Implementation of Model Predictive Control in Programmable Logic Controllers

Pablo Krupa, Daniel Limon, Teodoro Alamo

***Abstract*—In this paper we present an implementation of a low memory footprint Model Predictive Control (MPC) based controller in Programmable Logic Controllers (PLC). Automatic code generation of standardized IEC 61131-3 PLC programming languages is used to solve the MPC's optimization problem online. The implementation is designed for its application in a realistic industrial environment, including timing considerations and accounting for the possibility of the PLC not being exclusively dedicated to the MPC controller. We describe the controller architecture and algorithm, show the results of its memory footprint with regards to the problem dimensions and present the results of its implementation to control a hardware in the loop multivariable chemical plant.**

***Index Terms*—Model Predictive Control, Embedded systems, Programmable Logic Controller, Dual optimization, IEC 61131**

I. Introduction

Model Predictive Control (MPC) is an optimization based control strategy in which the control action is obtained from the solution of an optimization problem where a system model is used to predict the future evolution of the plant over a given prediction horizon. The optimization problem is posed as a minimization problem in which the cost function reflects the distance between the desired reference and the predicted system evolution [1]. One of the main draws of MPC is its inherent ability to consider and satisfy state and input constraints in multivariable systems.

The application of MPC has historically been confined to computationally powerful devices, such as PCs, because it requires the solution of an optimization problem at each sample time. This has limited its application in industrial settings, due to the fact that most control loops are implemented using embedded systems, whose resources are not suitable for solving the MPC optimization problem in real time. Specifically, Programmable Logic Controllers (PLCs) are the most widespread embedded system used in the industry for implementing low level control loops and automatons, due to their high reliability and robustness [2].

The implementation of MPC in PLCs is particularly attractive due to *(i)* their current prevalence in industrial settings, and *(ii)* the fact that they are usually used for implementing control loops at a fairly low level, where the use of MPC could (potentially) provide a performance improvement [3]. We are therefore interested in finding a way to efficiently implement MPC in embedded systems, and particularly interested in its implementation in PLCs for a realistic environment. Paper [4] provides a discussion and state-of-the-art on the implementation in embedded system of reliable MPC controllers in industrial settings and its challenges, including the implementation of MPC in PLCs.

One possible approach for implementing MPC in embedded systems is the use of explicit MPC, where the solutions of the set of MPC optimization problems are computed offline and the control action is then computed online using a lookup table [5]. This approach provides good results for small scale systems, but the computational burden and memory requirements can become prohibitive for higher order ones. Examples of explicit MPC being implemented in embedded systems include [6] and [7].

Another approach is to overcome the computational and memory limitations of embedded systems with the use of efficient algorithms capable of solving the optimization problem online. This has become possible due to recent advances in optimization algorithms for convex optimization problems and to the development of code generation tools that specifically tailor embedded systems, such as, to name a few of the more widespread ones, FiOrdOs [8], CVXGEN [9], FORCES [10] or qpOASES [11]. Examples of these tools being used to implement MPC in embedded platforms include [12], [13], [14] and [15]. Paper [16] provides an overview and comparison of the aforementioned tools for their use in embedded MPC.

Finally, another approach is implementations which are tailored to the specific optimization problem that arises from the MPC formulation, instead of relying on solutions for generic convex optimization problems such as the aforementioned tools. Some of these implementations rely on code generation to further take advantage of the specifics of the implementations, whereas others implement a hand tailored algorithm in the embedded system. Some noteworthy examples are [17], [18] and [19] for PLC implementations; [20], where the code generation tool μAO-MPC [21] is used, [22], [23] and [24] for FPGA implementations; [25] for microcontrollers; and [26], where in spite of the controller not being implemented in an embedded system, the underlying structure of the MPC's optimization problem was exploited.

