Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

PLC implementation of a real-time embedded MPC algorithm based on linear input/output models ⋆

Pablo Krupa * **Nilay Saraf** ** **Daniel Limon** *
Alberto Bemporad **

* *University of Seville (e-mail: pkrupa@us.es, dlm@us.es).*
** *IMT School for Advanced Studies Lucca (e-mail: nilay.saraf@alumni.imtlucca.it, alberto.bemporad@imtlucca.it)*

Abstract: How to efficiently implement Model Predictive Control (MPC) in embedded systems is a topic that is attracting a lot of research recently, due to its impact in practical applications. Implementing MPC in industrial Programmable Logic Controllers (PLCs) is of particular interest due to their widespread prevalence in the industry in comparison with other embedded systems, such as FPGAs or microcontrollers. In this paper, we present a PLC implementation of real-time embedded MPC for multivariable systems described by linear time-invariant input/output models subject to upper and lower bounds on input and output variables. The MPC algorithm uses a recently developed primal active-set method for bounded-variable least-squares problems. We highlight and address some crucial challenges that arise in implementing the MPC algorithm in a PLC. Possible extensions of the proposed methods are presented along with hardware-in-the-loop simulation results of controlling a nonlinear multivariable system using a real industrial PLC.

Keywords: Embedded optimization, programmable logic controllers, model predictive control, primal active-set method

1. INTRODUCTION

Model Predictive Control (MPC) is an advanced control method with increasing popularity in the industry due to its ability to optimize closed-loop performance of systems subject to input and output constraints (Camacho and Bordons Alba, 2013). In MPC, the control law is derived at each time step from the solution of an optimization problem. This leads to one of the main obstacles for the implementation of MPC in an industrial setting, which is the fact that control loops are typically implemented using embedded systems, whose limited computational and memory resources are not generally suitable for computing the MPC control law in real time. Specifically, the most common embedded control hardware used in the industry is the Programmable Logic Controller (PLC) (Alphonsus and Abdullah, 2016), which is a rugged and highly reliable digital computer specifically designed to cope with the environmental conditions of an industrial setting, and it typically has very limited resources.

One possible approach used to overcome this limited availability of resources is the use of *explicit* MPC (Bemporad, 2019), such as in (Valencia-Palomo and Rossiter, 2012), or (Raha et al., 2019), where the solution of the MPC control law is computed offline and stored in the embedded system as an explicit, continuous and piecewise affine function of the state (Tøndel et al., 2003). However, the memory requirements of this approach become prohibitive for medium to large-sized systems and/or MPC problems involving many constraints.

Another solution is to use efficient algorithms capable of solving the optimization problem on line that are tailored to embedded systems, such as CVXGEN (Mattingley and Boyd, 2012), FiOrdOs (Ullmann, 2011), qpOASES (Ferreau et al., 2014), and (Cimini et al., 2017). These tools have successfully been used to implement MPC in embedded systems, such as in (Huyck et al., 2012), (Kufoalor et al., 2015), and even in automotive mass production (Bemporad et al., 2018). Certain solvers are tailored to a certain MPC formulation and a specific embedded platform. Some examples of this approach are (Krupa et al., 2018), (Patrinos and Bemporad, 2013) and (Wang and Boyd, 2010).

Most of the advancements in recent years on this topic share two common features. First, they usually target embedded systems such as FPGAs or microcontrollers, instead of PLCs. Second, the prediction model is usually a state-space one, whereas in an industrial setting the use of an input/output (I/O) model for MPC design would be preferable, since systems are often identified from I/O data as step-response, impulse-response, or transfer-function models.

This paper presents the implementation on a PLC of the MPC approach based on I/O models developed by

⋆ The authors acknowledge MINERCO and FEDER funds for funding project DPI2016-76493-C3-1-R, and MCIU and FSE for the FPI-2017 grant. This work was partially supported by the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no. 674875 (oCPS).

(Saraf and Bemporad, 2017). Additionally, the cited MPC approach is extended to get offset-free control by adding an observer (Pannocchia et al., 2015). The MPC optimization problem is posed as a bounded-variable least-squares (BVLS) problem, which is solved on line using the primal active-set algorithm developed in (Saraf and Bemporad, 2019). In order to highlight the practical use of the proposed methods in an industrial setting, the linear MPC algorithm is implemented in a real industrial PLC for offset-free control of a nonlinear quadruple water-tank system in a hardware-in-the-loop (HIL) setting.

