Automatic Software and Computing Hardware Co-design for Predictive Control

Bulat Khusainov, Eric C. Kerrigan *Senior Member, IEEE,* George A. Constantinides *Senior Member, IEEE*,

Abstract—**Model Predictive Control (MPC) is a computationally demanding control technique that allows dealing with multiple-input and multiple-output systems, while handling constraints in a systematic way. The necessity of solving an optimization problem at every sampling instant often (i) limits the application scope to slow dynamical systems and/or (ii) results in expensive computational hardware implementations. Traditional MPC design is based on manual tuning of software and computational hardware design parameters, which leads to suboptimal implementations. This paper proposes a framework for automating the MPC software and computational hardware co-design, while achieving the optimal trade-off between computational resource usage and controller performance. The proposed approach is based on using a bi-objective optimization algorithm, namely BiMADS. Two test studies are considered: Central Processing Unit (CPU) and Field-Programmable Gate Array (FPGA) implementations of fast gradient-based MPC. Numerical experiments show that optimization-based design outperforms Latin Hypercube Sampling (LHS), a statistical sampling-based design exploration technique.**

Index Terms—**Model predictive control, Hardware-software co-design, FPGA, Multi-objective optimization, Design automation**

I. Introduction and related work

Model predictive controller design is a multidisciplinary problem that involves tuning several coupled design parameters. Traditionally MPC controllers were tuned manually, with a trial and error approach, which cannot be considered as a viable option for most present-day applications, considering the number of design parameters and design evaluation time [1]. Moreover, manual tuning often requires understanding the nature of the controlled dynamical system and MPC controller with the underlying optimization solver. Available tuning guidelines for model predictive control, including heuristic and systematic (but not automatic) approaches, are reviewed in [2]. Note that only high level optimal control problem parameters (e.g. horizon length, weights on states/inputs) are considered in the review paper, without regard to solving the underlying optimization problem.

The full design exploration approach, which can be considered as the simplest way of design automation, leads to unacceptable exponential complexity scaling with respect to the number of parameters. Statistical exploration methods, e.g. Monte Carlo methods [3] and Latin Hypercube Sampling (LHS) [4] attempt to accelerate exploration by randomising the sampling process. However, all above techniques *explore* the design space without *exploiting* the knowledge about evaluated designs. As a result, statistical algorithms often achieve uniform distribution in the parameters space, without giving priority to the most promising (in terms of performance criteria) areas and hence waste time evaluating unpromising implementations. However, Monte Carlo methods and LHS can be used for identifying an initial guess for other algorithms.

An alternative approach for taking humans out of the design loop is based on systematic optimization [5]. The following applications of optimization techniques to MPC design can be found in the literature:

- [6] presents an application of multi-objective optimization to the Van de Vusse reaction considering two contradicting objectives: maximizing the desired product and minimizing the unwanted product. Several multi-objective optimization methods are employed in this work: normalized normal constraint [7], normal boundary intersection [8] and weighted sum.
- Study [9] compares systematic multi-objective optimization-based parameter tuning using enhanced normalized normal constraint method [10] against Monte Carlo simulations. The case study considers NMPC applied to a biochemical system.
- Another algorithm for tuning NMPC controllers with application to chemical processes is presented in [11]. Instead of exploring the trade-offs the algorithm attempts to find a single compromise design using the approach [12].

Other examples of automatic tuning of MPC controllers can be found in [13] and [14]. The studies above share some common features:

- Multi-objective nature of MPC design problem is acknowledged and hence dedicated multi-objective optimization algorithms are used.
- Contradicting control objectives are considered as design goals without explicit optimization of computational complexity.
- Optimization solver parameters are not systematically tuned, it is assumed that optimal control problems are solved to optimality. This might be a valid assumption for slow dynamical systems, e.g. chemical reactors, but not for fast applications, e.g. robotics.
- Underlying computational platform parameters are not involved in the tuning process.

In contrast, in this work we automate MPC controller design considering computational resources as a design objective. The control performance can be traded off against resource usage by tuning both algorithmic and computational hardware-related parameters.

The paper is organised as follows. Following the introduction, the target computational platform is described in Section II. An approach for formulating predictive controller design as an optimization problem is presented in Section III. Possible ways of formalising design objectives and constraints are discussed within the section. Following that, Section IV reviews algorithms for solving the design optimization problem and justifies using the BiMADS algorithm [15] for solving MPC design problems. Two case studies are considered in Section V: CPU-based and FPGA-based designs for a fast gradient-based predictive controller. Section VI concludes the paper.

II. Target computational platform

In this work we prototype predictive controller implementations using Xilinx Zynq-7000 XC7Z020 System-on-a-Chip (SoC), which incorporates a dual-core ARM Cortex-A9 processor and Artix-7 FPGA logic. The FPGA fabric contains 53200 Lookup Tables (LUTs), 106400 Flip-Flops (FFs), 220 DSP blocks and 140 block RAMs (BRAMs) with total capacity of 4.9 Mb. Using SoC allows

This work was funded from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 607957 (TEMPO).

The authors are with the Department of Electrical & Electronics Engineering, Imperial College London, SW7 2AZ, U.K. E.C. Kerrigan is also with the Department of Aeronautics. email: bulat.khusainov@outlook.com, e.kerrigan@imperial.ac.uk, g.constantinides@imperial.ac.uk

prototyping both Central Processing Unit (CPU) and FPGA implementations using one single chip.

The following techniques can be used to accelerate optimization-based controllers on FPGAs:

- *Data-level parallelization* refers to splitting computations across multiple processors, so that different sets of data are processed simultaneously. Usually applied on regular data structures, e.g. arrays.
- *Data pipelining* is based on connecting processing units in series, so that the output of one unit is the input to the next one. The elements of this chain, i.e. pipeline, process data simultaneously.
- Tailoring *number representation* for a given algorithm, instead of using standard single/double precision floating point arithmetic, allows achieving better time vs resource usage trade-offs. For example, for fixed-point number representation arithmetic operations complexity is identical to that of integers.

