Explicit MPC: hard constraint satisfaction under low precision arithmetic

Andrea Suardi[1], Stefano Longo[2], Eric C. Kerrigan[3], George A. Constantinides[4]

Abstract

MPC is becoming increasingly implemented on embedded systems, where low precision computation is preferred either to reduce costs, speedup execution or reduce power consumption. However, in a low precision implementation, constraint satisfaction cannot be guaranteed. To enforce constraint satisfaction under numerical errors, we adopt tools from forward error analysis to compute an error bound on the output of the embedded controller. We treat this error as a state disturbance and use it to inform the design of a constraint-tightening robust controller. The technique is validated via a practical implementation on an FPGA evaluation board.

Keywords: Embedded systems, Predictive control, Robust control

1. Introduction

Since the widespread use of single- and double-precision floating-point arithmetic in computer architectures, control system designers routinely start with the assumption that computation is performed with infinite numerical precision. The consequence is that the two activities of control system design and its implementation are often decoupled. This is safe for simple and well-understood

[1]Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2AZ, UK (e-mail: a.suardi@imperial.ac.uk).

[2]Centre for Automotive Engineering, Cranfield University, Cranfield, MK43 0AL, UK (e-mail: s.longo@cranfield.ac.uk)

[3]Department of Electrical and Electronic Engineering and Department of Aeronautics, Imperial College London, London, SW7 2AZ, UK(e-mail: e.kerrigan@imperial.ac.uk).

[4]Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2AZ, UK(e-mail: g.constantinides@imperial.ac.uk).

algorithms. The control engineer worries about high-level issues, such as closed-loop performance, while the software engineer worries about implementation issues, such as code efficiency and timing [1].

In addition to high numerical precision, other factors such as high clock speed and small packaging have become standard features of modern embedded systems processors. Such advances in digital electronics (together with the development of sophisticated algorithms) have facilitated the spread of computationally-heavy control schemes in low-cost applications with relatively fast dynamics. The embedded control community has started exploring the hardware design dimension in order to reduce hardware costs and increase execution speed by, for example, implementing algorithms with finite and low precision arithmetic [2, 3].

It is well-known that low precision, especially if implemented in fixed-point, allows for much simpler circuits and greater computational speeds [4]. All of the above is at the expense of increased numerical errors that cannot and should not be ignored. There is a surprisingly small amount of theory for the design of such computer-based control systems. These issues could be considered as part of the emerging science called cyber-physical systems theory [5]. Cyber-physical systems are integrations of computation with physical processes and therefore would also embrace the problem of control algorithm performance under numerical errors.

Model Predictive Control (MPC) is a powerful control scheme that, due to the necessity of solving an optimization problem every sampling instant, has only recently found application outside the process industry. One of the often ignored drawbacks of MPC, however, is its sensitivity to numerical errors [6]. The use of different discretization methods has been proven to be an advantage when working with low precision [7]. Methods to avoid variable overflow have been proposed by constraining their ranges with carefully selected scaling methods [8]. However, for these approaches, stability and constraint satisfaction are not guaranteed and, in practice, the only solution to this problem is extensive simulation analysis.

In this paper, we extend the basic idea presented in [9] and we propose a method to guarantee hard constraint satisfaction of an explicit MPC scheme [10, 11] when the algorithm is implemented on a platform using finite and low precision arithmetic. Compared to [9], this paper adds a detailed algorithmic presentation, unveiling the machinery required for the robust controller design, adds a detailed implementation on a FPGA platform and provides experimental results. Furthermore, this paper uses the design tools presented in [12] and [13]. The idea is to quantify the maximum error made by the processor when evaluating the control policy. This is achieved by applying techniques from forward error analysis [14] to the explicit MPC controller. Considering the error as an additive disturbance to the plant dynamics, a controller that is robust to such a disturbance is designed.The resulting controller will therefore be robust against its own finite-precision implementation in a true cyber-physical sense. The proposed method requires the offline solution of an optimization problem, which is non-trivial but possible to automate. The validity of the method has been tested experimentally with a hardware-in-the-loop test ring where the controller has been implemented in a Xilinx Zynq Field-Programmable Gate Array (FPGA) platform.

In Section 2 the explicit MPC problem is formulated. In Section 3 the robust controller design methodology is presented. The procedure requires the analytical computation of error bounds, which is described in Section 4. The experimental validation setup is presented in Section 5 and test results of guaranteed robustness and implementation efficiency are discussed in Section 6.

2. Problem setup

Let us assume that we want to find a discrete-time feedback control law

$$u_k := \kappa(x_k), \tag{1}$$

where $\kappa : \mathbb{R}^n \to \mathbb{R}^m$ is designed to stabilize and guarantee some performance for the discretized plant

$$x_{k+1} = Ax_k + Bu_k, \tag{2}$$

where n is the number of states, m the number of inputs inputs, and $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are the discretized plant matrices. For simplicity we assume that we want to regulate the system from the current state x_0 to the origin. State and input variables are subject to the constraints

$$x_k \in \mathbb{X}, \qquad u_k \in \mathbb{U}, \tag{3}$$

where $\mathbb{X}$ and $\mathbb{U}$ are the polyhedral sets

$$\mathbb{X} = \{x \in \mathbb{R}^n : M_x x \leq k_x\}, \tag{4a}$$

$$\mathbb{U} = \{u \in \mathbb{R}^m : M_u u \leq k_u\}, \tag{4b}$$

containing the origin in their interior. The constraints on state and input may be physical or chosen by design. The finite horizon constrained linear quadratic regulator problem with horizon N is defined as

$$\min_{u_0,u_1,\ldots,u_{N-1}} \; x_N' P x_N + \sum_{i=0}^{N-1} \left(x_i' Q x_i + u_i' R u_i\right), \tag{5a}$$

$$\text{subject to:} \quad x_{k+1} = Ax_k + Bu_k, \quad (x_0 \text{ given}), \tag{5b}$$

$$x_k \in \mathbb{X}, \quad u_k \in \mathbb{U}, \tag{5c}$$

$$x_N \in \mathbb{X}_f, \tag{5d}$$

$$\forall k \in \{0, 1, \ldots, N-1\}, \tag{5e}$$

where $P, Q \in \mathbb{R}^{n\times n}$ are positive semidefinite matrices, $R \in \mathbb{R}^{n\times m}$ is a positive definite matrix, N is the length of the prediction horizon, $\mathbb{X}_f$ is the set in which the terminal state x_N is constrained to lie and, at sample instant k, the state vector $x_k \in \mathbb{R}^n$ is either measured or estimated. Solving (5) yields the optimal open-loop input sequence $u_0^*(x_0), u_1^*(x_0), \ldots, u_{N-1}^*(x_0)$, where, as per standard MPC, the first element u_0^* is applied and the optimization is repeated at

every time step in a receding horizon control fashion. MPC relies on a successive solution of the optimization problem in (5). Such an optimization problem can be expressed as a parametric Quadratic Programming (QP) problem for varying parameters x_0 defined as

$$\min_{U} \quad x_0' Y x_0 + U' H U + x_0' T U \tag{6a}$$

$$\text{subject to:} \quad GU \leq W + E x_0, \quad H > 0 \tag{6b}$$

where the vector $U = \begin{bmatrix} u_0' & u_1' & \dots & u_{N-1}' \end{bmatrix}' \in \mathbb{R}^{Nm}$ is the vector of decision variables and matrices H, T, Y, G, W and E are easily obtained from Q, R and P and by substituting $x_i = A^i x_0 + \sum_{j=0}^{i-1} A^j B u_{i-1-j}$ [15].