The paper is structured as follows. Section 2 describes the MPC and observer formulations, including the system description. Furthermore, it describes how offset-free control is attained. In Section 3 we discuss the implementation of the controller in the PLC, highlighting the associated challenges. Section 4 shows the results of the HIL tests, including the system description, closed-loop trajectories of reference tracking and disturbance rejection tests, and memory/computational requirements. Finally, Section 5 provides some final conclusions and future lines of work.

Notation: The component i of a vector x is denoted by x_i. Vector $x = (x_{(1)}, x_{(2)}, \dots, x_{(N)})$ is a column vector formed by the concatenation of vectors $x_{(1)}$ to $x_{(N)}$. Given a matrix $M \in \mathbb{R}^{n\times m}$, $M^\top$ denotes its transpose. For a vector x, $\|x\|_2$ is its Euclidean norm, i.e. $\|x\|_2 \doteq \sqrt{x^\top x}$. Given two vectors x and y, $x \leq (\geq)\ y$ denotes component-wise inequality. An identity matrix of dimension n is denoted by I_n. A matrix in $\mathbb{R}^{n\times m}$ whose elements are all zero is denoted by $0_{n\times m}$, and $0_{n\times n}$ is shortened as 0_n. The expression $M \succ 0$ denotes that a square matrix M is positive definite. Given a signal x, $x(k)$ is its value at time instant k, and $x(k|j)$ is the estimation of its value at time instant k based on the knowledge available at time instant j.

2. PROBLEM FORMULATION

We consider a system described by the following linear time-invariant I/O model in autoregressive exogenous form

$$y(k) = \sum_{j=1}^{n_a} A_j y(k-j) + \sum_{j=1}^{n_b} B_j u(k-j), \tag{1}$$

where $n_a, n_b > 0$, $A_j \in \mathbb{R}^{p\times p}$, $B_j \in \mathbb{R}^{p\times m}$, $y \in \mathbb{R}^p$ is the system output and $u \in \mathbb{R}^m$ is the system input. Additionally, we want (1) to satisfy the following box constraints

$$\begin{aligned} \underline{u} \leq u(k) \leq \overline{u} \\ \underline{y} \leq y(k) \leq \overline{y}. \end{aligned} \tag{2}$$

The control objective is to steer the measured system output $y_m \in \mathbb{R}^p$, i.e. the true system output, to a desired constant reference $r \in \mathbb{R}^p$ with zero offset while satisfying the system constraints (2), which is achieved by the use of the MPC controller and observer described in the following sections. Figure 1 shows a block representation of the controller architecture, that is very common in control applications.

2.1 Observer formulation

We include an observer in order to estimate steady-state offsets (or unmeasured constant disturbances), which may

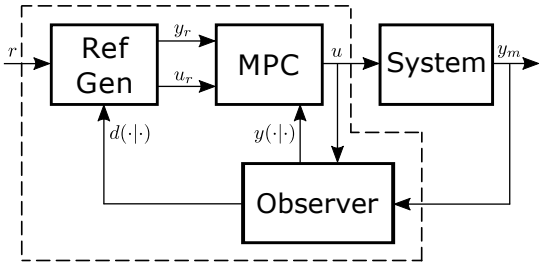

Fig. 1. Controller architecture.

arise, for instance, due to model mismatch. We consider the following augmented model

$$y(k) = \sum_{j=1}^{n_a} A_j y(k-j) + \sum_{j=1}^{n_b} B_j u(k-j) \tag{3a}$$

$$d(k+1) = d(k) \tag{3b}$$

$$\hat{y}_m(k) = y(k) + d(k), \tag{3c}$$

where $d \in \mathbb{R}^p$ is the disturbance, $\hat{y}_m \in \mathbb{R}^p$ is now modeling the measured output y_m, and where we are slightly abusing notation by using the same symbols as in (1). The controller is then designed to steer $\hat{y}_m(k)$ to r, or equivalently, to steer $y(k)$ to $r - d(k)$.

