High-Speed and Low-Cost Implementation of Explicit Model Predictive Controllers

Andrés Gersnoviez[1], María Brox[2] and Iluminada Baturone[3]

***Abstract*—This paper presents a new form of Piecewise-Affine (PWA) solution, referred to as PWAH, to approximate the explicit Model Predictive Control (MPC) law, achieving a very rapid control response with the use of very few computational and memory resources. This is possible because PWAH controllers consist of Single-Input Single-Output (SISO) PWA modules connected in cascade so that the parameters needed to define them increase linearly instead of exponentially with the input dimension of the control problem. PWAH controllers are not universal approximators but several explicit MPC controllers can be efficiently approximated by them. A methodology to design PWAH controllers is presented and validated with application examples already solved by MPC approaches. The designed PWAH controllers implemented in Field Programmable Gate Arrays (FPGAs) provide the highest control speed using the fewest resources compared to the other digital implementations reported in the literature.**

***Index Terms*—Field-programmable gate arrays (FPGAs), hierarchical systems, model predictive control (MPC), piecewise-affine (PWA) systems, PWA controllers.**

I. Introduction

Model Predictive Control (MPC), also called receding or rolling horizon control, obtains the control action by solving a finite horizon open-loop optimal control problem at each sampling instant [1]-[3]. The plant to be controlled is usually modeled as a linear discrete-time system:

$$\begin{cases} x(t+1) = Ax(t) + Bu(t). \\ y(t) = Cx(t). \end{cases} \tag{1}$$

Where $x(t) \in \mathbb{R}^n$ is the state vector, $y(t) \in \mathbb{R}^p$ is the output vector, and $u(t) \in \mathbb{R}^m$ is the control vector (input vector to the plant), which satisfy the constraints $y_{min} \leq y(t) \leq y_{max}$ and $u_{min} \leq u(t) \leq u_{max}$, for $t \geq 0$, and the pair (A, B) is stabilizable.

Assuming for simplicity that the control objective is to regulate the plant state to the origin, and that a full measurement of the state $x(t)$ is available at the current time t, MPC solves the constrained regulation problem by optimizing a performance index, which is usually expressed as a linear or quadratic criterion. For example, in the case of a quadratic criterion, the following optimization problem is solved at each time step t:

$$\min_{U} \left\{ \begin{array}{l} J(U, x(t)) = x^T_{t+N_y|t} P x_{t+N_y|t} + \\ + \sum_{k=0}^{N_y-1} [x^T_{t+k|t} Q x_{t+k|t} + u^T_{t+k} R u_{t+k}] \end{array} \right\} \tag{2}$$

Where:

$$U \triangleq \{u_t, \ldots, u_{t+N_u-1}\}. \tag{3}$$

is subject to:

$$\begin{array}{l} y_{min} \leq y_{t+k|t} \leq y_{max},\ k = 1, \ldots, N_c. \\ u_{min} \leq u_{t+k} \leq u_{max},\ \ k = 0, \ldots, N_c. \\ x_{t|t} = x(t). \\ x_{t+k+1|t} = Ax_{t+k|t} + Bu_{t+k},\ k \geq 0. \\ y_{t+k|t} = Cx_{t+k|t},\ k \geq 0. \\ u_{t+k} = Kx_{t+k|t},\ N_u \leq k < N_y. \end{array} \tag{4}$$

Where $x_{t+k|t}$ is the predictive state vector at time $t + k$, obtained by applying the input sequence $\{u_t, \ldots, u_{t+k-1}\}$ to the model (1), starting from the state $x(t)$; N_y, N_u and N_c are the output, input and constraint horizons, respectively; K is some feedback gain; and it is assumed in (2) that $Q = Q^T \succcurlyeq 0$, $R = R^T \succ 0$ and $P \succcurlyeq 0$, where Q^T denotes the transpose of Q [4].

The MPC control law is based on the idea that, at each time step t, the optimal solution $U^*(t) = \{u^*_t, \ldots, u^*_{t+N_u-1}\}$ for the problem (2) is calculated, but only the first action $u(t) = u^*_t$ is applied as input to the plant in (1) to obtain the new state $x(t+1)$. At the following time step, the optimization problem (2) is repeated over a shifted time-horizon and based on the new state vector [1]-[2].

An obstacle to a wider use of MPC controllers is the high computational cost involved to solve the optimization problem online. This has limited their use to applications of relatively slow dynamics (sample time measured in milliseconds or more even using computational methods for improving the speed of online optimization [3]). A solution proposed to reduce this shortcoming is the use of explicit MPC controllers which solve the optimization problem offline and compute the optimal control action, $u(t)$, as an "explicit" function of the state variables, $x(t)$, within a given domain D, which is assumed to be polytopic. Hence, online optimization is reduced to a simple function evaluation, $u(x) : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ [4]-[6]. In most cases, as explained in [6], such a function is piecewise affine (PWA) in the state variables, as follows:

Manuscript received October 28, 2016; revised March 16, 2017 and October 30, 2017; accepted November 3, 2017. Manuscript received in final form November 14, 2017. This work was supported by the Ministerio de Economía, Industria y Competitividad of the Spanish Government through the Project TEC2014-57971-R (cofunded by FEDER).

[1]A. Gersnoviez is with the Department of Electronic and Computer Engineering, Universidad de Córdoba, Córdoba, Spain (phone: +34 957212224; e-mail: andresgm@uco.es).

M. Brox is with the Department of Electronic and Computer Engineering, Universidad de Córdoba, Córdoba, Spain (e-mail: mbrox@uco.es).

I. Baturone is with the Instituto de Microelectrónica de Sevilla (IMSE-CNM), Universidad de Sevilla, CSIC, Seville, Spain (e-mail: lumi@imse-cnm.csic.es).

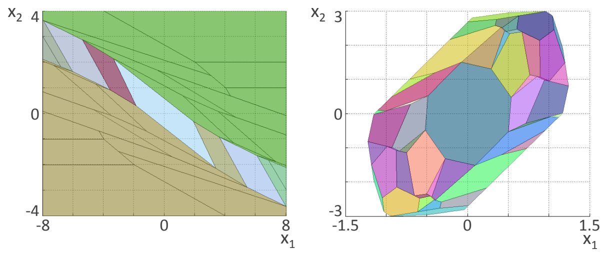


Fig. 1. Polyhedral partitions with (a) saturated regions and (b) without them.

$$u(x) = \begin{cases} F_1 x + g_1 & \text{if} \quad x \in \mathcal{P}_1. \\ & \vdots \\ F_I x + g_I & \text{if} \quad x \in \mathcal{P}_I. \end{cases} \tag{5}$$

Where $F_i \in \mathbb{R}^{m \times n}$, $g_i \in \mathbb{R}^m$ $(i = 1, \ldots, I)$, and $\mathcal{P}_i \subset D$ are I non overlapped regions ($\mathcal{P}_i \cap \mathcal{P}_j = \emptyset$ for $i \neq j$), called polytopes, which form a polyhedral partition of the domain D, fulfilling the relation $\cup_{i=1}^{I} \mathcal{P}_i = D$. The domain D is divided by H $n-1$ dimensional hyperplanes of the type $h_j^T x + k_j = 0$ $(j = 1, \ldots, H)$, with $h_j \in \mathbb{R}^n$ and $k_j \in \mathbb{R}$. Each hyperplane divides the domain into two parts and their intersection generates the edges of the I polytopes.

Several implementations of explicit MPC controllers have been proposed in the literature [7]-[28]. Some of them implement the optimal solution, $u(x)$ in Equation (5), without any approximation [7]-[19]. Others replace the optimal solution by a sub-optimal one, which is very similar to $u(x)$ but with reduced complexity so as to gain in latency as well as in memory requirements [20]-[28]. The problem is that the complexity of several sub-optimal explicit MPC implementations increases exponentially with the number of state variables [22]-[27].

This paper presents a new implementation of explicit MPC controllers which is referred to as PWAH (PWA Hierarchical) because it is composed of a hierarchy of very simple PWA modules. It approximates the optimal MPC control law, achieving a high speed and requiring very few resources which increase linearly instead of exponentially with the number of state variables. A very preliminary work about this subject can be seen in [28]. Since PWAH controllers are not universal approximators, a methodology is presented to evaluate if the optimal explicit MPC law can be well approximated by a PWAH approach and how to design it.

The paper is organized as follows. Section II summarizes the advantages and drawbacks of solutions reported in the literature to implement explicit model predictive controllers. Section III presents the PWAH solution, describing the shapes of their polytopes and describing their control response time and required resources. The capability of PWAH controllers to approximate optimal controllers is discussed in Section IV. Section V describes the methodology to design PWAH controllers. Application examples of applying the methodology and the results obtained are given in Section VI. Finally, Section VII shows conclusions.

II. Implementations of Explicit MPC

Once the state space is divided offline into I polytopes, the online computation of explicit MPC controllers (as shown in Equation (5)) is reduced to find the polytope which the current state variables belong to, and then to evaluate the control action as the linear function associated to that polytope. Since the polyhedral partition tends to be irregular, the problem of determining the polytope is known as the *point location problem* [7]-[9].

The so-called exhaustive or direct search explores sequentially all the polytopes to find where the point is located. Hence, the search time is linear in the number of polytopes. The implementation reported in [10] (henceforth denoted as PWA-direct) follows this approach. The search complexity can be reduced by designing a binary search tree. Using a balanced search tree, the search time is logarithmic in the number of polytopes [11]. Several implementations of explicit MPC controllers using a binary search tree have been proposed in the literature. They are called Generic PWA (PWAG) implementations [11]-[13]. The implementations in [11]-[12] use a VLSI (Very Large Scale Integration) device as target platform, and [13] uses an FPGA (Field Programmable Gate Array). The architecture employed in PWA-direct and PWAG approaches needs to store the parameters F_i and g_i $(i = 1, \ldots, I)$ in (5) as well as the parameters associated to the edges of the polytopes h_j and k_j $(j = 1, \ldots, H)$. As advantage, they can describe completely irregular polyhedral partitions, with any form of the polytopes as shown in Fig. 1. As drawbacks, the number of required parameters to represent the PWA function is significantly higher than in the other types of implementations, as shown in the following.

If the number of polytopes is large, which happens whenever many constraints are imposed and a large prediction horizon is used, the storage requirements as well as the tree complexity make PWA-direct and PWAG solutions prohibitive for low-cost implementations [9]. The concept of bounding boxes and interval trees is used in [14] to cope with a large number of polytopes. Although there are control problems for which this method offers a significant improvement, the worst-case complexity (in terms of memory and latency) is linear in the number of polytopes. Another solution to replace the possibly complex binary search tree is the two-stage hash-based algorithm proposed in [7], which uses a much simpler artificial hyperrectangular domain partition. The first stage identifies the subset of polytopes addressed by the hyperrectangle where the input is located and the second stage performs direct search on the subset. The implementation reported in [9] (henceforth denoted as PWA-hash) follows this approach. The cost to pay is higher latency than using the complex binary search tree.

Continuous PWA functions with completely irregular polyhedral partitions can also be represented by using a scheme based on the *lattice theory*, where the affine pieces of the function are properly selected without taking into account the edges of the polytopes explicitly [15]-[17]. Hence, the so-called Lattice PWA (PWAL) implementations require lower memory resources than PWA-direct and PWAG solutions. Besides, PWAL implementations cluster into super regions the polytopes which have the same linear control action (for example, all the polytopes filled with green color in Fig. 1a form a super region so that only one F_i and g_i parameters are stored for all of them). As drawback, the latency of the control action can be high if there are many polytopes which cannot be clustered into super regions.

Typical explicit MPC controllers usually contain super regions where the optimal control action is either constantly on the upper limit or on the lower limit. For those cases, the approach proposed in [18] to reduce on-line computation is to construct, explicitly or implicitly, a separator which indicates where the control action is saturated or not. Complexity reduction is proportional to the ratio between the number of unsaturated regions to the total number of regions.