In this work we synthesize FPGA circuits using vendor's high level synthesis tool, namely Vivado HLS. Vivado HLS allows describing FPGA circuit architecture with C code and additional optimization directives for implementing the above above-discussed acceleration techniques. The entire FPGA design flow involves multiple stages:

- *High-level synthesis*: Converting the C code with synthesis directives (e.g. loop unrolling or pipelining) into low level Hardware Description Language (HDL) code.
- *Synthesis*: The process of transforming HDL code into a netlist, a graph that defines the connection of all logic gates and registers.
- *Place-and-route (P&R)*: Solving a set of optimization problems in order to fit the netlist to a particular FPGA. The outcome of P&R is a bitstream that can be uploaded onto the FPGA.
- *Functional verification* of the circuit. For optimization-based control applications, the FPGA circuit has to be verified in the loop with a plant model.

We automate the above design flow with Protoip [16].

III. Formulating the design problem as an optimization problem

A. Design objectives and constraints

1) Performance: Quantifying controller performance is not a trivial task: depending on the nature of the dynamic system and design requirements several performance criteria might be considered. Fortunately, optimization-based controllers often perform explicit performance optimization and hence can be evaluated using objective measures. Depending on the application different control objectives might be selected, e.g. energy consumption or settling time. It should be emphasized that the solution of a single optimal control problem cannot serve as a performance indicator, since the ultimate goal is achieving desired closed-loop behaviour [17]. Instead, a closed-loop cost function should be calculated based on a closed-loop simulation.

Considering the closed-loop cost function as an objective within an optimization framework, it is important to take into consideration the continuity and monotonicity properties of the cost function. According to [18], even for a constrained LQR formulation, which is arguably the simplest MPC setup, neither continuity nor monotonicity with respect to horizon length and sampling time can be guaranteed in general. This significantly limits the range of optimization tools that can be used for MPC design optimization.

2) Computation time: In relation to CPU implementations, where several algorithms might share the same hardware platform, algorithm execution time becomes both a design objective and design constraint. On the one hand, minimizing algorithm execution time keeps processor load low and hence enables sharing processor time with other algorithms. On the other hand there is a fundamental constraint for MPC design problems: in order to implement the controller in real-time, the optimization algorithm execution time has to be smaller than the sampling time of the system. There are certain exceptions to this rule [19], which are not considered in this work.

In contrast, for FPGAs, where a circuit is synthesised for one particular algorithm, minimizing computation time would not give any benefits, since the logic cannot be reused by other algorithms. Moreover, for certain cases it could be worthwhile to increase computation time by decreasing the circuit size and hence saving resources. However, computation time can only be increased up to the sampling time, which can be formalized as a design constraint.

For a given algorithm the CPU implementation execution time might not be fully predictable, because of a complex memory hierarchy, sharing resources with other routines and having an operating system on the underlying level. For FPGA circuits, execution time (in terms of clock cycles) can often be determined based on the architecture and hence efficiently predicted before circuit synthesis.

3) Computational logic usage and energy consumption: An FPGA designer often aims to minimize the amount of silicon that is required for implementing a particular control algorithm to get a size- and cost-efficient solution. As discussed in Section II, a modern FPGA has the following resources: flip-flops, lookup tables, block RAMs and DSPs. We measure the silicon (or computational logic) usage as

$$R_{FPGA} = \sqrt{R_{FF}^2 + R_{LUT}^2 + R_{DSP}^2 + R_{BRAM}^2}, \qquad (1)$$

where R_{FF}, R_{LUT}, R_{DSP} and R_{BRAM} denote relative utilization of each resource type. The Euclidean norm is a compromise between the L^1 and L^∞ norms, where the former would not take into account a possible imbalance between different types of resources and the latter would penalize only the most heavily used resource.

There is seldom a linear correspondence between circuit size and energy consumption. For instance, in some cases it could be energy-efficient to create a large circuit by parallelizing all computations in order to quickly perform all calculations and switch to idle mode [17]. In such cases, energy consumption may be considered as a separate design objective, which is particularly important for energy-autonomous embedded platforms.

The above discussion is also valid for software (i.e. CPU-only) implementations. Similarly to FPGA implementations, CPU-only realizations might be constrained by the amount of available external RAM memory or electrical energy.

B. Design parameters

1) Horizon length and sampling time: Horizon length and sampling time are fundamental design variables that have a crucial impact both on closed-loop performance of a system and computational hardware requirements. These two parameters, being tightly coupled with each other, define the quantity of predicted information in a predictive controller. Horizon length defines the "vision distance", while sampling rate sets the "quality of the picture" [18]. Sampling frequency also defines the response delay of the controller. See [2] for an overview of techniques for manual tuning of these parameters.

2) Problem formulation: condensed vs non-condensed: Eliminating the states from decision variables by expressing them via the current state and input sequence leads to a compact condensed formulation [20], which results in a worst-case cubic scaling of the number of floating point operations in horizon length for the primal-dual interior point algorithm iteration. A sparse formulation implies the opposite approach: keeping the dynamics in constraints and treating both inputs and states as decision variables. Although the problem size becomes larger, exploiting the sparsity pattern allows linear scaling of computational effort in horizon length. The condensed and non-condensed formulations can be considered as two extreme points of a *sparsity level* design variable. Controlling the level of sparsity can be achieved by dividing the prediction horizon into sub-intervals and performing partial condensing. This provides the possibility of adjusting the block size of linear algebra problems in order to find the optimal level of sparsity in terms software and hardware resources utilization [21].

3) Optimization algorithm parameters: Optimization algorithms that solve optimal control problems often have many tuning parameters. The first fundamental design choice is the algorithm type [17]:

- first vs second order methods
- interior point vs active set algorithms
- gradient-based vs derivative-free approaches

In addition to making high level design decisions, it is important to tune low level parameters, which might include:

- number of iterations / termination criterion
- barrier parameter update strategy for interior-point algorithms
- globalization strategy (line search vs trust region)

The above parameters must be tuned with respect to closed-loop performance, which is not necessary correlated with a single optimization problem's optimality conditions [17]. For example, a sub-optimal controller with a longer prediction horizon might perform better than an optimal controller with a shorter horizon. More details on sub-optimal MPC can be found in [22].