The observer estimates, at each sample time k, the values of $y(k+1|k)$ to $y(k-n_a+2|k)$ and the value of $d(k+1|k)$ based on the augmented model (3) as follows,

$$e(k) = y_m(k) - y(k|k-1) - d(k|k-1) \tag{4a}$$

$$\begin{aligned} y(k+1|k) = &\sum_{j=0}^{n_a-1} A_j y(k-j|k-1) + \\ &\sum_{j=0}^{n_b-1} B_j u(k-j) + L_1 e(k) \end{aligned} \tag{4b}$$

$$y(k|k) = y(k|k-1) + L_2 e(k) \tag{4c}$$

$$\vdots$$

$$y(k-n_a+2|k) = y(k-n_a+2|k-1) + L_{n_a} e(k) \tag{4d}$$

$$d(k+1|k) = d(k|k-1) + L_d e(k), \tag{4e}$$

where matrices $L_i \in \mathbb{R}^{p\times p}$ for $i = 1 \dots n_a$ and $L_d \in \mathbb{R}^{p\times p}$ are chosen so that the observer is stable (see Proposition 2), and $e \in \mathbb{R}^p$ is the estimation error. Note that the estimation error $e(k)$ (4a) is the difference between the measured output $y_m(k)$ and its estimate

$$\hat{y}_m(k|k-1) \doteq y(k|k-1) + d(k|k-1).$$

Remark 1. *Observer (4) is constructed by taking the observer from (Maeder et al., 2009), which is built for a state-space model. Consider the state, input, and output vectors defined by*

$$\begin{aligned} \tilde{x}(k) &\doteq (y(k),\ y(k-1),\ \dots,\ y(k-n_a+1)), \\ \tilde{u}(k) &\doteq (u(k),\ u(k-1),\ \dots,\ u(k-n_b+1)), \\ \tilde{y}(k) &\doteq y(k), \end{aligned} \tag{5}$$

respectively. Then, model (1) can be rewritten as the state-space model $\tilde{x}(k+1) = \mathcal{A}\tilde{x}(k) + \mathcal{B}\tilde{u}(k)$, $\mathbf{y}(k) = \mathcal{C}\tilde{x}(k)$, *with*

$$\mathcal{A} = \begin{bmatrix} A_1 & A_2 & A_3 & \dots & A_{n_a} \\ I_p & 0_p & 0_p & \dots & 0_p \\ 0_p & I_p & 0_p & \dots & 0_p \\ 0_p & 0_p & \ddots & 0_p & 0_p \\ 0_p & \dots & \dots & I_p & 0_p \end{bmatrix}, \ \mathcal{C} = [\, I_p\ 0_p\ \dots\ 0_p \,]. \tag{6}$$

Observer (4) is then constructed as observer (9) from (Maeder et al., 2009) but written for the I/O augmented model (3) instead of using the state-space form (5). As such, the results from the cited paper apply to (4). Note that $\hat{y}_m(k)$ of (3c) is playing the role of output $y(t)$ of (3) and (4) in (Maeder et al., 2009).

In order to achieve offset-free control, the following two conditions must be satisfied (Maeder et al., 2009). First, the augmented model (3) must be observable. And second, the observer (4) must be stable. The satisfaction of these conditions is given by the following two propositions, obtained from (Maeder et al., 2009) and rewritten for our formulation and notation.

Proposition 1. *The augmented system (3) is observable if and only if*

$$\begin{bmatrix} \mathcal{A} - I_p & 0_{n\times p} \\ \mathcal{C} & I_p \end{bmatrix}$$

has full column rank, where $\mathcal{A}$ and $\mathcal{C}$ are given by (6).

Proposition 2. *The observer (4) is stable if and only if*

$$\begin{bmatrix} \mathcal{A} - L_x\mathcal{C} & -L_x \\ -L_d\mathcal{C} & I_p - L_d \end{bmatrix}$$

is a stabilizing matrix for a discrete LTI state-space model, where $L_x = \begin{bmatrix} L_1^\top & \dots & L_{n_a}^\top \end{bmatrix}^\top$, and $\mathcal{A}$ and $\mathcal{C}$ are given by (6).

Remark 2. *The reader may note that the results from (Maeder et al., 2009) require the system model to be observable, which we do not formally specify in the above due to our use of an I/O model instead of a state-space one. However, note that (6) is clearly observable. As such, the above results hold.*

2.2 MPC formulation

We consider the predicted outputs $\mathbf{y}(j)$, $j = 1, \dots, N_p$, and predicted inputs $\mathbf{u}(j)$, $j = 0, \dots, N_u - 1$, as decision variables, where N_p is the prediction horizon and N_u (with $N_u \leq N_p$) is the control horizon, i.e. $\mathbf{u}(j) = \mathbf{u}(N_u - 1)$, $\forall j \geq N_u$.

