Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 912603, 20 pages
doi:10.1155/2012/912603

Research Article

Towards Online Model Predictive Control on a Programmable Logic Controller: Practical Considerations

Bart Huyck,[1,2,3] Hans Joachim Ferreau,[3] Moritz Diehl,[3] Jos De Brabanter,[1,3] Jan F. M. Van Impe,[2] Bart De Moor,[3] and Filip Logist[2]

[1] *Department of Industrial Engineering, KAHO Sint-Lieven, Gebroeders De Smetstraat 1, 9000 Gent, Belgium*
[2] *BioTeC, Department of Chemical Engineering (CIT), KU Leuven, W. de Croylaan 46, 3001 Leuven, Belgium*
[3] *SCD, Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, 3001 Leuven, Belgium*

Correspondence should be addressed to Jan F. M. Van Impe, jan.vanimpe@cit.kuleuven.be

Received 10 July 2012; Revised 1 October 2012; Accepted 1 October 2012

Academic Editor: Wei-Chiang Hong

Given the growing computational power of embedded controllers, the use of model predictive control (MPC) strategies on this type of devices becomes more and more attractive. This paper investigates the use of online MPC, in which at each step, an optimization problem is solved, on both a programmable automation controller (PAC) and a programmable logic controller (PLC). Three different optimization routines to solve the quadratic program were investigated with respect to their applicability on these devices. To this end, an air heating setup was built and selected as a small-scale multi-input single-output system. It turns out that the code generator (CVXGEN) is not suited for the PLC as the required programming language is not available and the programming concept with preallocated memory consumes too much memory. The Hildreth and qpOASES algorithms successfully controlled the setup running on the PLC hardware. Both algorithms perform similarly, although it takes more time to calculate a solution for qpOASES. However, if the problem size increases, it is expected that the high number of required iterations when the constraints are hit will cause the Hildreth algorithm to exceed the necessary time to present a solution. For this small heating problem under test, the Hildreth algorithm is selected as most useful on a PLC.

1. Introduction

Model predictive control (MPC) has become a widely applied control technique in process industry for the control of large-scale installations, which are typically described by large-scale models with relatively slow dynamics [1, 2]. The key element in MPC is to repeatedly solve an optimization problem based on available measurements of the current state of the process. The advantages of MPC over classic PID control are its ability (i) to steer the process in an optimal way while taking proactively desired future behaviour into account, (ii) to tackle multiple inputs and outputs simultaneously, and (iii) to incorporate constraints [3]. In most cases, the MPC controller is hosted by a computer and employed as a supervisory controller, controlling the set-points of controllers closer to the process, for example, PIDs.

In the recent years, interest has grown to exploit *MPC on embedded systems*. Typical applications are, for example, mechatronic systems, which give rise to small-scale models with fast dynamics [4–6]. In these cases the MPC controller is not a supervisory controller anymore but directly steers the actuators and as such also the process itself.

Compared to a standard PC, which nowadays has several cores with speeds in the GHz range and several GBs of memory, embedded controllers are typically implemented on devices with much less computation power and memory. As indicated in Figure 1, a variety of devices exist. *Programmable logic controllers* (PLCs) are often exploited in industry for control tasks because of their robust operation, even in harsh conditions. They typically have a processing power in the order of only MHz and memory in the range from a few kB to several MBs. *Programmable automation controllers* (PACs) bridge the gap as they exhibit a processing power and memory that can go up to that found in PCs, combined with the I/O possibilities of a PLC. Hence, they can be employed to robustly fulfill also other tasks than control (e.g., data logging and maintaining network connections), as they can easily be integrated via standard communication protocols.

To deal with the computational limitations of embedded hardware, two approaches can be taken when implementing MPC on embedded hardware: explicit and online MPC. *Explicit MPC* [7, 8] precomputes all sets of possible solutions to the underlying optimization problem offline and stores them in a look-up tree. When running the controller online, mainly the right working set has to be selected each time. As this approach avoids the online solution of an optimization problem, high-speed algorithms are obtained for small-scale systems (e.g., single-input single-output (SISO) systems), with short prediction horizons. However, for larger systems (e.g., multiple-input multiple-output (MIMO) systems) the number of working sets quickly grows and the time to search the look-up tree becomes prohibitive. In these cases, *online MPC*, which exploits tailored algorithms for the online solution of the optimization problem [9], becomes attractive.

A prerequisite in industry is the use of reliable, easy-to-maintain control hardware, explaining the current dominance of PLCs today. Historically, PLC programming languages have focused on the relay ladder logic (RLL). Although more PC-like programming languages exist in the international standard IEC 61131-3, PLC programmers still more often use ladder languages [10]. As a result, MPC implementations on PLCs are scarce (see, e.g., [11, 12] for explicit MPC).

The aim of the current paper is to illustrate the practical feasibility of *online MPC with constraints on PLCs*. To this end, a test strategy is followed which exploits different types of control hardware with decreasing computation power. First, a PAC (CompactRIO, National Instruments) is used, and afterwards a PLC (S7-319, Siemens) is employed. Different approaches for solving the optimization problem are evaluated on a test setup. This online optimization problem boils down to the solution of a quadratic program (QP). The performance of three QP solvers will be compared when implemented on a PAC and a PLC. The first algorithm is the Hildreth QP algorithm [13], a classic but easy to implement algorithm with a limited number of code lines. It will be compared to qpOASES, a state-of-the-art online

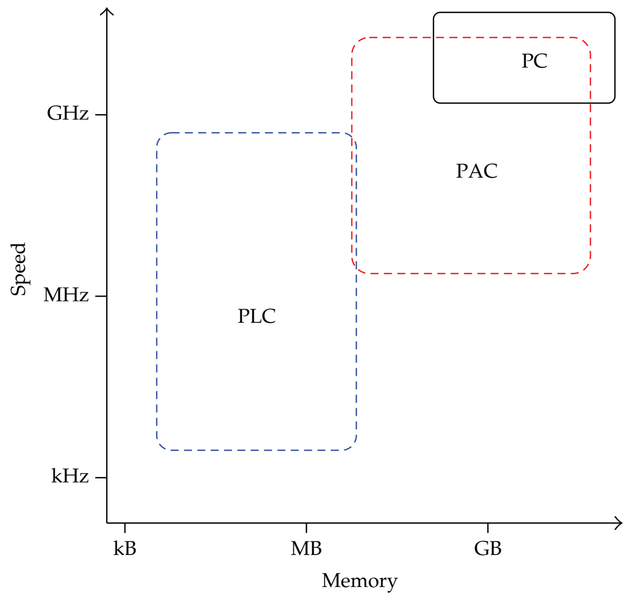


Figure 1: Graphical comparison where to situate PLC, PAC, and PC to each other in view of speed and memory.

active set algorithm [14] that is provided in C/C++. The third QP solver is CVXGEN, a C-code generator for QP-representable convex optimization problems. The practical test setup involves a heating device where fan speed and resistor power can be manipulated independently to control the air temperature. As such, this device can be regarded as a multiple-input single-output system. Although, small scale, it is an interesting application for testing, which has also been used by, for example, [11]. The observations allow the formulation of practical guidelines and warnings for possible pitfalls.