4) Number representation: There are usually two types of choices that have to be made in relation to number representation:

- The conceptual choice of data representation type: e.g. floating point or fixed point.
- The numbers of bits to be allocated for different parts of a number, e.g. mantissa and exponent in floating point arithmetic.

5) Data-level parallelism and pipelining: Data-level parallelism and pipelining were discussed in Section II. The main algorithmic choices in relation to these techniques are:

- The number of parallel processors.
- The number of pipeline stages, i.e. *pipeline depth*.

It should be emphasized that parallelism and pipelining affect algorithm execution time and resource usage, which may or may not have an impact on the closed-loop performance.

C. The resulting optimization problem

The design parameters considered in Section III-B can be classified in two categories: software parameters (e.g. prediction horizon length) and hardware parameters (e.g. number representation). Conventional approaches propose sequential design: initially the algorithm is designed at a high level of abstraction without regard to the intended hardware platform and, following this, the algorithm is implemented on a hardware platform by selecting appropriate hardware parameters. This decoupled approach usually leads to inefficient resource utilization [17]. In contrast, the co-design approach implies simultaneous design of both software and hardware components in order to achieve the best possible performance for a given set of available resources. However, improvement of the closed-loop performance cannot be considered as the only design objective. As can be seen from Section III-A there are often several contradicting design objectives, which might include performance and computational hardware resource usage. Instead of looking for one optimal design (which often does not exist due to conflicts between objectives), engineers might make a decision based on the whole series of Pareto optimal designs, i.e. designs that cannot be improved in terms of one objective without worsening at least one of the other objectives.

The problem of investigating design trade-offs is usually formalized as a multi-objective optimization (MOO) problem. The main bottleneck that prevents efficient solution of MOO design problems in relation to MPC are the properties of the objective functions:

- Long function evaluation time. Evaluation of the design objective functions requires time-consuming simulations.
- Absence of derivative information. There are no accurate analytical expressions for the derivates of the design objectives.
- Mixed domain. Design variables can be both discrete (e.g. number of bits for data representation) and continuous (e.g. sampling rate).
- Noisiness. For example, the same HLS code may result in different resource usage values depending on a vendor's software version or other factors that are not taken into account by conventional models.

The next section will review existing algorithms for solving multi-objective optimization problems with focus on BiMADs, a bi-objective version of a mesh adaptive direct search algorithm.

IV. Derivative-free multi-objective optimization

A. Problem statement

We consider the following multi-objective optimization problem:

$$\min_{p \in S} \quad f(p) := \left(f^{(1)}(p), \dots, f^{(l)}(p) \right) \tag{2}$$

where S is q-dimensional decision space. Since the objectives of MOO are often contradicting, there is no single solution to the problem. Instead, a set of *Pareto optimal* solutions can be obtained.

Definition IV.1. *A feasible solution $p^* \in S$ is Pareto optimal if there does not exist another feasible solution $p \in S$ such that $f_i(p) \leq f_i(p^*)$ for all $i \in \{1, \dots, l\}$ and $f_i(p) < f_i(p^*)$ for at least one index $j \in \{1, \dots, l\}$. The Pareto frontier is the set of all Pareto optimal points.*

Definition IV.2. *A point $y' = f(p')$ (strongly) dominates $y'' = f(p'')$ iff $\forall i \in \{1, \dots, l\} : y'_i \leq y''_i$ and $y' \neq y''$. The shorthand notation for this is $y' \prec y''$. Analogously, for weak dominance, $y' \preceq y''$ means $\forall i \in \{1, \dots, l\} : y'_i \leq y''_i$.*

$P(U)$ is the set of non-dominated points for a given set of evaluated points U, i.e. the current approximation of the Pareto frontier. The quality of a Pareto frontier approximation can be assessed by means of the *hypervolume*.

Definition IV.3. *For a given reference point y_{ref} and Pareto frontier approximation P, the hypervolume is defined as a set of points in the objective space $\{y \preceq y^{ref} \in \mathbb{R}^l | \exists y' \in P : y' \preceq y\}$.*

The quality of the approximation of P is defined to be the Lebesgue measure of the hypervolume $L(P, y_{ref})$, i.e. hypervolume space.

B. Review of derivative-free multi-objective optimization algorithms

1) Derivative-free single-objective optimization: Deterministic algorithms for single-objective derivative-free optimization can be classified into [23]:

- *Trust-region interpolation algorithms*. Trust-region algorithms propose building a local approximation of the objective function based on evaluated samples. Based on this approximation, the function is minimized inside the trust region.
- *Line search algorithms* for derivative-free optimization are conceptually similar to their derivative-based counterparts: they perform search along a particular direction. However, for derivative-free algorithms, the search direction is calculated without gradient information.

- *Direct Search Methods* (DSMs) do not attempt to approximate derivatives either explicitly or implicitly. Instead, optimization is based on evaluation of a finite set of points around the current solution guess, so that a point with smaller objective function value can be found.

Among the considered classes of algorithms, only direct search methods of directional type have been extensively studied in relation to multi-objective optimization [24], hence rest of the section will be focused on DSMs.

Each iteration of a DSM is split into two parts: a *search step* and *poll step*. The latter step must be rigidly defined to guarantee convergence, while the former is more flexible and can be tuned to improve numerical performance. Assuming there is a given feasible solution guess, the general structure of a DSM of directional type can be described as follows [23]:

- **Search step**. Evaluate the objective function at a finite set of points. If any of these points provide a better objective value compared to the current guess, declare the iteration as successful and skip the poll step.
- **Poll step**. Perform a local search around the current best point by evaluating a set of poll points, which are defined by poll directions and a step size. Depending on the result of this search, declare the iteration as successful or unsuccessful.
- **Mesh parameter update**. Reduce step size for unsuccessful iterations, increase or maintain step size for successful ones.

Practical examples of DSM algorithms of directional type are the coordinate-search method [23] and the Mesh Adaptive Direct Search (MADS) algorithm [25].

2) Derivative-free multi-objective optimization: There are two fundamentally different ways for tackling multi-objective optimization problems:

- *Direct* algorithms for multi-objective optimization attempt to approximate the Pareto frontier directly.
- *Scalarization-based* approaches propose converting the multi-objective problem into a sequence of single-objective problems.