At each time step k, the control action $u(k)$ is taken as the optimal value $\mathbf{u}(0)$ of the following convex quadratic optimization problem,

$$\min_{\mathbf{u},\mathbf{y}} \sum_{j=1}^{N_p} \frac{1}{2}\|W_y(\mathbf{y}(j) - y_r)\|_2^2 + \sum_{j=0}^{N_p} \frac{1}{2}\|W_u(\mathbf{u}(j) - u_r)\|_2^2 \tag{7a}$$

$$s.t.\ \mathbf{y}(j) = y(k+j|k-1), \forall j \in \{-n_a + 1, \dots, 0\} \tag{7b}$$

$$\mathbf{y}(j) = \sum_{i=1}^{n_a} A_i \mathbf{y}(j-i) + \sum_{i=1}^{n_b} B_i \mathbf{u}(j-i), \tag{7c}$$

$$\underline{y} \leq \mathbf{y}(j) \leq \overline{y}, \quad \forall j \in \{1, \dots, N_p\} \tag{7d}$$

$$\underline{u} \leq \mathbf{u}(j) \leq \overline{u}, \quad \forall j \in \{0, \dots, N_u - 1\} \tag{7e}$$

where $W_y \succ 0$ and $W_u \succ 0$ are tuning weights and y_r and u_r are the steady-state output and input references, respectively. In order to achieve offset-free control, we set

$$y_r = r - d(k|k-1),$$

and u_r is usually computed such that (y_r, u_r) is a steady-state of (1), or by solving a static optimization problem.

Note that the MPC is initialized with the estimates $y(k|k-1)$ to $y(k - n_a + 1|k - 1)$ provided by the observer (4). Additionally, the reference is corrected by the current estimate of the disturbance $d(k|k-1)$. We refer the reader to (Maeder et al., 2009) and (Pannocchia et al., 2015) for insights regarding how this approach is able to eliminate steady-state offsets.

Note that $\mathbf{y}(j)$ for $j \in \{-n_a + 1, \dots, 0\}$ (see Eq. (7b)) and $\mathbf{u}(j)$ for $j \geq N_u$ are not decision variables, but are included in (7) for simplicity of notation.

2.3 BVLS formulation

Problem (7) can be compactly recast as the following quadratic programming (QP) problem

$$\min_{\mathbf{z}} \frac{1}{2}\|W_z(\mathbf{z} - z_r)\|_2^2 \tag{8a}$$

$$s.t.\ G\mathbf{z} - g = 0, \tag{8b}$$

$$\underline{z} \leq \mathbf{z} \leq \overline{z}, \tag{8c}$$

where $W_z \in \mathbb{R}^{(N_u m + N_p p)\times(N_u m + N_p p)}$ is a block diagonal matrix constructed by stacking weights W_y and W_u according to the arrangement of the decision variables $\mathbf{z}$; vector z_r contains the steady state references y_r and u_r stacked accordingly; vectors $\underline{z}$ and $\overline{z}$ impose the constraints (7d) and (7e); and G, g contain the equality constraints (7c) and the values of $y(k|k-1)$ to $y(k - n_a + 1|k - 1)$.

Problem (8) can be in turn recast as the BVLS problem

$$\min_{\mathbf{z}} \quad \frac{1}{2}\left\| \begin{bmatrix} W_z \\ \sqrt{\rho}G \end{bmatrix} \mathbf{z} - \begin{bmatrix} W_z z_r \\ \sqrt{\rho} g \end{bmatrix} \right\|_2^2 \tag{9a}$$

$$s.t. \quad \underline{z} \leq \mathbf{z} \leq \overline{z} \tag{9b}$$

by relaxing the equality constraints (8b) using a quadratic penalty function weighted by a sufficiently large parameter $\rho > 0$ (Saraf, 2019, Section 3.5.3). A detailed discussion on the benefits of formulation (9) may be referred in (Saraf and Bemporad, 2017, Section II.B) and (Saraf, 2019).

3. PLC IMPLEMENTATION

This section discusses the implementation of the MPC controller in a PLC, highlighting the main challenges and describing how (9) is solved on line.

3.1 Controller implementation

The controller is programmed using the *FBD* programming language, in which functions are represented as blocks with a series of inputs and outputs connected to those of other blocks, fixing a layout that determines the execution order. The *FBD* programming language, along four others, are standardized in norm IEC 61131, which sets the guidelines and construction requirements for PLCs.

The controller is composed of the blocks shown in Figure 1 within the dashed line: the MPC, the observer and a block that computes (y_r, u_r) as described in Section 2.2. Additionally, the program contains blocks that interface the input and output signals of the PLC with those of the controller. The functions executed by the blocks are coded using *Structured Text*, which is an IEC 61131 standardized sequential programming language that resembles Pascal.