In the case of continuous PWA functions, the lattice-based orthogonal truncated binary search tree (LOTBST) methods proposed in [8] allow combining the low memory resources of lattice approaches with the low search time of binary trees. For more general partitions, e.g., with discontinuities and overlapping, the orthogonal truncated binary search tree (OTBST) can be combined with a direct search.

Another set of procedures approximate the optimal control action, $u(x)$, in (5) by a sub-optimal solution in order to reduce the complexity of the implementation. The idea in [20] is based on relaxing the first-order optimality conditions. The method in [21] employs a simpler polyhedral partition by using a lower value of the prediction horizon and minimizes the integrated squared error between the sub-optimal and the optimal controllers.

An improvement in the latency as well as a decrease in memory requirements are achieved if sub-optimal controllers consider regular partitions. The so-called Simplicial PWA (PWAS) implementations were proposed with this objective [22]. They describe simplicial partitions where the polytopes have a simplex shape [23]-[25]. The state space is partitioned into orthogonal hypercubes in [26]. The optimal solution is computed explicitly only at the vertices of these hypercubes, and those values are employed to approximate the sub-optimal solution valid in the whole hypercube. The point location problem is solved by a quad-tree or oct-tree, such that the search time is logarithmic with respect to the number of regions. Implementations of sub-optimal explicit MPC controllers that use these search trees have been proposed in the literature [27]. They are called Hyperrectangular PWA (PWAR) implementations.

All the optimal implementations of explicit MPC controllers commented above (PWAG-direct, PWAG, PWAG-hash and PWAL) increase in complexity (and, hence, in memory resources and latency) if the number of state variables is high. Most of the sub-optimal approaches presented in the literature provide theoretical bounds on the approximation error and guarantee closed-loop stability. In the case of the PWAS and PWAR approaches, which provide high-speed implementations, the price to pay is that the number of vertices of the polyhedral partition (and, hence, the memory required to store the related information) increases exponentially with the number of state variables. Next section describes how the memory required by the proposed PWAH approach increases linearly instead of exponentially with the input dimension. Next section and the following also explain the conditions that should be met by MPC controllers to be well approximated by PWAH approaches, since PWAH controllers are not universal approximators.

III. Hierarchical PWA Implementations (PWAH)

Let us consider an explicit MPC controller with $x(t) = \{x_1, \ldots, x_n\} \in \mathbb{R}^n$ as the state vector and $u(t) \in \mathbb{R}$ as the control action. The proposed PWAH implementation is formed exclusively by single-input and single-output (SISO) PWA modules, connected in cascade as shown in Fig. 2, with addition operators between them (the figure shows only subtraction operators for simplicity but only addition operators or a combination of both may also be used).

A SISO PWA$_j$, with input in_j and output out_j, divides the universe of discourse of in_j into I_j non overlapped intervals and provides a linear output according to the interval i which the input belongs to:

$$out_j = f_{ji} \cdot in_j + g_{ji}. \tag{6}$$

The number of polytopes in the PWAH implementation composed of n SISO modules is $\prod_{j=1}^{n} I_j$. The advantage is that the search complexity of the polytope $\mathcal{P}_i$ where the state vector is located is logarithmic in the number of polytopes, $\mathcal{O}(n)$, because each SISO PWA$_j$ explores its intervals I_j to find the interval i associated to the polytope $\mathcal{P}_i$ (referred to as $I_j^{(i)}$). That is, if $x \in \mathcal{P}_i$, then (as shown in Fig. 2) $x_1 \in I_n^{(i)}$, $sub_{n-1} = (y_n - x_2) \in I_{n-1}^{(i)}, \ldots$, and $sub_1 = (y_2 - x_n) \in I_1^{(i)}$.

Therefore, the module which provides the control action (PWA$_1$ in Fig. 2), verifies for the polytope $\mathcal{P}_i$ that:

$$\begin{aligned} u(x) &= f_{1i} \cdot sub_1 + g_{1i} = f_{1i}(y_2 - x_n) + g_{1i} \\ &= f_{1i} y_2 - f_{1i} x_n + g_{1i} \quad \text{with } (y_2 - x_n) \in I_1^{(i)}. \end{aligned} \tag{7}$$

Substituting the output y_2 by $(f_{2i} \cdot sub_2 + g_{2i})$ and sub_2 by $(y_3 - x_{n-1})$:

$$\begin{aligned} u(x) &= f_{1i}(f_{2i} \cdot sub_2 + g_{2i}) - f_{1i} x_n + g_{1i} \\ &= f_{2i} f_{1i}(y_3 - x_{n-1}) - f_{1i} x_n + f_{1i} g_{2i} + g_{1i} \\ &= f_{2i} f_{1i} y_3 - f_{2i} f_{1i} x_{n-1} - f_{1i} x_n + f_{1i} g_{2i} + g_{1i}, \\ &\text{with } (y_3 - x_{n-1}) \in I_2^{(i)} \text{ and } (y_2 - x_n) \in I_1^{(i)}. \end{aligned} \tag{8}$$

Repeating the steps above until the first module in the hierarchy (PWA$_n$ in Fig. 2), it can be seen that the control action provided for the polytope $\mathcal{P}_i$ is PWA in the state variables as follows:

$$\begin{aligned} u(x) =& f_{ni} f_{(n-1)i} \cdots f_{1i} x_1 - f_{(n-1)i} f_{(n-2)i} \cdots f_{1i} x_2 - \\ & \cdots - f_{2i} f_{1i} x_{n-1} - f_{1i} x_n + g_i \quad \text{if } x \in \mathcal{P}_i. \end{aligned} \tag{9}$$

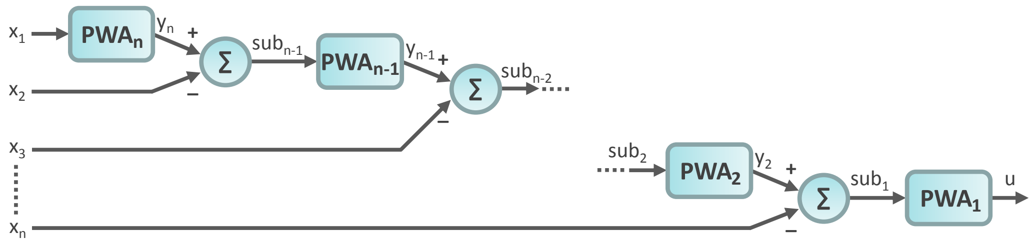


Fig. 2. Block diagram of a PWAH implementation.

Where $f_{1i}, \ldots, f_{ni}$ and g_i are constants $\in \mathbb{R}$, with:

$$\begin{aligned} g_i =& f_{(n-1)i} \cdots f_{1i} g_{ni} + f_{(n-2)i} \cdots f_{1i} g_{(n-1)i} + \\ & \cdots + f_{1i} g_{2i} + g_{1i} \quad \text{if } x \in \mathcal{P}_i. \end{aligned} \tag{10}$$

Hence, a PWAH implementation provides a multi-input single-output (MISO) PWA controller by a simple combination of SISO PWA modules. A multi-input multi-output (MIMO) controller can be implemented by using a structure as shown in Fig. 2 for each output.

A. The shape of the polytopes

As $u(x)$ is a PWA function of the state variables $\{x_1, \ldots, x_n\}$, y_2 is a PWA function of the state variables $\{x_1, \ldots, x_{n-1}\}$, $\ldots$, y_{n-1} is a PWA function of the state variables $\{x_1, x_2\}$ and y_n is a PWA function of the state variable x_1. While the influence of x_n on the control action can be only modified by the module PWA_1, the influence of x_1 on the control action can be modified by all the modules. This produces a relationship between the state variables as described in the following Lemma.

Lemma 1: Given a PWAH implementation with n inputs, as shown in Fig. 2, it is verified that the values of the state variables $\{x_1, \ldots, x_n\}$ that provide the same value of control action (thus forming a level curve or, in general, a level hypersurface) verify that:

$$x_n = \mathcal{F}_{PWA}(x_1, \ldots, x_{n-1}) + b. \tag{11}$$

Where $\mathcal{F}_{PWA} : \mathbb{R}^{n-1} \to \mathbb{R}$ is a PWA function which is equal for all the level hypersurfaces and it is provided by the output of the module PWA_2, and $b \in \mathbb{R}$ is a constant that depends on the constant value of the control action and the interval $I_1^{(i)}$ where $(y_2 - x_n)$ belongs to.

The proof of Lemma 1 is as follows. If $(y_2 - x_n)$ belongs to $I_1^{(i)}$, then the control action takes the expression in Equation (7). If the values of the state variables $\{x_1, \ldots, x_n\}$ provide the same value of control action, $u(x) = C_{qu}$, then, according to Equation (7), it follows that:

$$u(x) = C_{qu} = f_{1i} \cdot y_2 - f_{1i} \cdot x_n + g_{1i}. \tag{12}$$

The output of the module PWA_2 y_2, is a PWA function of $\{x_1, \ldots, x_{n-1}\}$. If it is named as $\mathcal{F}_{PWA}(x_1, \ldots, x_{n-1})$, Equation 12 can be rewritten as:

$$x_n = \mathcal{F}_{PWA}(x_1, \ldots, x_{n-1}) + \frac{g_{1i} - C_{qu}}{f_{1i}}. \tag{13}$$

Where $(g_{1i} - C_{qu})/f_{1i}$ is a constant that depends on the constant value of the control action, C_{qu}, and the parameters, f_{1i} and g_{1i}, associated to the interval $I_1^{(i)}$ where $(y_2 - x_n)$ belongs to.

Likewise, the outputs of SISO modules from PWA_2 to PWA_{n-1} have similar features to those stated in Lemma 1 for the control action.

For example, let us consider the system with two inputs shown in Fig. 3a, formed by two SISO PWA modules, PWA_1 and PWA_2, which are characterized by the parameters shown in Fig. 3b and 3c. The control action (u) versus the state variables is the surface shown in Fig. 4a. Several level curves obtained when the control action takes the same value between the minimum and the maximum are shown in Fig. 4b. According to Lemma 1, it can be seen that they follow Equation (11), in this case:

$$x_2 = \mathcal{F}_{PWA}(x_1) + \frac{g_{1i} - C_{qu}}{f_{1i}} = y_2 + \frac{g_{1i} - C_{qu}}{f_{1i}}. \tag{14}$$

Let us analyze the polyhedral partitions provided by PWAH implementations. For example, let us consider again the system in Fig. 3a whose partition in polytopes is shown in Fig. 4c. The three polytopes which has the same maximum control action can be merged into a super region. Similarly, the three polytopes with the same minimum control action can be merged. Analyzing the polyhedral partition, it can be seen how the slopes of some of the polytopes' edges are the same as the slopes of the three pieces described by the system PWA_2 (Fig. 3b). The rest of the edges are parallel to the axis x_2, which is the variable that is added (subtracted) to the output of the system PWA_2. As a matter of fact, the PWA function described by PWA_2, which is seen in the level curves, can be seen also repeated four times in the polyhedral partition, displaced in parallel to the axis x_2. The number of repetitions (four) is defined by the number of breakpoints (four) in the system PWA_1 (Fig. 3c). Their displacement of 0.75, 0.5 and 0.75 along the axis x_2 are defined by the distance between the breakpoints measured in the input variable of the system PWA_1 (a distance of 0.75 in sub_1 input between the first and the second breakpoint, 0.5 between the second and the

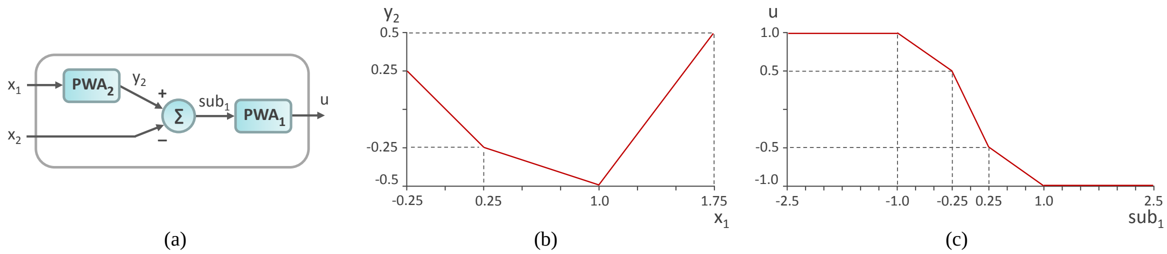


Fig. 3. (a) PWAH system with two inputs; (b) PWA function of the module PWA_2; (b) PWA function of the module PWA_1.