The Direct Multisearch (DMS) algorithm [26], which belongs to the class of direct multi-objective optimization algorithms, is a generalization of DSM algorithms for single-objective optimization. The main components of a DMS are the same as DSM as discussed in the previous subsection: the search step, the poll step and mesh update. The algorithm keeps a list of non-dominated points, which represents the current Pareto frontier approximation. The local poll search is performed around several non-dominated points. Successfulness of an iteration is decided based on the changes in the Pareto frontier approximation. The search step, similarly to single-objective DSM algorithms, is flexible and is not required for convergence [24]. However, it can help to improve the distribution of the points along the Pareto frontier, although this is not a systematic way of ensuring uniformity of the frontier.

A classical example of scalarization-based algorithms is the weighted-sum method. The method scalarizes the objective vector into a single objective by taking an affine combination of the objectives. Tuning the weights of the scalarized function allows movement along the Pareto frontier, but in a non-systematic way. As a result, some regions become over-represented and others may suffer from lack of information. In addition, this approach cannot deal with non-convexities and discontinuities in the Pareto frontier and hence loses some Pareto optimal solutions.

The BiMADS algorithm [15] performs scalarization in a different way. For a problem with two objectives the scalarized function is

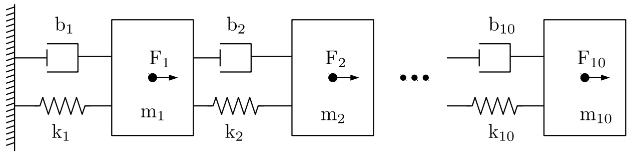


Fig. 1. Mass-spring-damper system.

$$\phi_r(p) := \begin{cases} -\prod_{i=1}^{2}(r_i - f^{(i)}(p))^2 & \text{if } f(p) \preceq r, \\ \sum_{i=1}^{2}(\max\{0, (f^{(i)}(p) - r_i)\})^2 & \text{otherwise,} \end{cases} \tag{3}$$

where r is a reference point in the objective space. The problem of minimizing $\phi_r(\cdot)$ is solved by the MADS [25] algorithm. The formulation (3) with appropriate selection of a reference point allows the recovery of all Pareto optimal solutions, which is not the case for the weighted-sum reformulation. For achieving good distribution of non-dominated solutions, the reference point must be selected in a particular way. The BiMADS algorithm proposes a formal method for identifying the biggest gap in the Pareto frontier approximation and presents a procedure for selecting an appropriate reference point. See [15] for details.

The BiMADS algorithm will be used for solving the multi-objective co-design problem. The main advantages are:

- Both BiMADS and the underlying single-objective MADS solver are supplied with a mathematical proof of convergence [15], [25], which is not the case for other considered algorithms.
- BiMADS has proven to be efficient for solving real-world engineering problems [27].
- There is a free software application that implements the algorithm, namely NOMAD [28]
- NOMAD provides various interfaces, including a Matlab front-end, which simplifies integration with Protoip [16].

Although there is ongoing work on extending BiMADS for handling three and more objectives [29], at the moment the algorithm is limited to bi-objective problems, which is the main drawback. Potentially, handling more than two objectives can be achieved by aggregating the problem into a bi-objective formulation. However, aggregation-based approaches are not considered in this work.

The next section will present an application of BiMADS to predictive controller design problems. Although only a subset of design parameters, objectives and constraints described in Section III will be considered, the proposed approach can be extended to a wider range of design problems, which includes handling parameters, objectives and constraints not described in this paper.

V. Case studies

A. Fast gradient-based controller for a mass-spring-damper system: CPU-only implementation

The system under control represents a chain of ten masses that are connected via springs and dampers (Figure 1). The first mass is also connected to a fixed wall. Each mass can be actuated with an input force that has input and output limits. It is assumed there is no gravitational force. The system can be modelled with a continuous-time linear state space model:

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{4}$$

where $A_c \in \mathbb{R}^{n \times n}$, $B_c \in \mathbb{R}^{n \times m}$ are continuous-time state and input matrices. For a system of ten masses $n = 20$ and $m = 10$.

The following optimal control formulation is considered:

$$\begin{aligned}\underset{u,x}{\text{minimize}} \quad & \int_0^T \left(\frac{1}{2}x^T(t)Q_c x(t) + \frac{1}{2}u^T(t)R_c u(t)\right. \\ & \left. + x^T(t)W_c u(t)\right) dt + \frac{1}{2}x^T(T)P_c x(T) & \text{(5a)} \\ \text{subject to } & x(0) = \hat{x} & \text{(5b)} \\ & \dot{x}(t) = A_c x(t) + B_c u(t),\ \forall t \in [0,T] & \text{(5c)} \\ & u_{min} \le u(t) \le u_{max},\ \forall t \in [0,T] & \text{(5d)}\end{aligned}$$

where $Q_c \in \mathbb{S}^n_+$, $R_c \in \mathbb{S}^m_{++}$, $W_c \in \mathbb{R}^{n \times m}$ and $P_d \in \mathbb{S}^n_{++}$ are state, input, cross and terminal penalty matrices accordingly. $\mathbb{S}^n_{++}(\mathbb{S}^n_+)$ denotes a set of positive (semi-)definite matrices.

In this experiment the following prediction matrices were used:

$$R_c = I_{m \times m} \otimes [0.0001], \quad Q_c = I_{n \times n} \otimes \begin{bmatrix} 1 & 0 \\ 0 & q_{speed} \end{bmatrix}, \\ W_c = 0_{m \times n}, \quad P_c = Q_c, \quad (6)$$

where I and $0_{m \times n}$ denote identity and zeros matrices accordingly. Tuning q_{speed}, which is a design parameter, allows for changing the ratio between penalising positions and velocities for all the masses. A small penalty is applied to inputs to ensure numerical stability, while allowing aggressive controller response. Terminal penalty is selected a priory: $P_c = Q_c$, although the test setup can be improved further by considering P_c as another design parameter.