All of the above blocks are programmed on the PLC's *main task*, which is executed cyclically. At the start of

each cycle there is an overhead due to the execution of diagnostic processes and to the reading/writing of PLC's inputs/output signals. This can limit the application of the PLC for controlling systems with fast dynamics.

PLCs have limited computational and memory resources, as is common in many embedded systems. As an example, the resources of a Schneider Electric® Modicon M340 with a BMXP3420302 processor module, which is a medium-range industrial PLC that we used for the HIL tests, are: 256Kbytes of data memory (user declared data), 3840Kbytes of program memory (user defined executable code, diagnostics, system, etc.), and an execution time of 0.12μs for boolean operations and 1.16μs for floating point operations. The overhead for the main task is 0.7ms.

3.2 Challenges

The following list summarizes the main challenges of implementing MPC in real time on a PLC:

(i) Even though the syntax of *Structured Text* resembles that of high level programming languages such as Pascal or C, it does not support the use of external libraries. It is thus preferable to use a library-free MPC algorithm.
(ii) Most PLCs work with single-precision arithmetic. Therefore, the MPC algorithm must be robust against numerical errors due to limited computing precision.
(iii) It is desirable for the entire source-code of the MPC algorithm to be a single set of instructions in order to avoid overhead due to 'function' calls, which might not even be supported by the PLC.

3.3 Optimization solver

In order to efficiently solve problem (9) while addressing the aforementioned challenges, we propose to use the BVLS solver of (Saraf and Bemporad, 2019), which is computationally efficient, library-free, easy to code in *Structured Text*, and stable in single precision. Under the presence of numerical errors, typically encountered in single-precision, theoretical convergence properties of active-set methods may not hold. In our implementation we use anti-cycling procedures in order to detect numerical errors and to ensure that in such cases the algorithm terminates in finite iterations to a primal feasible (possibly suboptimal) solution of (9).

The proposed BVLS algorithm solves a sequence of over-determined linear systems using recursive thin QR factorization for computational efficiency. Numerical stability for the recursive factorization method is provided using the *reorthogonalization* procedure from (Daniel et al., 1976), which is performed when numerical cancellation is detected. The reader is referred to (Saraf and Bemporad, 2019; Saraf, 2019) for further details on the solver.

4. HARDWARE-IN-THE-LOOP TESTS

4.1 Quadruple water-tank system

We consider the quadruple water-tank system described in (Alvarado et al., 2006), which is shown in Figure 2. This system consists of four water tanks, two of which

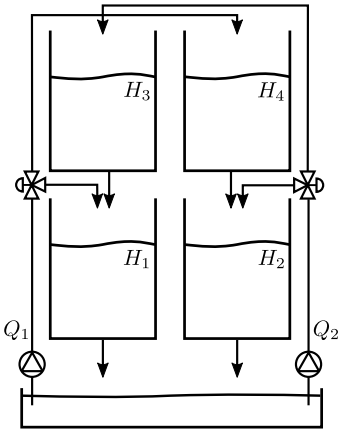

Fig. 2. Quadruple water-tank system.

are located above the other two. Water can be directed to the tanks via two separate pumps, each one of which feeds water into one of the upper tanks and one of the lower ones as shown in the figure. The nonlinear system dynamics are described by the following set of ordinary differential equations

$$A_t \frac{dH_1}{dt} = -a_1\sqrt{2gH_1} + a_3\sqrt{2gH_3} + \gamma_1 \frac{Q_1}{3600} \tag{10a}$$
$$A_t \frac{dH_2}{dt} = -a_2\sqrt{2gH_2} + a_4\sqrt{2gH_4} + \gamma_2 \frac{Q_2}{3600} \tag{10b}$$
$$A_t \frac{dH_3}{dt} = -a_3\sqrt{2gH_3} + (1-\gamma_2) \frac{Q_2}{3600} \tag{10c}$$
$$A_t \frac{dH_4}{dt} = -a_4\sqrt{2gH_4} + (1-\gamma_1) \frac{Q_1}{3600}, \tag{10d}$$

where $H_i[m]$ is the height of water tank i, $Q_i[m^2/h]$ is the water flow rate of pump i and $g = 9.81[m/s^2]$. The parameters of the system are $A_t = 0.03$, $a_1 = 1.3104\cdot 10^{-4}$, $a_2 = 1.5074 \cdot 10^{-4}$, $a_3 = 9.2673 \cdot 10^{-5}$, $a_4 = 8.8164 \cdot 10^{-5}$, $\gamma_1 = 0.3$, and $\gamma_2 = 0.4$.