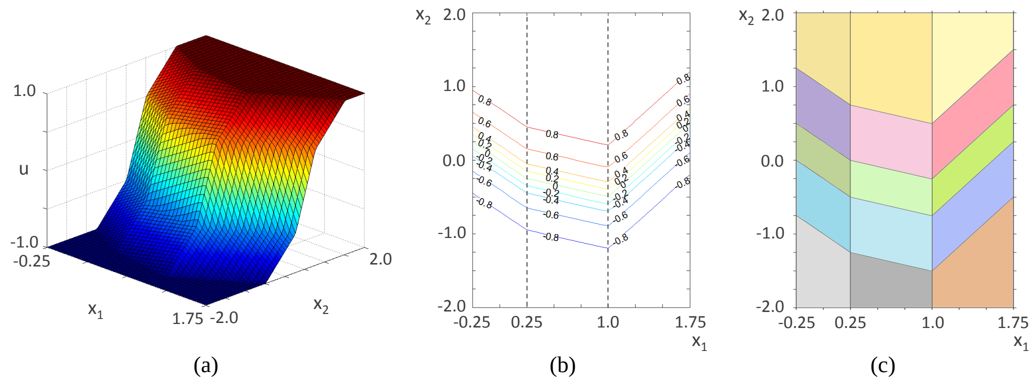


Fig. 4. (a) Output versus inputs of the system described in Fig. 3; (b) Level curves of the output surface; (c) Polyhedral partition.

third breakpoint, and 0.75 between the third and the fourth breakpoint). In general, the following Lemma is verified.

Lemma 2: Given a PWAH implementation with n inputs, as shown in Fig. 2, it is verified that: (a) the unsaturated polytopes are hyperparallelepipeds which always have two of their edges parallel to the axis x_n, which is the variable that is finally added (subtracted) in the cascade of PWA modules, and (b) the other edges of the polytopes, which are parallel between them in pairs, are linear in the state variables $\{x_1, \ldots, x_{n-1}\}$ and their slopes are given by the output of the module PWA_2, like the slopes of the level hypersurfaces.

The proof of Lemma 2 is as follows. The affine functions of the output of the last module in the hierarchy (PWA_1 in Fig. 2) depend on which interval, $I_1^{(i)}$, the value of $sub_1 = y_2 - x_n$ belongs to, according to Equation (7). The state variable x_n only introduces a displacement (offset) in the value of sub_1. Depending on such displacement (parallel to the axis x_n) the value of sub_1 belongs to an interval, $I_1^{(i)}$, or another, that is, the state variables $\{x_1, \ldots, x_n\}$ belong to a polytope or another. Since the displacement is parallel to the axis x_n, there are always two edges of the polytopes which are parallel to the axis x_n. Since the displacement with regards to y_2 is constant to belong to an interval or another, the other edges of the polytopes are parallel between them in pairs. Those pairs of edges are linear in the state variables $\{x_1, \ldots, x_{n-1}\}$ because they are provided by y_2, which is a PWA function of $\{x_1, \ldots, x_{n-1}\}$, and, hence, their slopes depend on the interval, $I_2^{(i)}$, which the value of sub_2 belongs to.

B. Control response time and required resources

The basic modules in a PWAH implementation are the SISO PWA modules. The point location problem with only one input is solved simply by comparing the input with the breakpoints, $I_j - 1$, of the univariate PWA_j function. The fastest way to realize comparisons with digital circuitry is to use combinatorial logic based on a bank of $(I_j - 1)$ comparators. Since the logic is not complex, high clock frequency constraints can be satisfied when synthesizing the combinatorial circuit. Once the interval the input belongs to has been identified, an adder and a multiplier are required to implement the Equation (6). If 2 combinatorial adders and 1 multiplier are employed, only one clock cycle is required to process the input through a SISO PWA module and to obtain the valid input for the following SISO module in the cascade. The circuit latency (the number of clock cycles between the arrival of a new state vector value and the calculation of the corresponding control action) is, therefore, n clock cycles if n SISO modules are employed, as shown in Fig. 2.

If the circuit throughput (the number of clock cycles between two successive valid control actions and, hence, the sampling time of state variables) is also n clock cycles, the same adders and the multiplier can be exploited to implement successively the n affine functions. The throughput (and sampling time) can be reduced to only one clock cycle by using pipeline techniques that divide the system into as many stages as SISO modules. Thus, every SISO module is computing a SISO PWA function every clock cycle. For example, the module PWA_1 can be providing the control action corresponding to the state variables sampled n clock cycles before, while PWA_2 can be computing the output y_2 corresponding to the state variables sampled $(n-1)$ clock cycles before,$\ldots$, while PWA_n can be computing the output y_n corresponding to the state variables most recently sampled. In that case, adders and multipliers cannot be reused so that $2n-1$ adders and n multipliers are required. The latency is n clock cycles if n SISO modules are employed, but there is a trade-off between throughput and resources depending on the number of pipeline stages employed. Hence, the approach called intra-delay sampling can be implemented [29].

If the number of pieces of a SISO PWA_i module is I_i, then the input value should be compared to $I_i - 1$ breakpoints to solve the point location problem. If the number of bits to represent the breakpoints and the parameters f_{ji} and g_i of the affine pieces is n_{bit}, the number of bits required by the PWAH implementation is:

$$n_{bit} \cdot \sum_{i=1}^{n} \left[(I_i - 1) + 2I_i\right] = n_{bit} \cdot \sum_{i=1}^{n} (3I_i - 1). \tag{15}$$

The proposed PWAH implementation is compared with other PWA implementations reported in the literature in Table I. The symbols employed are: the input dimension of the PWA function (n), the depth of the binary search tree (d), the number of nodes in the search tree (N), the number of different local affine functions (W), the number of edges defining the polytopes (E), the number of regions involved in the direct search of the PWA-hash (m_0), the subset of polytopes that are directly searched ($\mathcal{J}$), the number of edges of polytope $\mathcal{P}_k$ (E_k), the number of bits representing the input (n_{bit}), the number of vertices in the simplicial partition ($n_{bit} \cdot \prod_{i=1}^{n} (I_i + 1)$, where I_i is the number of partitions for each input), the order of the trees in the Multitree (M), the internal height of the Multitree (h_M), the number of rows in the simplified structure matrix of the lattice representation (X), and U, which is related to the number of super regions in the lattice representation.

The proposed PWAH implementation approximates the optimal MPC, like PWAS and PWAR realizations. The latency provided is similar to serial PWAS and PWAR realizations (and faster than PWAG and PWAL approaches), using the same number of multipliers but with the advantage of requiring a much smaller memory which increases linearly instead of exponentially with the input dimension. Hence, if the optimal MPC can be approximated adequately by a PWAH solution, a high speed is provided with low cost in computational and memory resources. These advantages are illustrated quantitatively in the examples shown in Section VI. In addition, that section will show that, for particular applications, further simplifications can reduce resource requirements as well as increase the control speed.

IV. PWAH Approximation of MPC

PWAS and PWAR implementations employ polyhedral partitions based on simplexes and hyperrectangles, respectively.

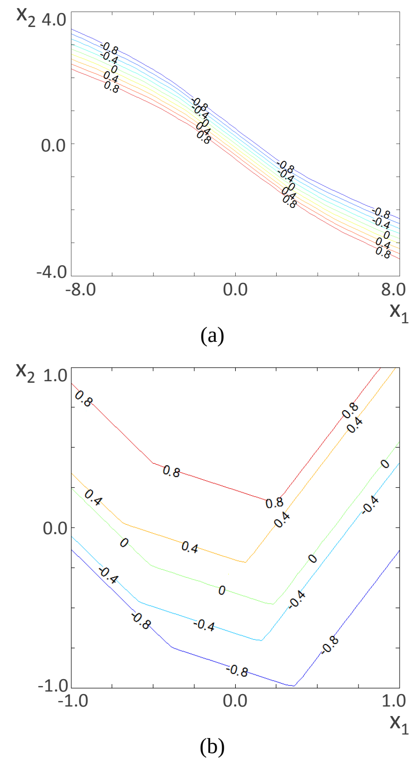


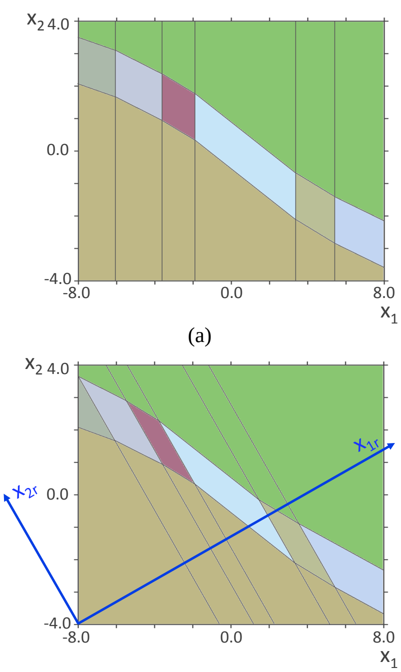
Fig. 5. (a) Level curves of the control action of the system shown in Fig. 1a; (b) Example of level curves that are not suitable for PWAH approach.

The affine control actions provided by PWAS and PWAR implementations are independent of the edges of the polytopes where the state variables are located. They can approximate any explicit MPC controller with an arbitrarily small error, using small simplexes and hyperrectangles, although this can require a huge number of polytopes. In the other side, PWAH implementations employ polyhedral partitions based on hyper-parallelepipeds that fulfill the conditions expressed in Lemma 2. Besides, the affine control actions provided by PWAH implementations fulfill the conditions expressed in Lemma 1, so that they are related to the edges of the polytopes where the state variables are located. With such constraints, PWAH cannot achieve an arbitrarily good approximation of any controller, but there are many explicit MPC controllers that can be approximated by PWAH implementations with a small error (although not arbitrarily small), using a small number of polytopes.

The approximation errors introduced by PWAH implementations depend on the shapes of the level hypersurfaces and polytopes of the optimal MPC to approximate. To illustrate this point, let us consider the level curves shown in Fig. 5a corresponding to the optimal MPC controller whose polytopes are shown in Fig. 1a. Those level curves and polytopes can be approximated with a small error by level curves and polytopes that fulfill the conditions expressed in Lemma 1 and Lemma 2, as illustrated in Fig. 6a. On the other hand, level curves such as those in Fig. 5b or polytopes such as those in Fig. 1b

TABLE I
COMPARISON BETWEEN PWA IMPLEMENTATIONS

	Latency (number of clock cycles)	Memory (bits to store)	Multipliers	Exact optimal MPC
PWAG-direct [10]	I	$n_{bit}(n+1)(I+E)$	1	yes
PWAG (FPGA) [13]	$n+2\cdot d+n\cdot d+2$	$n_{bit}(n+1)(I+E)$	1	yes
PWAG (ASIC) [12]	$n+2\cdot d$	$n_{bit}(n+1)(D+E)+N\log_2(I+E)$	n	yes
PWAG-hash [9]	$12+I+6\cdot m_0+\max\left(\sum_{k\in\mathcal{J}}E_k\right)$	$n_{bit}(n+1)(D+E)+I\cdot(\log_2 E+1)$	n	yes
PWAL [17]	$n+U+1$	$n_{bit}(n+1)W+U\cdot(\log_2 X+1)$	$2n$	yes
MultiTree [19]	h_M+2	-	$n\cdot M$	yes
PWAS(serial) [25]	$n+4$	$n_{bit}\prod_{i=1}^{n}(I_i+1)$	1	no (approx.)
PWAS(parallel) [25]	3	$n_{bit}(n+1)\prod_{i=1}^{n}(I_i+1)$	$n+1$	no (approx.)
PWAR(serial) [27]	$n+2$	$n_{bit}(n+1)\prod_{i=1}^{n}(I_i+1)$	1	no (approx.)
PWAR(parallel) [27]	2	$n_{bit}(n+1)\prod_{i=1}^{n}(I_i+1)$	n	no (approx.)
PWAH	n	$n_{bit}\sum_{i=1}^{n}(3I_i-1)$	1	no (approx.)