In this paper we present a code generation tool for the implementation of MPC based controllers on PLCs where the optimization problem is solved online using first order methods that have been specifically tailored to solve the MPCs

Pablo Krupa, Daniel Limon and Teodoro Alamo are at the Systems Engineering and Automation department, University of Seville. `pkrupa@us.es`, `dlm@us.es`, `talamo@us.es`
The authors acknowledge MINERCO and FEDER funds for funding project DPI2016-76493-C3-1-R, and MCIU and FSE for the FPI-2017 grant.

optimization problem efficiently. The tool has been designed to fulfill the following criteria: *(i)* high level problem definition, *(ii)* automatic code generation, *(iii)* streamlined implementation, *(iv)* code tailored to specific MPC formulation and system, *(v)* optimization problem solved online, *(vi)* practical considerations specific to PLCs, *(vii)* the use of the native standardized IEC 61131-3 PLC programming languages, and *(viii)* minimization of the memory footprint and computational complexity of the optimization algorithm. Additionally, the proposed control scheme guarantees offset free tracking for feasible references. This paper extends and improves upon the preliminary results shown in the conference paper [27].

One of the main advantages of the proposed implementation is that the memory footprint grows linearly with respect to the prediction horizon of the MPC due to taking advantage of the structure of the controller's optimization problem by means of a banded Cholesky factorization of one of the ingredients of the optimization algorithm. This approach can be applied to a wide range of MPC formulations. This paper focuses on two of them.

To the best of the author's knowledge no previous work collectively addresses all the aforementioned points in order to deliver an MPC implementation tailored to a realistic industrial implementation. The objective is to develop a tool that would allow a streamlined way to implement a state-space model predictive controller in a PLC. To this end, we present the results of a hardware-in-the-loop (HIL) implementation of the proposed controller to control a multivariable chemical plant.

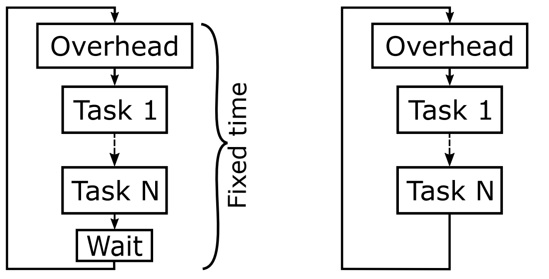
The paper is structured as follows. Section II presents the architecture of the controller. In Section III, we present the optimization algorithms that can be implemented. In Section IV we show how the optimization problems described in Section II are implemented using a minimal memory footprint, including how linear memory growth with respect to the prediction horizon is achieved. Section V describes the overall controller implementation, including its general algorithm. Section VI shows the results of a hardware in the loop implementation of the controller on a chemical plant. Section VII shows the results of tests studying the memory requirements of the controller with regards to the problem dimensions. Finally, Section VIII summarizes the main conclusions.

Additionally, we include the following subsection, which provides some details about PLCs which are of importance for contextualizing and illustrating the design decisions described in the remainder of this paper.

Programmable Logic Controllers

The construction guidelines and requirements of PLCs are standardized by the international norm IEC 61131, which includes the description of its standard programming languages. Some high-end PLCs also offer the possibility of programming in C/C++. However, all modern PLCs are programmable using the standard IEC 61131 programming languages. For future reference, we highlight the following two:

- **Structured Text (ST)**. ST is a high level sequential text-based programming language which resembles Pascal.
- **Function Block Diagram (FBD)**. FBD is a graphical language in which functions are represented by blocks with a series of inputs and outputs connected to those of other blocks, fixing a layout that determines the execution order. Each block calls a function, which can be programmed in any of the five IEC 61131 programming languages.



(a) Periodic mode (b) Cyclic mode

Fig. 1: PLC's main loop operation modes.

The basic operating cycle of a PLC is the following,

1) Perform diagnosis and overhead operations.
2) Digital and analog input signals are sampled and their values are stored in memory.
3) User defined tasks are executed sequentially in a pre-specified order.
4) Values from the outputs of the user defined programs are written to the PLC's output connections.

This cycle can be run in two different modes: *cyclic* mode, where the first point of the operating cycle starts as soon as the last one has finished; or *periodic* mode, where the cycle runs periodically with a fixed time. Figure 1 illustrates both operating modes. The diagnosis and overhead operations incur in a computational overhead typically in the order of milliseconds, which can limit the application of PLCs to systems with high sampling frequencies.