This paper is structured as follows Section 2 briefly repeats the MPC formulation and the steps required to obtain a linear system model. Section 3 describes the practical implementation of the controller and the QP solvers used. A description of the experimental setup can be found in Section 4. Section 5 contains the results for the model identification as well as for the control of the setup using the PAC and the PLC. Finally, the main conclusions are summarized in Section 6.

2. Steps towards Online MPC Implementation

First of all, the model predictive control needs a process model. For the design of the controller, several decisions need to be taken, for example the length of the different horizons, the selection of input, state or output constraints. Once these decisions have been made, the controller can be implemented and tested.

2.1. Modelling the Process

There are several ways to obtain a model for control. Based on the physical and chemical laws underlying the process, a *white-box* model can be deduced. This modeling procedure is often time intensive and, hence, expensive for large and complex systems. As time is money in industry, faster ways are often preferred. Alternatively, available process data can be used to fit a *black-box* model based on generic mathematical relations. There are different black-box modelling techniques, linear [15–17] as well as nonlinear [18, 19]. In this paper, an MPC controller will be used that exploit a linear state space model constructed based on black-box techniques.

2.2. Model Predictive Control Formulation

Linear model predictive control is well known in the literature [3, 20, 21], and the reader is invited to read these works for a detailed description. The basic formulation is briefly given below.

A linear, time-invariant discrete-time system is described by:

$$\begin{aligned} \mathbf{x}(k+1) &= A\mathbf{x}(k) + B\mathbf{u}(k), \\ \mathbf{y}(k) &= C\mathbf{x}(k), \end{aligned} \tag{2.1}$$

with $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$. Here m, n, and p are the number of inputs, states, and outputs, respectively. The objective of the controller is to find the optimal input for this system by means of minimizing a cost function:

$$J = \sum_{i=1}^{H_p} \|\hat{\mathbf{y}}(k+i \mid k) - \mathbf{y}_{\text{ref}}(k+i \mid k)\|_{W_y}^2 + \sum_{j=0}^{H_c-1} \|\Delta\mathbf{u}(k+j \mid k)\|_{W_u}^2. \tag{2.2}$$

$\mathbf{y}_{\text{ref}}$ is the output reference and $\hat{\mathbf{y}}$ is the predicted output. The formulation $\mathbf{y}(k+i \mid k)$ represents the vector $\mathbf{y}$ on sample time $k+i$ at calculation time k. The change of the input is $\Delta\mathbf{u}(k+j \mid k) = \mathbf{u}(k+j \mid k) - \mathbf{u}(k+j-1 \mid k)$. H_p and H_c (with $H_c \le H_p$) are, respectively, the prediction and control horizon of the controller. $W_y \in \mathbb{R}^{p\times p}$ and $W_u \in \mathbb{R}^{m\times m}$ are positive definite weighting matrices.

One of the key elements of MPC is the possibility to handle constraints. For this paper only input constraints are taken into account:

$$\begin{aligned} \mathbf{u}(j) &\le \mathbf{u}_{\text{Max}}, \\ \mathbf{u}(j) &\ge \mathbf{u}_{\text{Min}}. \end{aligned} \tag{2.3}$$

Output and state constraints are omitted in the current study but can easily be introduced. The optimization problem can be formulated as the minimization of (2.2), subject to (2.1) and (2.3). In order to solve this problem, the optimization problem is reformulated by elimination of the states in the form of a QP:

$$\min_{\theta} J = \frac{1}{2}\theta^T H\theta + g^T\theta, \tag{2.4}$$

$$\text{subject to}: P\theta \le \alpha, \tag{2.5}$$

with (2.4) the quadratic objective function, (2.5) the linear inequalities constraints, and θ the decision variables. To reformulate the problem, following steps are taken. First, the state space model (2.1) is rewritten in terms of $\Delta\mathbf{u}$ and a new state

$$\mathbf{x}^{\star} = \begin{bmatrix} \Delta\mathbf{x} \\ \mathbf{y} \end{bmatrix}, \tag{2.6}$$

with $\Delta\mathbf{x}(k) = \mathbf{x}(k) - \mathbf{x}(k-1)$ and $\mathbf{y}$ the current measured output [21]. The prediction over the prediction horizon is written in matrix formulation and is formulated as

$$\widehat{\mathbf{Y}} = F\mathbf{x}^{\star} + Q\Delta\mathbf{U}, \tag{2.7}$$

with $\mathbf{Y}$ and $\Delta\mathbf{U}$ column matrices of predicted outputs and delta inputs, respectively. For example, $\Delta\mathbf{U} \in \mathbb{R}^{nH_c\times 1}$ composed of $\Delta\mathbf{u}(k \mid k)$ to $\Delta\mathbf{u}(k + H_c - 1 \mid k)$. The matrices F and Q can be found in many works on MPC [3, 21]. The matrix Q is postprocessed by including the weight matrices as follows:

$$S = Q^T W_{y_{bd}} Q + W_{u_{bd}}, \tag{2.8}$$

with $W_{y_{bd}}$ and $W_{u_{bd}}$ block diagonal matrices of W_y and W_u, respectively. As for the purpose of this work, the aim is to minimize the online calculation work; matrices that can be computed in advance are calculated offline. Matrix S is one of the precomputed matrices that does not change during runtime. Vector g from (2.4) has to be calculated online. It contains two parts depending on the current state and the reference of the in- and outputs. So, g has to be calculated according to

$$g = G_1\mathbf{x}^{\star} - G_2\mathbf{Y}_{\text{ref}}, \tag{2.9}$$

where G_1 and G_2 are gradient matrices. $\mathbf{Y}_{\text{ref}}$ is an $\mathbb{R}^{pH_c\times 1}$ matrix of the references $\mathbf{y}_{\text{ref}}(k \mid k)$ to $\mathbf{y}_{\text{ref}}(k + H_p \mid k)$. The gradient matrices are constant and are computed offline:

$$\begin{aligned} G_1 &= S^T W_{y_{bd}}^T F, \\ G_2 &= S^T W_{y_{bd}}^T. \end{aligned} \tag{2.10}$$