(a)

(b)

Fig. 6. PWAH partition of the system shown in Fig. 1a: (a) Using the original reference system; (b) Using a new rotated reference system.

cannot be well approximated by a PWAH structure.

If an explicit MPC can be well approximated by a PWAH implementation, several implementations can be considered, depending on the approximation error desired. Continuing with the case of Fig. 1a, the PWAH structure shown in Fig. 3a can be used, setting the module PWA_2 in order to reproduce the PWA function displaced along x_2 in the polytopes, and repeating it the right number of times with the module PWA_1. The polyhedral partition shown in Fig. 6a was obtained by using 6 and 3 pieces in PWA_2 and PWA_1, respectively (which gives 6 unsaturated polytopes and 2 saturated regions). Although this approximation can be pretty good, the approximation error can be reduced if the original reference system, formed

TABLE II
PWAH APPROXIMATION OF THE EXPLICIT MPC IN FIG. 1A WITH DIFFERENT POLYTOPES

No. of unsaturated polytopes	Rotation angle (θ_r)	RMSE
1	$0°$	6.2%
5	$0°$	1.2%
8	$0°$	1.2%
8	$30°$	0.1%

by the state variables $\{x_1, x_2\}$, is rotated to obtain a new one, $\{x_{1r}, x_{2r}\}$, so that the PWA function generated by the PWA_2 module is displaced in parallel to the new axis x_{2r}, as shown in Fig. 6b. The proposed PWAH implementation shown in Fig. 2 can operate with the original state variables $\{x_1, \ldots, x_n\}$ or with the variables $\{x_{1r}, \ldots, x_{nr}\}$ that result from rotation. Rotations are linear operations so that they can be considered as part of the linear conditioning that is always applied to the inputs of any controller. This will be explained more in detail in Sections V and VI. Table II illustrates the approximation accuracy provided by PWAH implementations when the explicit MPC in Fig. 1a is considered and much or fewer polytopes (with or without rotation) are used. Small errors are achieved with few polytopes.

There are many explicit MPC controllers that have many saturated polytopes and a small number of unsaturated polytopes, as shown in [18]. Hence, if they can be well approximated by a PWAH approach, the complexity of the implementation is small to approximate the small number of unsaturated polytopes. In addition, the implementation complexity becomes smaller as the constraints on the control action are tighter and the penalty on it is lower, because fewer polytopes are unsaturated and more become saturated. Another advantage of PWAH implementations is that their complexity does not increase significantly with the prediction horizon, since a high prediction horizon usually increases the saturated polytopes significantly but only slightly the

TABLE III
PWAH APPROXIMATION OF THE EXPLICIT MPC IN FIG. 1A WITH DIFFERENT HORIZONS

Hori-zon	Polytopes (Optimal)	Unsat. polytopes (PWAH)	Rot. angle (θ_r)	RMSE
4	25	5	0°	1.2%
6	57	5	0°	1.2%
8	87	5	0°	1.2%
10	119	5	0°	1.2%
12	149	5	0°	1.2%

unsaturated polytopes. Table III illustrates the approximation accuracy provided by PWAH implementations with the same number of polytopes when the explicit MPC in Fig. 1a (with a prediction horizon of 4) is obtained with higher prediction horizons. Similar and small errors are achieved in all the cases.

An important feature concerning approximation that can be met by a PWAH implementation is local optimality, that is, zero approximation error around the origin. This feature makes the PWAH implementation inherit local stability and local frequency response properties of the optimal MPC controller [24]. This is formalized in the following Lemma.

Lemma 3: Let us consider $\mathcal{P}_{eq}^{(opt)}$ is the optimal polytope that contains the origin, $(x_1,\ldots,x_n)=(0,\ldots,0)$, where the optimal MPC controller provides an optimal control action equal to $F_{eq}\cdot x = F_{1eq}\cdot x_1 - F_{2eq}\cdot x_2 - \cdots - F_{neq}\cdot x_n$. The PWAH implementation achieves local optimality if the approximated polytope $\mathcal{P}_{eq}$ that contains the origin verifies that $\mathcal{P}_{eq}\subseteq\mathcal{P}_{eq}^{(opt)}$ and, according to equations (9) and (10):

$$\begin{array}{l} f_{1eq}=F_{neq}, \quad f_{2eq}=F_{(n-1)eq}/F_{neq},\\ f_{3eq}=F_{(n-2)eq}/F_{(n-1)eq},\ldots,\ f_{neq}=F_{1eq}/F_{2eq},\\ F_{2eq}\cdot g_{neq}+F_{3eq}\cdot g_{(n-1)eq}+\cdots+F_{neq}\cdot g_{2eq}+g_{1eq}=0. \end{array} \tag{16}$$

The proof follows directly from equations (9) and (10) to equal optimal control action at origin and around.

For simplicity in the design, the offset terms g_{neq} to g_{1eq} can be set to zero, as employed in the methodology described in the following section.

V. DESIGN METHODOLOGY OF PWAH CONTROLLERS

Let us consider an explicit MPC controller with $x(t)=\{x_1,\ldots,x_n\}\in\mathbb{R}^n$ as the state vector and $u(t)\in\mathbb{R}$ as the control action, fulfilling the Equation (5). If $u(t)\in\mathbb{R}^m$ then this methodology is applied to the m components. The starting point to find a PWAH solution is to select one of the n state variables, x_i, to be separated from the rest, decomposing the system in the structure of Fig. 7. According to equation (7), the aim is to express $u(x)$ as:

$$\begin{array}{c} u(x)=f_{1j}\cdot x_i^{lev}-f_{1j}\cdot x_i+g_{1j}, \quad \text{with } (x_i^{lev}-x_i)\in I_1^{(j)}\\ \text{and } x_i^{lev}=\mathcal{F}_1(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n). \end{array} \tag{17}$$

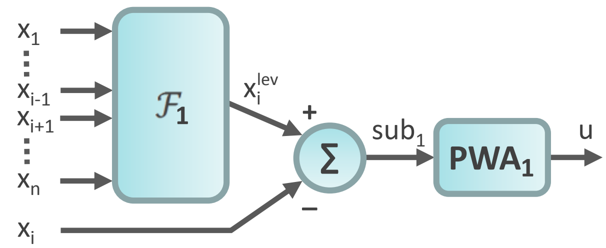


Fig. 7. First hierarchical decomposition of a system with n inputs.

A. Selection of the state variable to be separated

The procedure is summarized in Algorithm 1 and explained in the following.

Since it is supposed that the optimal control action can be computed for any state variables, the first step in Algorithm 1 is to consider (z_i+1) points in the universe of discourse $[x_{imin},x_{imax}]$ of each state variable, x_i, as follows:

$$x_i^{qi}=x_{imin}+\frac{x_{imax}-x_{imin}}{z_i}\cdot q_i \quad \text{with, } q_i=0,\ldots,z_i. \tag{18}$$

Then, a grid partition of the state variables is considered from the Cartesian product of the points at each variable, thus resulting $Z=(z_1+1)\times(z_2+1)\times\cdots\times(z_n+1)$ points. The optimal control action corresponding to each point in the grid is computed so as to generate a set $\mathbb{X}$ with Z points:

$$\begin{array}{l} \mathbb{X}=\{(x_1^{q1},\ldots,x_n^{qn},u_z), \quad \text{with}\\ q_1=0,\ldots,z_1;\ldots;q_n=0,\ldots,z_n;z=1,\ldots,Z\}. \end{array} \tag{19}$$

The second step in Algorithm 1 is to find $(L-1)$ subsets of points (level hypersurfaces), $\mathcal{L}_{qu}\subset\mathbb{X}$, with $q_u=1,\ldots,(L-1)$. Each level hypersurface contains N_{qu} points, denoted as $(x_{1lu},\ldots,x_{nlu},u_{qu})$ with $lu=1,\ldots,N_{qu}$, verifying that:

$$\begin{array}{l} \mathcal{L}_{qu}=\{(x_{1lu},\ldots,x_{nlu},u_{qu}), \quad \text{with}\\ |u_{qu}-C_{qu}|\le\epsilon_0 \quad \text{and}\\ C_{qu}=u_{min}+\frac{u_{max}-u_{min}}{L}\cdot q_u\}. \end{array} \tag{20}$$

As the value of ϵ_0 is smaller, the level hypersurfaces are more exact. The extreme values of the control action, u_{min} and u_{max}, are not considered since they are usually saturation values (Fig. 4b and Fig. 5 show examples of level curves).

The steps (3) and (4) in Algorithm 1 evaluate for each state variable, x_i, and each level hypersurface, $\mathcal{L}_{qu}$, if it is possible to meet Lemma 1, that is, to express the N_{qu} points as (according to Equation (17)):

$$\begin{array}{l} x_{ilu}=x_{ilu}^{lev}+\frac{g_{1j}-C_{qu}}{f_{1j}}=x_{ilu}^{lev}+b_{1ju},\\ \text{with } x_{ilu}^{lev}=\mathcal{F}_1(x_{1lu},\ldots,x_{(i-1)lu},x_{(i+1)lu},\ldots,x_{nlu})\\ \text{and } b_{1ju} \text{ a constant for the interval } I_1^{(j)}. \end{array} \tag{21}$$

The fourth step in Algorithm 1 checks that $\mathcal{F}_1(.)$ is a well-defined function, that is, it cannot associate different outputs to the same input. Hence, it checks that there are no equal points

$(x_{1lu},\ldots,x_{(i-1)lu},x_{(i+1)lu},\ldots,x_{nlu})$ in $\mathcal{L}_{qu}$ as a necessary condition to select x_i, because, otherwise, $\mathcal{F}_1(.)$ would be multivalued. Formally:

$$\begin{array}{l}\forall l_{uv}, l_{uw} \in \{1,\ldots,N_{qu}\},\\ (x_{1luv},\ldots,x_{(i-1)luv},x_{(i+1)luv},\ldots,x_{nluv}) \neq \\ (x_{1luw},\ldots,x_{(i-1)luw},x_{(i+1)luw},\ldots,x_{nluw}).\end{array} \tag{22}$$

For example, the coordinates x_{1lu} of the state variable x_1 in the points (x_{1lu},x_{2lu},u_{qu}) representing each of the 9 level curves in Fig. 4b are different because the state variable x_2 can be separated. On the other hand, several coordinates x_{2lu} of the state variable x_2 in the points (x_{1lu},x_{2lu},u_{qu}) are repeated, so that the state variable x_1 cannot be separated.

The fourth step also checks if all the level hypersurfaces can be represented by the same function displaced in parallel to state variable x_i (in the example of Fig. 4b, the 9 level curves can be seen as the same function in x_1 displaced in parallel to axis x_2). For this purpose, the level hypersurface, $\mathcal{L}_{min}$, with the minimum number of points, N_{min}, is selected to evaluate if the coordinates corresponding to the state variables x_j ($j = 1$ to $n; j \neq i$) in the points of $\mathcal{L}_{min}$ also appear in the points of any other level hypersurface, $\mathcal{L}_u$. Formally:

$$\begin{array}{l}\forall lr \in \{1,\ldots,N_{min}\}\exists ls^* \in \{1,\ldots,N_u\}/|x_{jlr}-x_{jls^*}| \leq \epsilon_1,\\ \text{with } j=1,\ldots,i-1,i+1,\ldots,n.\end{array} \tag{23}$$

The fourth step also checks if the differences of the N_{min} values of the coordinates corresponding to the state variables x_i in each pair of level hypersurfaces, $\mathcal{L}_{min}$ and $\mathcal{L}_u$, are equal or very similar, that is:

$$\begin{array}{l}\exists c \in \mathbb{R}/|(x_{ilr}-x_{ils^*})| = c + \epsilon_2\\ \forall lr \in \{1,\ldots,N_{min}\} \text{ and } ls^* \in \{1,\ldots,N_u\}.\end{array} \tag{24}$$

As the values of ϵ_1 and ϵ_2 are smaller, the approximation error of the PWAH approach is smaller.

