values $\tilde{f}_i$ and their corresponding inputs w_i, being these terms arrays of lengths n_y and n_w, respectively. To represent these data, fixed-point format is proposed [27]. In general, the use of this type of representation, in comparison with floating-point representation [28], has the advantage that numbers are easier to handle, as they are treated as integers by the processing unit, and arithmetic operations are computed faster. The con is that fixed point representation provides a worse precision for the same number of bits than floating point representation [29], making necessary an study of the calculation error of the algorithm due to this representation.

Figure 7 shows an schematic of how data are organized inside the BRAMs. Each memory address of the BRAMs allocates a row of data consisting of the n_y outputs and the n_w inputs. Each of these values is represented in fixed point format with 1 bit for the sign, ι bits for the integer part and φ bits for the fractional part, resulting a total of $\iota + \varphi + 1$ bits.

The selection of the number of bits to be used in the data representation is crucial in the design of the implementation, since this determines the area and energy consumption as well as the precision of the calculations to be carried out [30], [31]. The greater the number of bits, the greater the amount of area and energy that are consumed and the greater the precision of

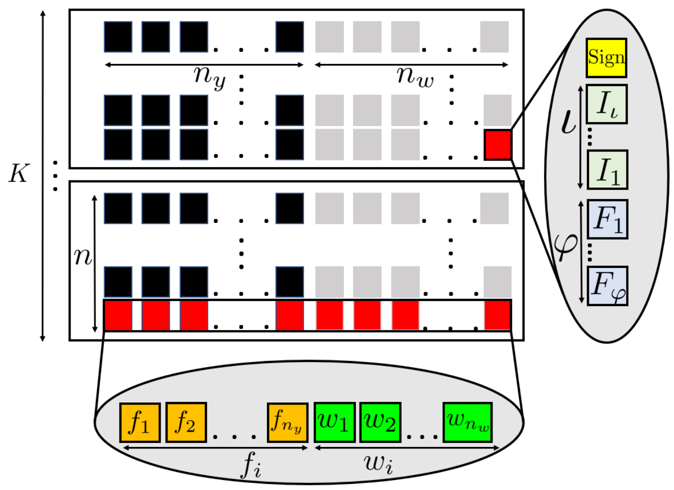


Fig. 7: Data format and memory organization.

the proposed architecture. Hence, the number of bits ι and φ must be selected to in such a way that the system is optimized while some specifications are met. The resulting optimization problem is studied and detailed in Section V.

One of the specifications that the representation of the data must fulfill is that this can represent every possible value resulting of the algorithm, avoiding overflow and ensuring sufficient precision. The number of bits of the integer part ι must be large enough to avoid overflow regardless the value of the input signals, while the number of bits of the fractional part φ must be large enough to meet the precision requirement.

A traditional approach to determine the minimum number of bits would be to first fix the value of bits of the fractional part and then perform a range evaluation analysis for every operand in the system taking into account that the error committed can make the integer value fluctuate. The range evaluation problem has commonly been dealt with via three different approaches: Monte Carlo analysis [32], transfer-function-based analysis [33] and interval analysis [34], [35].

In this work, and based on the principles described in [36], where affine arithmetic models [37] are used to represent fixed-point numbers, we propose the following approach tailored to the LI algorithm. First, a range evaluation analysis for all signals in the system is performed considering no error is committed due to the fractional part. This will allow us to find the overflow constraint. Knowing that this value can later fluctuate due to the error committed, an extra bit must be added to the integer part. Once this is done, an analysis of how the error propagates through the system is performed, and the minimum number of bits of the fractional part is selected so that the maximum error meets the desired limit. Note that these two processes could have been carried out using probabilistic simulation-based methods, but because of the simplicity of the algorithm, a conservative analytical approach can be derived.

As it was previously stated, the first problem to be addressed is to find the overflow constraint. For that purpose, a range evaluation analysis of all signals in the system is performed using the principles of interval arithmetic [38]. In this analysis it is not considered the representation error of the stored data or the query point. The intervals to which the values of the input signals belong to can be expresses as

$$\begin{aligned} &q_j \in [q^-, q^+], \quad w_{i,j} \in [w^-, w^+], \quad \tilde{f}_{i,k} \in [\tilde{f}^-, \tilde{f}^+], \\ &i = 1, ..., N_{\bar{\mathcal{D}}}, j = 1, ..., n_w, k = 1, ..., n_y. \end{aligned} \tag{5}$$

With this, the intervals to which the rest of the signals belong to are

$$q_j - w_{i,j} \in [q^- - w^+, q^+ - w^-], \tag{6}$$

$$\|q - w_i\| \in \left[0, \max(|q^- - w^+|, |q^+ - w^-|)\right], \tag{7}$$

$$\mathfrak{u}_i \in \left[\tilde{f}^-, \tilde{f}^+ + \max\left(|q^- - w^+|, |q^+ - w^-|\right)\right], \tag{8}$$

$$\mathfrak{l}_i \in \left[\tilde{f}^- - \max\left(|q^- - w^+|, |q^+ - w^-|\right), \tilde{f}^+\right], \tag{9}$$

$$\begin{aligned} \tilde{\mathfrak{f}} \in \Bigg[&\frac{1}{2}(2\tilde{f}^- - \max\left(|q^- - w^+|, |q^+ - w^-|\right)), \\ &\frac{1}{2}(2\tilde{f}^+ + \max\left(|q^- - w^+|, |q^+ - w^-|\right))\Bigg]. \end{aligned} \tag{10}$$

Once the ranges of values of all signals in the system have been evaluated, we must ensure that the number of bits selected for the integer part ι is such that all signals can be represented without suffering overflow. Suppose all input signals are scaled in the range [0,1]. Let $\underline{\nu}$ and $\overline{\nu}$ be the minimum and maximum values among all signals in the circuit, i.e.

$$\underline{\nu} \;=\; \min\{(q_j - w_{i,j})^-, \mathfrak{l}_i^-\}, \tag{11a}$$

$$\overline{\nu} \;=\; \max\{\|q - w_i\|^+, \mathfrak{u}_i^+\}. \tag{11b}$$

Then, the minimum number of bits to be selected so that none of the signals suffers overflow [39] is calculated as

$$\iota_{\min} = \eta(\underline{\nu}, \overline{\nu}) = \lceil \log_2(\max(\underline{\nu}, \overline{\nu})) \rceil + 1, \tag{12}$$

where an extra bit has been added as a safety margin. Note that this value is obtained following a conservative approach and in practice the range that the real values could take is smaller than this. Because of this, a simpler and less conservative simulation-based approach [40] could have been followed, but the results obtained following this type of methods is less robust than the solution here proposed.