For the purpose of digital control, the continuous-time state-space model (4) and optimal control problem (5) are discretized, assuming a zero-order hold to give

$$\begin{aligned}\underset{u_0 \ldots u_{N-1}, x_0 \ldots x_N}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \left(\frac{1}{2}x_k^T Q_d x_k + \frac{1}{2}u_k^T R_d u_k\right. \\ & \left. + x_k^T W_d u_k\right) + \frac{1}{2}x_N^T P_d x_N & \text{(7a)} \\ \text{subject to } & x_0 = \hat{x} & \text{(7b)} \\ & x_{k+1} = A_d x_k + B_d u_k,\ \forall k \in \{0, \ldots, N-1\} & \text{(7c)} \\ & u_{min} \le u_k \le u_{max},\ \forall k \in \{0, \ldots, N-1\} & \text{(7d)}\end{aligned}$$

Note that the discrete-time penalty matrices Q_d, R_d, W_d and P_d are model-dependent [18]. The optimal control problem (7) can be transformed into a condensed quadratic programming problem by eliminating the states, which leads to the following formulation

$$\begin{aligned}\text{minimize} \quad & \frac{1}{2}\theta^T H \theta + \theta^T h & \text{(8a)} \\ \text{subject to} \quad & \theta_{min} \le \theta \le \theta_{max} & \text{(8b)}\end{aligned}$$

Note that the gradient term h depends on the current state, while the Hessian H is fixed and hence can be precalculated offline. More details on condensed and sparse formulations for predictive control can be found in [20]. Since (8b) has the form of box constraints, calculating projection on the feasible set becomes computationally cheap. This facilitates using Nesterov's projected gradient algorithm [30], also known as the Fast Gradient Method (FGM). The method proposes moving in the anti-gradient direction and performing projection $P(\cdot)$ on the feasible set after each iteration (Algorithm 1). The extra momentum step with a parameter β achieves an optimal convergence rate. The constant step scheme [30] implies $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$, where L is the largest eigenvalue of the Hessian H and μ is the convexity parameter, which is equal to minimum eigenvalue of the Hessian.

The algorithm was implemented on an ARM Cortex A9 processor of the Xilinx Zynq-7000 XC7Z020 system-on-a-chip employing one processing core and using single precision arithmetic for data representation. Using the Protoip toolbox allowed for fast verification of the controller in the loop with the plant model as shown in Figure 2. For the considered setup, the computational delay of the MPC controller is neglected.

Algorithm 1 Projected fast gradient algorithm for constrained optimization with constant step size.

```
Initial guess: $\theta_0$
$v_0 = \theta_0$
for $i = 0$ to $N_{FGM}$ do
    $\theta_{i+1} = (I - (1/L)H)\nu_i - (1/L)h$        ▷ anti-gradient step
    $z_{i+1} = P(\theta_{i+1})$        ▷ projection on the feasible set
    $v_{i+1} = (1+\beta)z_{i+1} - \beta z_i$        ▷ extra-momentum step
end for
```

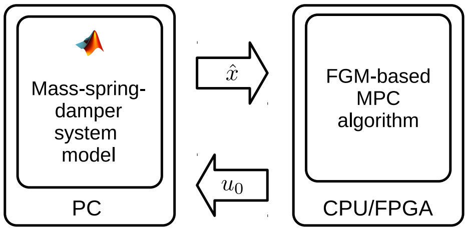


Fig. 2. CPU in the loop test setup.

The problem of interest is automatic design of a fast gradient-based controller. The following design objectives are considered:

- *Controller performance* is judged based on settling time. In this experiment, settling time is defined as the time elapsed from the beginning of closed-loop simulation to the time at which $\|x(t)\|_2 \le \epsilon$, where $\epsilon = 0.01$. Several simulations with different initial conditions were performed in order to calculate performance measure as a sum of settling times for different initial conditions; see [31] for details. Since the performance criterion and MPC objective are different, it is essential to tune the prediction matrices in order to achieve the desired performance.
- *Algorithm computational time*. As discussed in Section III-A2, computational time is the main measure of algorithm complexity for CPU implementations.

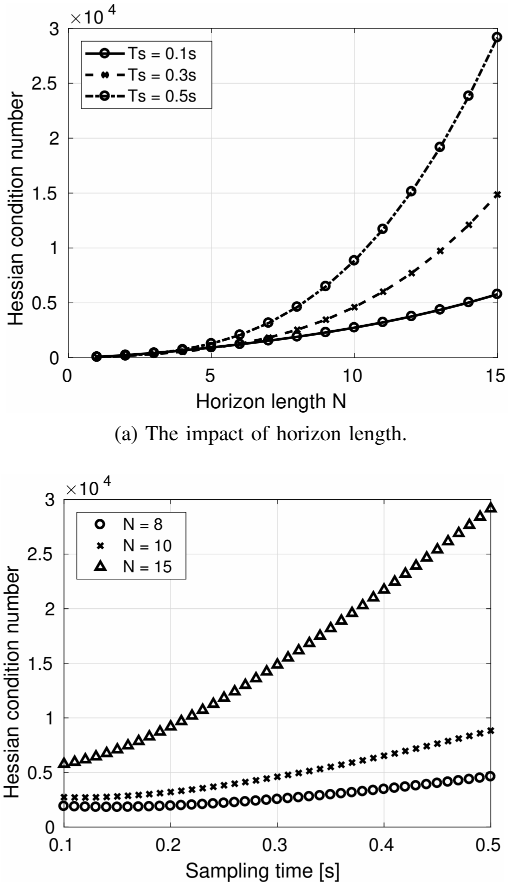
Design constraints:

- *Algorithm computational time*, in addition to being a design objective, appears in a constraint function: in order to implement the controller in real-time the algorithm execution time has to be smaller than the sampling time of the system (Section III-A2).
- *Stability constraint* captures whether the controller was able to stabilize the system. Unstable response might happen due to short horizon, numerical errors or other reasons.

Note that the former constraint is *quantifiable* while the latter is *non-quantifiable*. A quantifiable constraint is a constraint for which the degree of feasibility and violation can be quantified [32]. In this work, non-quantifiable constraints are handled with the *extreme barrier* approach, which implies setting the objective to infinity for all infeasible points and therefore not allowing infeasible iterations. For quantifiable constraints the *progressive barrier* approach is adopted. Progressive barrier constraint handling allows exploiting knowledge of the violation degree by accepting infeasible iterations. Both extreme and progressive barrier approaches are implemented in NOMAD. More details can be found in [32], [28].