If the above conditions are fulfilled by several state variables, x_i and others x_j, the level hypersurface $\mathcal{L}_{eq}$ corresponding to the control action in the equilibrium state, $u = 0$, is analyzed. This level hypersurface is analyzed in detail since it will be used to define the module $\mathcal{F}_1$, as commented in the following subsection. If the data of this hypersurface are $(x_{1leq},\ldots,x_{nleq},u_{leq})$ with $l_{eq} = 1,\ldots,N_{eq}$, the variable selected to be separated, x_i, is that whose percentage of universe of discourse is the least covered by that hypersurface. This is done at step (6) in Algorithm 1. Formally:

$$\begin{array}{l}\frac{|\max_{l_{eq}}\{x_{il_{eq}}\}-\min_{l_{eq}}\{x_{il_{eq}}\}|}{|x_{imax}-x_{imin}|} \leq \\ \frac{|\max_{l_{eq}}\{x_{jl_{eq}}\}-\min_{l_{eq}}\{x_{jl_{eq}}\}|}{|x_{jmax}-x_{jmin}|} \quad \forall j \neq i.\end{array} \tag{25}$$

For the example in Fig. 5a, if the above conditions would be fulfilled by state variables x_1 and x_2, the variable selected to be separated would be x_2 since its universe of discourse is the least covered in percentage by the hypersurface $\mathcal{L}_{eq}$.

If the above conditions are not fulfilled by any of the state variables, the analysis is repeated with $M-1$ discrete rotations ($\theta_r = \frac{90}{M}\cdot r$ with $r = 1,\ldots,M-1$) of two by two variables. This is done at steps (7) and (8) in Algorithm 1. The rotation matrix applied if x_a and x_b axis are rotated an angle θ_r using as rotation axis the $n-2$ dimensional hyperplane formed by the other variables, follows the formula of n-dimensional rotations proposed in [30]:

Algorithm 1 Selection of the state variable to be separated

Input = $u(x); z_1,\ldots,z_n; L; \epsilon_0; \epsilon_1; \epsilon_2; E = []$;

(1) Generate a set of points $\mathbb{X}$ as described in Equations (18)-(19);

(2) Find $(L-1)$ level hypersurfaces, $\mathcal{L}_{qu} \subset \mathbb{X}$, as described in Equation (20);

(3) Select an unexplored state variable. If no such variable exists, go to step (6).

(4) Check if Equation (22), (23) and (24) are verified by all the $\mathcal{L}_{qu}$. If not, mark the state variable as explored and go to step (3).

(5) Add the state variable to the list E, mark it as explored and go to step (3).

(6) If E is empty go to step (7). If it has one variable, select it as the variable to be separated and finish. If it has more than one, select the variable that meets Equation (25) and finish.

(7) Select an unexplored rotation. If no such rotation exists, finish.

(8) Replace the state variables by the rotated ones, mark the rotation as explored and go to step (3).

Output = The variable, if any, to be separated.

$$R_{ab}(\theta_r) = \left\{ r_{ij};\ \begin{array}{l} r_{ii} = 1; i \neq a, i \neq b\\ r_{aa} = \cos\theta_r\\ r_{bb} = \cos\theta_r\\ r_{ab} = -\sin\theta_r\\ r_{ba} = \sin\theta_r\\ r_{ij} = 0 \text{ otherwise}\end{array} \right\} \tag{26}$$
with $i,j = 1,\ldots,n$.

If after all the possible combinations of rotations none of the rotated x_{ir} inputs fulfill the conditions in Equations (22)-(24) (replacing the state variables x_i by the rotated ones, x_{ir}), then a decomposition as shown in Fig. 7 is not possible. Otherwise, the methodology continues with the design of the SISO PWA$_1$ module in Fig. 7 as follows.

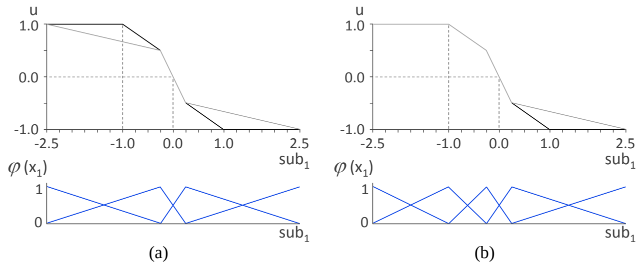


Fig. 8. Design example of a SISO PWA module to approximate the function of Fig. 3c (the target curve is depicted in black and the output of the module in grey): (a) with four kernel functions at the first step (two of them fixed by Lemma 3); (b) with five functions at the second step.

B. Design of the SISO PWA modules

Let us suppose that the state variable x_i has been selected to be separated, decomposing the system in the structure of Fig. 7. Hence, the points of the level hypersurfaces can be expressed as in Equation (21). In the case of the level hypersurface $\mathcal{L}_{eq}$ corresponding to the control action in the equilibrium state, $u = 0$, let us assume for simplicity that $g_{1eq} = 0$. Hence, this level hypersurface is used to define the module $\mathcal{F}_1$, since it provides:

$$x_{ileq} = x_{ileq}^{lev} = \mathcal{F}_1(x_{1leq}, \ldots, x_{(i-1)leq}, x_{(i+1)leq}, \ldots, x_{nleq}), \text{ with } \quad leq = 1, \ldots, N_{eq}. \tag{27}$$

This is equivalent to assume that if $sub_1 = x_{ileq}^{lev} - x_{ileq} = 0 \in I_1^{(eq)}$, then $u = f_{\text{PWA}_1}(sub_1) = f_{\text{PWA}_1}(0) = 0$. In addition, this is equivalent to assume that $\mathcal{F}_1(0) = 0$, because the state variables and control action should be zero in the equilibrium.

Since the level hypersurfaces can be represented by the same function, $\mathcal{F}_1$, displaced in parallel to x_i, the points in the level hypersurfaces $\mathcal{L}_{qu}$ verify that (according to Equations (21), (23), and (27)):

$$x_{ils*} = x_{ils*}^{lev} + b_{1ju} = x_{ileq*} + b_{1ju}. \tag{28}$$

Therefore, if $sub_1 = (x_{ileq*} - x_{ils*}) \in I_1^{(j)}$, then $u = f_{\text{PWA}_1}(-b_{1ju}) = C_{qu}$. Hence, the module PWA_1 in Fig. 7 is adjusted to minimize the sum of squared errors obtained when considering the points $sub_1 = (x_{ileq*} - x_{ils*})$ as inputs and C_{qu} as desired output. For that objective, the univariate PWA function provided by the module PWA_1 is obtained by the linear combination of triangular kernel functions, $\varphi(sub_1)$, that cover the input universe of discourse as a partition of unity (that is, the sum of all the kernel functions evaluated at every input value is always the unity), as shown in Fig. 8:

$$f_{\text{PWA}_1}(sub_1) = \sum_{i=1}^{P} w_n \cdot \varphi_n(sub_1). \tag{29}$$

The algorithm employed to adjust the number, P, and location of the triangular kernel functions is based on the proposal in [31]. In the first step, if the conditions stated in Lemma 3 are not imposed, only two triangles are considered, with their centers at the extreme points of the input universe of discourse, each of them weighted by the desired output at the two extreme points of the input universe of discourse. However, if the conditions stated in Lemma 3 are imposed, the piece associated to the origin and around ($x_i = 0$, $\mathcal{F}_1(0) = 0$, and, hence, $sub_1 = 0$) should be approximated without error. Hence, two kernels with their two weights are fixed to meet local optimality, as shown in the center of Fig. 8a. Besides, kernel functions should be located at the extreme points of the input universe of discourse, weighted by the desired output at those extreme points, as shown on the left and right of Fig. 8a. In the second step, a new kernel function is located with its vertex at the input value sub_1 at which the absolute error between the output f_{PWA_1} and the target output provided by the points in the level hypersurfaces is the highest, as shown in Fig. 8b. The weight of the new kernel function is the value of the target output provided by the data. More steps are done successively until the approximation error is smaller than a given goal error or the number of kernel functions is over a given limit. In the end, the weights, w_n, of the kernel functions that are not fixed by Lemma 3 are adjusted to minimize the sum of squared errors by using Levenberg-Marquardt algorithm [32][33].

Once the system in Fig. 7 is completely designed, the methodology applies again the steps described in Algorithm 1 to now explore if the block $\mathcal{F}_1$ can also be decomposed similarly, that is, if one of its inputs can be separated and another block $\mathcal{F}_2$ and SISO module PWA_2 can be introduced. The set of points in step (1) of Algorithm 1, now referred to as $\mathbb{X}_1$, are:

$$\mathbb{X}_1 = (x_{1leq}, \ldots, x_{(i-1)leq}, x_{(i+1)leq}, \ldots, x_{nleq}, x_{ileq}), \text{ with } \quad leq = 1, \ldots, N_{eq}. \tag{30}$$

The new level hypersurfaces in step (2) of Algorithm 1 now correspond to constant values of the state variable x_i. The level hypersurface associated to a zero value of x_i will be used to define the block F_2 and to evaluate further decompositions (assuming $g_{2eq} = 0$).

This is done iteratively until the PWAH implementation in Fig. 2 is found, if possible. If it is found, the first SISO PWA module (PWA_n in Fig. 2) is also adjusted to select the adequate number and location of kernel functions and their weights. In any case, the final step is to apply again Levenberg-Marquardt algorithm to readjust the weights of the kernel functions of all the SISO modules found by the methodology (not fixed by Lemma 3) so that the final implementation reduces further the sum of squared errors.

VI. Application Examples

In order to demonstrate the effectiveness of the PWAH approach, this section compares it with other PWA solutions presented in the literature. The section begins with a benchmark problem of control, such as the control of the double integrator. Then, a case of a highly unstable multi-input system is studied. Finally, a problem of recent commercial interest is faced, such as a car adaptive cruise control system.

The optimal explicit MPC controllers used as reference to be approximated by PWAH systems, as well as their simulation results, were obtained thanks to MOBY-DIC Toolbox [34], Hybrid Toolbox [35] and Multi-Parametric Toolbox [36], all for Matlab-Simulink. The design and adjustment of the SISO PWA modules at each hierarchical decomposition step as well as the final re-adjustment of the global solution were performed with the description and tuning tools of Xfuzzy environment [37]. Xfuzzy not only has CAD tools to describe and adjust piecewise-polynomial systems but also synthesis tools for hardware implementations, in particular, it has a tool which automates the communication with Xilinx System Generator, which is based on Matlab-Simulink [38]-[39].

The PWAH solutions obtained were designed using the Xilinx System Generator tool and implemented in Xilinx FPGAs. Although lower-cost FPGAs could have been chosen, Spartan 3 FPGAs were selected in all the examples in order to compare the implementation results of the PWAH controllers with other PWA approaches reported in the literature which were implemented in Spartan 3 FPGAs. The synthesis and implementation of the controllers were carried out with the Xflow tool from Xilinx ISE 14.7. The descriptions make use of registers for the required constants and *Convert* blocks to saturate the universe of discourse of the signals. In order to implement the PWAH controllers with fixed-point arithmetic, the values of state variables and the control action provided were transformed to the interval [0,1].

To verify the behavior of the hardware implementation in the FPGA, hardware-in-the-loop (HIL) co-simulations with models of the plants in Matlab-Simulink were carried out.