The control objective is to steer the system output $y = (H_1, H_2)$ to the desired heights by acting on the input $u = (Q_1, Q_2)$. The input and output of the system are subject to the following constraints

$$\begin{aligned} (0,0) \le u_i \le (3, 2.1) \\ (0,0) \le y_i \le (1.2, 1.2). \end{aligned} \tag{11}$$

We obtain a discrete time-invariant linear I/O model (1) of the system around the operating point

$$H_1^0 = 0.7175,\ H_2^0 = 0.7852,\ H_3^0 = 0.6594,\ H_4^0 = 0.8950,$$
$$Q_1^0 = 1.9,\ Q_2^0 = 2.0 \tag{12}$$

in Matlab by using the System Identification Toolbox™. The dataset for identification is obtained by the following procedure:

1 A pseudorandom input sequence $\hat{u}$ with a holding time of 15s is generated around the operating point input $u^0 \doteq (Q_1^0, Q_2^0)$. The range of the signal is $u_i^0 - 0.2 \le \hat{u}_i \le u_i^0 + 0.2$.
2 Signal u_n is obtained by adding white noise to $\hat{u}$.
3 The system is simulated for the input sequence u_n starting at the operating point by numerically integrating (10), providing the output signal $\hat{y}$.
4 Signal y_n is obtained by adding white noise to $\hat{y}$.

5 An I/O model (1) with a sampling time of 5s is obtained with the System Identification Toolbox in Matlab using the input/output sequences $\hat{u}$ and y_n.

This procedure aims at mimicking the data that would be available in a real industrial setting. The amount of white noise added to the signals has a magnitude of 0.2% of the signal value.

We obtain the following I/O model with $n_a = n_b = 4$,

$$A_1 = \begin{bmatrix} 0.68391 & -0.35285 \\ -0.31309 & 0.58412 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0.34456 & -0.09245 \\ -0.08057 & 0.31709 \end{bmatrix}$$
$$A_3 = \begin{bmatrix} -0.06323 & 0.26467 \\ -0.08908 & 0.30807 \end{bmatrix} \quad A_4 = \begin{bmatrix} 0.00723 & 0.16411 \\ 0.45853 & -0.23229 \end{bmatrix}$$
$$B_1 = \begin{bmatrix} 0.01374 & 0.00076 \\ 0.00079 & 0.01816 \end{bmatrix} \quad B_2 = \begin{bmatrix} 0.00359 & 0.00722 \\ 0.00533 & 0.00617 \end{bmatrix}$$
$$B_3 = \begin{bmatrix} -0.00087 & 0.00881 \\ 0.00631 & 0.00077 \end{bmatrix} \quad B_4 = \begin{bmatrix} 0.00098 & 0.00419 \\ 0.00775 & -0.00356. \end{bmatrix},$$

and an 84-85% fit to training and validation data for each output channel.

4.2 Simulation setup

The PLC used is a Modicon M340 from Schneider Electric® equipped with a BMXP3420302 processor module, an AMI0410 analog input module, and an AMO0210 analog output module. The input and output modules of the PLC, which read and write the output and control input of the system, respectively, are connected to a National Instruments USB-6211 data acquisition card, which is in turn connected to a PC where the quadruple water-tank system gets simulated in real-time by integrating the ODEs (10) using Simulink with the QUARC® software package. The interface between the user and the PLC is done by using the PLC's programming software Unity Pro XL.

The program described in Section 3.1 is loaded to the PLC, occupying a total of 35.95Kbytes of data memory (14.04% of the total available data memory) and 312.53Kbytes of program memory (8.14% of the total available program memory). However, we note that an empty project occupies a total of 23.13Kbytes and 160.56Kbytes of data and program memory, respectively, due to the storage of system and diagnostic information. Therefore, the complete MPC setup itself only occupies 12.82Kbytes (5.01%) of data memory and 151.97Kbytes (3.96%) of program memory. [1]

4.3 Hardware-in-the-loop results

We show the results of two closed-loop tests, one for reference tracking and one for disturbance rejection. In each test, the system is initialized at the operating point (12) and with the operating point control input $u^0 = (Q_1^0, Q_2^0)$ being manually applied to it. Then, the PLC is engaged with the reference $r = (H_1^0, H_2^0)$. After some time, either the reference is changed or a disturbance is added to the system.

The tuning parameters of the MPC and observer are $Q = 10I_4$, $R = 5I_2$, $N_p = 9$, $N_u = 2$, $L_1 = 0.5I_2$, $L_d = 0.6I_2$, and $L_i = 0_2$ for $i = 2, 3, 4$ – for which Propositions 1 and 2 are satisfied.