The constraints, that is, the minimum and maximum admissible values for $\Delta\mathbf{U}$, are calculated online. $\Delta\mathbf{U}_{\text{Max}}$ is a column matrix of H_c times $\Delta\mathbf{u}_{\text{Max}} = \mathbf{u}_{\text{Max}} - \mathbf{u}(k-1)$. $\Delta\mathbf{U}_{\text{Min}}$ is a column matrix of H_c times $\Delta\mathbf{u}_{\text{Min}} = \mathbf{u}_{\text{Min}} - \mathbf{u}(k-1)$. Finally, the QP problem to be solved is

$$\begin{aligned} \min_{\Delta\mathbf{U}} \quad & \frac{1}{2}\Delta\mathbf{U}^T S \Delta\mathbf{U} + g\Delta\mathbf{U}, \\ \text{subject to}: \quad & \Delta\mathbf{U}_{\text{Min}} \le \Delta\mathbf{U} \le \Delta\mathbf{U}_{\text{Max}}, \\ & g = G_1\mathbf{x}^{\star} - G_2\mathbf{Y}_{\text{ref}}. \end{aligned} \tag{2.11}$$

The Hildreth algorithm [13], qpOASES [14], and CVXGEN [22] will be used to solve this QP problem.

2.3. Implementation of the MPC Controller

When the model is known and all parameters in the cost function are fixed, the controller can be simulated and implemented. The hardware determines the speed of calculation and restricts the size of the problem. Certainly in an embedded environment, this is an important factor. The following section deals with these problems when aiming for online MPC on a PLC.

3. The Approach

When the process model has been identified, it is possible to simulate the process and tune the controller to find valid and useful settings. Going towards online MPC on a PLC means that we have to deal with a shrinking amount of memory and a decreasing CPU speed. To this end, the MPC algorithm is analysed and parts that remain unchanged during runtime are precomputed and lifted out of the online calculations. The limited amount of memory limits the size of the problem.

3.1. The Hardware

For the practical implementation, two different controllers are used. First, a National Instruments CompactRIO is used. This PAC controller consists of a NI cRio–9024 Real-Time Controller (800 MHz CPU, 512 Mb memory), a cRIO–9114 Reconfigurable Chassis, an NI 9265 Analogue Current Output module, and an NI 9217 RTD 24-Bit Analogue Input Module. This Real-Time controller is programmed with LabVIEW and is able to run a software library compiled from C/C++ code via a *call library function*. The library is compiled for the VxWorks 6.3 operating system with the gcc-compiler version 3.4.4. Second, a Siemens CPU319-3DP/PN PLC is used. The base memory of this CPU is increased to the maximum allowed 8 Mb. This CPU is the fastest S7-300 CPU. It takes 40 ns for one floating-point operation. An additional SM334 analogue card is employed to connect to the solid-state relays and DC drive. The Siemens CPU is programmed using the Step 7 Professional 2010 software. To code the problem, the *structured control language* (S7-SCL) is used. This programming language corresponds to STL in the standard IEC 61131-3.

3.2. Practical Implementation

To compute a new input for the process, the following sequence of actions, presented in Algorithm 1 are implemented. In advance, constant matrices are precomputed and the reference trajectory for the output is selected.

3.2.1. Programming the PAC

The CompactRIO is running VxWorks as its operating system. A compiler exists to convert C/C++ code. All implemented QP solvers are originally written in C or C++ and are converted into a library. The preparative calculations, for instance, the scaling of the in- and outputs, the estimation of the state, and the selection of the current reference, are programmed in LabVIEW.

```
Offline: Calculate H', G1 and G2
Online: Start PLC or PAC
Store all precomputed matrices in working memory, together with the reference for
inputs and output
while CPU is running do
    if 1 second passed since last call then
        Scale the temperatures
        Calculate current state
        Calculate $g$, $U'_{\min}$ and $U'_{\max}$
        case Hildreth algorithm
            Calculate the unconstrained inputs of the process.
            if unconstrained inputs violate constraints then
                while maximum numbers of iterations is not reached and solution not
                found do
                    Solve one iteration of the QP
                end
                if maximum numbers of iterations is reached then
                    Use unconstrained solution with inputs violating the constraint
                    limited to the constraint
                end
            end
        endsw
        case qpOASES
            while maximum numbers of iterations is not reached and solution not
            found do
                Solve one iteration of the QP
            end
            if maximum numbers of iterations is reached then
                Use last sub-optimal solution
            end
        endsw
        case CVXGEN
            while maximum numbers of iterations is not reached and solution not
            found do
                Solve one iteration of the QP
                end
            if maximum numbers of iterations is reached then
                Use unconstrained solution with inputs violating the constraint limited
                to the constraint
            end
        endsw
        Apply the inputs to the system
    end
end
```

Algorithm 1: Steps to compute the inputs of the experimental set-up.

3.2.2. Programming the PLC

There exists no compiler to transform the C/C++ source code to a running binary on a Siemens PLC. Therefore, the C/C++ code has to be translated into S7-SCL (STL). Although possible, this is a time consuming step. In this project, the qpOASES and hildreth solvers are translated to S7-SCL. The qpOASES solver is translated without the hotstarting possibilities

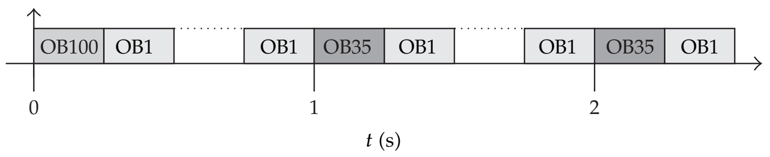


Figure 2: Schematic overview of the different organization blocks in the PLC.

and the general constraint handling code. Instead, only the part that handles bounds is translated. CVXGEN cannot generate STL code, and a manual translation of the generated code is impossible, hence; it is not used.

To calculate the appropriate inputs of the system and solve the QP, following built-in function blocks (FBs) and organization blocks (OBs) are programmed. Organization blocks are built-in functions called by hardware interrupts. Function blocks are user defined functions with corresponding data stored in a data block (DB) with the same number. Figure 2 depicts the order in which these blocks are called.

B 100: Cold Start

This block is called once when the controller is started. It calls function FB 1, which is used to initialize the precomputed matrices larger than the 256 elements which are stored in DB 2 linked to FB 2. This procedure is followed to overcome the limitation that an array of constants cannot be larger the 256 elements at compilation time. In the case a matrix needs to contain more than 256 elements, more arrays are combined in this block at runtime into a combined array. For these experiments, only matrix $G2$ is initialized in this function.

OB 1: Main Loop

This loop is started as soon as OB 100 is finished. When this function finishes, it restarts again. This loop is used to program standard tasks of the PLC. In this experiment, it is not used. It will run during the idle time of the CPU between two OB35 calls.