The design parameters are the following:

- Horizon length, N in (7); bounds: $1 \le N \le 12$.



(a) The impact of horizon length.

(b) The impact of sampling time.

Fig. 3. The impact of horizon length and sampling time on Hessian condition number.

- Sampling time, T_s; bounds: $0.02 \leq T_s \leq 0.5$.
- Number of fast gradient algorithm iterations, N_{FGM} in Algorithm 1; bounds: $20 \leq N_{FGM} \leq 200$.
- State penalty matrix, particularly q_{speed} parameter in (6); bounds: $0.2 \leq q_{speed} \leq 5$.

The above parameters are tightly coupled with each other. For example, consider Figure 3 that illustrates the impact of horizon length and sampling time on the Hessian condition number.

It can be observed that both parameters have a significant impact on the condition number, which in turn affects the convergence rate of the fast gradient algorithm [30]. As a result, the number of fast gradient algorithm iterations N_{FGM} required for convergence will also change. However, N_{FGM} must be selected with respect to closed-loop performance, rather than open-loop optimality conditions, which complicates the tuning process even more.

The above design problem was solved using the BiMADS algorithm. The results are compared to LHS, which is a statistical sampling method commonly used for design exploration and for design-of-experiments in particular. Compared to 'simple' random sampling, LHS achieves more evenly distributed sampling points across all possible values [4]. For both experiments the number of evaluations was restricted to 200. Design evaluation involves compiling the source code and performing processor-in-the-loop tests, which takes 1-4 minutes for the considered setup, depending on a design complexity and the sampling time. Observe that full design exploration is not a viable approach: even using a coarse grid with ten points for the continuous variables, full exploration will require 217200 evaluations. As can be seen from Figure 4, the Pareto frontier returned by BiMADS dominates the front identified by LHS, i.e. any design sampled by LHS is dominated by at least one design identified by BiMADS (see Definition IV.2). Moreover, according to the hypervolume profile (Figure 5), BiMADS provides a satisfactory approximation of the Pareto frontier on the early stages, which opens the possibility of early termination, depending on the time and/or simulation resource availability.

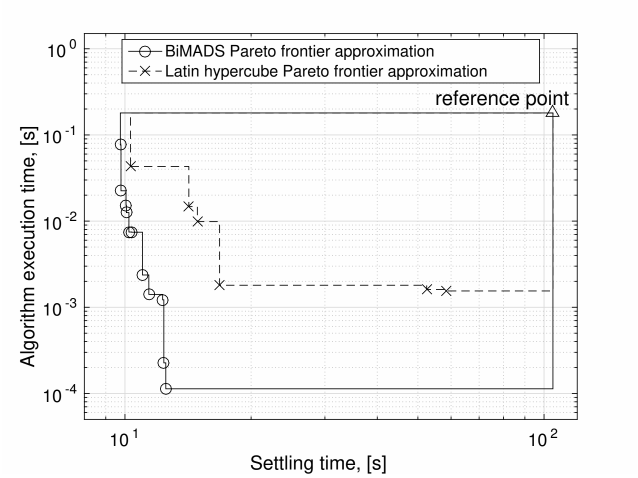


Fig. 4. Pareto frontier approximation for CPU implementations of the FGM: BiMADS vs LHS.

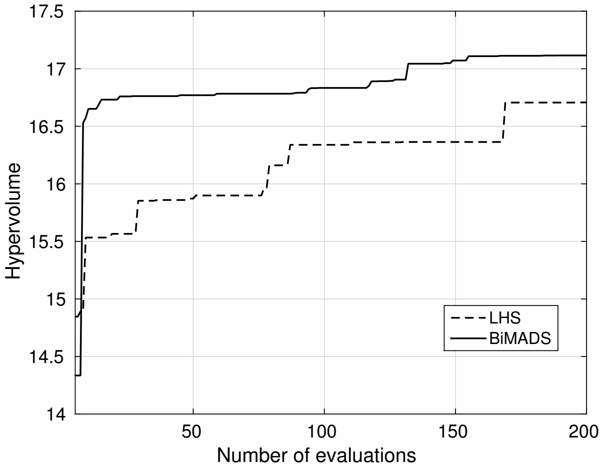


Fig. 5. Hypervolume profiles for CPU implementations of the FGM: BiMADS vs LHS.

Given the Pareto frontier (Figure 4) a designer will be able to select an implementation based on the available processing time, e.g.

- For the computational budget of 100 ms, LHS-based exploration achieves a settling time of 10.31 s, while optimization-based design allows settling of the plant in 9.75 s.
- Tightening the computational budget to 2 ms leads to sampling times of 16.82 s and 11.40 s for LHS- and optimization-based approaches accordingly.

B. Fast gradient-based controller for a mass-spring-damper system - FPGA implementation

This case study considers implementation of the Algorithm 1 with fixed point arithmetic on the FPGA logic of Xilinx Zynq-7000 XC7Z020 SoC. The testing setup is similar to that of Section V-A, both in terms of the optimal control problem formulation (5) and the plant model (Figure 1). The algorithm was implemented with the Vivado HLS FPGA synthesis tool using Protoip for automatic deployment and verification in the loop with the plant model (Figure 2). As can be seen from Algorithm 1, FGM relies only on addition and multiplication operators, while all divisions can be precalculated offline. The following techniques were used to accelerate vector-vector and matrix-vector operations (lines 4-7):

- *Loop pipelining* [33]. Data pipelining, as a general acceleration technique, was discussed in Section II. In relation to loops, pipelining implies overlapping iterations, i.e. starting a new iteration before finishing the previous. For the considered implementation, the initiation interval was set to one clock cycle and was not treated as a design parameter.
- *Loop flattening* [33] is transforming nested loops into a single loop with multiple counters. Flattening allows efficient pipelining of nested loops. This technique was applied to matrix-vector multiplication (line 4), where loop nests arise when iterating over matrix rows and columns.