Figures 3 and 4 show the resulting outputs and inputs of a reference tracking test where the reference r is changed from $r = (H_1^0, H_2^0)$ to $r = (0.7754, 0.9)$. Inputs and outputs are represented in solid lines and the reference in dashed lines [2]. As can be seen in the figures, offset-free control is attained in spite of the nonlinearity of the simulated process. Additionally, note that the control input Q_2 has reached its upper bound. The maximum computation time of the controller is 28ms, including an overhead of 9ms.

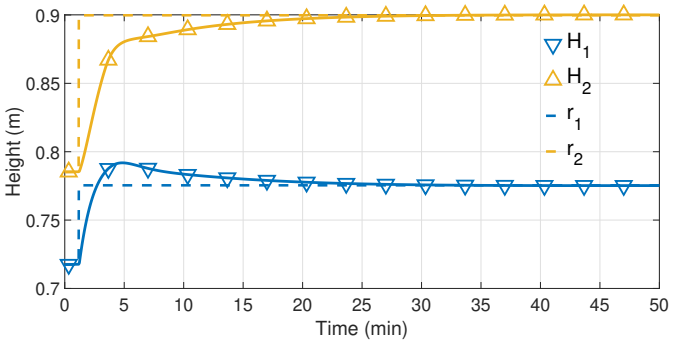

Fig. 3. Reference tracking: system output.

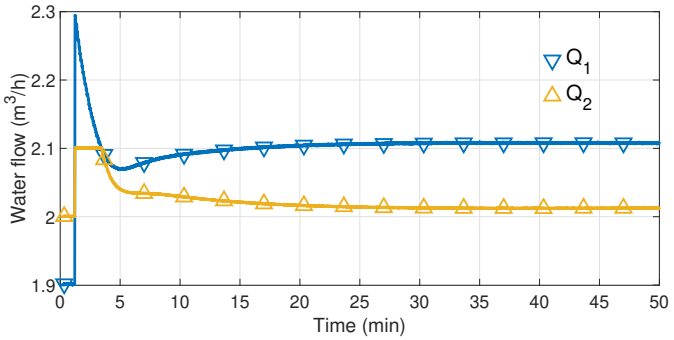

Fig. 4. Reference tracking: system input.

Figures 5 and 6 show the resulting outputs and inputs of a disturbance rejection test where tank 1 receives an instantaneous addition of water, resulting in a sudden increase of H_1. The maximum computation time of the controller is 23ms, including an overhead of 7ms.

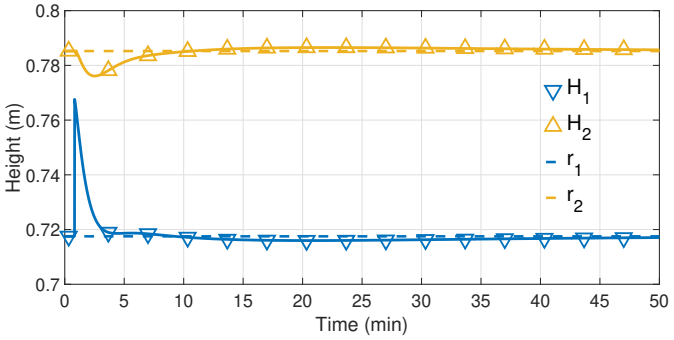

Fig. 5. Disturbance rejection: system output.

[1] Percentages are obtained by taking 1Kbyte = 1024bytes.

[2] The markers are added to help identifying the signals and have no relation with the sample time.

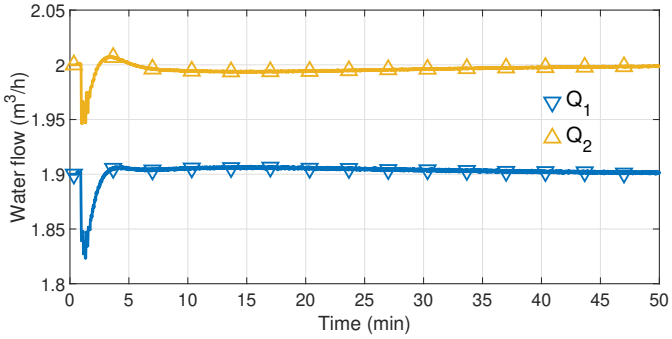

Fig. 6. Disturbance rejection: system input.

The median computation time of the controller is 16ms in both tests, without including the overhead due to other PLC tasks. We note that the overhead of the PLC may vary each cycle, ranging from 6 to 9ms.