OB 35: Timed Loop

This organization block is called every second. It contains the necessary code to read the current inputs. This information is scaled and employed to calculate the current state (FB 3). Together with the reference for the in- and outputs, the state is used to update vector g (FB 2). Now, the QP is solved and the scaled solution will be passed to the outputs of the PLC.

3.3. A Solution to the QP: Algorithms

3.3.1. The Hildreth Algorithm

The Hildreth algorithm has been chosen for its limited number of code lines which makes it easy to implement. It has been written in C for the PAC and in S7-SCL for the PLC. This algorithm calculates the solution in two steps [21]. First, the unconstrained solution is calculated, and if no constraints are violated, this solution is adopted. If a constraint is violated, a constrained QP is solved. The solution of the QP is then passed to the inputs of the heating device. For more information about the solution routing, see [13, 21, 23]. If a solution to the QP could not be found, the unconstrained solution is compared to the constraint. If a constraint is violated, that entry of the unconstraint solution is limited to the constraint.

3.3.2. qpOASES

qpOASES is an open-source C++ implementation of the recently proposed online active set strategy [14]. It builds on the idea that the optimal sets of active constraints do not differ much from one QP to the next. At each sampling instant, it starts from the optimal solution of the previous QP and follows a homotopy path towards the solution of the current QP. Along this path, constraints may become active or inactive as in any active set QP solver and the internal matrix factorizations are adapted accordingly. While moving along the homotopy path, the online active set strategy delivers sub-optimal solutions in a transparent way. Therefore, such suboptimal feedback can be reasonably passed to the process in case the maximum number of iterations is reached.

A simplified version of qpOASES has been translated to S7-SCL and was used for controlling the heating device. Note that our simplified implementation does not allow for hot starting the QP solution and is not fully optimized for speed. Moreover, it only handles bounds on the control inputs but no general constraints. On the PAC the plain ANSI C implementation of qpOASES has been used. Although the full version of qpOASES is perfectly suited for hot starting, this is not used as the employed implementations of neither the Hildreth algorithm nor the CVXGEN have possibilities to use hotstart. Moreover, based on the knowledge that a solution is found in one step if no constraints are active, the algorithm is only used with cold starts. This makes it possible to start the search for a solution with offline computed matrices. On the other hand, qpOASES will most probably benefit from hotstarting if constraints are active as the number of required iterations decreases.

3.3.3. CVXGEN

According to the website (http://www.cvxgen.com/), CVXGEN [22] generates fast custom code for small, QP-representable convex optimization problems, using an online interface with no software installation. With minimal effort, a mathematical problem description can be turned into a high speed solver. The generated code is C-code that should run on any device supporting this programming language. It works best for small problems, where the final KKT-matrix has up to 4000 nonzero matrix entries. CVXGEN does not work well for larger problems. The mathematical representation of the QP problem is presented to the web interface and the generated code is compiled for the VxWorks operating system of the CompactRIO. Similar to the Hildreth algorithm, the unconstrained solution, limited to the constraints when violated, is employed if a solution to the QP cannot be found.

4. Experimental Setup

The temperature control setup (Figure 3) consists of a resistor and a fan which can be manipulated separately. The fan is driven by a 24 V DC motor, and the resistor has a maximum power of 1400 W. The heating power delivered by the resistor can be adapted by solid-state relays with analog control (Gefran GTT 25 A 480 VAC—analogue control voltage 0–10 V).

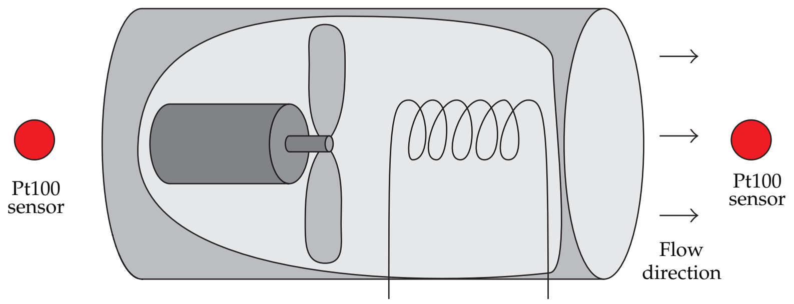


Figure 3: Schematic overview of the temperature control setup.

The fan is manipulated by a custom made DC drive based on a Texas Instruments DRV102T chip and adapted for an analogue control voltage of 0–10 V. Temperature sensors measure the environmental temperature and the temperature of the heated air as indicated in Figure 3. Both sensors are of the PT100 class B type.

5. Results

5.1. Model Identification

To control the experimental setup, a two-input (fan speed and resistor power) and single-output (temperature) black-box model is constructed with the Matlab System Identification Toolbox [24]. A linear, low-order, continuous-time transfer function of the form

$$G(s) = \frac{K_p}{1 + T_{p1}s} e^{-T_d s}. \tag{5.1}$$

is fitted to the data and named *P1D*. Afterwards, this model is discretized and converted to statespace of the form (2.1). This model is called *P1DSS*. The excitation signal of the identification experiment is a multisine with frequencies within the range 0.05–0.00125 Hz. After detrending and normalization, this dataset, is divided in an estimation and validation set with each a length of 250 s. To determine the model quality, a fit measure is defined over the validation data:

$$\text{fit} = 100\% \left(1 - \frac{\left\| \hat{Y} - Y \right\|_2}{\left\| Y - \overline{Y} \right\|_2} \right), \tag{5.2}$$

where $\hat{Y}$ is the simulated output, Y the measured output, and $\overline{Y}$ the mean of the measured output. A fit value of 100% means that the simulation is the same as the measured output.

The final identified P1D model is

$$T_n = \left[\frac{-0.98}{1 + 5.17s} e^{-1.34s} \; \frac{0.83}{1 + 7.17s} e^{-1.53s} \right] \begin{bmatrix} u_{\text{Fan},n} \\ u_{\text{Power},n} \end{bmatrix}, \tag{5.3}$$

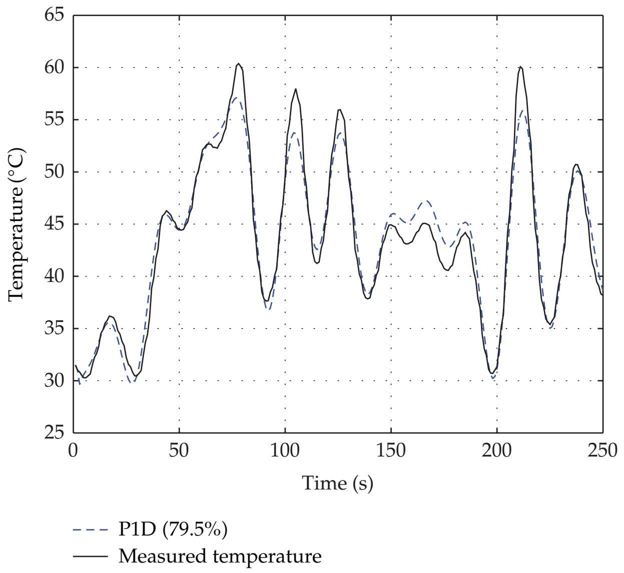


Figure 4: Validation of the P1D model on multisine validation data.

where the index n indicates a normalised variable. $u_{\text{Fan},n}$ and $u_{\text{Power},n}$ are, respectively, the normalized and detrended actuator voltage for the motor drive and the solid state relays of the resistor. After detrending, a zero output of the model corresponds to 42°C. For the inputs, the zero input corresponds to 5 V. Conversion to state space of the *P1D* model with a discretisation interval of 1 s, results in a discrete state space model of order 4. This model is controllable, observable, and stable. The validation of the *P1D* model is depicted in Figure 4. The fit is 79% for both the *P1D* and *P1DSS* model.