Fixed point arithmetic often introduces overflow and round-off errors. The former issue can be addressed by precalculating the upper bound on the largest absolute value of algorithm iterates using interval arithmetic. Regarding round-off errors, the number of fraction bits has to be sufficiently large to maintain numerical stability of an iterative algorithm. A procedure for precalculating the minimum number of integer and fraction bits for fixed point implementations of a fast-gradient algorithm is presented in [34]. However, [34] does not attempt to formalize the problem of selecting the number of fraction bits for optimal resource usage vs performance trade-offs.

The following objectives are considered in this design optimization:

- *Controller performance* is measured similarly to the previous case study (Section V-A) with $\epsilon = 0.02$.
- *FPGA logic usage*. As discussed in Section III-A FPGA designers often aim to minimize the amount of logic used for a particular algorithm. Logic usage is measured as the Euclidean norm of relative utilization of each resource type, see (1) in Section III-A.

Design constraints:

- *Algorithm execution time*. Similarly to the previous case study, in order to implement the controller in real-time, the algorithm execution time has to be smaller than the sampling time of the system. This constraint is treated with the progressive barrier approach. Note that for FPGA setup (in contrast to CPU), computational time does not appear as an objective. This is explained by the fact that FPGA logic is synthesized for a particular algorithm and cannot be reused for other applications.
- *Objective function convexity constraint*. Due to assumptions on the weight matrices in formulation (5), the Hessian of the objective function (8a) is positive definite. However, a fixed point representation of the true Hessian may be non-convex because of truncation errors, which might affect convergence of the fast-gradient algorithm. To avoid non-convex formulations, a positivity constraint on the smallest eigenvalue of the fixed point representation of the Hessian must be set. Although this is a quantifiable constraint, which potentially can be treated with a progressive barrier approach, we will use the extreme barrier method that rejects all infeasible iterations. Since identifying Hessian convexity is significantly faster compared to the full design evaluation, which involves circuits synthesis and closed loop simulation, rejecting infeasible iterations allows saving design time.
- *Stability constraint*, similarly to the previous case study.

The following design parameters are considered:

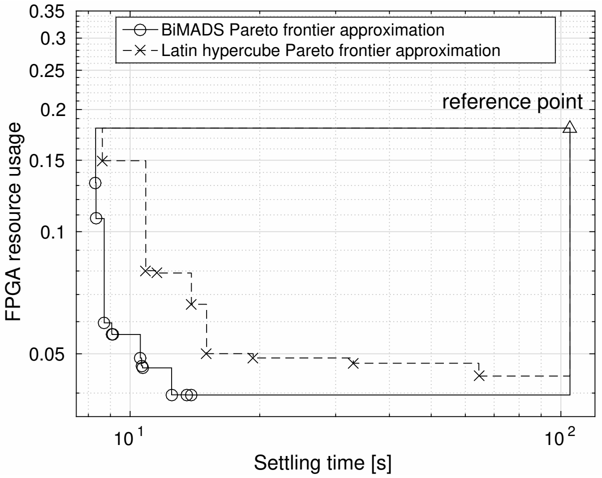


Fig. 6. Pareto frontier approximation for FPGA implementations of the FGM: BiMADS vs LHS.

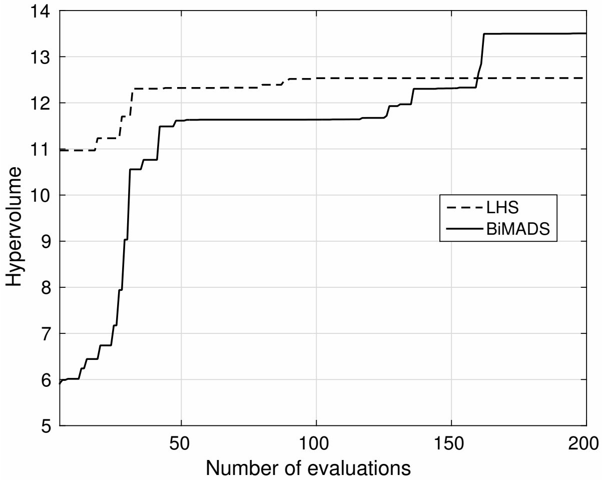


Fig. 7. Hypervolume profiles for FPGA implementations of the FGM: BiMADS vs LHS.

- Horizon length, N in (7); bounds: $1 \leq N \leq 12$.
- Sampling time, T_s; bounds: $0.02 \leq T_s \leq 0.5$.
- Number of fast gradient algorithm iterations, N_{FGM} in Algorithm 1; bounds: $20 \leq N_{FGM} \leq 200$.
- State penalty matrix, particularly q_{speed} parameter in (6); bounds: $0.2 \leq q_{speed} \leq 5$.
- Number of fraction bits for fixed point number representation, N_{frac}; bounds: $5 \leq N_{frac} \leq 25$.

Similarly to the previous case study, the multi-objective optimization problem with the above design objectives and constraints was solved using BiMADS algorithm and results were compared to Latin hypercube sampling allowing 200 evaluations for both algorithms. In contrast to software compilation, FPGA circuit synthesis is a time-consuming process, which leads to the design evaluation time in the range of 20-35 minutes. For this case study full design exploration would require 4561200 evaluations, assuming a grid of ten points for the continuous parameters.

It can be observed from Figure 6 that BiMADS outperforms LHS, i.e. the BiMADS Pareto frontier dominates frontier identified by LHS. However, unlike with the previous case study, LHS outperforms BiMADS in the early stages of design exploration, which can be visualized with the hypervolume profile in Figure 7. This might happen due to bad selection of initial guesses for BiMADS. Moreover, LHS, being a statistical method, might occasionally identify Pareto optimal designs faster than deterministic algorithms. It can be observed from Figure 7 that, after achieving a certain hypervolume space in the beginning of exploration process, LHS does not improve the Pareto frontier approximation significantly further. In contrast, BiMADS, having a poor initial guess, improves the solution and outperforms LHS when reaching an evaluation limit.

Given the Pareto frontier (Figure 6) a designer will be able to select a particular implementation based on the available FPGA resources (R_{FPGA}). For example:

- For $R_{FPGA} = 0.1$, LHS-based exploration achieves a settling time of 10.87 s, while optimization-based design allows settling of the plant in 8.71 s.
- Tightening the resource limit to $R_{FPGA} = 0.05$ leads to sampling times of 19.31 s and 10.56 s for LHS- and optimization-based approaches accordingly.