A. Double Integrator

Let us consider the problem of regulating to the origin the double integrator system:

$$\ddot{y} = u(x). \tag{31}$$

Its equivalent discrete-time state-space representation, with a sampling time $T_s = 1s$, is given by the expression:

$$x(t+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t). \tag{32}$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t). \tag{33}$$

Fulfilling the restriction:

$$u_{min} = -1 \leq u(t) \leq 1 = u_{max}. \tag{34}$$

The optimal explicit PWA controller is obtained by the MOBY-DIC Toolbox as the first element of U from (2) with the following horizons (N_y, N_u) and weight matrices (Q, R):

$$N_y = N_u = 4; \;\; Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}; \;\; R = 0.1. \tag{35}$$

and the state domain is given by $D = [-8, 8] \times [-4, 4]$. Fig 1a illustrates the optimal polyhedral partition.

According to the methodology described in the previous section, the level curves of the control action were found in the set $\mathbb{X} = \{y^{q1}, \dot{y}^{q2}, u_z\}$ generated to be approximated by the PWAH solution. In this case, 9 level curves were analyzed, which are shown in Fig. 5a.

Condition of Equation (22) was fulfilled by both state variables $\{y, \dot{y}\}$, but Equations (23) and (24), using relatively small values of ϵ_1 and ϵ_2, were only fulfilled by separating the state variable $\dot{y}$. If the values of ϵ_1 and ϵ_2 are chosen smaller to reduce the approximation error, then the methodology finds that the best inputs are $\{y_r, \dot{y}_r\}$, obtained by a rotation of the $\{y, \dot{y}\}$ plane an angle $\theta_r = 30°$ and selects $\dot{y}_r$ as the input to be separated. The results in Table II show the approximation errors obtained with several solutions. Since the PWAH solution without applying rotation and using one unsaturated polytope (first row in Table II) provided a competitive approximation error, it was selected for simplicity. Therefore, the double integrator PWAH controller has the structure shown in Fig. 9a, with a module PWA_2 that approximates the nonlinear behavior of $\dot{y}$ as a function of y in the level curves (Fig. 9b) and a module PWA_1 that approximates how that function is displaced in parallel to $\dot{y}$ (Fig. 9c).

As can be seen in Fig. 9b, the module PWA_2 applies a linear transformation to the input y, so that it can be performed by the signal conditioning (which has to transform the input values to the range [0,1]). In addition, the module PWA_1 applies also a linear transformation to its input, $\dot{y}^{lev} - \dot{y}$, which is saturated for values that go beyond the interval [0,1]. This is achieved simply by using the output signal as an unsigned type signal with N bits and the decimal dot in the N position, applying saturation outside that range. Hence, no multipliers are required.

The hardware implementation in a Xilinx Spartan 3 (XC3S200) FPGA, with 12 bit of precision for the inputs and outputs, consumes 17 slices (approximately 0.9% of the available slices on the FPGA). The implementation is completely combinational so that a delay block was added to provide the output synchronously at each clock cycle. Therefore, the controller is able to operate at a maximum frequency of 759.9MHz.

Table IV compares the implementation results with other PWA controllers reported in the literature. The PWAG(FPGA) [13], PWAS [25], PWAR [27] and PWAL [17] controllers are implemented in a Xilinx Spartan 3 (XC3S200) FPGA (like the PWAH controller described herein), while the PWAG(ASIC) controller [12] is implemented in a 90-nm technology ASIC. All controllers use a 12-bit precision and the latency values correspond to a 20-MHz frequency. The error (MRE) is calculated as the maximum absolute difference between the optimal MPC control output and the control output in hardware.

As can be seen in Table IV, the PWAH controller occupies the least percentage of slices and does not employ any multiplier or memory resources available in the FPGA. In addition, it is the controller with the lowest latency and, hence,

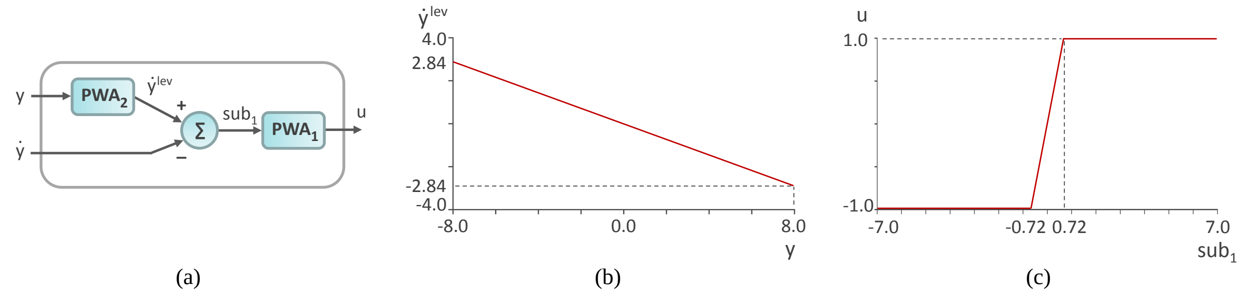


Fig. 9. (a) PWAH structure for the double integrator; (b) Behavior of the module PWA_2; (c) Behavior of the module PWA_1.

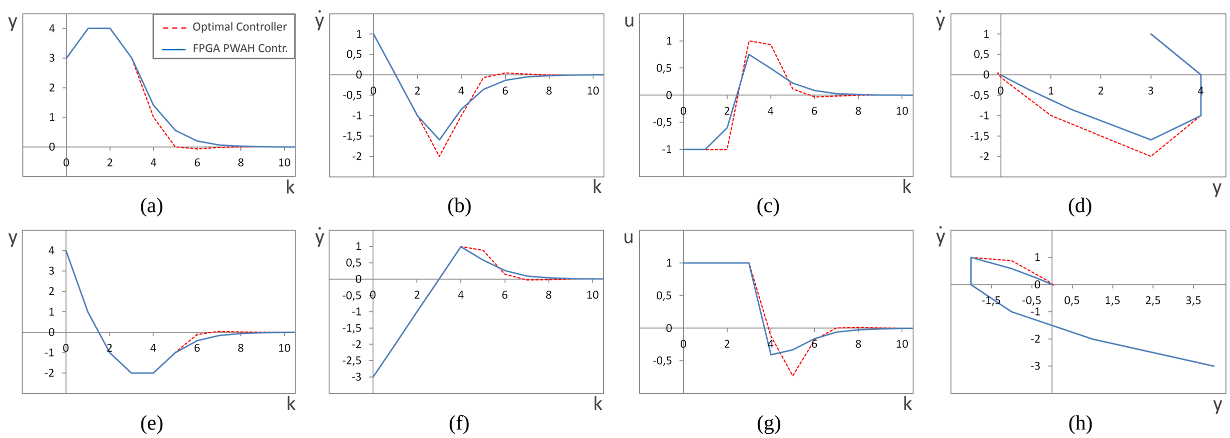


Fig. 10. Comparison of the optimal controller (simulation) with the PWAH controller implemented in a FPGA (HIL) for the double integrator: (a) and (e) first state variable, (b) and (f) second state variable, (c) and (g) control action, and (d) and (h) state space evolutions.

TABLE IV
COMPARISON OF PWA CONTROLLERS FOR THE DOUBLE INTEGRATOR

Controller	Slices (%)	Latency (μs)	Clock cycles	Memory (KB)	Multipliers	MRE
PWAG(FPGA)	6	1.6	32	0.41	1	0.0013
PWAG(ASIC)	-	1.4	28	0.99	2	0.016
PWAL	5	0.6	12	0.03	4	0.00077
PWAS(serial)	7	0.3	6	0.38	1	0.45
PWAS(parallel)	7	0.15	3	1.13	3	0.45
PWAR(serial)	2	0.2	4	1.13	1	0.29
PWAR(parallel)	1	0.1	2	1.13	2	0.29
PWAH	0.9	0.05	1	0	0	0.25

the fastest one. Since PWAG and PWAL approaches do not approximate the optimal MPC, their MRE values are very small. Among PWAS, PWAR and PWAH approaches, which implement sub-optimal MPC controllers, the PWAH controller achieves the smallest error.

Performance of the PWAH controller in the FPGA was evaluated by co-simulating it working in a closed loop (HIL) with a model of the plant described in Matlab-Simulink. Fig. 10 shows two examples with two different initial plant states (Fig. 10a-d the first one, and Fig. 10e-h the second one). It can be seen that the PWAH controller is able to stabilize the plant in a way that approximates the optimal MPC controller. Since conditions of Lemma 3 were not imposed in the design of this PWAH controller, the evolution of the state variables and control action to the origin are slightly different, but both controllers stabilize the plant practically at the same time, being the PWAH controller slightly softer. Simulation results of the PWAH solution that applies rotation and uses 8 unsaturated polytopes (fourth row in Table II) are quite similar to the results of the optimal MPC controller.

B. Multi-input System

The next application considers the problem of regulating to the origin a highly unstable multi-input system whose equivalent discrete-time state-space representation, with a sampling time $T_s = 1s$, is given by the expression:

$$x(t+1) = \begin{bmatrix} 1.3 & 1 \\ 0 & 1.1 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} u(t). \quad (36)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t). \quad (37)$$

Fulfilling the restriction:

$$u_{min} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq u(t) \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} = u_{max}. \quad (38)$$

The optimal explicit PWA controller is obtained by the MOBY-DIC Toolbox as the first element of U from (2) with the following horizons (N_y, N_u) and weight matrices (Q, R):

$$N_y = N_u = 4;\ \ Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix};\ \ R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}. \quad (39)$$

The multi-input system is decomposed into two MISO systems, MISO_1 and MISO_2, which are approximated separately. Hence, two sets of points, $\mathbb{X}_1 = \{x_1^{q1}, x_2^{q2}, u_{1z}\}$ and $\mathbb{X}_2 = \{x_1^{q1}, x_2^{q2}, u_{2z}\}$, are generated from a grid partition of the state domain given by $D = [-5, 5]^2$.

In the case of MISO_1, 9 level curves were analyzed. Equation (22) was not fulfilled without rotation. Using small values of ϵ_1 and ϵ_2, Equations (23) and (24), were only fulfilled by separating the variable x_{2r} of the new state variables $\{x_{1r}, x_{2r}\}$ obtained by a rotation of the $\{x_1, x_2\}$ plane an angle $\theta_r = 44°$. The PWAH structure designed, which is illustrated in Fig. 11, provides an RMSE of 1.2%.

In the case of MISO_2, 9 level curves were also analyzed. The methodology found that Equations (23) and (24) were only verified if the variable x_2 is separated. The PWAH structure obtained, which is illustrated in Fig. 12, provides an RMSE of 1.7%.

The SISO modules PWA_{11} and PWA_{12} were implemented as the PWA_1 module of the double integrator. The point location problem for the SISO modules PWA_{21} and PWA_{22} was solved in parallel using combinatorial logic. Although these modules have very few pieces, they were further reduced exploiting the symmetry of odd functions (i. e., $\text{PWA}(-x) = -\text{PWA}(x)$), so that only 2 and 3 pieces were required by PWA_{21} and PWA_{22} modules, respectively.

The resulting implementation contains 14 adders, 6 2-to-1 multiplexers, 2 4-to-1 multiplexers, 5 comparators and 7 multipliers (4 multipliers that carry out the rotation of the state variables and 3 multipliers for the SISO modules). Using a Xilinx Spartan 3 (XC3S200) FPGA, with 12 bits of precision for inputs and outputs, the PWAH controller consumes 215 slices (approximately 11% of the available slices on the FPGA) and uses 7 of the 12 multipliers available in the FPGA. The controller is able to operate at a maximum frequency of 47MHz and has a latency of two clock cycles (the MISO_1 employs one clock cycle in the rotation and another in the nonlinear transformation of the modules PWA_{21} and PWA_{11}, and the MISO_2 is synchronized with it).

TABLE V

COMPARISON OF PWA CONTROLLERS FOR THE MULTI-INPUT SYSTEM

Controller	Slices (%)	Latency (μs)	Clock cycles	Mem. (KB)	Mult.
PWAG	12	0.36	38	0.27	1
PWAS(serial)	11	0.27	19	0.37	1
PWAS(parallel)	35	0.04	1	1.13	3
PWAH	11	0.04	2	0	7

Table V allows the comparison of several PWA implementations that control this multi-input system. The PWAG, PWAS(serial) and PWAS (parallel) approaches are implemented in a Xilinx FPGA Spartan 3 (XC3S200) [40], like the PWAH controller developed in this section. The latency values are provided for maximum frequency of each implementation. It can be seen that the PWAH controller is as fast as the fastest, but occupying a percentage of slices as the least and without using memory resources.