Notation: Set $\mathbb{Z}_+$ is the set of non negative integer numbers. $(x_{(1)}, x_{(2)}, \dots, x_{(N)})$ is a column vector formed by the concatenation of column vectors $x_{(1)}$ to $x_{(N)}$. Depending on the context, x_i can either denote the i-th element of vector x, or the i-th element of a sequence $\mathbf{x} = \{x_0, x_1, \dots, x_N\}$. Given a matrix $A \in \mathbb{R}^{n \times m}$, $A_{i,j}$ denotes its element (i, j), $A^\top$ its transposed and A^{-1} its inverse (if A is square). The identity matrix of dimension n is denoted by I_n. Given $x \in \mathbb{R}^n$, $\|x\|_Q$ is the weighted Euclidean norm $\|x\|_Q \doteq \sqrt{x^\top Q x}$, and $\|x\| \doteq \|x\|_{I_n}$ is the standard Euclidean norm. $\langle x, y \rangle$ is the inner product of vectors x and y, i.e. $\langle x, y \rangle = x^\top y$. For a vector x, $x(k|j)$ denotes the prediction of x at time k based on the knowledge available at time j. $\mathcal{D}^n_{(+)}$ is the set of (strictly) positive definite diagonal matrices of dimension n. $A \succ 0$ denotes that matrix A is strictly positive definite. Given two vectors x and y, $x \leq (\geq)\ y$ denotes componentwise inequality.

II. Controller Architecture

This section presents the controller's architecture, which is shown in Figure 2 within the dashed line. The blocks in the figure represent each of the elements included in the controller.

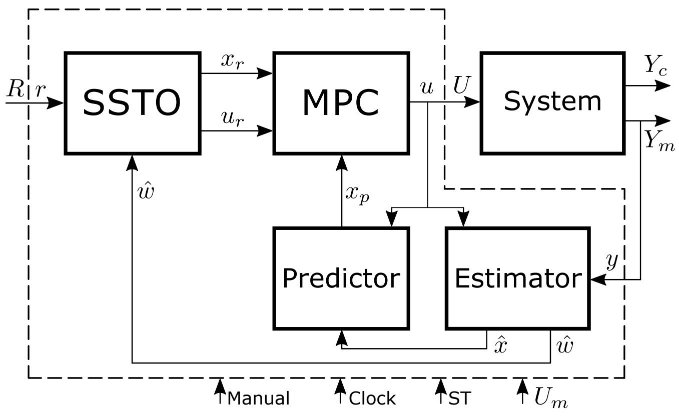


Fig. 2: Controller architecture.

We consider a plant with measured output $Y_m \in \mathbb{R}^p$, constrained outputs $Y_c \in \mathbb{R}^{n_c}$ and input $U \in \mathbb{R}^m$, where the input U and output Y_c are subject to hard constraints,

$$\underline{Y}_c \leq Y_c \leq \overline{Y}_c, \quad \underline{U} \leq U \leq \overline{U}.$$

It is assumed that the plant is operating around a steady state U^0, Y_m^0, Y_c^0, and that a linearized state-space model of the system around the operating point is available.

The objective is to develop an MPC based controller to steer the controlled system output, which is assumed to be the measured output Y_m, to the given reference $R \in \mathbb{R}^p$ (if it is a feasible reference) while satisfying the system constraints. The controller must guarantee offset free tracking for feasible references and delay compensation.

A. Prediction Model

The prediction model is a discrete linear time invariant model linearized around the operating point U^0, Y_m^0, Y_c^0. The inputs and outputs are scaled in order to reduce possible numerical issues by means of the diagonal scaling matrices N_u, N_m and N_c, resulting in the linearized variables $u(k) = N_u\left(U(k) - U^0\right)$, $y_m(k) = N_m\left(Y_m(k) - Y_m^0\right)$, $y_c(k) = N_c\left(Y_c(k) - Y_c^0\right)$, where k is the sample time counter. The resulting model can be written as,

$$x(k+1) = Ax(k) + Bu(k-d) \tag{1a}$$
$$y_m(k) = Cx(k), \tag{1b}$$

where $x(k) \in \mathbb{R}^n$ is the state at sample time k, and $d \in \mathbb{Z}_+$ is the system delay, measured in number of sample times.