5.2. Controller Design

The *P1DSS* model is selected as controller model. The control horizon H_c of the controller is set to 7 and the prediction horizon H_p is 22. These horizons have been chosen similar to those in [25], to compare the different controllers and control algorithms for this temperature control setup. These horizons turned out to be the maximum settings if an MPC controller is built with CVXGEN for use on the PAC.

5.3. Controlling the Setup

On each controller device, three experiments, each time with a different QP algorithm, are executed. The weight matrix W_u in the cost function is the identity matrix:

$$W_u = I_{2\times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{5.4}$$

W_y is set to one. Each experiment consists of a reference trajectory for the temperature of 10 minutes. This reference is first at a constant temperature of 40°C during 100 s. To ensure that the data logging program is ready and the estimator has reached a steady-state value on both PAC and PLC, the inputs calculated by the controller are only applied from 30 s on. During the first 30 s, the fan speed is set to 20% of its maximum speed and the resistor power is 0. After 100 s, the setpoint jumps to 45°C and 60°C for 60 s each. This stair is followed by a half period of a cosine that should bring the temperature at 20°C. This is below the environmental temperature of approximately 22°C and therefore unreachable. This part of the trajectory is added to make sure the controller hits the constraints for a number of seconds. From 320 s on, the temperature reference is set to 30°C for 60 s, followed by a ramp of 30 s with a slope 0.1°C/s. At the end of the ramp there is a jump towards 50°C. This temperature is kept constant for 60 seconds and followed by another set-point change to 30°C for 60 seconds. To end the experiment the temperature is fixed at 40°C. It has to be noted that all algorithms and implementations have tested beforehand in simulation. In this case identical results have been obtained as the solution to the QP is unique. The simulations have been executed in Matlab and LabVIEW. The latter are hardware-in-the-loop simulations. Thereto we have used the code and libraries also used for the experiments on the real setup, but instead of the real setup, a linear model is used.

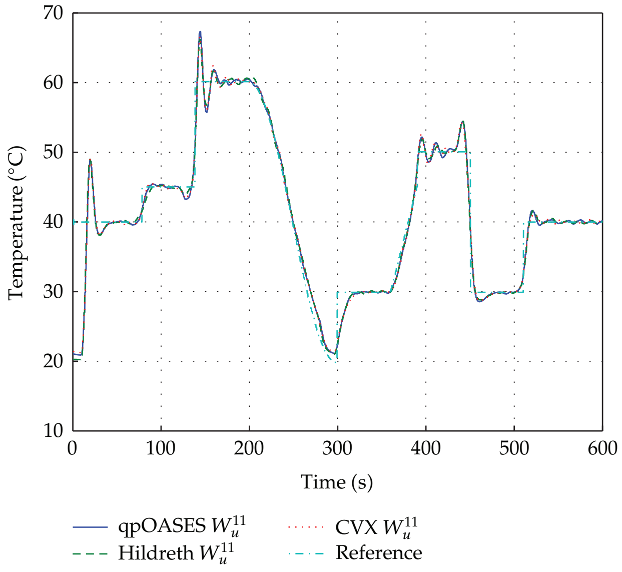
5.3.1. MPC on a PAC

All three QP algorithms are tested on the CompactRIO PAC system. Figure 5 depicts the controlled temperature along the reference and the corresponding inputs.

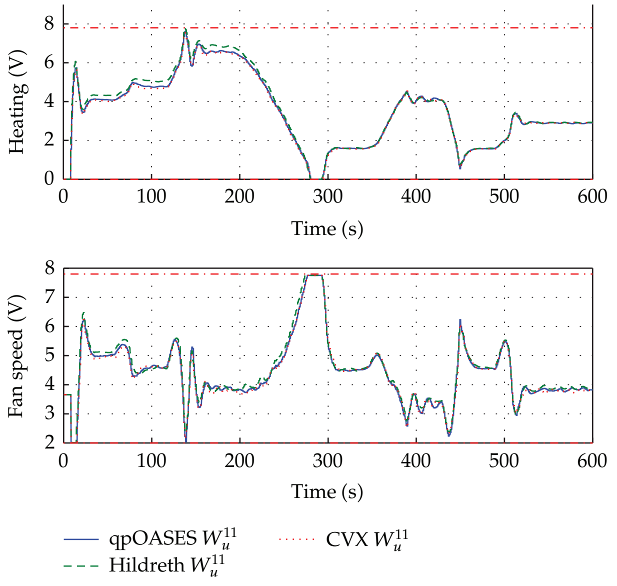
The Resulting Temperature Control

The controlled temperature follows the reference accurately when all transient effects have faded. The incorporated integrator eliminates the steady-state error. The plots show oscillating behaviour at steps when the reference is above 45°C. This is explained by the limited validity of the linear model at these temperatures. This oscillation behavior is unwanted and has to be corrected with different controller settings, for example, an increasing W_u. In order to make a fair comparison with the PLC experiments these settings are not changed. All three algorithms start the experiment with a large overshoot. It is caused by the large step from the environmental temperature to 40°C and the mismatch between the estimated state from the linear model and the real system. For the next step around 80 seconds, an overshoot is hardly seen. The mismatch between the model and the real heating setup is small at this point. The next jump towards 60°C is outside the validity region of the model and this is clearly seen as an overshoot followed by oscillations. The cosine function is followed accurately, except at the end, where the set-point is below the environmental temperature. This makes it impossible to track the desired temperature. After 300 s, the set-point evolves slowly to 30°C as both constraints are active. The next ramp is followed with a delay of one to two seconds. At the end of the ramp, the 10°C jump causes again a large overshoot as the mismatch between the model and real system is large at 50°C. The transition to 30°C leads to a small nonoscillating undershoot.