In order to find the error bound, the next step is to study how the representation error propagates. According to the principles of affine arithmetic, an uncertain number can be expressed as a nominal value plus some sources of uncertainty. Thus, the affine representation of a variable x can be expressed as

$$x = x_0 + \sum_{i=1}^{p} x_i \epsilon_i, \tag{13}$$

where x_0 is the nominal value of the signal, each $\epsilon_i \in [-1, 1]$ is an independent uncertainty factor, each $x_i \in \mathbb{R}$ the magnitude of each component and p is the number of sources of uncertainty. Any number represented in this way can also be represented using interval notation as

$$x \in \left[x_0 - \sum_{i=1}^{p} |x_i|, x_0 + \sum_{i=1}^{p} |x_i|\right]. \tag{14}$$

In the specific case of fixed-point representation of bounded random values (as in our case), two main sources of uncertainty are commonly taken into account. The first one stems from the range of values the signal may take. The second one is the quantization error due to the number of bits chosen to represent the fractional part. Considering these two sources of uncertainty, the affine fixed-point representation x_f of a number x can be expressed as

$$x_f = x_0 + x_1\epsilon_{1,x} + a\epsilon_{2,x}, \tag{15}$$

where x_0 is the mean of the interval of values the number x can take, x_1 is the deviation this value can suffer to cover the whole range of values, $a = 2^{-(\varphi+1)}$ is the maximum absolute representation error and $\epsilon_{1,x}$ and $\epsilon_{2,x}$ are random variables in the range [-1,1]. Because the range evaluation analysis has already been performed for the integer part, in this case we only consider the uncertainty due to the representation error.

Next, consider that the same number of bits φ is employed to represent the fractional part of all signals. Let $\mathbb{B}(\cdot,\cdot)$ be an interval defined as

$$\mathbb{B}(x_c, r) = \{x_c + ru : |u| \leq 1\}, \tag{16}$$

where $x_c \in \mathbb{R}$ is the center of the interval and $r \in \mathbb{R}$ is its radius. Then, the inputs of each ECAU in the system can be represented in affine fixed-point format as

$$\begin{aligned}
q_{fj} &\in \mathbb{B}(q_j, a), && \text{(17a)}\\
w_{fi,j} &\in \mathbb{B}(w_{i,j}, a), && \text{(17b)}\\
\tilde{f}_{fi,k} &\in \mathbb{B}(\tilde{f}_{i,k}, a), && \text{(17c)}\\
i &= 1, ..., N_{\mathcal{D}}, &&\\
j &= 1, ..., n_w, &&\\
k &= 1, ..., n_y.,&&
\end{aligned}$$

where the subscript f is used to denote the fixed-point representation of the real number (for instance, q_f denotes the fixed-point representation of q). In the first stage of the ECAUs, each new input query point q is subtracted from each w_i and then the infinite norm is calculated. Using the principles of interval arithmetics, this results in

$$\|q_f - w_{fi}\|_f \in \mathbb{B}(\|w_i - q\|, 2a). \tag{18}$$

Consequently, the *ceiling* and *floor* terms associated to each data point i in $\bar{\mathcal{D}}$ can be calculated as

$$\mathfrak{u}_{fi,k} \in \mathbb{B}(\tilde{f}_{i,k} + \|w_i - q\|, 3a), \tag{19}$$

$$\mathfrak{l}_{fi,k} \in \mathbb{B}(\tilde{f}_{fi}(k) - \|w_i - q\|, 3a). \tag{20}$$

Then, the minimum among all *ceiling* terms and the maximum among all *floor* ones are given by

$$\begin{aligned}\mathfrak{u}_{fk} = \min_i \mathfrak{u}_{fi,k} &\in \mathbb{B}(\min_i(\tilde{f}_i + \|w_i - q\|), 3a)\\ &= \mathbb{B}(\mathfrak{u}_k, 3a),\end{aligned} \tag{21}$$

$$\begin{aligned}\mathfrak{l}_{fk} = \max_i \mathfrak{l}_{fi,k} &\in \mathbb{B}(\max_i(\tilde{f}_i - \|w_i - q\|), 3a)\\ &= \mathbb{B}(\mathfrak{l}_k, 3a).\end{aligned} \tag{22}$$

Finally, both *ceiling* and *floor* terms are added together and divided by two. This division is carried by simply shifting one bit to the right, and thus, no uncertainty is introduced by the multiplication operation. The output signal can now be calculated as

$$\begin{aligned}\tilde{\mathfrak{f}}_{fk} &\in \mathbb{B}\left(\frac{1}{2}(\mathfrak{u}_k + \mathfrak{l}_k), 3a\right)\\ &= \mathbb{B}\left(\tilde{\mathfrak{f}}_k, 3a\right).\end{aligned} \tag{23}$$

On the basis of these results, it can be seen that the maximum error committed throughout the system $\rho(\varphi)$ is given by

$$\rho(\varphi) \leq 3a = 3 \cdot 2^{-(\varphi+1)}. \tag{24}$$

Thus, the minimum number of bits of the fractional part $\varphi_{\min}$ to be employed so that the maximum error committed $\rho(\varphi)$ is bounded by some value $\rho_{\max}$ is given by

$$\varphi_{\min} = -\log_2\left(\frac{\rho_{\max}}{3}\right) - 1. \tag{25}$$

V. Design Methodology

In this section, the calculation of the optimal number of bits for the integer (ι) and fractional (φ) parts of the fixed-point representation of all signals in the system is described. These values are determined by solving an optimization problem where a certain performance measure of the design is minimized, subject to the constraints that ensure the satisfaction of the requirements on the range of representation to avoid overflow, as well as on the specified precision.

The performance of the design is measured by the area and dynamic energy consumption [1]. Assuming that all signals in the system employ the same number of bits, the area consumption and the energy consumption can be posed as functions of the number of bits, namely $C_A(\varphi, \iota) : \mathbb{R}^2_{\geq 0} \to \mathbb{R}_{\geq 0}$ and $C_E(\varphi, \iota) : \mathbb{R}^2_{\geq 0} \to \mathbb{R}_{\geq 0}$, respectively.

Then, the optimal representation is derived from the solution of the following optimization problem:

$$\begin{aligned}
\underset{\varphi,\iota}{\text{minimize}} \quad & J(\varphi,\iota) = C_A(\varphi,\iota) + \alpha C_E(\varphi,\iota) && \text{(26a)}\\
\text{s.t.} \quad & \varphi \geq \varphi_{\min} && \text{(26b)}\\
& \iota \geq \iota_{\min}, && \text{(26c)}
\end{aligned}$$

where $\alpha \geq 0$ is a weighing factor that allows the user to balance between the optimization of the area and the energy consumption. The limit values of the constraints, $\iota_{\min}$ and $\varphi_{\min}$, are given by (12) and (25), respectively.