When N, n, and m are small, we can compute an MPC feedback controller κ explicitly by solving a multi-parametric optimization problem. In parametric programming, the goal is to find the solution of (6) for a range of parameters values, or equivalently, the closed form solution $x_0 \mapsto U(x_0)$ of (6) for any feasible x_0. This problem could be solved by using the freely available Multi Parametric Toolbox (MPT) [16] written for MATLAB® and the Model Predictive Control Toolbox™ embedded into MATLAB®. The resulting κ is a continuous piecewise affine (PWA) function defined over a polyhedral partition of the state space. Therefore, computing (1) requires:

1. the solution of a point location problem to determine in which polytope — defined by a linear inequality ($Hx_k \leq k$) — the current state x_k belongs to
2. the evaluation of a control law of the form

$$u_k = F x_k + g \tag{7}$$

   associated with the selected region in step 1.

A variety of algorithms have been proposed to solve the point location problem, since this is the most time-consuming task [17, 18, 19, 20]. Such algorithms range from simple ones (a *sequential* search through the regions of the partition) to more complex ones where the region is found via a binary *search tree*. In either

case, the solution of the point location problem and the evaluation of the control law are operations that have to be performed online on the target hardware.

If infinite-precision arithmetic was available, the control action u_k could be computed exactly without introducing any numerical errors, hence complying with the QP problem theoretical guarantees such as constraint satisfaction. However, computing u_k in a processor that works with finite precision (typically any processor) results in the introduction of an error. Such an error is the combination of two factors: first, the selection of the wrong region due to the point location algorithm and second, numerical errors due to the computation of the control law in (7).

It should be noted that this is not a particular feature of fixed-point arithmetic, since errors are also introduced when computations are performed in other (finite) arithmetics, such as IEEE floating-point double-precision. However, in high precision, this error can often be safely ignored. In the sequel, when we refer to 'infinite precision' we are in practice performing computations in a desktop PC using floating-point double-precision (high enough not to cause noticeable failures in the practical problems we have studied).

The computational error cannot be computed offline because this error depends on the online value of the current state x_k. However, it will be shown in Section 4 that such an error can be bounded. If we therefore consider this error as a bounded additive disturbance w_k to the plant dynamics, such as

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{8}$$

a robust controller can be designed for which constraint satisfaction is guaranteed [21, 22, 23, 24]. The interesting point here is that the newly designed robust controller will result in a new control scheme with possibly different error bounds. Hence, the proposed error analysis must be re-applied and the controller design process is repeated (Section 3). In practice, only a few design iterations are required. Constraint satisfaction for the resulting explicit MPC scheme is guaranteed for the chosen finite and low precision arithmetic, if a controller realization exists. We will carry out the analysis for a fixed-point

implementation because this is more suitable for inexpensive applications with fast dynamics [2] and we will assume that enough bits are used for the integer part to avoid overflow. However, a similar procedure could be applied to other number representations, including custom floating-point.

3. Iterative controller design

Let the chosen finite-precision implementation of the feedback control law (1), at time k, produce a numerical error q_k. We can define the finite precision control action $\hat{u}_k$ as

$$\hat{u}_k := u_k + q_k, \tag{9}$$

where u_k is the control action obtained if computations were performed in infinite precision. By applying (9) to the discrete plant dynamics we get

$$\begin{aligned} x_{k+1} &= Ax_k + B\hat{u}_k \\ &= Ax_k + Bu_k + Bq_k \\ &= Ax_k + Bu_k + w_k, \end{aligned} \tag{10}$$

where $w_k \in \mathbb{R}^n$ and $w_k := Bq_k$ is an additive state disturbance to the plant states. Equation (10) is in the same form as (8) and therefore, if the upper bound vector $\overline{w}$ and the lower bound vector $\underline{w}$ on w_k were known, a robust MPC law can be designed using a minimax approach (e.g. [21]). However, such a *new* controller may, and most likely will, produce numerical errors with different bounds.

This procedure is repeated iteratively up to L times, where L could be as small as 10. Using larger values for L will only increase the likelihood of finding a design that tightens the constraints less conservatively. The computational effort for one step of the algorithm is proportional to the square of the number of regions (see Algorithm 3). It can be measured in the order of few seconds when the number of regions is smaller than 100 using a typical laptop's CPU. Once this pool of robust controllers is available with their respective error bounds,

Algorithm 1 Iterative controller design

Set the arithmetic precision (based on the target hardware)

Set the state disturbance bounds: $\overline{w} := 0$ and $\underline{w} := 0$

for $l = 1 : L$ **do**

DESIGN A ROBUST CONTROLLER with disturbance bounds $\overline{w}$ and $\underline{w}$ [21]

COMPUTE ERROR BOUND $\underline{w}_{new}$ and $\overline{w}_{new}$ (Section 4)

$\overline{w} := \overline{w}_{new}$ and $\underline{w} := \underline{w}_{new}$

end for

Within the pool of robust controllers generated, select the controller that generates the smallest error bound and produces an error bound smaller than the error bound used to design the controller itself.

we can proceed with selecting the most appropriate controller. Clearly, we would like to select the controller that generates the smallest error so this will guarantee that the worst numerical error produced by such a controller is always smaller than the error the controller was designed to be robust against, hence guaranteeing constraint satisfaction.

The algorithmic procedure for the controller design is summarized in Algorithm 1.

Example 1. This core idea is now illustrated via an example. Figure 1 shows the numerical error upper bounds $\overline{w}$ obtained at each iteration. For instance, the controller obtained from iteration 5 was designed to be robust against the error bounds produced by the controller obtained from iteration 4, which was designed to be robust against the error bounds produced by the controller obtained from iteration 3 and so forth. It can be noticed that the controller obtained at iteration 2 was designed for an error bound of zero (iteration 1), which is how we have chosen to initialize this procedure. If we decide to use the controller obtained from iteration 4 (see *Circle* in Figure 1), we can indeed guarantee constraint satisfaction. This is because its finite precision implementation will produce errors bounded by the value given at iteration 5, which is smaller than the error bound value this controller was designed for. On the other hand,

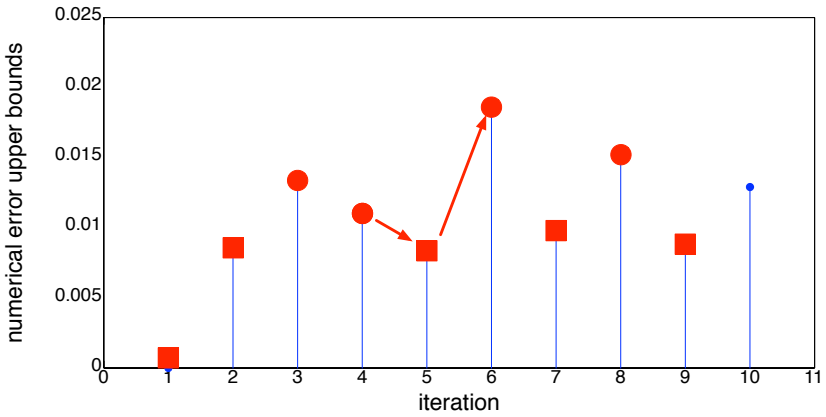


Figure 1: Numerical error upper bounds $\overline{w}$ of sequentially designed controllers. The controller is designed to be robust against an error bounded by the value at iteration i will generate a numerical error bounded by the value given at iteration $i+1$. The controllers designed at iterations $\{3, 4, 6, 8\}$ (marked with *Circles*) guarantee constraint satisfaction, while the controllers designed at iterations $\{1, 2, 5, 7, 9\}$ (marked with *Squares*) do not guarantee constraint satisfaction.

if we decide to use the controller obtained from iteration 5 (see *Square* in Figure 1), we would not be able to guarantee constraint satisfaction. This is the case because this controller's error bound (given at iteration 6) is larger than the bound it was designed to be robust against. A similar argument can be made for the lower bound.