Important to notice is that all three algorithms behave almost identically. The mean-squared-error (MSE) values are calculated and differ by at most 3%. The small differences in the output (Figure 5(a)) and inputs (Figure 5(b)) are caused by different environmental conditions. An open window is responsible for a soft breeze in the room from time to time.



(a) Measured temperature. The controlled output of the system



(b) The applied inputs to the system

Figure 5: The in- and outputs for the Hildreth, CVXGEN, and qpOASES algorithm on the PAC.

The QP Solvers

Figure 6 depicts the number of iterations needed to solve the QP. The maximum was set to 20 for qpOASES, 25 for CVXGEN, and 60 for Hildreth. The latter is so high as the number of iterations grows linearly with the problem size [26]. None of these maxima were reached, so the QP solvers always converged during this experiment. qpOASES needs only 5 iterations

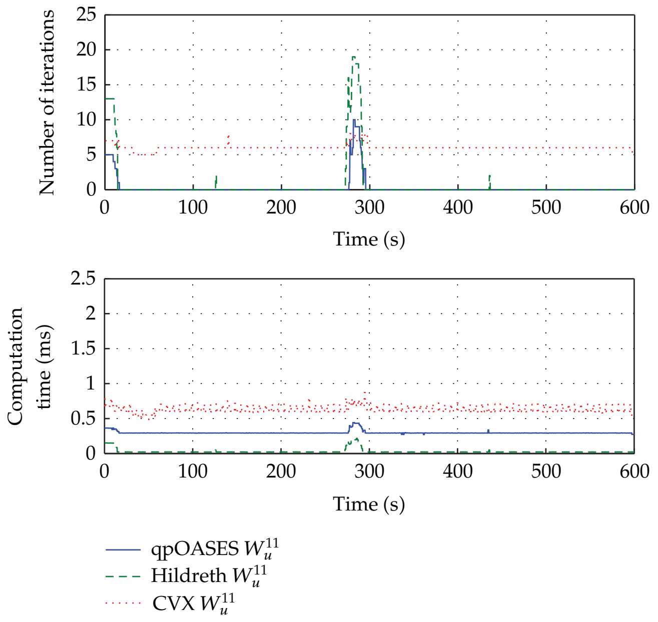


Figure 6: The number of iterations and corresponding calculation time to solve the QP problem on the PAC system for the three algorithms.

at the start while only one constraint is active. When two constraints are active, at maximum 10 iterations are needed to calculate the solution. If no constraint is active, no iterations are needed to solve the problem. The corresponding time is 0.48 ms for 10 iterations, 0.37 ms for 5 iterations, and if no iteration is performed, the solution is presented after 0.29 ms. CVXGEN needs at minimum 6, but never more than 8 iterations during this experiment. The corresponding time is 0.62 and 0.82 ms. Hildreth needs at maximum 19 iterations during this experiment, which takes 0.22 ms.

The bottom plot of Figure 6 depicts the time needed to present a solution by the different algorithms. For CVXGEN, the calculation time for zero constraints and one active constraint is indistinguishable as the number of iterations is identical.Only with two active constraints the number of needed iterations increases. The number of needed iterations and the corresponding time fluctuate more frequently for the other two algorithms. It is not possible to determine from these plots if one or two constraints, are active. For this setup, the Hildreth algorithm needs less time to solve the QP than the other algorithms. CVXGEN needs the most time to present a solution. It is expected that the need for more iterations to solve the problem is a disadvantage for the Hildreth algorithm if the problem size increases.

All three algorithms are integrated in a C/C++ library called by the LabVIEW code. The design choices of the QP developers have important consequences. Although CVXGEN delivers ready-to-use code in a fast and straightforward way, it has to be mentioned that the choice for memory before allocation results in large code if the problem size increases. A test to solve a QP problem with a hessian of size 30 resulted in a code of size 430 kB and a hessian of size 40 even resulted in 996 kB, while only 80 kB was needed with a hessian of size 14 in these experiments. In [25] it has been observed that the code of the problem cannot be higher than 900 kB as the code simply will not run due to limitations of the PAC. This means a hessian of size 40 is the limit with CVXGEN. In the case that one wants MPC to control a multiple input or multiple output system, this limit is most likely too low for practical use on the CompactRIO. This code generator delivers C-code for the declared mathematical description. A small altering of this description means a regeneration of the C-code. If a different language is needed, this code generator cannot be used.

An increase of the size of the hessian hardly increases the code size of qpOASES. The compiled library is about 112 kB large. The compiled code for Hildreth is only 7 kB. This size of the latter two algorithms will only grow with increasing size of the hessian which is coded in the library. These algorithms are programmed generally, so the same code can be used for instance, for different hessian sizes. This code can be translated in a different language.

To employ these algorithms on a PLC, the limited speed and memory size have to be taken into account. As CVXGEN needs the most time to solve the problem and consumes the most memory, this algorithm is not preferred for this problem.

Conclusion. The three algorithms are incorporated in a library compiled from the C-code. The provided code for qpOASES and CVXGEN could be implemented easily. With growing QP problem size, the compiled CVXGEN code is increasing very fast, limiting the size of the QP to 40 on this device. This is most likely not enough for a lot of MIMO systems. qpOASES and Hildreth do not encounter this problem. From the control point of view, there is hardly any difference for the three algorithms. From the implementation side of view, Hildreth is a simple algorithm with a small footprint that is easily implemented. Also qpOASES is easily implemented with the freely available C++ code, but the more complex code takes more time to be evaluated. On the other hand, the number of iterations is less than that for the Hildreth algorithm, which can be an advantage when the size of the QP problem increases.

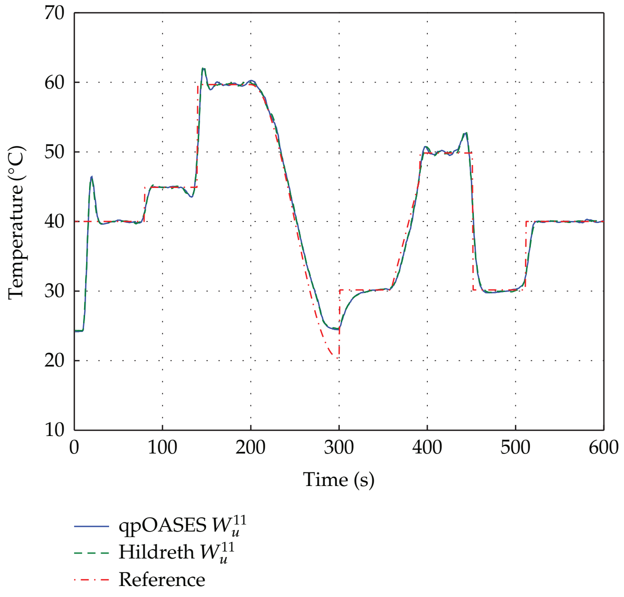
5.3.2. MPC on a PLC

As no STL code can be generated with the CVXGEN code generator for the PLC, only the Hildreth and qpOASES algorithms are tested on the PLC. Figure 7(a) depicts the measured temperature and its reference for the Hildreth and qpOASES algorithms. The corresponding inputs are displayed in Figure 7(b).