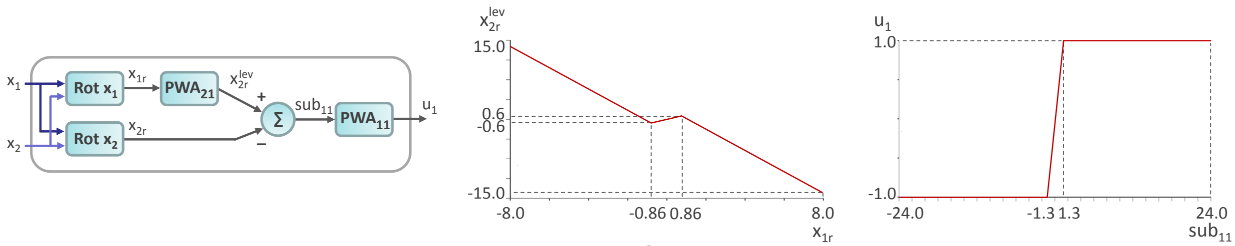
The PWAH controller implemented in the FPGA was co-simulated in a closed loop with a model of the multi-input system described in Matlab-Simulink. Two simulation examples with two different initial plant states are shown in Fig. 13 (Fig. 13a-d the first one, and Fig. 13e-h the second one). The degree of similarity between the behavior of the PWAH controller and the optimal controller is so high that it is hard to distinguish one from the other (conditions of Lemma 3 were imposed in the design of this PWAH controller).

C. Adaptive Cruise Control (ACC)

An ACC system is the evolution of the standard cruise control (CC) system used in most of the cars today. The CC system is responsible for maintaining the vehicle at the steady speed set by the driver by controlling only the throttle. In addition, an ACC is able to adapt the speed of the car (*host vehicle*) to the speed of another car located ahead (*target vehicle*), controlling both the throttle and the brake.

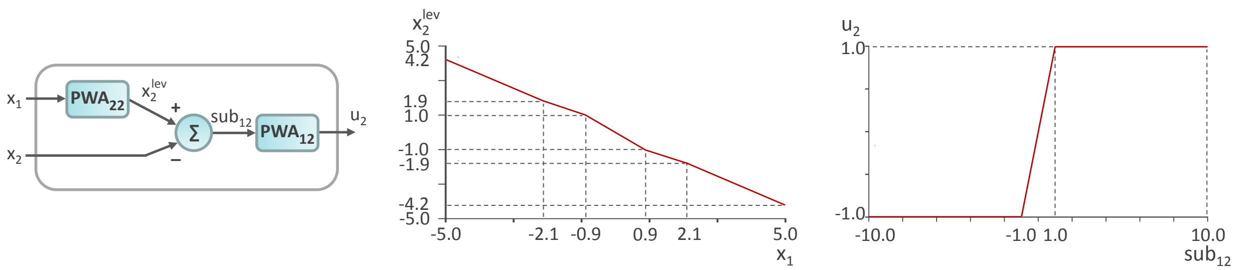
In this section, the ACC to be implemented is described in [41]. In this model, the speed of the *host vehicle* (v_h) and its acceleration (a_h) are available, while the relative distance between the two vehicles (x_r) and the relative velocity ($v_r = v_t - v_h$) are measured by a radar located at the *host vehicle*. The goal is to keep the *host vehicle* to a desired distance x_r from the *target vehicle*. To define this distance, it is often used the desired headway time ($t_{hw,d}$), so that $x_{r,d} = x_{r,0} + v_h t_{hw,d}$, where $x_{r,0}$ is a constant that represents the desired distance at standstill. Therefore, the tracking error will be defined by $e = x_{r,d} - x_r$.

The model described in [41], along with the considerations taken in [42], is presented in the form:



(a) (b) (c)

Fig. 11. (a) PWAH structure for the MISO$_1$; (b) Behavior of the module PWA$_{21}$; (c) Behavior of the module PWA$_{11}$.



(a) (b) (c)

Fig. 12. (a) PWAH structure for the MISO$_2$; (b) Behavior of the module PWA$_{22}$; (c) Behavior of the module PWA$_{12}$.

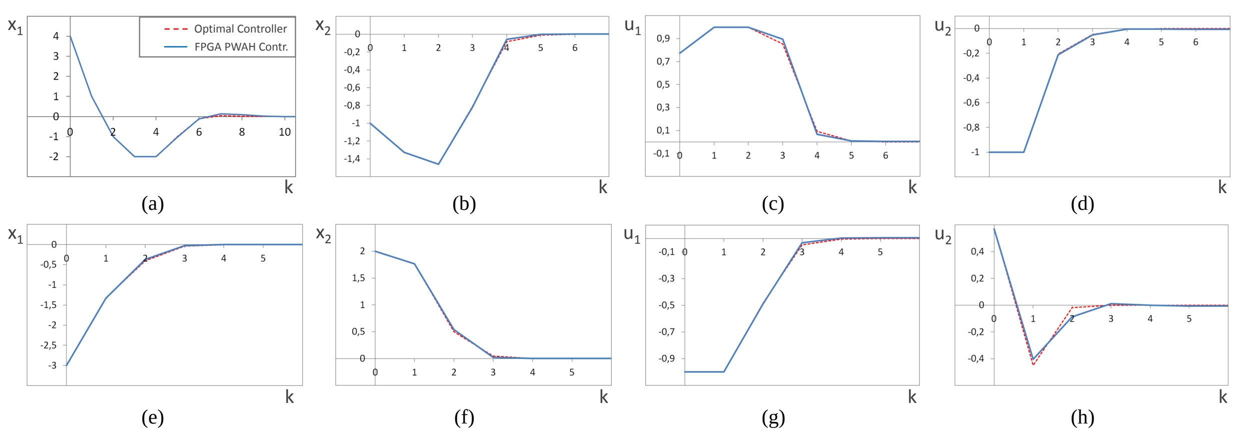


Fig. 13. Comparison of the optimal controller (simulation) with the PWAH controller implemented in a FPGA (HIL) for the multi-input system: (a) and (e) first state variable, (b) and (f) second state variable, (c) and (g) first control action, and (d) and (h) second control action evolutions.

$$x(t+1) = \begin{bmatrix} 1 & -T_s & 0 & Z_s \\ 0 & 1 & 0 & T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(t). \quad (40)$$

Where $Z_s = T_s t_{hw,d} + \frac{1}{2}T_s^2$ and the state variable $x = [e \;\; v_r \;\; v_t \;\; a_h]^T$. The sampling time is $T_s = 0.1s$.

The values of the constants mentioned above are $x_{r,0} = 3.5m$, $t_{hw,d} = 1.5s$, $v_{t,max} = 50m/s$, $v_{h,max} = 50m/s$, $a_{h,min} = -3m/s^2$, $a_{h,max} = 2m/s^2$. The constraints of the host jerk (derivative of the acceleration) are $j_{h,min} = -0.3m/s^3$ and $j_{h,max} = 0.3m/s^3$, and the radar range is $x_{rr} = 200m$.

The optimal explicit PWA controller is obtained as the first element of U from (2) with the following horizons (N_y, N_u) and weight matrices (Q, R):

$$N_y = N_u = 4; \quad Q = \begin{bmatrix} 2.5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad R = 1. \quad (41)$$

The PWAH controller to design has three inputs because the variable v_t is considered to be constant by the model in Equation (40). The set of points $\mathbb{X} = \{e^{q1}, v_r^{q2}, a_h^{q3}, u_z\}$ to be approximated are obtained from a grid partition of the state domain given by $D = [-196.5, 78.5] \times [-50, 50] \times [-3, 2]$.

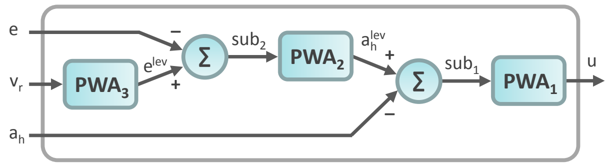


Fig. 14. PWAH structure for the ACC.

TABLE VI
COMPARISON OF PWA CONTROLLERS FOR THE ACC

Controller	Slices (%)	Latency (μs)	Clock cycles	Mem. (KB)	Mult.
PWAG	87	5.4	108	3.3	1
PWAS(serial)	31	0.4	8	11.5	1
PWAS(parallel)	95	0.15	3	57.6	5
PWAL	5	1.65	33	1.06	8
PWAH	0.8	0.05	1	0	1

Since $u_{min} = -0.3 \leq u \leq 0.3 = u_{max}$, 5 level hypersurfaces were analyzed (from $u = -0.2$ to $u = 0.2$). The methodology first selected the variable a_h to be separated, obtaining a first hierarchical decomposition as in Fig. 7. The next iteration, which analyzed if the module $\mathcal{F}_1$ could be decomposed, found that Equations (22) to (24) were verified by separating the state variables e and v_r, but e was selected according to condition (25).

The PWAH structure designed, which is shown in Fig. 14, provides a good trade-off between high simplicity and small approximation error (the RMSE is 0.31%). As in the case of the double integrator, the SISO module PWA_3 (Fig. 15a) provides a linear function that can be implemented by the signal conditioning circuitry, so that it is eliminated of the controller. The resulting implementation contains two adders and one multiplier.

Using a Xilinx Spartan-3AN (XC3S700AN) FPGA, with 16 bits of precision for inputs and outputs, the PWAH controller consumes 47 slices (approximately 0.8% of the available slices on the FPGA) and uses only 1 of the 20 multipliers available in the FPGA. The controller has a latency of one clock cycle and is able to operate at a maximum frequency of 753MHz.

Table VI compares the implementation results of several PWA controllers. All the controllers (PWAG and PWAS in [42], and PWAL in [17]) are implemented in a Xilinx FPGA Spartan 3AN (XC3S700AN), like the PWAH controller developed in this section. The latency values are provided for a frequency of 20MHz. It can be seen that the PWAH controller is the fastest, occupies the least percentage of slices, and does not require any memory.

The hardware implementation of the controller in the FPGA was co-simulated with a software description of the ACC plant in Matlab-Simulink. Two of these simulations with two different initial plant states are shown in Fig. 16 (Fig. 16a-d the first one, and Fig. 16e-h the second one). The control achieved by the PWAH system is so similar to the optimal control that it is difficult to differentiate one from the other.

VII. CONCLUSIONS

The PWAH controllers presented in this paper are very simple to implement since they only use SISO PWA modules connected in cascade with addition/subtraction operations between them. A methodology has been described to design them starting from the optimal explicit PWA controllers. The methodology results have been tested successfully in many model predictive control problems. Although not all the optimal explicit PWA controllers are able to be well approximated by the PWAH form (the methodology finds which are able and not), there are many controllers that allow this type of decomposition.

Many examples even allow that the SISO PWA modules provide linear functions with or without saturation, thus resulting efficient controllers whose hardware implementation is much simpler than simplifications achieved by other techniques such as the PWAS and PWAR approaches. This has been proven with results from FPGA implementations and hardware-in-the-loop simulations. For a benchmark problem of control, such as the control of the double integrator, a multi-input system, and a problem of recent commercial interest, such as a car adaptive cruise control system, the PWAH controllers designed were the fastest with the lowest cost in resources, compared to other PWA controllers reported in literature, achieving a very similar performance to the optimal explicit MPC controller.

ACKNOWLEDGMENT

The authors would like to thank M.C. Martínez-Rodríguez for helpful discussions on the optimal controllers design.