We consider the following box constraints,

$$\underline{x} \leq x(k) \leq \overline{x} \tag{2a}$$
$$\underline{u} \leq u(k) \leq \overline{u}, \tag{2b}$$

where $\underline{u} = N_u\left(\underline{U} - U^0\right)$, $\overline{u} = N_u\left(\overline{U} - U^0\right)$.

We note that, in order to enforce the constraints on Y_c, the state x is selected so that the linearized constrained outputs y_c are contained within it, i.e. for each $i \in \{1, \dots, n_c\}$ we have that $y_{c,i} \equiv x_j$ for some $j \in \{1, \dots, n\}$. The constraints on y_c can then be written as constraints on x by setting $\underline{x}_j = N_{c,i}(\underline{Y}_{c,i} - Y_{c,i}^0)$ and $\overline{x}_j = N_{c,i}(\overline{Y}_{c,i} - Y_{c,i}^0)$ accordingly. Note that some elements of x may be unbounded.

B. State and disturbance estimator

We consider the following augmentation of system (1),

$$x(k+1) = Ax(k) + Bu(k-d) \tag{3a}$$
$$y_m(k) = Cx(k) + w(k) \tag{3b}$$
$$w(k+1) = w(k), \tag{3c}$$

where $w(k)$ is the output disturbance at sample time k, which is included to account for steady-state offset due to the mismatch between the linear model and the plant, and we slightly abuse notation by using the same symbols as in (1).

At each sample time k, the state and disturbance estimator (henceforth *estimator*), calculates the predicted system state $\hat{x}(k+1) \in \mathbb{R}^n \doteq x(k+1|k)$ and the predicted disturbance $\hat{w}(k+1) \in \mathbb{R}^p \doteq w(k+1|k)$, with x given by (3a) and w by (3c). The predicted state and disturbance are calculated from,

$$\hat{x}(k+1) = A\hat{x}(k) + Bu(k-d) + L_x(y_m(k) - C\hat{x}(k) - \hat{w}(k)) \tag{4a}$$
$$\hat{w}(k+1) = \hat{w}(k) + L_w\left(y_m(k) - C\hat{x}(k) - \hat{w}(k)\right), \tag{4b}$$

where $y_m(k)$ is the linearized measured system output at sample time k and matrices L_x and L_w are calculated so that the estimator is stable and $\hat{w}$ can be used to achieve offset free control by following the indications in [28].

C. Steady State Target Optimizer

The Steady State Target Optimizer (SSTO) computes, at each sample time k, the feasible steady state $(x_r(k), u_r(k))$ of prediction model (1) that minimizes the distance between $y_r(k) \doteq Cx_r(k) + \hat{w}(k)$ and the filtered reference $r_f(k)$,

$$r_f(k) = \eta_r r_f(k-1) + (1-\eta_r)r(k), \tag{5}$$

where $0 \leq \eta_r < 1$, $r(k) = N_m(R(k) - Y_m^0)$, and the inclusion of $\hat{w}$ from (4b) serves to provide offset-free control under the conditions described in [28].

Pair $(x_r(k), u_r(k))$ is taken as the pair (x_r^*, u_r^*) obtained from solving the following optimization problem,

$$(x_r^*, u_r^*, h^*) = arg \min_{x_r, u_r, h} \|h\|_{T_h}^2 + \|u_r\|_{T_u}^2 \tag{6}$$
$$s.t. \quad x_r = Ax_r + Bu_r \tag{6a}$$
$$h = r_f(k) - Cx_r - \hat{w}(k) \tag{6b}$$
$$\underline{x} \leq x_r \leq \overline{x} \tag{6c}$$
$$\underline{u} \leq u_r \leq \overline{u}, \tag{6d}$$

where $T_h \in \mathcal{D}_+^p$ and $T_u \in \mathcal{D}^m$.