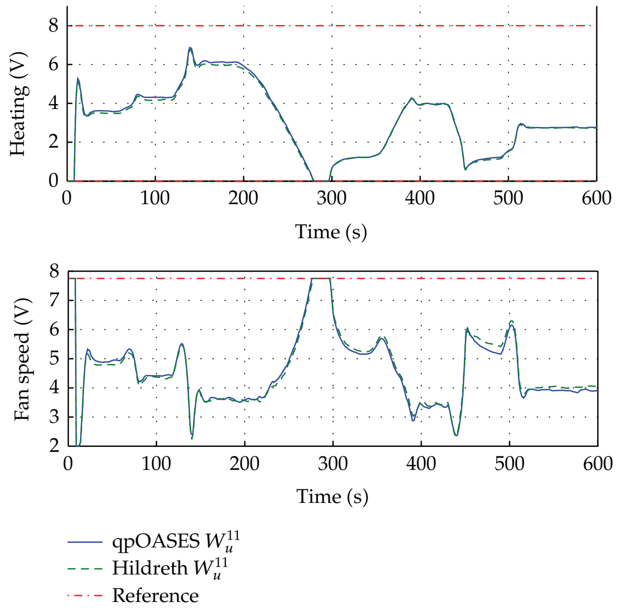
The Resulting Temperature Control

Both, the Hildreth and qpOASES algorithms follow the desired reference temperature accurately. The large overshoot in the beginning is also caused by an inaccurate estimate of the temperature, combined with a large step from the environmental temperature towards 40°C. The jump towards 50°C is taken without overshoot. The set-point change to 60°C causes an overshoot but after about 20 s, the reference is accurately followed again. Also the cosine function is closely followed. Around 300 s, at the end of the cosine, the reference temperature is lower than the environmental temperature, which makes it impossible to reach this temperature without cooling, causing both the heating and fan constraint to be hit (Figure 7(b)). The next set-point of 30°C is reached, but as still both constraints are active, the temperature evolves slowly to the desired set-point. The ramp and step towards 50°C is followed with a small delay. The large decrease of the temperature from 50 to 30°C is reached with a very small undershoot. The final jump toward 40°C is smooth and without overshoot.

Both algorithms behave similarly. The MSE for all PLC experiments differs 2% with qpOASES having the highest value. As the stop criteria for both QP algorithms are identical,



(a) Measured temperature. The controlled output of the system



(b) The applied inputs to the system

Figure 7: The in- and outputs for the Hildreth and qpOASES algorithm on the PLC.

the different environmental conditions are the main reason for this difference. In case a constraint is active, the delay caused by the calculation time needed to solve the QP problem differs nearly 150 ms between both algorithms (Figure 8). This also might have a small influence.

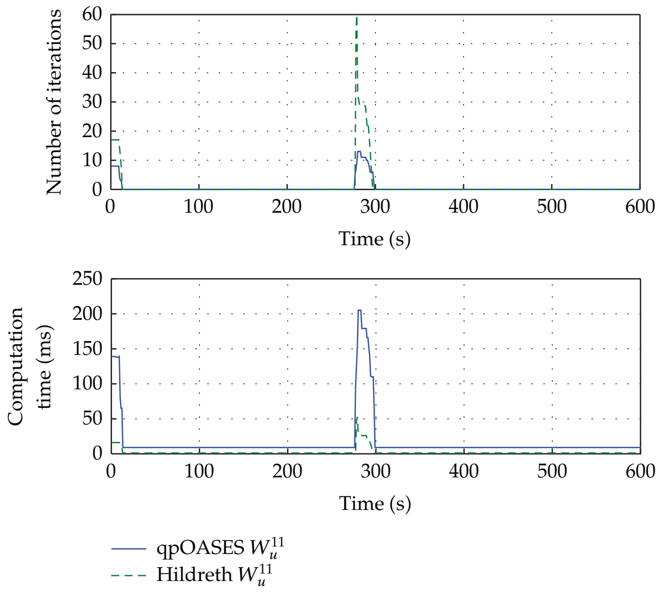


Figure 8: Number of iterations and corresponding calculation time needed to solve the QP problem on the PLC for both algorithms.

The QP Solvers

The top plot in Figure 8 depicts the required number of iterations for the employed algorithms. Each iteration is a backsolve operation, which corresponds to a check of one constraint at one time instance during the control horizon, followed by an update of the active set. The bottom plot of Figure 8 plots the corresponding calculation time. For Hildreth, the maximum number of iterations is 60 at 279 s after the start of the experiment. As this is the maximum allowed, no solution was found to the QP and an unconstrained, but limited to the constraints, solution is delivered. As stated earlier, the applied input was manually fixed during the first 30 s of the experiment. The Hildreth algorithm needs approximately 1 ms to calculate one iteration.

If no constraint is hit for the qpOASES algorithm, no iteration of the QP is needed. Due to the more complex code of the algorithm, it takes about 10 ms to calculate the input. If a constraint is hit, it takes between 15 and 16 ms to process 1 iteration. At the start, at maximum 8 iterations, corresponding with 140 ms of calculation time, are needed to come up with a solution. The maximum number of iterations is set to 20. This number is never reached, so an optimal solution is always provided to the system for these experiments. If only one constraint is hit, 7 iterations are sufficient. No constraints are violated.

Both algorithms solve the QP online. The Hildreth algorithm is faster in time compared to qpOASES. On the other hand, the maximum number of iterations is less for qpOASES and still an optimal solution is always found. It has to be noted that at time instance of 279 s, the Hildreth algorithm needs 60 iterations which is the maximum allowed. Hence, no (optimal) solution is available. In such a case, the algorithm delivers the unconstrained solution, with all entries violating their constraints limited to there respectively, constraints. It is to be expected that this causes no harm, but it is not the optimal solution. This situation can last for several time instances. qpOASES on the other hand always gives an optimal solution without reaching the maximum number of iterations. This makes the latter more suited for MPC.

Both algorithms are active set algorithms. This means it is possible to stop the algorithms early. It has to be noted that a minimum number of iterations is needed. Although the qpOASES algorithm should be able to reach the optimum in maximum 2 to 5 iterations according to [14] if hot starting is applied, at least 7 if one constraint is active or 14 if two constraints are active are needed for these experiments if no hot start is executed. With a cold start, at least all constraints at every time instance of the control horizon need to be checked. For one active constraint, this means 7 checks are to be performed for this experimental setup. Two active constraints need 14 checks. qpOASES is in this case able to solve all optimization problems in one check of the constraints, as it needs at maximum 14 iterations. At 279 s, Hildreth needs to check more than 4 times all of the constraints and is still not at the optimal solution. It is clear that qpOASES evolves faster to the optimum than Hildreth. For practical use, qpOASES is therefore preferred.