VI. Conclusions and Future Work

This paper proposed automating predictive control design by employing systematic optimization. It was shown that the bi-objective optimization-based design outperforms a statistical exploration technique and allows systematic investigation of resource-performance trade offs. Two case studies considered CPU and FPGA implementations accordingly, although the proposed approach can be applied to a broader range of computing architectures, including heterogeneous computers.

In this work parameter tuning problem was solved for a *predefined* algorithm, namely FGM. Further work might be focused on formalizing the problem of algorithm selection, e.g. FGM vs splitting algorithms or first-order vs second order algorithms. Another direction for further research is extending the proposed approach to the problems with more than two contradicting objectives in order to be able to capture a wider range of real-world problems.

References

[1] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems*, 36(6):45–64, Dec 2016.

[2] Jorge L. Garriga and Masoud Soroush. Model predictive control tuning methods: A review. *Industrial & Engineering Chemistry Research*, 49(8):3505–3515, 2010.

[3] Jack P. C. Kleijnen. *Design and Analysis of Monte Carlo Experiments*, pages 529–547. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[4] Felipe AC Viana. Things you wanted to know about the latin hypercube design and were afraid to ask. In *10th World Congress on Structural and Multidisciplinary Optimization, Orlando, Florida, USA (cf. p. 69)*, 2013.

[5] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, Jan 2016.

[6] Mattia Vallerio, Jan Van Impe, and Filip Logist. Tuning of NMPC controllers via multi-objective optimisation. *Computers & Chemical Engineering*, 61:38–50, 02 2014.

[7] A. Messac, A. Ismail-Yahaya, and C.A. Mattson. The normalized normal constraint method for generating the pareto frontier. *Structural and Multidisciplinary Optimization*, 25(2):86–98, Jul 2003.

[8] Indraneel Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[9] D. Telen, B. Houska, M. Vallerio, F. Logist, and J. Van Impe. A study of integrated experiment design for NMPC applied to the droop model. *Chemical Engineering Science*, 160:370 – 383, 2017.

[10] J. Sanchis, M. Martínez, X. Blasco, and J. V. Salcedo. A new perspective on multiobjective optimization by enhanced normalized normal constraint method. *Structural and Multidisciplinary Optimization*, 36(5):537–546, Nov 2008.

[11] Federico Lozano Santamaría and Jorge M. Gómez. An algorithm for tuning NMPC controllers with application to chemical processes. *Industrial & Engineering Chemistry Research*, 55(34):9215–9228, 2016.

[12] V. M. Zavala. Real-time resolution of conflicting objectives in building energy management: An utopia-tracking approach. In *Proc. 5th National Conference of IBPSA-USA*, 2012.

[13] A.S. Yamashita, A.C. Zanin, and D. Odloak. Tuning of model predictive control with multi-objective optimization. *Brazilian Journal of Chemical Engineering*, 33(2):333–346, 2016.

[14] André Shigueo Yamashita, Antonio Carlos Zanin, and Darci Odloak. Tuning the model predictive control of a crude distillation unit. *ISA Transactions*, 60:178 – 190, 2016.

[15] Charles Audet, Gilles Savard, and Walid Zghal. Multiobjective optimization through a series of single-objective formulations. *SIAM Journal on Optimization*, 19(1):188–210, 2008.

[16] B. Khusainov, E. C. Kerrigan, A. Suardi, and G. A. Constantinides. Nonlinear predictive control on a heterogeneous computing platform. In *IFAC World Congress 2017*, July 2017.

[17] E. C. Kerrigan. Co-design of hardware and algorithms for real-time optimization. In *Control Conference (ECC), 2014 European*, pages 2484–2489, June 2014.

[18] Vincent Bachtiar, Eric C. Kerrigan, William H. Moase, and Chris Manzie. Continuity and monotonicity of the MPC value function with respect to sampling time and prediction horizon. *Automatica*, 63:330 – 337, 2016.

[19] Dominic Buchstaller, Eric C. Kerrigan, and George A. Constantinides. Sampling and controlling faster than the computational delay. *IFAC Proceedings Volumes*, 44(1):7523 – 7528, 2011. 18th IFAC World Congress.

[20] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides. A condensed and sparse QP formulation for predictive control. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 5217–5222, Dec 2011.

[21] Daniel Axehill. Controlling the level of sparsity in MPC. *Systems and Control Letters*, 76(0):1 – 7, 2015.

[22] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings. Suboptimal model predictive control (feasibility implies stability). *IEEE Transactions on Automatic Control*, 44(3):648–654, Mar 1999.

[23] A. Conn, K. Scheinberg, and L. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, 2009.

[24] A.L. Custodio, M. Emmerich, and J.F.A. Madeira. Recent developments in derivative-free multiobjective optimisation. *Computational Technology Reviews*, 5:1 – 30, 2012.

[25] Charles Audet and Jr. J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[26] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and L. N. Vicente. Direct multisearch for multiobjective optimization. *SIAM Journal on Optimization*, 21(3):1109–1140, 2011.

[27] Aïmen E. Gheribi, Jean-Philippe Harvey, Eve Bélisle, Christian Robelin, Patrice Chartrand, Arthur D. Pelton, Christopher W. Bale, and Sébastien Le Digabel. Use of a biobjective direct search algorithm in the process design of material science applications. *Optimization and Engineering*, 17(1):27–45, 2016.

[28] Sébastien Le Digabel. Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm. *ACM Trans. Math. Softw.*, 37(4):44:1–44:15, February 2011.

[29] Sbastien Le Digabel, Christophe Tribes, and Charles Audet. *NOMAD User Guide*.

[30] Yurii Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

[31] Bulat Khusainov. *Co-design of FPGA Implementations for Model Predictive Control*. PhD thesis, Imperial College London, 2017.

[32] Charles Audet and Jr. J. E. Dennis. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472, 2009.

[33] Xilinx. *Vivado Design Suite User Guide. High-Level Synthesis*, May 2014.

[34] J. L. Jerez, S. Richter, P. J. Goulart, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at Megahertz rates. *Automatic Control, IEEE Transactions on*, 59(12):3238–3251, Dec 2014.