Remark 1. *If $p < m$, the system is controllable, and C is full rank, there are additional degrees of freedom, i.e. there are infinite inputs u_r such that the steady state (x_r, u_r) satisfies $r_f(k) = y_r(k)$. In this case, the term $\|u_r\|_{T_u}^2$ is included, with $T_u \in \mathcal{D}_+^m$, so that the optimal set is a singleton. In any other case, the inclusion of term $\|u_r\|_{T_u}^2$ may provide (x_r^*, u_r^*, h^*) that does not minimize $\|r_f(k) - y_r(k)\|$. This can be avoided by taking $T_u = 0$.*

D. Open loop predictor

The open loop predictor calculates, at each sample time k, the predicted system state $x_p(k) = x(k+d|k) \in \mathbb{R}^n$, where d is the system delay. The predicted state is used in the MPC controller instead of the current state. The advantage of doing so is that the prediction model used by the MPC does not depend on the system delay, as can be seen in the following section. This results in the MPC's optimization problem not depending on the value of d. As such, its structure can be more easily exploited. The use of the open loop predictor enables the implementation of memory efficient controllers for systems with large delays [29].

At each sample time k, the predicted state is calculated as,

$$x_p(k) = A^d \hat{x}(k) + \sum_{i=1}^{d} A^{d-i} B u(k-d-1+i), \tag{7}$$

where $\hat{x}$ is the estimated state (4a). In order to avoid storing matrices $A^{d-1} \dots A$, $x_p(k)$ is calculated recursively using Algorithm 1. Note that $x_p(k) = \hat{x}(k)$ if $d = 0$.

Algorithm 1: Open loop predictor

```
  Require: $\hat{x}(k)$, $[u(k-d), \dots, u(k-1)]$
1 $x_p(k) = \hat{x}(k)$
2 for $i = 1$ to $d$ do
3     $x_p(k) = A x_p(k) + B u(k-d-1+i)$
4 end for
  Output: $x_p(k)$
```

E. Model Predictive Control

We consider two possible MPC formulations, given respectively by the two following optimization problems:

$$\min_{\mathbf{x},\mathbf{u}} \sum_{i=0}^{N-1} \|x_i - x_r(k)\|_Q^2 + \|u_i - u_r(k)\|_R^2 \tag{8}$$

$$s.t.\ x_{i+1} = Ax_i + Bu_i, \tag{8a}$$

$$\underline{u} \le u_i \le \overline{u}, \quad i = 0, \dots, N-1 \tag{8b}$$

$$\underline{x} \le x_i \le \overline{x}, \quad i = 1, \dots, N-1 \tag{8c}$$

$$x_0 = x_p(k) \tag{8d}$$

$$Ax_{N-1} + Bu_{N-1} = x_r(k), \tag{8e}$$

and

$$\min_{\mathbf{x},\mathbf{u}} \sum_{i=0}^{N-1} \|x_i - x_r(k)\|_Q^2 + \|u_i - u_r(k)\|_R^2 + \|x_N - x_r(k)\|_T^2 \tag{9}$$

$$s.t.\ x_{i+1} = Ax_i + Bu_i, \tag{9a}$$

$$\underline{u} \le u_i \le \overline{u}, \quad i = 0, \dots, N-1 \tag{9b}$$

$$\underline{x} \le x_i \le \overline{x}, \quad i = 1, \dots, N-1 \tag{9c}$$

$$x_0 = x_p(k), \tag{9d}$$

where here x_i and u_i denote the elements of the sequence of predicted states $\mathbf{x} = \{x_0, x_1, \dots, x_{N-1}\}$ and system inputs $\mathbf{u} = \{u_0, u_1, \dots, u_{N-1}\}$, respectively. Vectors $x_r(k)$ and $u_r(k)$ are the state and input references given by the SSTO (6), respectively. Vector $x_p(k)$ is the predicted state provided by the predictor (7). Matrices $Q \in \mathcal{D}_+^n$, $R \in \mathcal{D}_+^m$ and $T \in \mathcal{D}_+^n$ are the cost function matrices.

In formulation (8), constraint (8e) is included in order to guarantee stability [30]. However, the resulting MPC typically requires long prediction horizons in order for the optimization problem to be feasible, since the state must be able to reach the reference in N steps.

For this reason, we also include MPC formulation (9), which drops the terminal constraint (8e) in favor of a terminal cost $\|x_N - x_r(k)\|_T^2$. This formulation will be more suitable when dealing with systems for which the prediction horizon of formulation (8) would need to be set prohibitively high. The stability of (9) can be guaranteed by a proper selection of the cost function matrix T [31].