From a theoretical and intuitive perspective, the numerical error bounds do not need to converge. It should be noted that, running extensive simulations we have been able to verify that the preliminary results presented in [9] where misleading. In most of the cases the numerical error does not converge. Moreover, no clear pattern seem to exists and the behavior is not predictable. For example, Figure 2 shows the first 100 iterations. It is difficult to see a clear pattern, which confirms our intuition that such a pattern may not exist.

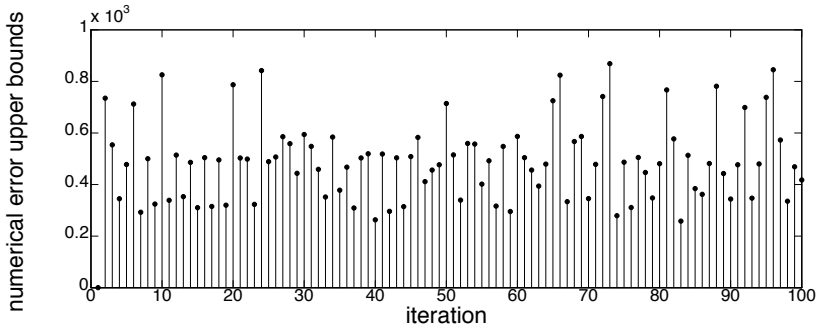


Figure 2: Numerical error upper bounds $\overline{w}$ of 100 sequentially designed controllers.

4. Error Bound Computation

We will now discuss one of the main contributions of this paper, which is how the error bounds $\underline{w}$ and $\overline{w}$ in Algorithm 1 are computed. To compute such bounds, all the possible sources of error q_k, generated by the control law, need to be investigated. Errors are generated by two factors: (i) the selection of a wrong region due to quantization of the (measured or estimated) current state x_k and/or quantization of the polyhedral partition and (ii) the numerical errors introduced by the computation of the control law (7) using finite precision.

The analysis of these two sources of error will be considered in Sections 4.1. The quantification of the error bounds, achieved by solving an optimization problem, will be given in Section 4.3.

4.1. Point Location Algorithm: Binary Search Tree

A more efficient approach to solve the point location problem consists of building a binary search tree off-line [17] and traverse the tree on-line as described in Algorithm 2

When using a search tree point location algorithm, it is possible to avoid the limitations of the *sequential* search algorithm.

The state space is uniquely and fully covered by all leaves. This is shown graphically in Figures 3-a (infinite precision) and 3-b (finite precision). In Figure

3-b (finite precision) overlaps and uncovered areas do not occur. An explanation for this follows.

Compared to the case of infinite precision, in finite precision some leaves might not be reachable due to numerical errors. This becomes more likely to happen as the arithmetic precision is reduced. On the other hand, in finite precision the reachable convex sets are able to cover the areas that were associated with the unreachable leaves, as shown in Figure 4. However, even if the search tree point location algorithm can guarantee to locate a unique region, computational error q_k might still occur. This is because a different region would have been selected when compared to the infinite precision case, due to numerical errors introduced when representing the convex sets associated with the leaves.

Let us define $\mathbb{P} := \{x | Mx \leq s\}$ to be the convex set associated with a leaf of the search tree implemented in infinite precision and $\hat{\mathbb{P}} := \{\hat{x} | \hat{M}\hat{x} \leq \hat{s}\}$ to be the convex set associated with a leaf of the search tree in finite precision. An error in the region identification can occur if the convex set, resulting from the intersection $\mathbb{P} \cap \hat{\mathbb{P}}$, contains at least one finite precision value of the state vector x_k. This happens when $\mathbb{P}$ and $\hat{\mathbb{P}}$ do not represent the same convex set.

4.2. Function Evaluation Analysis

Once a region is selected, the associated control law in (7) is computed. Here, errors are introduced because of the finite-precision arithmetic used to

Algorithm 2 Binary search tree

Let each node of the tree (N_k) be described by a unique hyperplane

$N_k \leftarrow$ tree root node

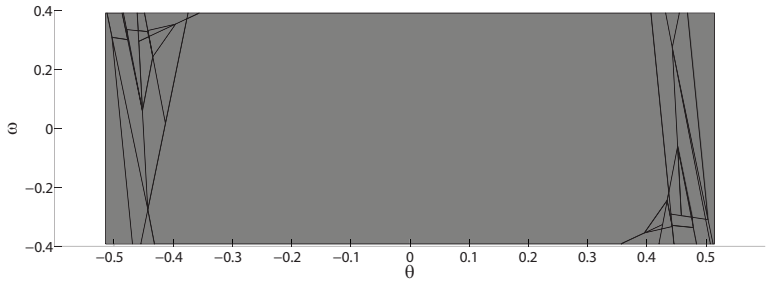
while N_k is a not a leaf node **do**

$N_k \leftarrow$ child node of N_k according to the sign of the evaluated hyperplane

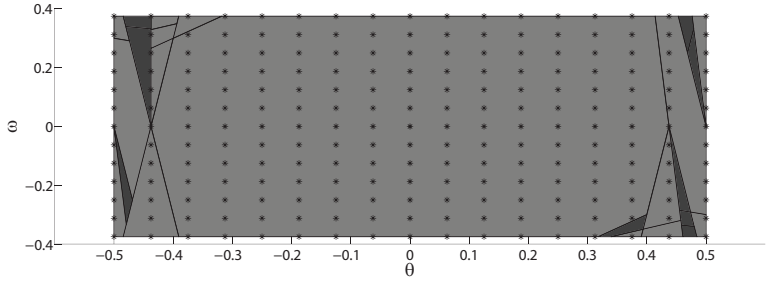
end while

The reached leaf node represents a unique convex set, made of the hyperplanes encountered while traversing the tree

return the region associated to the leaf node



(a)



(b)

Figure 3: Explicit MPC polyhedral partition of the inverted pendulum system generated with MPT: (a) *binary search* tree for point location with *infinite* precision (double floating-point); (b) *search tree* for point location with *finite* precision (fixed-point, 4 bits fraction length). In (b) stars point represents all the feasible discretized x value and light gray are areas exclusively associated with one polytope and dark gray are areas covered by any feasible discretized x value.

perform the algebraic operations.