The determination of the functions $C_A(\varphi, \iota)$ and $C_E(\varphi, \iota)$ is in general complex, and it depends on the blocks used in the design and the options used in the generation of the layout. In practice, these functions are typically empirically obtained by extensive simulation, as shown it will be shown in Section VI. If the cost function $J(\varphi, \iota)$ turns to be a non-convex function,

[1] In this analysis static energy is not considered because it is mainly due to the reverse bias leakage currents of the PN junctions and it does not depend on the design but on the selected platform [41].

then nonlinear programming methods must be used to solve the resulting optimization problem.

Next, we show that, under some mild and practical assumptions, explicit expressions for $C_A(\varphi,\iota)$ and $C_E\varphi,\iota)$ can be estimated allowing us to find the optimal values of φ and ι.

The total area and energy consumed can be approximated by the sum of areas and energies consumed by each of the elements in the system. In this estimation, the following statements must be taken into account: (i) the area consumed by the output ALU is not significant compared to the rest of the blocks; (ii) the area consumed by the FSM can be taken as a constant offset value that does not influence the optimization problem, (iii) only BRAM blocks are employed to store data so no LUTs are consumed and (iv) the employment of digital signal processors (DSPs) is avoided by a right configuration of the FPGA syntethizer and avoiding using multiplications. Therefore, area and energy consumption functions can be estimated by considering only the contribution of all ECAUs and comparators, resulting in

$$C_A^K(\varphi,\iota) \approx \sum_{i=1}^{K} C_{A,i}^{\text{ECAU}} + \sum_{j=1}^{K-1} C_{A,j}^{\text{COMP}}, \quad (27a)$$

$$C_E^K(\varphi,\iota) \approx \sum_{i=1}^{K} C_{E,i}^{\text{ECAU}} + \sum_{j=1}^{K-1} C_{E,j}^{\text{COMP}}, \quad (27b)$$

where K is equal to the number of data points to be processed in parallel, $K-1$ is the total number of comparators, $C_{A,i}^{\text{ECAU}}$ and $C_{A,j}^{\text{COMP}}$ the amount of area consumed by the j-th ECAU and comparator, respectively, and $C_{E,i}^{\text{ECAU}}$ and $C_{E,j}^{\text{COMP}}$ the amount of energy consumed by the j-th ECAU and comparator, respectively. Supposing all ECAUs and comparators consume the same amount of area and energy (which is normally the case), area and energy consumption functions can be expressed as

$$C_A^K \approx K\cdot C_A^{\text{ECAU}} + (K-1)\cdot C_A^{\text{COMP}}, \quad (28a)$$

$$C_E^K \approx K\cdot C_E^{\text{ECAU}} + (K-1)\cdot C_E^{\text{COMP}}, \quad (28b)$$

where C_A^{ECAU} and C_E^{ECAU} are the area and energy consumed by a single ECAU, respectively, and C_A^{COMP} and C_E^{COMP} the area and energy consumed by a single comparator, respectively. These terms are in turn functions of the type of resources employed to implement these blocks, mainly look-up tables (LUTs) and DSPs [42], according to the configuration of the resource allocation algorithm [43]. Since the proposed architecture avoids using multiplications (cf. Section II), the resource allocation algorithm (with the adequate configuration) only uses LUTs to implement the system. In this case, the cost function of the problem is linear with respect to the total number of bits ($\varphi+\iota$) employed to represent data (see Figures (12) and (13) in the case study), and the optimization problem can be expressed as

$$\underset{\iota,\varphi}{\text{minimize}} \quad J = \beta(K)\cdot(\iota+\varphi) \quad (29a)$$

$$\text{s.t.} \quad \varphi \geq \varphi_{\min} \quad (29b)$$

$$\iota \geq \iota_{\min}, \quad (29c)$$

where $\beta(\cdot):\mathbb{R}_{\geq 0}\to\mathbb{R}_{\geq 0}$ is a function that depends on the number of ECAUs and comparators employed, and thus on the number of data points processed in parallel.

From this analysis we can state that under the presented simplifying assumptions (which are typically fulfilled by the proposed architecture), the optimal number of bits that minimizes area and energy consumption functions are the minimum values of ι and φ that meet overflow and error constraints, irrespective of the size of the system (given by K), i.e.

$$(\varphi^*,\iota^*) = (\varphi_{\min},\iota_{\min}). \quad (30)$$

Notice that this reasoning would also be valid if the cost function were continuous and monotonically increasing with respect to the number of bits.

From this study, the following optimal design methodology is proposed:

Design Methodology

1) Begin with a given data set $\mathcal{D}$ and a Lipschitz constant L. Store $\tilde{f}_j = \hat{f}_j/L_j$, $j=1,\cdots,n_y$, as explained in Section II-B.
2) Perform a range evaluation analysis using the expressions proposed in Section IV and obtain the overflow constraint. Note that no simulation-based analysis is required.
3) Define the maximum allowable error to be committed, $\rho_{\max}$, and analytically find the error constraint using the expressions proposed in Section IV.
4) Set φ and ι to $\varphi_{\min}$ and $\iota_{\min}$, respectively.
5) Determine the number of data that can be processed in parallel K so that the constraint imposed by the number of available LUTs is satisfied.
6) Build the system as described in Section III.

VI. Case Study

In order to illustrate the proposed structure and methodology design of Lipschitz interpolation algorithms in an FPGA, this has been tested on a challenging real-time control problem: the real-time implementation of a constrained nonlinear model predictive control law for a self-balancing vehicle.

The resulting design has been experimentally implemented and tested on the target platform Xilinx Zynq-7000 system on chip embedded on a Zybo z7 board which integrates a dual-core ARM Cortex-A9 processor with a Xilinx 7-series FPGA [44]. The board is shown in Figure 8.

In this section, the design steps described in the previous sections are followed, demonstrating and justifying the proposed design methodology. Standard Verilog has been employed to implement the system on the FPGA as well as to perform all analysis and verification tests. For comparison purposes, the sequential version of the algorithm running on the ARM processor has been programmed using C language.