5. CONCLUSIONS

The memory requirements of the proposed MPC implementation are well within that of medium-range industrial PLCs such as the one used for the HIL tests. Furthermore, the computational requirements are in the order of milliseconds, making the proposed implementation suitable for the systems typically controlled by PLCs, which rarely require higher sampling frequencies. The computation time and memory requirements could both be further improved with the development of a sparse implementation of the BVLS solver, specific to the structured optimization problem arising in MPC. Additionally, the implementation of nonlinear MPC in a PLC is another future line of work.

REFERENCES

Alphonsus, E.R. and Abdullah, M.O. (2016). A review on the applications of programmable logic controllers (PLCs). *Renewable and Sustainable Energy Reviews*, 60, 1185–1205.

Alvarado, I., Limon, D., Garcia-Gabin, W., Alamo, T., and Camacho, E. (2006). An educational plant based on the quadruple-tank process. *IFAC Proceedings Volumes*, 39(6), 82–87.

Bemporad, A. (2019). Explicit model predictive control. In J. Baillieul and T. Samad (eds.), *Encyclopedia of Systems and Control*, 1–7. Springer, London, UK.

Bemporad, A., Bernardini, D., Long, R., and Verdejo, J. (2018). Model predictive control of turbocharged gasoline engines for mass production. In *WCX^{TM}: SAE World Congress Experience*. Detroit, MI, USA.

Camacho, E. and Bordons Alba, C. (2013). *Model Predictive Control*. Springer Science & Business Media.

Cimini, G., Bemporad, A., and Bernardini, D. (2017). ODYS QP Solver. ODYS S.r.l. (`https://odys.it/qp`).

Daniel, J.W., Gragg, W.B., Kaufman, L., and Stewart, G.W. (1976). Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Mathematics of Computation*, 30(136), 772–795.

Ferreau, H., Kirches, C., Potschka, A., Bock, H., and Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4), 327–363.

Huyck, B., Callebaut, L., Logist, F., Ferreau, H.J., Diehl, M., De Brabanter, J., Van Impe, J., and De Moor, B. (2012). Implementation and experimental validation of classic MPC on programmable logic controllers. In *2012 20th Mediterranean Conference on Control & Automation (MED)*, 679–684.

Krupa, P., Limon, D., and Alamo, T. (2018). Implementation of model predictive controllers in programmable logic controllers using IEC 61131-3 standard. In *2018 European Control Conference (ECC)*, 1–6.

Kufoalor, D.K.M., Binder, B., Ferreau, H.J., Imsland, L., Johansen, T.A., and Diehl, M. (2015). Automatic deployment of industrial embedded model predictive control using qpOASES. In *2015 European Control Conference (ECC)*, 2601–2608.

Maeder, U., Borrelli, F., and Morari, M. (2009). Linear offset-free model predictive control. *Automatica*, 45(10), 2214 – 2222.

Mattingley, J. and Boyd, S. (2012). CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1), 1–27.

Pannocchia, G., Gabiccini, M., and Artoni, A. (2015). Offset-free MPC explained: novelties, subtleties, and applications. *IFAC-PapersOnLine*, 48(23), 342–351.

Patrinos, P. and Bemporad, A. (2013). An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Transactions on Automatic Control*, 59(1), 18–33.

Raha, A., Chakrabarty, A., Raghunathan, V., and Buzzard, G.T. (2019). Embedding approximate nonlinear model predictive control at ultrahigh speed and extremely low power. *IEEE Transactions on Control Systems Technology*.

Saraf, N. (2019). *Bounded-Variable Least-Squares Methods for Linear and Nonlinear Model Predictive Control*. Ph.D. thesis, IMT School for Advanced Studies Lucca.

Saraf, N. and Bemporad, A. (2017). Fast model predictive control based on linear input/output models and bounded-variable least squares. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 1919–1924.

Saraf, N. and Bemporad, A. (2019). A bounded-variable least-squares solver based on stable QR updates. *IEEE Transactions on Automatic Control*.

Tøndel, P., Johansen, T.A., and Bemporad, A. (2003). An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3), 489–497.

Ullmann, F. (2011). FiOrdOs: A MATLAB toolbox for C-code generation for first order methods. *MS thesis*.

Valencia-Palomo, G. and Rossiter, J. (2012). Novel programmable logic controller implementation of a predictive controller based on Laguerre functions and multi-parametric solutions. *IET Control Theory & Applications*, 6(8), 1003–1014.

Wang, Y. and Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2), 267.