Let us define the finite-precision representation $\hat{\alpha}$ of a real constant $\alpha \in \mathbb{R}$ as

$$\hat{\alpha} := \alpha + \epsilon_{\hat{\alpha}}, \tag{11}$$

where $\epsilon_{\hat{\alpha}} \in \mathbb{R}$ is the quantization error. If a fixed-point representation is used, the quantization error due to truncation is $\epsilon_{\hat{\alpha}} \in \left(-2^{-l}, 0\right]$ and $l \in \mathbb{N}^+$ is an integer that defines the fraction length, in terms of the number of bits. The assumption here is that we use enough bits for the integer part so that overflow does not occur.

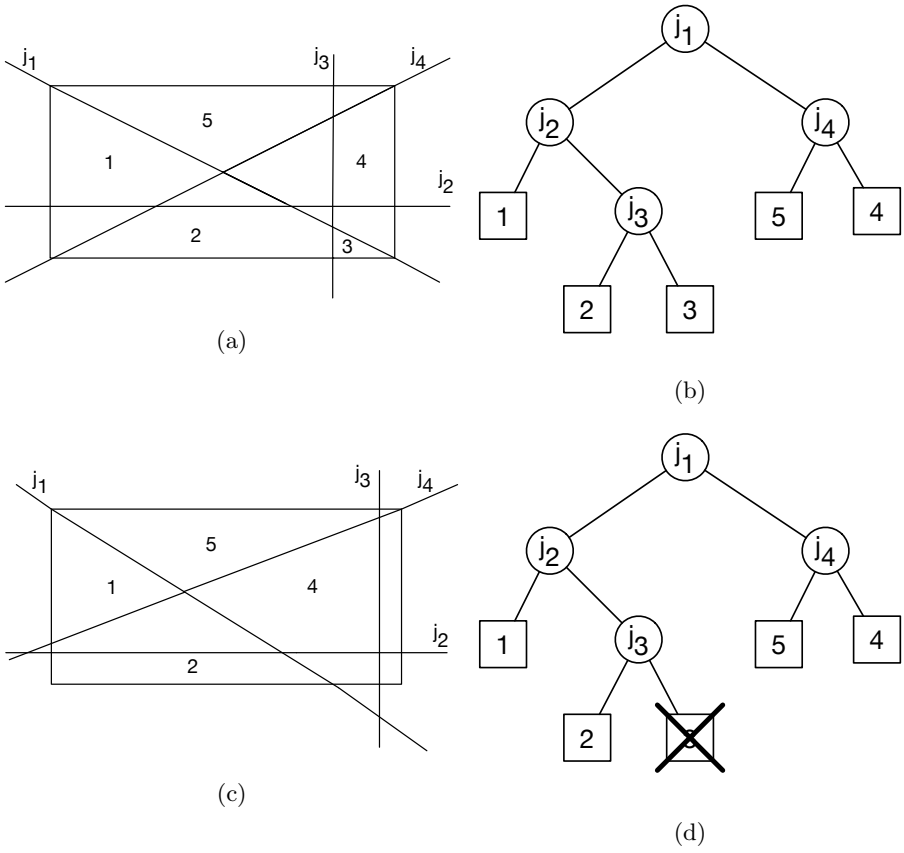


Figure 4: (a) polyhedral partition and (b) associated binary search tree with *infinite* precision (double floating-point); (c) polyhedral partition and (d) associated binary search tree with *finite* precision (fixed-point, 4 bits fraction length). Using *finite* precision, the leaf number 3 becomes unreachable, the area associated to leaf number 3 is covered now by leaf number 4, thus the state space fully coverage is preserved.

By applying the finite-precision representation (11) to the control law (7) we define the finite-precision control action $\hat{u}_k$ associated with the convex set $\hat{\mathbb{P}}$ as

$$\hat{u}_k = \hat{F}\hat{x}_k + \hat{g}, \tag{12}$$

where

$$\hat{x}_k = x_k + e_{\hat{x}_k}, \tag{13}$$

and $\hat{F}$ and $\hat{g}$ are the finite-precision representation of F and g, respectively. The values in $\hat{F}$ and $\hat{g}$ can be computed exactly, since their infinite-precision values are known and fixed. The state vector x_k, however, is unknown and thus is its quantization error vector $e_{\hat{x}} \in \mathbb{R}^n$.

By considering the infinite-precision control action (7) associated with the convex set $\mathbb{P}$

and substituting (12) into (9), we can compute the control law error q_k and express the additive state disturbance w_k to the plant as

$$\begin{aligned} w_k &= Bq_k \\ &= B\left(\hat{u}_k - u_k\right) \\ &= B\left[\left(\hat{F} - F\right)\hat{x}_k + Fe_{\hat{x}_k} + (\hat{g} - g)\right]. \end{aligned} \tag{14}$$

The state disturbance w_k cannot be computed exactly because the value of $\hat{x}_k$ and $e_{\hat{x}_k}$, as well as the error in the point location algorithm, are unknown. We will now show how to compute the maximum $\overline{w}(n_i)$ and minimum $\underline{w}(n_i)$ bounds (worse case scenario) for each element $n_i \in \{1, 2, \ldots n\}$ of the disturbance vector w_k (Section 4.3) considering all possible sources of error.

Let us define index $i \in \{1, 2, \ldots N_{leaf}\}$ to denote $\mathbb{P}^i$ — the convex set associated with leaf i of the search tree implemented in *infinite* precision — and index $j \in \{1, 2, \ldots \hat{N}_{leaf}\}$ denote $\hat{\mathbb{P}}^j$ – the convex set associated with leaf j of the search tree in *finite* precision. Here, N_{leaf} and $\hat{N}_{leaf}$ are the number of leaves of the search tree in infinite and finite precision, respectively. Therefore, every possible permutation of i and j has to be exhaustively investigated to determine when the point location algorithm makes an error when selecting the region.

This leads to an algorithm complexity proportional to $N_{leaf} * \hat{N}_{leaf}$, thus proportional to the square of the number or regions. The procedure is outlined in Algorithm 3.

Algorithm 3

for $n_i = 1, 2, ..., n$ **do**
  for $i = 1, 2, ..., N_{leaf}$ **do**
    for $j = i, i+1, ..., \hat{N}_{leaf}$ **do**
      if $\exists\, \hat{x}_k \subset \mathbb{P}^i \cap \hat{\mathbb{P}}^j$ **then**
        COMPUTE BOUNDS $\overline{w}^{ij}(n_i)$ and $\underline{w}^{ij}(n_i)$ associated with $\mathbb{P}^i \cap \hat{\mathbb{P}}^j$ (Section 4.3)
      end if
    end for
  end for
  $\overline{w}(n_i) := \max_{i,j} \overline{w}^{ij}(n_i)$ and $\underline{w}(n_i) := \min_{i,j} \underline{w}^{ij}(n_i)$
end for

4.3. Maximum and Minimum Bound Computation

Based on the considerations above, the task of computing the upper $\overline{w}^{ij}(n_i)$ and lower $\underline{w}^{ij}(n_i)$ bounds associated with the polytope intersection $\mathbb{P}^i \cap \hat{\mathbb{P}}^j$ is translated into solving two optimization problems: a maximization and a minimization, respectively.