A. Model predictive control of a two-wheel self-balancing robot

The system to be controlled is a self-balancing two-wheel robot presented in [45] (and represented in Figure 9). The

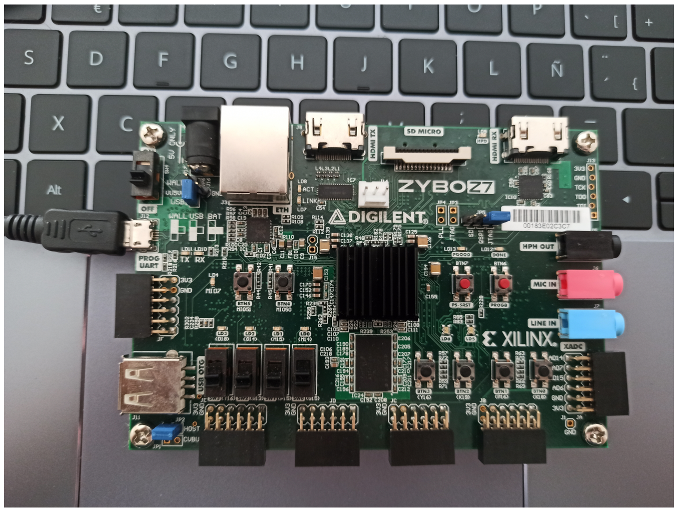


Fig. 8: Zybo z7 board.

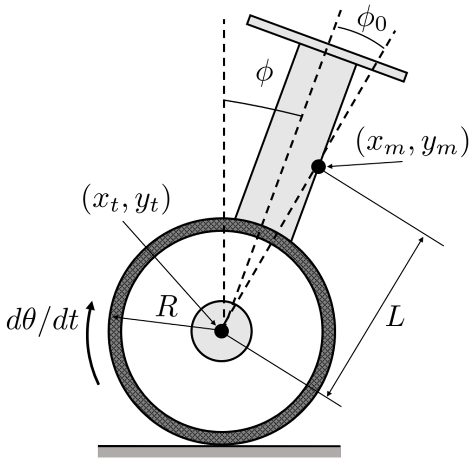


Fig. 9: Scheme of the two-wheel robot, reproduced from [45].

objective is to stabilize this constrained nonlinear system on the upper position by manipulating the acceleration of the wheels.

The state of the system comprises the tilt angle ϕ (rad), its velocity $\dot{\phi}$ (rad/s) and the velocity of the angle between the wheel spin and the vertical, $\dot{\theta}$ (rad/s) (see Figure 9). The control action is the angular acceleration of the wheels, $\ddot{\theta}$ (rad/s^2). The nonlinear model is discretized such that $x(k+1) = g(x(k), u(k))$. The model function g and its parameters can be found in [45, Appendix]. The sampling time to satisfactorily control this system has been fixed to $40\,\text{ms}$, making its real-time implementation on an embedded platform difficult.

The following stabilizing nonlinear model predictive control (MPC) law has been designed, taking into account the constraints on the inputs ($|\ddot{\theta}| \leq 50\,\text{rad/s}^2$), to obtain the desired control action, which is applied in a receding horizon manner [46]:

$$\min_u \quad \sum_{j=0}^{N-1} \|\hat{x}(j|k)\|_Q + \|u(j)\|_R \tag{31a}$$
$$\text{s.t.} \quad \hat{x}(0|k) = x(k) \tag{31b}$$
$$\hat{x}(j+1|k) = g(\hat{x}(j|k), u(j)), \tag{31c}$$
$$u(j) \in \mathbb{B}(0, 50), \forall j = 0, \ldots, N-1, \tag{31d}$$
$$\hat{x}(N|k) = [0, 0, 0]^T. \tag{31e}$$

The prediction horizon is set to $N = 4$, and the stage cost weights to $Q = \text{diag}(10, 1, 10)$ and $R = 0.1$.

Solving such constrained nonlinear optimization problem in less than $40\,\text{ms}$ on an embedded platform is hard. To overcome this problem, the resulting control law is learnt using Lipschitz interpolation, based on a suitable data set obtained from off-line simulations of the proposed MPC. Thus, the function to be learnt is the control law, as a function of the robot's state, i.e.

$$u(k) = \ddot{\theta}(k) = \kappa_{\text{MPC}}(x(k)) = f(\phi(k), \dot{\phi}(k), \dot{\theta}(k)). \tag{32}$$

Note that the function has three inputs and one output, i.e. $n_y = 1$ and $n_w = 3$.

Then, the proposed Lipschitz interpolation algorithm is implemented in the FPGA allowing the real time execution of the control law. In the following, the steps followed for this design are showcased.

B. Obtaining the data set

Once that the stabilizing nonlinear MPC control law has been designed, the data set for the learning method is obtained carrying out several closed-loop simulations in which the robot is balanced in the vertical position, subject to:

- Different random initial states, in which $\phi(0)$ and $\dot{\phi}(0)$ are uniformly distributed in $\mathbb{B}(0, 0.6)\,\text{rad}$ and $\mathbb{B}(0, 5)\,\text{rad/s}$, respectively, and $\dot{\theta}$ is kept at $0\,\text{rad/s}$.
- Additive random sensors' noise in the measure of the angle ϕ, normally distributed with 0 mean and standard deviation of $0.05\,\text{rad}$.

This process aims to obtain a data set rich enough to learn the control action, and thus a data set with $N_{\mathcal{D}} = 14000$ is collected. An example of the closed-loop performance of this MPC is shown in Figure 10. Each control action is calculated in approximately $70\,\text{ms}$ in Matlab, on an Intel® Core™ i7-6700HQ CPU @ 2.60GHz 12GB RAM, which is more than the $40\,\text{ms}$ required by the system.

All the signals are scaled to range [0,1], and the Lipschitz constant has been estimated as in [23], yielding $L = 4.67$. Next, the loop is closed applying in each iteration the control action obtained by

$$u(k) = L\tilde{\mathfrak{f}}(x(k); \bar{\mathcal{D}}). \tag{33}$$

The learnt control law is then validated. Its performance relies on the obtained data set. To this aim, closed-loop simulation applying the Lipschitz interpolation-based control law are carried out, subject to the same previous conditions (i.e., random initial states and random sensor's noise), checking

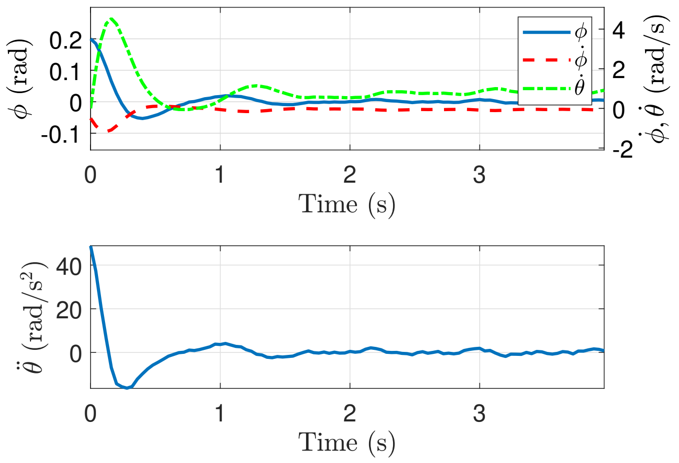


Fig. 10: Closed-loop simulation of the robot controlled by the MPC.