The solution of (8) or (9) provides the sequences $\mathbf{x}^* = \{x_0^*, \dots, x_{N-1}^*\}$ and $\mathbf{u}^* = \{u_0^*, \dots, u_{N-1}^*\}$. At each sample time k, the control input $u(k)$ is taken as u_0^*.

The optimization problems of the SSTO (6) and MPC (8), (9) can be posed as QP problems which are solved online using one of the algorithms described in the following section.

III. Dual Optimization Methods

This section describes two optimization algorithms for solving Quadratic Programming (QP) problems using the dual formulation: FISTA [32] and ADMM [33], [34].

Consider the following QP problem,

$$\min_{z} \frac{1}{2} z^\top H z + q^\top z \tag{10}$$

$$s.t.\ z \in \mathcal{Z} \tag{10a}$$

$$Gz = b, \tag{10b}$$

where $z \in \mathbb{R}^{n_z}$ are the decision variables and $G \in \mathbb{R}^{m_z \times n_z}$.

Assumption 1. *We assume that,*

(i) The Hessian $H \in \mathcal{D}_+^{n_z}$.
(ii) $\mathcal{Z} = \{z \in \mathbb{R}^{n_z} \mid \underline{z} \le z \le \overline{z}\}$ defines a set of box constraints.
(iii) $n_z > m_z$ and rank$(G) = m_z$.
(iv) There exists $\hat{z} \in ri(\mathcal{Z})$ such that $G\hat{z} = b$, where $ri(\cdot)$ stands for relative interior.

Let $J(z) = \frac{1}{2} z^\top H z + q^\top z$ and z^* be the cost function and optimal solution of problem (10), respectively. We denote $J^* = J(z^*)$ the optimal cost.

We note that if the Hessian H is not diagonal, but is positive definite, then the problem can re recast into a QP that also satisfies Assumption 1.i by means of an appropriate addition of decision variables and equality constraints.

A. Fast Iterative Shrinking-Threshold Algorithm

We present a variation on the Fast Iterative Shrinking-Threshold Algorithm (FISTA) [32], which is an accelerated gradient method based on Nesterov's fast gradient method [35].

Consider QP problem (10) where Assumption 1 holds. Let the Lagrange function $L : \mathbb{R}^{n_z} \times \mathbb{R}^{m_z} \to \mathbb{R}$, the dual function $\psi : \mathbb{R}^{m_z} \to \mathbb{R}$, and $z(\lambda)$ be defined as,

$$L(z, \lambda) = \tfrac{1}{2} z^\top H z + q^\top z - \lambda^\top (Gz - b) \tag{11}$$

$$\psi(\lambda) = \inf_{z \in \mathcal{Z}} L(z, \lambda) \tag{12}$$

$$z(\lambda) = \arg\min_{z \in \mathcal{Z}} L(z, \lambda), \tag{13}$$

where $\lambda \in \mathbb{R}^{m_z}$ is the dual variable.

Due to Assumption 1, we have that strong duality holds [36, Proposition 5.3.3], i.e. $J^* = \psi(\lambda^*)$ and $z^* = z(\lambda^*)$ with,

$$\lambda^* = \arg\max_{\lambda} \psi(\lambda). \tag{14}$$

Definition 1. *Given a QP problem (10) with Hessian $H \in \mathcal{D}_+^{n_z}$ and equality constraint matrix $G \in \mathbb{R}^{m_z \times n_z}$, we denote W_H as,*

$$W_H = G H^{-1} G^\top \in \mathbb{R}^{m_z \times m_z}. \tag{15}$$

Additionally, given a matrix W_H, we denote by "W_H-system" a system of equations of the form $W_H y = c$.