As an example, let us consider the maximization problem (a similar approach can be used for the minimization). Because the state $\hat{x}_k$ has fixed-point values, it can be scaled by a factor of 2^l and expressed as an integer $z_k \in \mathbb{Z}^n$, i.e.

$$z_k = \hat{x}_k \cdot 2^l. \tag{15}$$

This will lead to the mixed-integer linear programming (MILP) problem

$$\max_{z_k, e_{\hat{x}_k}} \quad b_{n_i} \left[\left(\hat{F}^j - F^i \right) 2^{-l} z_k + F^i e_{\hat{x}_k} + \left(\hat{g}^j - g^i \right) \right] \tag{16a}$$

$$s.t. \quad z_k \in \mathbb{Z}^n \tag{16b}$$

$$z_k 2^{-l} \in \mathbb{P}^{ij} \tag{16c}$$

$$-2^{-l} < e_{\hat{x}_k} \leq 0, \tag{16d}$$

where z_k is the integer decision variable and the vector b_{n_i} is the row n_i of

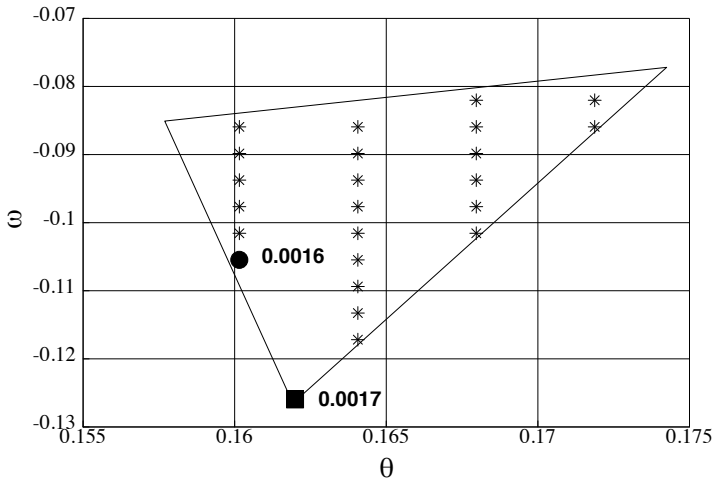


Figure 5: Solution for the maximization problems: MILP and LP. *Stars* show the feasible values of $\hat{x}_k$ inside the polytopes intersection $\mathbb{P}^i \cap \hat{\mathbb{P}}^j$. The *Circle* shows the solution of the MILP problem. The *Square* shows the solution of the LP.

matrix B. The scaling factor 2^l is also applied to the maximization function and to the left-hand-side of the inequality constraint (16c).

The solution of a MILP problem is computationally demanding [25], especially when the number of bits l used for the fraction length is large. This is because the search space area given by the inequality constraints (16c) increases proportionally with 2^l. Our proposed solution is to assume that the variable $\hat{x}_k$ is continuous. This will allow us to solve, instead of the MILP in (16), the linear programming (LP) problem

$$\begin{aligned} \max_{\hat{x}_k, e_{\hat{x}_k}} \quad & b_{n_i} \left[\left(\hat{F}^j - F^i \right) \hat{x}_k + F^i e_{\hat{x}_k} + \left(\hat{g}^j - g^i \right) \right] \\ s.t.: \quad & \hat{x}_k \in \mathbb{P}^{ij} && \text{(17a)} \\ & -2^{-l} < e_{\hat{x}_k} \leq 0. && \text{(17b)} \end{aligned}$$

The solution, as shown for an example in Figure 5, will be a worst case approximation of the real solution provided by (16). Although slightly more conservative, this is admissible since only a bound is computed.

5. Experimental setup

In this section we describe the setup for the hardware-in-the-loop (HIL) test rig we have built to show the validity of the robust controller design approach described above. This setup consists of a low-end FPGA evaluation board, where the controller is implemented at variable finite precision arithmetic, connected in a feedback loop to a real-time host system that simulates a classical unstable system, namely the inverted pendulum. The FPGA board where the control algorithm has been implemented is a Xilinx Zynq FPGA (xc7z020) [26] mounted on the ZedBoard evaluation module [27]. In order to set up the controller for the HIL implementation and the whole of the HIL simulation, a new FPGA development tool has been built and used. This tool is called Protoip [12] and more details can be found in [13]. Protoip is open source and allows one to automatically build and deploy an algorithm written in C/C++ onto an FPGA. Protoip builds an HIL setup composed of two main hardware/software parts: (i) a Xilinx Zynq FPGA on which a microprocessor (ARM-based) and the electronic circuit representing the control algorithm are running; (ii) a host system with MATLAB software.

As shown in Figure 6, the host system and the FPGA communicate via Ethernet interface by means of UDP/IP or TCP/IP packets. On the FPGA side, the microprocessor runs a bare-metal software application. UDP/IP or TCP/IP server bridges the communication between the physical Ethernet interface, the DDR memory used as a shared memory space and the user algorithm. On the other side, the host system runs a UDP/IP or TCP/IP client accessible via a Matlab API provided by Protoip. The complete setup is shown diagrammatically in Figure 7.

Once the prototype has been built, Protoip also provides measurements of the hardware characteristics of the digital circuit being built in terms of the amount of hardware resources (e.g. memory) used and an accurate estimate of its energy consumption.

The system to be controlled (the inverted pendulum) is described by the

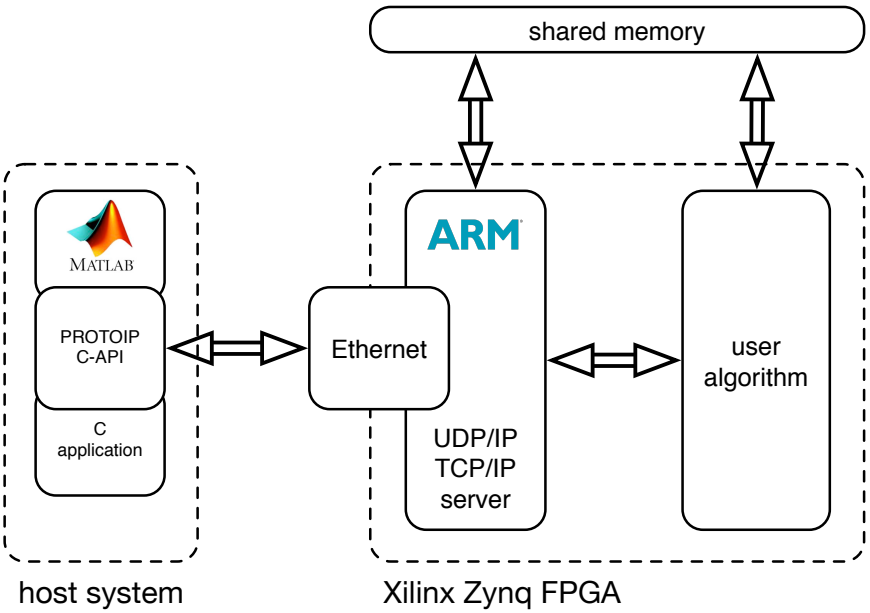


Figure 6: Hardware In the Loop (HIL) setup built by Protoip using the Xilinx Vivado FPGA compiling tool.