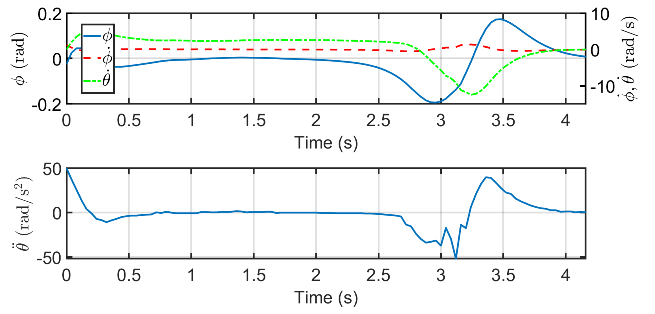


Fig. 11: Closed-loop simulation of the robot controlled by the LI controller.

whether the robot is stabilized towards the vertical position. An example of this learnt control law is shown in Figure 11, illustrating how the learnt control law also stabilizes the system.

C. Area and energy consumption

Once the data set has been defined, area and energy consumption functions are studied. As it was mentioned in Section II, avoiding multiplications simplifies the optimization problem, because no digital signal processor (DSPs) are used by the FPGA to implement the system. Only LUTs are employed, and area and energy consumption functions are linear with respect to the number of bits employed to represent data.

Since this will depend on the employed platform and the resource allocation configuration, as mentioned before, next we experimentally demonstrate that this is the case for the considered target platform. To obtain experimentally the area and energy consumption functions, several designs have been implemented, employing different number of bits $(\varphi + \iota)$ and considering an ambient air temperature of $T = 25\,°\mathrm{C}$ and a constant airflow of $A = 75\,\mathrm{m/s}$. The results obtained from the implementation of the different components on the real board

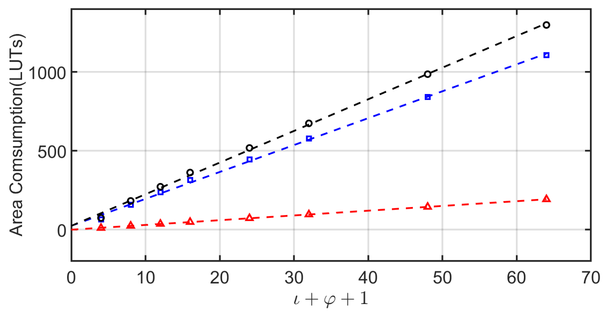


Fig. 12: Area consumption: (red) area consumption of a single comparator, C_A^{COMP}, (blue) area consumption of a single ECAU, C_A^{ECAU}, (black) sum of the areas consumed by an ECAU and a comparator.

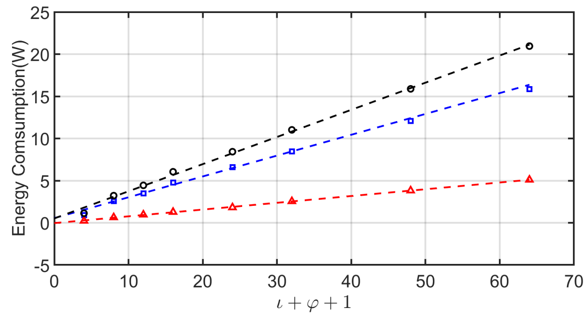


Fig. 13: Energy consumption: (red) energy consumption of a single comparator, C_E^{COMP}, (blue) energy consumption of a single ECAU, C_E^{ECAU}, (black) sum of the energies consumed by an ECAU and a comparator.

are represented in Figures 12 and 13, demonstrating that these are linear functions as assumed.

D. Data format analysis

1) Range evaluation: In order to calculate the minimum number of bits devoted to the integer part so that overflow is avoided, a range evaluation analysis supposing no representation error is carried out by using expressions (5) to (10). Since the signals to be applied are scaled in the range [0,1], the results obtained show that all signals in the system lie within the interval [-1,2], so 2 bits are enough to represent the whole range of values, while ensuring no overflow appears. An extra bit has been added to ensure no overflow appears due to the approximation error, resulting that minimum number of 3 bits is required to represent the integer part of all signals in the system.

In order to validate this range analysis, an experiment in which a total of 10^5 random input values scaled in the range [0,1] are applied to the algorithm, has been performed. The results (the values that all signals take assuming no approximation error) are shown in Figure 14. It can be seen that all signals lie in the estimated interval [-1,2].

In addition, note that the interval algebra evaluation yields a range of $[-0.5, 1.5]$ for the output $\tilde{\mathfrak{f}}$, provided that the ceiling and floor terms lied in the ranges $[0, 2]$ and $[-1, 1]$,

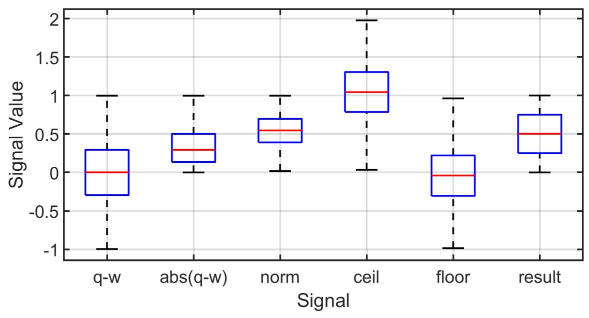


Fig. 14: Range evaluation analysis.

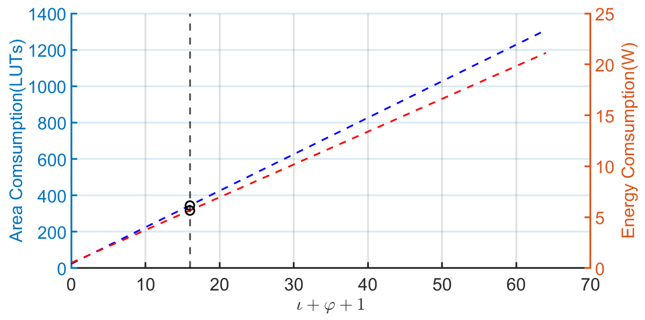


Fig. 16: Optimization problem solution.

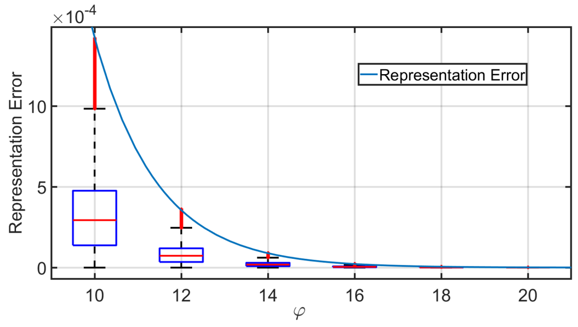


Fig. 15: Representation error.