Conclusion. Two online QP solvers have been successfully tested. They are used for a model-based control of a heating device on a PLC. Despite the substantially higher calculation time to present a solution, qpOASES performs not substantially better than Hildreth. For this particular MPC study on a small setup, the additional calculation time needed for qpOASES to solve the problem with less iterations is not needed, on the other hand leads the high number of iterations from time to time to suboptimal solutions for Hildreth. For larger systems it is to be expected that the reduced number of needed iterations for qpOASES can lead to a shorter solution time of the QP, certainly if also hotstarting is used.

6. Conclusion

Three online QP algorithms have been investigated for their applicability to solve an online model-based control problem on industrial hardware. All three algorithms perform similarly on a small-scale test setup, but were obtained in a different way. The first algorithm is generated by a code generator (CVXGEN). This is an easy and fast way to get the needed code. On the other hand, the use of a code generator sticks the user to the offered programming language, and it is impossible to change concept decisions of the developer resulting in unwanted effects such as large memory consumption. The second algorithm (qpOASES) uses off-the-shelf code. This offers the user high-quality and easy-to-implement code. The third algorithm is programmed based on the theoretical concept of the Hildreth algorithm. Starting from scratch takes a lot of (debugging) time and is fault sensitive. All algorithms have been implemented on a PAC. As a different programming language is needed, only the latter two algorithms have been implemented on a PLC. As the accuracy of all three algorithms used on the PAC is comparable, qpOASES is preferred for use on a PAC based on flexibility, for instance, the size of the problem, and user friendliness of the algorithm. Although not tested, qpOASES most probably will benefit from its hot starting ability. The limited amount of memory and calculation speed of a PLC, make this device not suited for controlling MIMO systems with more than two to five in- and outputs. This makes an algorithm with a small footprint and fast calculations such as Hildreth, most suited for MPC implementations on a PLC.

Acknowledgments

This work is supported in part by the Katholieke Universiteit Leuven: OT/10/035, OPTEC Center-of-Excellence Optimization in Engineering (PFV/10/002), SCORES4CHEM (KP/09/005); by the Belgian Federal Science Policy Office: the Belgian Program on Interuniversity Poles of Attraction; by the European Commission: Interreg IVa-7-022-BE i-MOCCA. J. F. V. Impe holds the chair Safety Engineering sponsored by the Belgian Chemistry and Life Sciences Federation Essenscia. The authors thank Andreas Potschka for his Matlab implementation of qpOASES. They also thank Lennert Callebaut for his help with the translation of the Matlab implementation of qpOASES to STL.

References

[1] S. J. Qin and T. A. Badgwell, “A survey of industrial model predictive control technology,” *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[2] U. Mathur, R. D. Rounding, D. R. Webb, and R. J. Conroy, “Use model-predictive control to improve distillation operations,” *Chemical Engineering Progress*, vol. 104, no. 1, pp. 35–41, 2008.

[3] J. Maciejowski, *Predictive Control with Constraints*, Pearson Education Limited, 2002.

[4] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[5] L. Van den Broeck, M. Diehl, and J. Swevers, “A model predictive control approach for time optimal point-to-point motion control,” *Mechatronics*, vol. 21, no. 7, pp. 1203–1212, 2011.

[6] N. Giorgetti, G. Ripaccioli, A. Bemporad, I. V. Kolmanovsky, and D. Hrovat, “Hybrid model predictive control of direct injection stratified charge engines,” *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 5, pp. 499–506, 2006.

[7] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, “The explicit linear quadratic regulator for constrained systems,” *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[8] P. Tøndel, T. A. Johansen, and A. Bemporad, “An algorithm for multi-parametric quadratic programming and explicit MPC solutions,” *Automatica*, vol. 39, no. 3, pp. 489–497, 2003.

[9] G. Pannocchia, J. B. Rawlings, and S. J. Wright, “Fast, large-scale model predictive control by partial enumeration,” *Automatica*, vol. 43, no. 5, pp. 852–860, 2007.

[10] J. R. Wright, “The debate over which PLC programming language is the state-of-the-art,” *Journal of Industrial Technology*, vol. 15, no. 4, pp. 2–5, 2006.

[11] G. Valencia-Palomo and J. A. Rossiter, “Efficient suboptimal parametric solutions to predictive control for PLC applications,” *Control Engineering Practice*, vol. 19, no. 7, pp. 732–743, 2011.

[12] M. Kvasnica, I. Rauová, and M. Fikar, “Automatic code generation for real-time implementation of model predictive control,” in *Proceedings of IEEE International Symposium on Computer-Aided Control System Design (CACSD ’10)*, pp. 993–998, September 2010.

[13] C. Hildreth, “A quadratic programming procedure,” *Naval Research Logistics Quarterly*, vol. 4, pp. 79–85, 1957.

[14] H. J. Ferreau, H. G. Bock, and M. Diehl, “An online active set strategy to overcome the limitations of explicit MPC,” *International Journal of Robust and Nonlinear Control*, vol. 18, no. 8, pp. 816–830, 2008.

[15] T. Soderstrom and P. Stoica, *System Identification*, Prentice-Hall, London, UK, 1989.

[16] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory, Implementation, Applications*, Kluwer Academic, 1996.

[17] L. Ljung, *System Identification: Theory for the User*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1999.

[18] C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms*, Prentice-Hall, 1998.

[19] J. A. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*, World Scientific, Singapore, 2002.

[20] E. F. Camacho and C. Bordons, *Model Predictive Control*, Springer, 2003.

[21] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB*, Springer, London, UK, 2009.

[22] J. Mattingley and S. Boyd, "CVXGEN: a code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[23] D. G. Luenberger, *Optimization by Vector Space Methods*, John Wiley & Sons, New York, NY, USA, 1969.

[24] L. Ljung, *System Identification Toolbox Users Guide*, The MathWorks, Natick, Mass, USA, 2009.

[25] B. Huyck, F. Logist, J. De Brabanter, J. Van Impe, and B. De Moor, "Constrained model predictive control on a programmable automation system exploiting code generation: practical considerations," in *Proceedings of the 18th World Congress of the International Federation of Automatic Control*, pp. 12207–12212, Milano, Italy, 2011.

[26] A. N. Iusem and A. R. De Pierro, "On the convergence properties of Hildreth's quadratic programming algorithm," *Mathematical Programming*, vol. 47, no. 1, pp. 37–51, 1990.