Given Assumption 1 and [37, Lemma 3.1], we have that $\psi(\cdot)$ is a strongly smooth function, i.e. it is continuously differentiable with a Lipschitz gradient. Moreover, it can be shown that the smoothness parameter is matrix W_H given by Definition 1, i.e. $\psi(\cdot)$ satisfies

$$\psi(\lambda+\Delta\lambda) \geq \psi(\lambda) - \Delta\lambda^\top (Gz(\lambda) - b) - \frac{1}{2} \Delta\lambda^\top W_H \Delta\lambda. \tag{16}$$

Therefore, given λ_k, a value of $\lambda_{k+1} = \lambda_k + \Delta\lambda_k$ such that $\psi(\lambda_{k+1}) \geq \psi(\lambda_k)$ can be obtained by taking $\Delta\lambda_k$ as the solution of,

$$\Delta\lambda_k = \arg\max_{\Delta\lambda} -\Delta\lambda^\top (Gz(\lambda_k) - b) - \frac{1}{2} \Delta\lambda^\top W_H \Delta\lambda, \tag{17}$$

whose solution can be obtained by solving the system of equations,

$$W_H \Delta\lambda_k = -(Gz(\lambda_k) - b). \tag{18}$$

The optimum $\psi(\lambda^*)$ is attained when $\psi(\lambda_{k+1}) = \psi(\lambda_k) = \psi(\lambda^*)$, which occurs when $\Delta\lambda_k = 0$, or alternatively when $Gz(\lambda_k) - b = 0$. Since this condition is impractical, the suboptimal solution λ_k that satisfies,

$$\| Gz(\lambda_k) - b \| \leq \epsilon, \tag{19}$$

is taken instead, where $\epsilon > 0$ is an accuracy parameter and here $\| \cdot \|$ can be any norm.

Remark 2. *Note that if Assumption 1 holds, the elements of $z(\lambda)$, denoted as $z(\lambda)_i$ for $i = 1 \ldots n_z$, can be calculated by solving the following optimization problems, since there is no coupling between variables $z(\lambda)_i$,*

$$z(\lambda)_i = \arg\min_{\hat{z} \in \mathbb{R}} \frac{1}{2} H_{i,i} \hat{z}^2 + \left(q_i - \sum_{j=1}^{m_z} G_{j,i} \lambda_j \right) \hat{z} \tag{20}$$

$$s.t. \quad \underline{z}_i \leq \hat{z} \leq \overline{z}_i, \tag{20a}$$

which has the explicit solution,

$$z(\lambda)_i = \max \left\{ \min \left\{ \frac{-\left(q_i - \sum\limits_{j=1}^{m_z} G_{j,i} \lambda_j \right)}{H_{i,i}},\ \overline{z}_i \right\},\ \underline{z}_i \right\}. \tag{21}$$

Algorithm 2 shows FISTA algorithm. The convergence rate of this algorithm has been well established [38], [39].

Algorithm 2: FISTA

```
   Require: $\lambda_0$
1  $k = 0$, $\eta_0 = \lambda_0$, $t_0 = 1$
2  repeat
3      $k = k + 1$
4      Obtain $z(\lambda_{k-1})$ using (21)
5      Obtain $\Delta\lambda_{k-1}$ by solving $W_H$-system (18)
6      $\eta_k = \lambda_{k-1} + \Delta\lambda_{k-1}$
7      $t_k = \frac{1}{2}\big(1 + \sqrt{1 + 4t_{k-1}^2}\big)$
8      $\lambda_k = \eta_k + \frac{t_{k-1} - 1}{t_k}(\eta_k - \eta_{k-1})$
9      Compute residual $\Gamma = Gz(\lambda_k) - b$
10 until $\|\Gamma\| \leq \epsilon$
   Output: $z^* = z(\lambda_k)$
```

B. Alternating Direction Method of Multipliers

We present an implementation of the Alternating Direction Method of Multipliers (ADMM) algorithm based on Algorithm 1 in [33].

Let Assumption 1 hold and consider the following equivalent formulation of problem (10),

$$\min_{z,v} \frac{1}{2} z^\top \hat{H} z + \frac{1}{2} v^\top \hat{H} v + q^\top v \tag{22}$$

$$s.t.\ v \in \mathcal{Z} \tag{22a}$$

$$Gz = b \tag{22b}$$

$$v - z = 0, \tag{22c}$$

where $\hat{H} = H/2$.