linearized continuous-time dynamics

$$\begin{bmatrix} \dot{\vartheta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{g}{L} & -\frac{b}{mL^2} \end{bmatrix} \begin{bmatrix} \vartheta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{mL^2} \end{bmatrix} u, \tag{18a}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \vartheta \\ \omega \end{bmatrix}, \tag{18b}$$

where the states ϑ and ω are, respectively, the angular displacement measured from the equilibrium position and the angular velocity; u is the input torque, $g = 9.81\text{m/s}^2$ the gravitational force, $m = 344\text{kg}$ the mass, $b = 0.48\text{Ns/m}$ the rotation friction and $L = 1.703\text{m}$ the length of the pendulum. Given the continuous-time weight matrix on the states $Q_c = I$ and on the inputs $R_c = 0.1$, the continuous-time plant matrices and weight matrices have been discretized with a sampler with period $T_s = 0.1\text{s}$ and a zero-order-hold. An explicit MPC controller has been formulated with a time prediction horizon of $T = 0.4\text{s}$. State

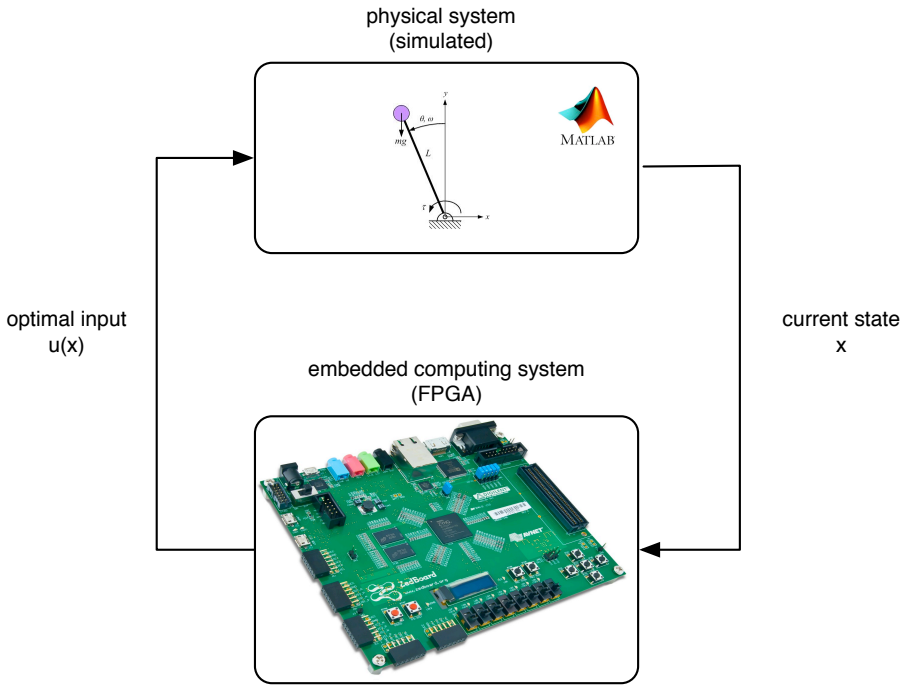


Figure 7: Closed-loop tests using the HIL setup. The plant is emulated in Matlab, while the controller is running on the FPGA.

constraints have been set to

$$\begin{bmatrix} -\pi \\ -\pi/8 \end{bmatrix} \leq \begin{bmatrix} \vartheta \\ \omega \end{bmatrix} \leq \begin{bmatrix} \pi \\ \pi/8 \end{bmatrix}. \tag{19}$$

The procedure for the experimental results is the following:

1. Decide on a finite precision (fixed-point) representation. Use enough bits for the integer part to avoid overflow and we set the number of bits for the fraction length to be equal to an arbitrary l. Using MATLAB and MPT toolbox Software, execute, off-line, the iterative controller design procedure described in Section 3.

2. Within the pool of resulting robust controllers, select the controller that guarantees constraints satisfaction with minimum conservativeness as described in Section 3.

3. Build and deploy the controller on the FPGA using the Protoip tool.
4. Run closed loop tests using the HIL setup provided by Protoip. Perform extensive testing (statistics collected from about 1000 simulations) from random initial conditions to steady states using the setup shown in Figure 7.

5.1. Test case description

In order to validate our claims, we have devised two sets of experiments using the design procedure described above.

The *first* set, composed of tests 1, 2 and 3, is aimed at showing how a finite precision arithmetic impacts on constraint satisfaction when a controller is not designed to be robust against it. Thus, we have designed a nominal controller (equivalent to setting the state disturbance bounds $\overline{w} := 0$ and $\underline{w} := 0$) and implemented it into the FPGA using:

- *test ID 1*: single precision floating-point arithmetic.
- *test ID 2*: $l = 8$ bits fraction length fixed-point arithmetic.
- *test ID 3*: $l = 12$ bits fraction length fixed-point arithmetic.

The *second* set, composed of tests 4 and 5, is aimed at showing how constraint satisfaction can be guaranteed by design even with finite precision arithmetic. We have run the proposed iterative controller design procedure (Algorithm 1) for $L = 10$ with the fixed-point fraction length $l = 8$ and we have then calculated the state disturbance bounds, such as, for instance, the upper bound $\overline{\omega}$ shown in Figure 8. It should be noted that, for fraction lengths smaller than 8 bits, it is not possible to generate a robust controller. This is because the computed error bound using Algorithm 3 results to be too big to be used as states disturbance for the formulation of a new controller.

Among the pool of designed controllers, we have run the closed-loop simulations using two particular ones:

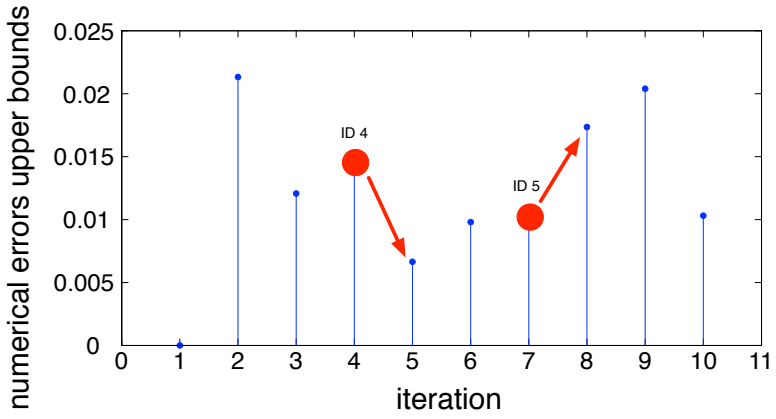


Figure 8: Values of upper bound state ω disturbance running Algorithm 1 for 10 iterations when fraction length is set to 8 bits.

- *test ID 4*: the controller designed at iteration 4 in Figure 8 guarantees constraint satisfaction while being the least conservative in constraints tightening.
- *test ID 5*: the controller designed at iteration 5 in Figure 8 does not guarantee constraint satisfaction despite being less conservative.

6. Experimental results

6.1. Closed-loop simulation results

Consider the first set of tests based on an explicit MPC controller robust to the (very small) error introduced by floating-point double precision arithmetic (this will be almost equivalent to the design of a 'non-robust' explicit MPC). If this controller is implemented on-line using finite-precision arithmetic (fixed-point), constraint violations might occur due to the numerical errors arising from the implementation. The lower the precision, the larger the errors, as shown in Figure 9 for an example.

On the other hand, if it is known that finite-precision arithmetic will be used, then the controller can be designed using Algorithm 1. Constraint satisfaction

can be guaranteed by design (test ID 4) only if the selected controller generates error bounds than are smaller that the error bounds that were used for the design, as shown in Figure 10. However, if this is not the case (test ID 5), constraints might be violated (Figure 11).