REFERENCES

[1] J.M. Maciejowski, *Predictive control with constraints*, Prentice Hall, Englewood Cliffs, NJ, USA, 2002.
[2] D.Q. Mayne, J.B. Rawlings, C.V. Rao and P.O.M. Scokaert (2000), “Constrained model predictive control: Stability and optimality”, *Automatica 36(6)*, pp. 789-814.
[3] Y. Wang and S. Boyd (2010), “Fast model predictive control using online optimization”, *IEEE Trans. Control Syst. Technol. 18(2)*, pp. 267-278.
[4] A. Bemporad, M. Morari, V. Dua and E.N. Pistikopoulos (2002), “The explicit linear quadratic regulator for constrained systems”, *Automatica 38(1)*, pp. 3-20.
[5] A. Bemporad., F. Borrelli and M. Morari (2002), “Model predictive control based on linear programming - The explicit solution”, *IEEE Trans. Autom. Control 47(12)*, pp. 1974-198.
[6] A. Bemporad (2015), “A multiparametric quadratic programming algorithm with polyhedral computations based on nonegative least squares”, *IEEE Trans. Autom. Control 60(11)*, pp. 2892-2903.
[7] F. Bayat., T.A. Johansen and A.A. Jalali (2011), “Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control”, *Automatica 47(3)*, pp. 571-577.
[8] F. Bayat., T.A. Johansen and A.A. Jalali (2012), “Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC”, *IEEE Trans. Control Syst. Technol. 20(3)*, pp. 632-640.
[9] A. Oliveri, C. Gianoglio, E. Ragusa and M. Storace (2015), “Low-complexity digital architecture for solving the point location problem in explicit model predictive control”, *J. Franklin Inst. 352(6)*, pp. 2249-2258.

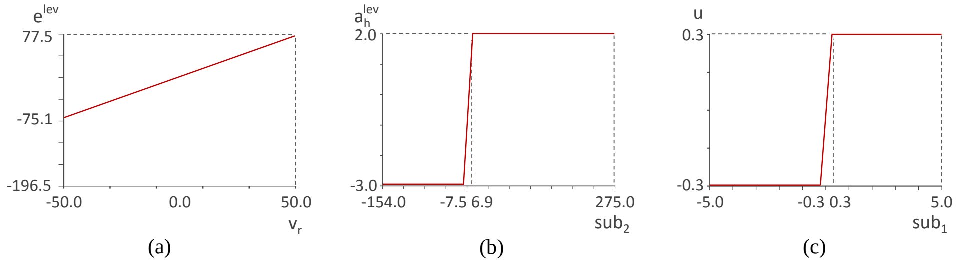


Fig. 15. Behavior of the modules of Fig. 14: (a) PWA_3; (b) PWA_2; (c) PWA_1.

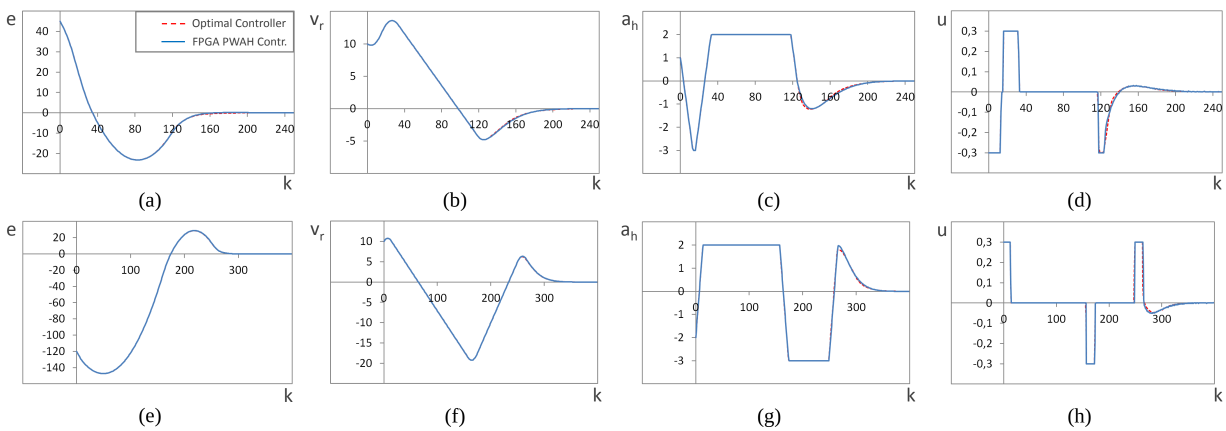


Fig. 16. Comparison of the optimal controller (simulation) with the PWAH controller implemented in a FPGA (HIL) for the ACC: (a) and (e) first state variable, (b) and (f) second state variable, (c) and (g) third state variable, and (d) and (h) control action evolutions.

[10] D. Ingole, J. Holaza, B. Takács and M. Kvasnica, "FPGA-based explicit model predictive control for closed-loop control of intravenous anesthesia", in *Proc. 20th Int. Conf. on Process Control (PC'15)*, 2015, pp. 42-47.

[11] T.A. Johansen, W. Jackson, R. Schreiber, P. Tøndel (2007), "Hardware synthesis of explicit model predictive controllers", *IEEE Trans. Control Syst. Technol. 15(1)*, pp. 191-197.

[12] P. Brox, J. Castro-Ramírez, M.C. Martínez-Rodríguez, E. Tena, C.J. Jiménez, I. Baturone and, A.J. Acosta (2013), "A programable and configurable ASIC to generate Piecewise-Affine functions defined over general partitions", *IEEE Trans. Circuits Syst. I: Reg. Papers 60(12)*, pp. 3182-3194.

[13] A. Oliveri, A. Oliveri, T. Poggi and M. Storace, "Circuit implementation of piecewise-affine functions base on a binary search tree", in *Proc. Eur. Conf. Circuit Theory and Design (ECCTD'09)*, 2009, pp. 145-148.

[14] F.J. Christophersen, M. Kvasnica, C.N. Jones and M. Morari, "Efficient evaluation of piecewise control laws defined over a large number of polyhedra", in *Proc. 2007 Eur. Control Conf. (ECC'07)*, 2007, pp. 2360-2367.

[15] J.M. Tarela and M.V. Martínez (1999), "Region configurations for realizability of lattice piecewise-linear models", *Mathematical and Computer Modelling 30(11-12)*, pp. 17-27.

[16] C. Wen, X. Ma and B.E. Ydstie (2009), "Analytical expression of explicit MPC solution via lattice piecewise-affine function", *Automatica 45(4)*, pp. 910-917.

[17] M.C. Martínez-Rodríguez, P. Brox and I. Baturone (2015), "Digital VLSI implementation of piecewise-affine controllers based on lattice approach", *IEEE Trans. Control Syst. Technol. 23(3)*, pp. 842-854.

[18] M. Kvasnica, J. Hledík, I. Rauová and M. Fikar (2013), "Complexity reduction of explicit model predictive control via separation", *Automatica 49(6)*, pp. 1776-1781.

[19] M. Mönnigmann and M. Kastsian, "Fast explicit MPC with multiway trees", in *Proc. 18th IFAC World Congr. (IFAC'11)*, 2011, pp. 1356-1361.

[20] A. Bemporad and C. Filippi (2003), "Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming", *J. Opt. Theory Appl. 117(1)*, pp. 9-38.

[21] J. Holaza, B. Takács, M. Kvasnica and S. Di Cairano (2015), "Nearly optimal simple explicit MPC controllers with stability and feasibility guarantees", *Opt. Control Appl. Meth. 36*, pp. 667-684.

[22] P. Julian, A. Desages and O. Agamennoni (1999), "High-level canonical piecewise linear representation using a simplicial partition", *IEEE Trans. Circuits Syst. I: Fund. Theory Appl. 46(4)*, pp. 463-480.

[23] R. Rovatti, C. Fantuzzi and S. Simani (2000), "High-speed DSP-based implementation of piecewise-affine and piecewise-quadratic fuzzy systems", *Signal Process. 80(6)*, pp. 951-963.

[24] A. Bemporad, A. Oliveri, T. Poggi and M. Storace (2011), "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations", *IEEE Trans. Autom. Control 56(12)*, pp. 2883-2897.

[25] M. Storace and T. Poggi (2011), "Digital architectures realizing piecewise-linear multivariate functions: Two FPGA implementations", *Int. J. Circuit Theory and Appl. 39(1)*, pp. 1-15.

[26] T.A. Johansen and A. Grancharova (2003), "Approximate explicit constrained linear model predictive control via orthogonal search tree", *IEEE Trans. Autom. Control 48(5)*, pp. 810-815.

[27] F. Comaschi, B.A.G. Genuit, A. Oliveri, W.P.M.H. Heemels and M. Storace (2012), "FPGA implementations of piecewise affine functions based on multi-resolution hyperrectangular partitions", *IEEE Trans. Circuits Syst. I: Reg. Papers 59(12)*, pp. 2920-2933.

[28] I. Baturone, M.C. Martínez-Rodríguez, P. Brox, A. Gersnoviez and S. Sánchez-Solano,"Digital implementation of hierarchical piecewise-affine controllers", in *Proc. 2011 IEEE Int. Symp. on Industrial Electronics (ISIE'11)*, 2011, pp. 1497-1502.

[29] D. Buchstaller, E.C. Kerrigan and G.A. Constantinides (2012),"Sampling and controlling faster than the computational delay", *IET Control Theory & Applications 6(8)*, pp. 1071-1079.

[30] K. Duffin and W. Barret, "Spiders: A new user interface for rotation and visualization of n-dimensional point sets", in *Proc. 1994 IEEE Conf. on Visualization (Visualization'94)*, 1994, pp. 205-211.

[31] C.M. Higgins and R.M. Goodman (1994), "Fuzzy rule based networks for control", *IEEE Trans. Fuzzy Syst. 2(1)*, pp. 82-88.

[32] R. Battiti (1992), "First- and second-order methods for learning: between steepest descent and Newton's method", *Neural Comput. 4(2)*, 141-166.

[33] R. Fletcher, *Practical methods of optimization*, John Wiley & Sons, Ltd., 2013.

[34] A. Oliveri *et al.*, "MOBY-DIC: A Matlab toolbox for circuit-oriented design of explicit MPC", *IFAC Proc. Volumes*, vol. 45, no. 17, pp. 218-225, 2012.

[35] Hybrid Toolbox. Accessed: Nov. 13, 2017. [Online]. Available at: http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox

[36] Multi-Parametric Toolbox. Accessed: Nov. 13, 2017. [Online]. Available at: http://people.ee.ethz.ch/~mpt/3

[37] F.J. Moreno-Velo, I. Baturone, A. Barriga and S. Sánchez-Solano (2007), "Automatic tuning of complex fuzzy systems with Xfuzzy", *Fuzzy Sets Syst. 158(18)*, pp. 2026-2038.

[38] I. Baturone, S. Sánchez-Solano, A. Gersnoviez and M. Brox, "An automated design flow from linguistic models to piecewise polynomial digital circuits", in *Proc. 2010 IEEE Int. Symp. on Circuits and Systems (ISCAS'10)*, 2010, pp. 3317-3320.

[39] M. Brox, S. Sánchez-Solano, E. del Toro, P. Brox and F.J. Moreno-Velo (2013), "CAD Tools for Hardware Implementation of Embedded Fuzzy Systems on FPGAs", *IEEE Trans. Ind. Informat. 9(3)*, pp. 1635-1644.

[40] T. Poggi, I. Baturone and M. Storace, "Selection of architectures for PWA functions implementation", document MOBY-DIC Project FP7-INFSOICT-248858, 7th Framework Programme, European Community, 2010.

[41] G.J.L. Naus, J. Ploeg, M.J.G. Van de Molengraft, W.P.M.H. Heemels and M. Steinbuch (2010), "Design and implementation of parameterized adaptive cruise control: An explicit model predictive control approach", *Control Eng. Pract. 18(8)*, pp. 882-892.

[42] A. Oliveri, G.J.L. Naus, M. Storace and W.P.M.H. Heemels, "Low-complexity approximations of PWA functions: A case study on adaptive cruise control", in *Proc. Eur. Conf. Circuit Theory and Design (ECCTD'11)*, 2011, pp. 669-672.