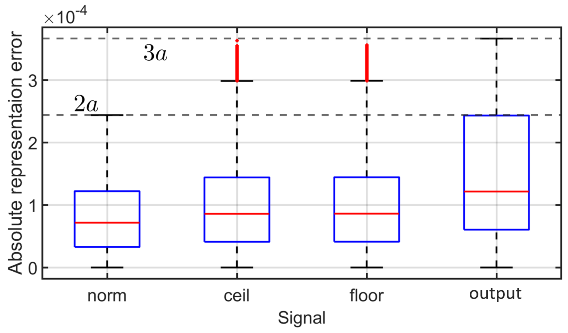


Fig. 17: Representation error of all signals in the system.

respectively (cf. eq. (23)). However, this is a conservative overestimation of the actual range, which results in $[0, 1]$, as illustrated in Figure 14.

2) Error analysis: In order to find the minimum number of bits devoted to the fractional part is necessary to know the maximum allowable error. In this case, and based on the nature of the problem, a maximum error of $\rho_{\max} = 4 \times 10^{-4}$ has been selected, since it has been numerically tested that the predictive controller maintains a good performance in case that a disturbance in the input of the system within this range is applied. From expression (24), we have that taking $\varphi_{\min} = 12$ guarantees a maximum error of $\rho(\varphi) \leq 3 \cdot 2^{12+1} \approx 3.66 \times 10^{-4}$.

This bound has also been experimentally validated evaluating 10^5 random inputs for a collection of data format with different number of bits of the fractional part (φ). The obtained results are shown in Figure 15. As it can be seen, the maximum absolute representation error exponentially decreases with the number of bits φ. It is also interesting to demonstrate that the maximum error committed when representing all signals in the system are those given by expressions (18) to (23), demonstrating that these expressions are valid.

E. Optimal design

In the previous subsections, we have determined the area and energy consumption functions, and overflow and error constraints. Therefore, we are ready to calculate the optimal design of the proposed implementation, which is derived from the solution of the optimization problem proposed in Section V. Since, as assumed, both area and energy consumption functions are monotonically increasing, the solution of the problem is the minimum allowable values of φ and ι, i.e. $\iota = \iota_{\min} = 3$ and $\varphi = \varphi_{\min} = 12$. Taking an extra bit to describe the sign, the optimal number of bits to be employed to represent data is equal to 16.

In Figure 16 the cost functions to be optimized together with the achieved minimum for $K = 1$ are illustrated, proving that this choice provides the minimum number of LUTs and energy consumption.

Next, we experimentally validate that this optimal number of bits is appropriate for every signal of the implementation. To this aim, an experiment in which a total of 10^5 random input values scaled in the range [0,1] have been applied to the system has been performed. The results are shown in Figure 17.

F. Architecture configuration

The final step in the design process is to configure the architecture and to find out how many data points can be processed in parallel. To maximize this number, the area required for different numbers of K has been calculated and the results are shown in Figure 18. The red dashed line represents the number of LUTs available to implement the system (53200 LUTs for the chosen target platform) while the blue line represents the number of LUTs consumed as a function of K. From this, it is derived that a total of $K = 256$ is the maximum number of data that can be processes in parallel.

G. Experimental results

The designed implementation has been experimentally validated in real time in the Xilinx Zynq-7000 platform, by means

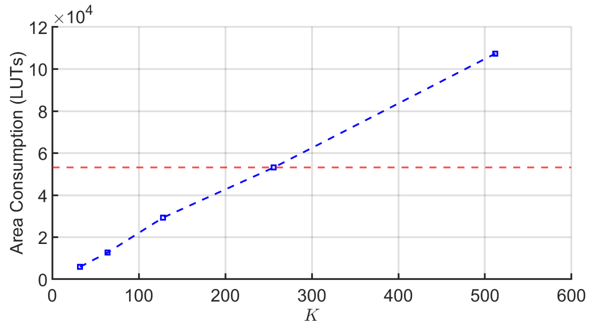


Fig. 18: System area consumption.

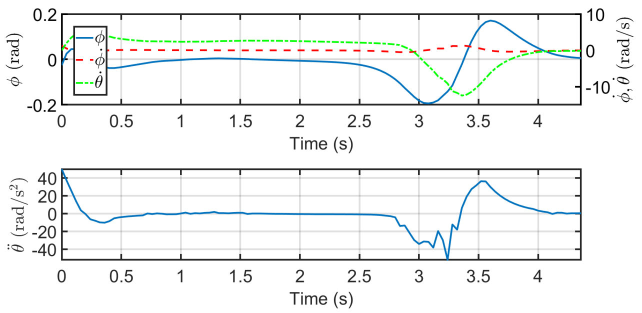


Fig. 19: Close-loop simulation of the robot controlled by the parallel LI controller.

of Processor in the Loop (PiL) architecture. The nonlinear predictive control law implemented in the FPGA is used to control in real time the two-wheel self-balancing robot model. It is simulated on one of the two ARM cores available on the SoC, connected by AXI-Lite interface.

Simulation results of the closed-loop PiL system for random initial conditions and under the presence of bounded sensor noise and disturbances are shown in Figure 19. As it can be seen, the implemented control law stabilizes the system towards the vertical position.

An analysis of the error made for 2500 queries due to the parallelization is represented in Figure 20, comparing three different implementations of the predictive control law: (a) the MPC law, i.e. the ground truth function calculated in Matlab with double precision; (b) the LI prediction implemented in Matlab using double precision; and (c) the proposed parallel LI prediction implemented in the FPGA. Note that the error between the LI implemented in Matlab (b) and in the FPGA (c) is of the order of 1×10^{-5}, which is even less than the maximum representation error established $\rho_{\max}$.