6.2. Performance analysis

We have summarized all the results in Table 1. We compare the controller performance in terms of implementation complexity, closed-loop performance. For implementation complexity we measure the number of regions and hyperplanes generated, the maximum depth of the search tree, the memory utilized and the hardware energy consumption. For the closed-loop performance we measure the closed-loop cost J_{cl}, the percentage of the times that constraints have been violated and the largest constraint violation.

The closed-loop cost of constrained systems controller in presence of round-off errors cannot be computed analytically. This is measured by averaging the costs of 1000 simulations of the controlled plant evolution from a set of random and uniformly distributed initial condition to steady state. Hence, we define J_{cl} as

$$J_{cl} := x'_{N_{cl}} P x_{N_{cl}} + \sum_{j=0}^{N_{cl}-1} x'_j Q x_j + u'_j R u_j \tag{20}$$

where N_{cl} is the number of simulation steps chosen long enough so that $x_j \approx 0$ and $u_j \approx 0$ for all $j \geq N_{cl}$ and all feasible initial conditions.

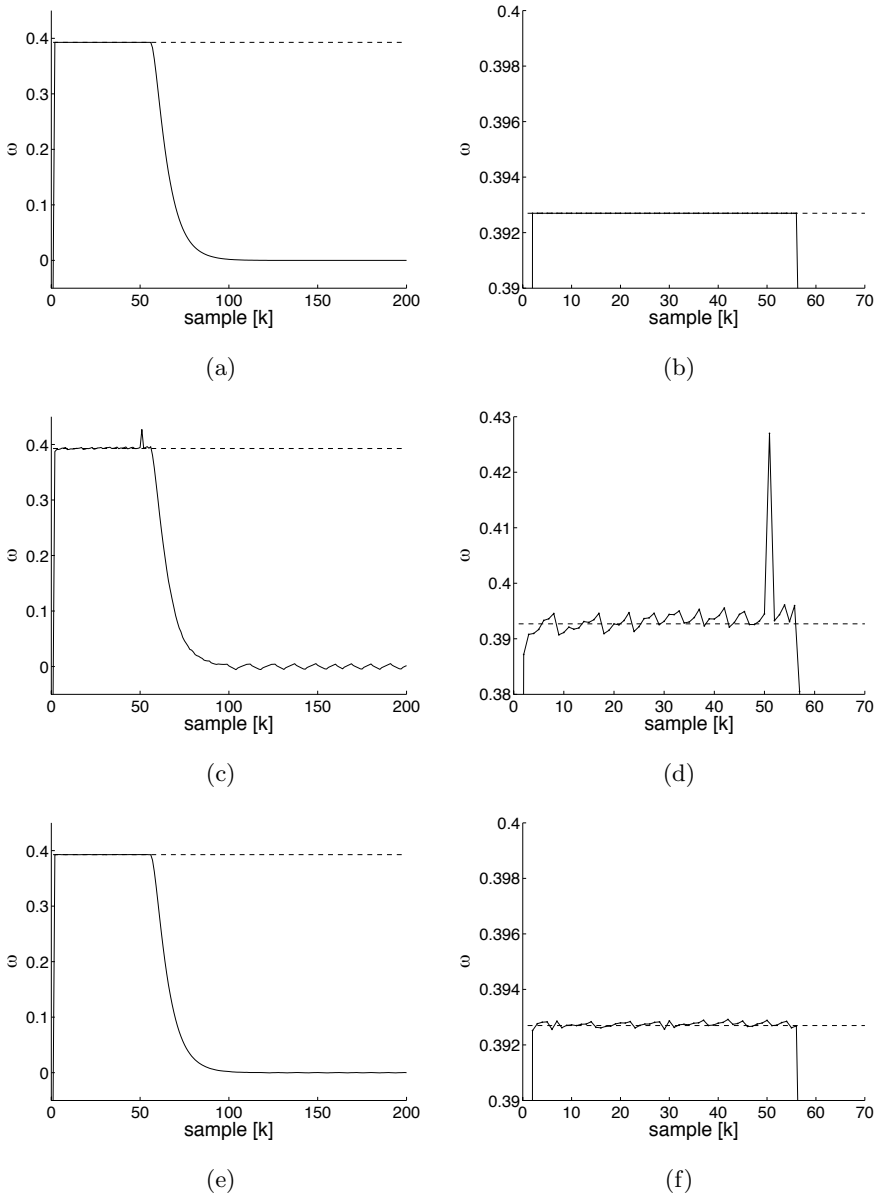
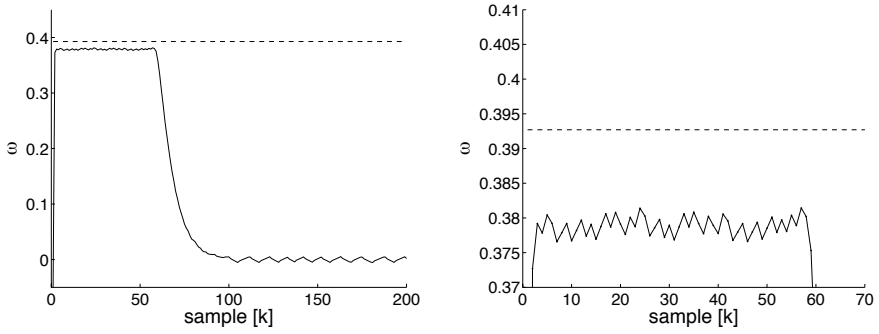
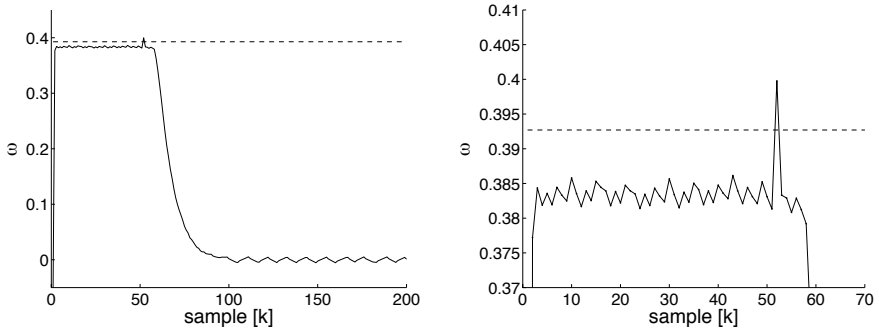


Figure 9: Evolution of the state ω of the inverted pendulum system during a closed-loop simulation. Test ID 1 using floating-point single precision the constraint is guaranteed: (a) full evolution, (b) detail when the state ω constraint is activated. Test ID 2 using fixed-point with 8-bit fraction length the constraint is not guaranteed: (c) full evolution, (d) detail when the state ω constraint is not guaranteed. Test ID 3 using fixed-point 12-bit fraction length the constraint is not guaranteed: (e) full evolution, (f) detail when the state ω constraint is not guaranteed. The dashed line represents the state constraint. It should be noted that implementing a nominal controller using fixed-point arithmetic constraint violation occurs.



(a) (b)

Figure 10: Test ID 4: evolution of the state ω of the inverted pendulum system during a closed-loop test: (a) full evolution, (b) detail when the state ω is close to the constraint, thus the constraint is guaranteed. The dashed line represents the state constraint.