After performing a time analysis, a clock signal of $15\,\mathrm{ns}$ has been selected, which provides a positive WNS (Worst Negative Slack). Since data uploading from the BRAM memories is synchronized with the clock signal, new output terms are generated every $15\,\mathrm{ns}$. As it was previously shown, the maximum number of data that can be processed in parallel is $K = 256$, a total of $n = 55$ iterations are required to process all data, being this equal to the depth of each BRAM in the system. Thus the computation time of the resulting implementation of the control law results to be $825\,\mathrm{ns}$, four orders of magnitude

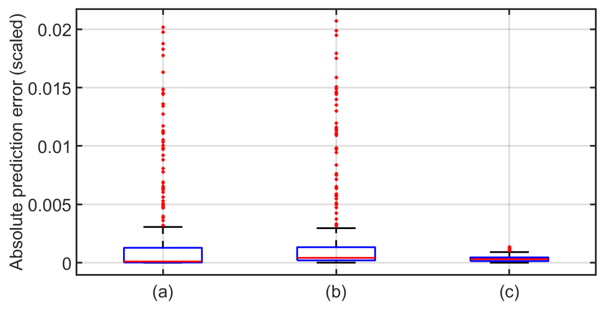


Fig. 20: Prediction error for 2500 queries. (a) MPC law vs standard LI (b) MPC law vs parallel prediction. (c) Standard LI vs parallel predictions.

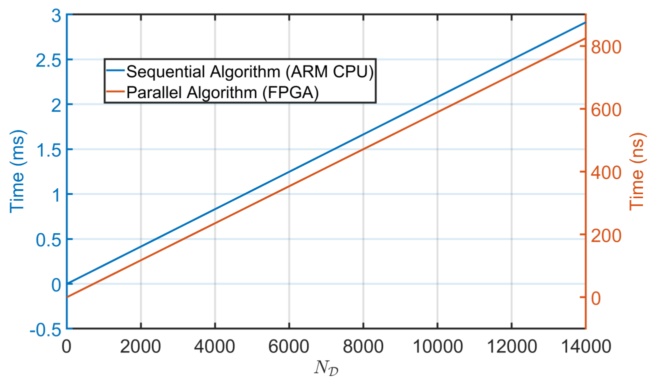


Fig. 21: Computation time w.r.t the size of the data set.

faster than the specified sampling time.

Finally, to illustrate the benefits of the parallel implementation in the FPGA of the LI algorithm versus its serial implementation in the embedded ARM processor available in the SoC, we have studied their open-loop computation time for a single query point w.r.t. the number of data points on the data base $N_{\mathcal{D}}$. The results are shown in Figure 21 and clearly demonstrate that the parallel version of the algorithm implemented on the FPGA device is significantly faster (three orders of magnitude in this case) than the sequential algorithm, irrespective of the size of the data set.

VII. Conclusion

In this work, a FPGA-based architecture was proposed to parallelize the inference method known as Lipschitz interpolation. The algorithm was posed in a suitable form to be parallelized, avoiding multiplications, and a suitable architecture was proposed for its implementation. In addition, a design methodology was presented to optimize the design in terms of area and energy consumption while the given precision specification is ensured by design. This results not only in a better use of the FPGA resources but also in increasing the number of data that can be processed simultaneously, leading to a significant reduction of the computational time.

The architecture and the design methodology were validated in an experimental real-time case study devoted to learn a nonlinear model predictive control law that stabilizes a two-wheel self-balancing robot. The resulting control law implementation successfully stabilizes the system yielding a computation time four orders of magnitude faster than the original controller. As it has been demonstrated, the proposed algorithm is three orders of magnitude faster than the standard sequential algorithm and the prediction error committed falls below the specified limit.

References

[1] Y. Liu, Q. Leng, and S. Wang, "Learning medical diagnosis via scaled convex hull-based SK algorithm," in *2019 IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS)*, 2019, pp. 377–381.

[2] T. Alobaidi and W. B. Mikhael, "A modified discriminant sparse representation method for face recognition," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, 2018, pp. 727–730.

[3] L. Li, Q. Zhang, X. Wang, J. Zhang, T. Wang, T. Gao, W. Duan, K. K. Tsoi, and F. Wang, "Characterizing the propagation of situational information in social media during covid-19 epidemic: A case study on weibo," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 2, pp. 556–562, 2020.

[4] S. Lucia, D. Navarro, O. Lucia, 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.

[5] C. Wu, J. Chen, C. Xu, and Z. Liu, "Real-time adaptive control of a fuel cell/battery hybrid power system with guaranteed stability," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1394–1405, 2016.

[6] A. G. Gonzalez, M. V. Alves, G. S. Viana, L. K. Carvalho, and J. C. Basilio, "Supervisory control-based navigation architecture: a new framework for autonomous robots in industry 4.0 environments," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1732–1743, 2017.

[7] Y. Zhang, M. Yutaka, M. Sasabe, and S. Kasahara, "Attribute-based access control for smart cities: A smart contract-driven framework," *IEEE Internet of Things Journal*, 2020.

[8] K.-L. Tsai, F.-Y. Leu, and I. You, "Residence energy control system based on wireless smart socket and IOT," *IEEE Access*, vol. 4, pp. 2885–2894, 2016.

[9] D. Hu, D. Feng, Y. Xie, G. Xu, X. Gu, and D. Long, "Efficient provenance management via clustering and hybrid storage in big data environments," *IEEE Transactions on Big Data*, vol. 6, no. 4, pp. 792–803, 2019.

[10] R. Chapman and T. S. Durrani, "Ip protection of DSP algorithms for system on chip implementation," *IEEE Transactions on signal processing*, vol. 48, no. 3, pp. 854–861, 2000.

[11] D. B. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, gpus, FPGAs, and massively parallel processor arrays for random number generation," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2009, pp. 63–72.

[12] N. S. Alizadeh and M. Momtazpour, "Machine learning-based interference detection in GPGPU concurrent kernel execution," in *2020 25th International Computer Conference, Computer Society of Iran (CSICC)*, 2020, pp. 1–4.

[13] M. Andersch, J. Lucas, M. A. LvLvarez-Mesa, and B. Juurlink, "On latency in gpu throughput microarchitectures," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 169–170.

[14] N. Shah, P. Chaudhari, and K. Varghese, "Runtime programmable and memory bandwidth optimized FPGA-based coprocessor for deep convolutional neural network," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 5922–5934, 2018.

[15] D. Tong, Y. R. Qu, and V. K. Prasanna, "Accelerating decision tree based traffic classification on FPGA and multicore platforms," *IEEE transactions on parallel and distributed systems*, vol. 28, no. 11, pp. 3046–3059, 2017.

[16] C.-F. Juang, C.-M. Lu, C. Lo, and C.-Y. Wang, "Ant colony optimization algorithm for fuzzy controller design and its FPGA implementation," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 3, pp. 1453–1462, 2008.

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

[18] 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.

[19] J.-P. Calliess, "Conservative decision-making and inference in uncertain dynamical systems," Ph.D. dissertation, University of Oxford, 2014.

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

[21] ——, "Robust learning-based MPC for nonlinear constrained systems," *Automatica*, vol. 117, p. 108948, 2020.

[22] J.-P. Calliess, "Lipschitz optimisation for Lipschitz interpolation," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 3141–3146.