(a) (b)

Figure 11: Test ID 5: evolution of the state ω of the inverted pendulum system during a closed-loop test: (a) full evolution, (b) detail when the state ω constraint is not guaranteed. The dashed line represents the state constraint.

Table 1: Prototype Performances

Test ID	#Regions	#Hyperplanes	Max depth search tree	Memory [KByte]	Power [Watts]	Latency [μs]	J_{cl}	Constraint violation [%]	MAX value above constraint $\pi/8$
1	75	193	10	31.90	0.202	2.59	32.54	0%	0
2	75	193	10	19.04	0.186	1.14	32.51	87%	0.0357
3	75	193	10	22.26	0.186	1.15	32.55	85.4%	0.0002
4	28	48	7	5.18	0.183	0.9	33.61	0%	0
5	27	46	7	4.98	0.184	0.9	33.26	9.7%	0.0093

From the table, the following can be observed:

- As expected, constraints are violated more frequently the smaller the number of bits used when implementing a nominal controller.
- When implementing a nominal controller in fixed-point, the constraints are violated by a factor of 10 times (for this specific test case) the computational quantization error.
- Constraint satisfaction is guaranteed at all times (0% constraint violation) if a proper state disturbance bound (test ID 4) is selected. If not (test ID 5), Algorithm 1 does not guarantee constraint satisfaction for all the tested cases (constraints are violated 9.7% of the time).
- The controller implementation complexity (number of regions and hyperplanes) decreases when using our procedure. The resources used by the controller (memory) is reduced because the number of regions and its energy consumption is lowered since the time needed to execute the controller is smaller (shallower search tree). This can be seen when comparing the nominal design in test ID 2 (nominal controller) against our robust design in tests ID 4 and 5 (robust controller).
- Although robustness is guaranteed by design, the closed-loop performance of the robust controller is slightly worse when compared to the nominal controller due to constraint tightening.

The proposed controller design and the presented HIL test procedure have been indeed applied to various plants in different configuration scenarios. As far as we have observed, we can say that all the tests gave us the expected behaviour: constraint satisfaction is guaranteed at all times if a proper state disturbance bound is selected and it is not guaranteed otherwise. These results have not been included in the manuscript as we thought they were not adding any additional insight.

7. Conclusions

We have proposed, and verified via an FPGA implementation in an HIL setup, an explicit MPC design that robustly guarantees hard constraint satisfaction in the presence of finite-precision arithmetic errors introduced by the controller's own implementation. We have shown that robust constraints satisfaction can indeed be guaranteed by the robust controller implemented in low numerical precision while this was not the case for the nominal controller. The controllers were tested experimentally with precision as low as 8 bits. Some side benefits of our approach include a reduction in complexity due to a reduction in number of regions. This, together with the reduction in hardware resources given by the low precision would allow explicit MPC to be implemented in small, inexpensive and energy efficient platforms. Future work may include the extension of this technique to other controllers design methods.

References

[1] J. Teich, Hardware/software codesign: The past, the present, and predicting the future, Proceedings of the IEEE 100 (Special Centennial Issue) (2012) 1411–1430.

[2] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. Kerrigan, M. Morari, Embedded online optimization for model predictive control at Megahertz rates, IEEE transaction on automatic control 59 (2014) 3238–3251.

[3] G. A. Constantinides, Tutorial paper: Parallel architectures for model predictive control, in: Proc. European Control Conference 2009, Budapest, 2009, pp. 138–143.

[4] D. A. Patterson, J. L. Hennessy, Computer architecture: a quantitative approach, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[5] W. Wolf, Cyber-physical systems, Computer 42 (3) (2009) 88–89.

[6] A. Hasan, E. C. Kerrigan, G. A. Constantinides, Control-theoretic forward error analysis of iterative numerical algorithms, IEEE Transactions on Automatic Control 58 (6) (2013) 1524–1529.

[7] S. Longo, E. C. Kerrigan, G. A. Constantinides, Constrained LQR for low-precision data representation, Automatica 50 (1) (2014) 162 – 168.

[8] J. L. Jerez, G. A. Constantinides, E. C. Kerrigan, A low complexity scaling method for the lanczos kernel in fixed-point arithmetic, IEEE Transactions on Computers 99 (PP).

[9] A. Suardi, S. Longo, E. C. Kerrigan, G. A. Constantinides, Robust explicit mpc design under finite precision arithmetic, World Congress of the International Federation of Automatic Control 19 (2014) 2939–2944.

[10] A. Bemporad, M. Morari, V. Dua, E. N. Pistikopolous, The explicit linear quadratic regulator for constrained systems, Automatica 38 (1) (2002) 3–20.

[11] M. Kvasnica, M. Fikar, Performance-lossless complexity reduction in explicit MPC, in: 49th IEEE Conference on Decision and Control (CDC), 2010, pp. 5270–5275.

[12] A. Suardi, E. C. Kerrigan, G. A. Constantinides, Fast FPGA prototyping toolbox for embedded optimization, in: Proc. European Control Conference, 2015.

[13] Andrea Suardi, Protoip (Dec. 2014).
URL `http://github.com/asuardi/protoip`

[14] N. J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[15] J. M. Maciejowski, Predictive control with constraints, Prentice-Hall, 2002.

[16] M. Herceg, M. Kvasnica, C. Jones, M. Morari, Multi-Parametric Toolbox 3.0, in: Proc. of the European Control Conference, Zürich, Switzerland, 2013, pp. 502–510, `http://control.ee.ethz.ch/~mpt`.

[17] P. Tøndel, T. A. Johansen, A. Bemporad, Evaluation of piecewise affine control via binary search tree, Automatica 39 (5) (2003) 945 – 950.

[18] C. Jones, P. Grieder, S. Rakovic, A logarithmic-time solution to the point location problem for parametric linear programming, Automatica 42 (12) (2006) 2215–2218.

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

[20] M. Monnigmann, M. Kastsian, Fast explicit MPC with multiway trees, in: Proc. of the 18th IFAC World Congress, 2011, 2011.

[21] A. Bemporad, F. Borrelli, M. Morari, Min-max control of constrained uncertain discrete-time linear systems, IEEE Transactions on Automatic Control 48 (9) (2003) 1600–1606.

[22] M. Baotic, F. Borrelli, A. Bemporad, M. Morari, Efficient On-Line Computation of Constrained Optimal Control, SIAM Journal on Control and Optimization 47 (5) (2008) 2470–2489.

[23] E. C. Kerrigan, J. Maciejowski, Robustly stable feedback min-max model predictive control, in: American Control Conference, 2003. Proceedings of the 2003, Vol. 4, 2003, pp. 3490–3495 vol.4.

[24] E. C. Kerrigan, D. Mayne, Optimal control of constrained, piecewise affine systems with bounded disturbances, in: Decision and Control, 2002, Proceedings of the 41st IEEE Conference on, Vol. 2, 2002, pp. 1552–1557 vol.2.

[25] D. Vivek, E. N. Pistikopoulos, An algorithm for the solution of multiparametric mixed integer linear programming problems, Annals of operations research 99 (1-4) (2000) 123–139.

[26] Xilinx, Xilinx website (Feb. 2015).
URL `http://www.xilinx.com/`

[27] ZedBoard, Zedboard (Oct. 2014).
URL `http://zedboard.org/`