[23] 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.

[24] Z. B. Zabinsky, R. L. Smith, and B. P. Kristinsdottir, "Optimal estimation of univariate black-box Lipschitz functions with upper and lower error bounds," *Computers & Operations Research*, vol. 30, no. 10, pp. 1539–1553, 2003.

[25] C. P. Kruskal, L. Rudolph, and M. Snir, "A complexity theory of efficient parallel algorithms," *Theoretical Computer Science*, vol. 71, no. 1, pp. 95–132, 1990.

[26] I. Chiuchisan, A. D. Potorac, and A. Graur, "Finite state machine design and vhdl coding techniques," *Development and Application Systems*, p. 75, 2010.

[27] R. Yates, "Fixed-point arithmetic: An introduction," *Digital Signal Labs*, vol. 81, no. 83, p. 198, 2009.

[28] J. M. Muller, N. Brisebarre, F. de Dinechin, C. P. Jeannerod, L. Vincent, G. Melquiond, N. Revol, D. Stehl, and S. Torres, "Handbook of floating point arithmetic, 2010," *Google Scholar Google Scholar Digital Library Digital Library*.

[29] O. Sicoe and M. Popa, "Comparison metrics for the output of a mixed fixed point and floating point graphics pipeline," in *2017 25th Telecommunication Forum (TELFOR)*. IEEE, 2017, pp. 1–4.

[30] H. Thibault, O. Hacène, and L. Benoit, "Optimal word-length allocation for the fixed-point implementation of linear filters and controllers," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 175–182.

[31] R. Nehmeh, D. Menard, E. Nogues, A. Banciu, T. Michel, and R. Rocher, "Fast integer word-length optimization for fixed-point systems," *Journal of Signal Processing Systems*, vol. 85, no. 1, pp. 113–128, 2016.

[32] P. Saracco, M. Batic, G. Hoff, and M. Pia, "Uncertainty quantification (UQ) in generic Monte Carlo simulations," in *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*. IEEE, 2012, pp. 651–656.

[33] D. Menard and O. Sentieys, "Automatic evaluation of the accuracy of fixed-point algorithms," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 2002, pp. 529–535.

[34] J. Cong, K. Gururaj, B. Liu, C. Liu, Z. Zhang, S. Zhou, and Y. Zou, "Evaluation of static analysis techniques for fixed-point precision optimization," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE, 2009, pp. 231–234.

[35] M. Grailoo, B. Alizadeh, and B. Forouzandeh, "UAFEA: Unified analytical framework for IA/AA-based error analysis of fixed-point polynomial specifications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 10, pp. 994–998, 2016.

[36] 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)*. IEEE, 2003, pp. 275–282.

[37] S. Vakili, J. P. Langlois, and G. Bois, "Finite-precision error modeling using affine arithmetic," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 2591–2595.

[38] T. Hickey, Q. Ju, and M. H. Van Emden, "Interval arithmetic: From principles to implementation," *Journal of the ACM (JACM)*, vol. 48, no. 5, pp. 1038–1068, 2001.

[39] V. Kotlyar and M. Moudgill, "Detecting overflow detection," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2004, pp. 36–41.

[40] W. Sung and K.-I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE transactions on Signal Processing*, vol. 43, no. 12, pp. 3087–3090, 1995.

[41] K. K. Poon, A. Yan, and S. J. Wilton, "A flexible power model for FPGAs," in *International Conference on Field Programmable Logic and Applications*. Springer, 2002, pp. 312–321.

[42] I. Kuon, R. Tessier, and J. Rose, *FPGA architecture: Survey and challenges*. Now Publishers Inc, 2008.
[43] L. J. Hwang and R. L. Sanchez, "Method and system for resource allocation in FPGA-based system-on-chip (SoC)," Jun. 6 2006, uS Patent 7,058,921.
[44] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014.
[45] C. Gonzalez, I. Alvarado, and D. Muñoz de la Peña, "Low cost two-wheels self-balancing robot for control education," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9174–9179, 2017.
[46] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

J.M. Nadales received his BSc in Electronic Engineering from the University of Cordoba in 2017. He moved to the University of Seville where he received his MSc in Control and System Engineering in 2018, specializing in nonlinear model predictive control, and his MSc in Microelectronic Design in 2019, specializing in the implementation of bio-inspired algorithms on configurable embedded platforms. He is currently a PhD candidate in the University of Seville where he is specializing in data-driven predictive control and metaheuristic optimization techniques. His main research line is the application of new predictive control strategies and optimization techniques to different fields such as supply chain logistics, power electronic systems, or intelligent air conditioning systems.

J.M. Manzano received his MSc degree in Industrial Engineering and his PhD in Automation Engineering from the University of Seville in 2016 and 2020, respectively. He carried out his research in the Department of Systems and Automation of the University of Seville, and was a visiting researcher at the University of Oxford in 2019. He is currently a Lecturer in the Department of Engineering at Universidad Loyola Andalucía, Spain. His research interests merge nonlinear model predictive control and data-based learning algorithms. He focuses on the study of the stability and robustness properties of MPCs whose models are inferred from observed data, using machine learning techniques.

A. Barriga is Full Professor in the Dpt. of Electronics and Electromagnetism at the University of Seville. Currently he is also at the Microelectronic Institute of Seville from the National Microelectronic Centre of Spain. He was the leader of the research unit called "Digital and Mixed Signal Design" for twenty years. His current research interest are in areas such as design methodologies of VLSI integrated circuits, in particular, CMOS digital circuit design, digital implementation of neuro-fuzzy systems, and CAD tools for fuzzy logic based systems. He is author of close to two hundred publications in books, magazines, and congresses. He has taken part in numerous research projects (national and European founded projects) as well as industrial development contracts.

D. Limon received the M.Eng. and Ph.D. degrees in electrical engineering from the University of Seville, Seville, Spain, in 1996 and 2002, respectively. From 1999 to 2007, he was an Assistant Professor with the Departamento de Ingeniería de Sistemas y Automática, University of Seville, where he was an Associate Professor from 2007 to 2017 and has been a Full Professor since 2017. He has been a Visiting Researcher with the University of Cambridge, Cambridge, U.K., and Mitsubishi Electric Research Laboratories in 2016 and 2018, respectively. His current research interests include model predictive control, stability and robustness analysis, tracking control, and data-based control. Dr. Limon has been a Keynote Speaker at the International Workshop on Assessment and Future Directions of Nonlinear Model Predictive Control in 2008 and a Semiplenary Lecturer at the IFAC Conference on Nonlinear Model Predictive Control in 2012. He has been the Chair of the fifth IFAC Conference on Nonlinear Model Predictive Control in 2015.