The Lagrangian function $L : \mathbb{R}^{n_z} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_z} \to \mathbb{R}$ is

$$L(z, v, \lambda) = \frac{1}{2} z^\top \hat{H} z + \frac{1}{2} v^\top \hat{H} v + \langle q, v \rangle + \langle \lambda, z - v \rangle. \tag{23}$$

For some given $v_{k-1} \in \mathcal{Z}$ and λ_{k-1}, the step of ADMM applied to (22) is given by,

$$z_k = \arg\min_{z} \frac{1}{2} z^\top \hat{H} z + \langle \lambda_{k-1}, z \rangle + \frac{\beta}{2} \| z - v_{k-1} \|^2 \quad s.t.\ Gz = b \tag{24a}$$

$$v_k = \arg\min_{v} \frac{1}{2} v^\top \hat{H} v + q^\top v + \langle \lambda_{k-1}, -v \rangle + \frac{\beta}{2} \| z_k - v \|^2 \quad s.t.\ v \in \mathcal{Z} \tag{24b}$$

$$\lambda_k = \lambda_{k-1} + \beta(z_k - v_k), \tag{24c}$$

where $\beta > 0$ is the penalty parameter.

Step (24a) can be expressed as,

$$z_k = \arg\min_{z} \frac{1}{2} z^\top \hat{H}_\beta z + q_{z,k}^\top z \tag{25}$$

$$s.t.\ Gz = b, \tag{25a}$$

where $\hat{H}_\beta = \hat{H} + \beta I_{n_z}$ and $q_{z,k} = \lambda_{k-1} - \beta v_{k-1}$.

From [40, Subsection 10.1.1] we have the following proposition, which states the optimality conditions of (25).

Proposition 1. *Consider problem (25) for which Assumption 1 holds. Then, z_k is optimal if and only if there is a $\mu_k \in \mathbb{R}^{m_z}$ such that,*

$$Gz_k = b \tag{26}$$

$$\hat{H}_\beta z_k + q_{z,k} + G^\top \mu_k = 0, \tag{27}$$

which using simple algebra leads to,

$$W_{\hat{H}_\beta} \mu_k = -(G\hat{H}_\beta^{-1} q_{z,k} + b) \tag{28}$$

$$z_k = -\hat{H}_\beta^{-1}(G^\top \mu_k + q_{z,k}). \tag{29}$$

Step (24b) can be expressed as,

$$v_k = \arg\min_v \frac{1}{2} v^\top \hat{H}_\beta v + q_{v,k}^\top v \tag{30}$$

$$s.t.\ v \in \mathcal{Z}, \tag{30a}$$

where $q_{v,k} = q - \lambda_{k-1} - \beta z_k$.

Note that Remark 2 can also be applied to the elements of v_k, denoted as $(v_k)_i$ for $i = 1 \dots n_z$, leading to,

$$(v_k)_i = \max\left\{ \min\left\{ \frac{-(q_{v,k})_i}{(\hat{H}_\beta)_{i,i}},\ \overline{z}_i \right\},\ \underline{z}_i \right\}. \tag{31}$$

Algorithm 3 shows the ADMM algorithm, which makes use of (28), (29) and (31). The convergence rate of this algorithm has been well established, see [33] and [41] - which also discusses the selection of parameter β.

Algorithm 3: ADMM

```
Require: $v_0 \in \mathcal{Z}$, $\lambda_0$, $\beta > 0$
1 $k = 0$
2 repeat
3     $k = k + 1$
4     Obtain $\mu_k$ by solving $W_{\hat{H}_\beta}$-system (28)
5     $z_k = -\hat{H}_\beta^{-1}(G^\top \mu_k + q_{z,k})$
6     Obtain $v_k$ using (31)
7     $\lambda_k = \lambda_{k-1} + \beta(z_k - v_k)$
8     Compute residual $\Gamma = z_k - v_k$
9 until $||\Gamma|| \leq \epsilon$
Output: $z^* = z_k$
```

Remark 3. *This section describes an implementation of ADMM algorithm derived from Alg. 1 in [33]. There are other possible implementations with comparable implementation complexity, such as Alg. 7 in [33].*

Remark 4. *Note that both of the above algorithms show similar complexity with regards to their implementation in an embedded platform. Step 4 from FISTA algorithm is equivalent to step 6 from ADMM algorithm. Furthermore, notice the resemblance between step 5 of FISTA algorithm and step 4 of ADMM algorithm. Both steps require solving a W-system of equations with the same matrix G and different matrix H. The main complication in the implementation of these algorithms in an embedded system is finding an efficient way to solve these W-system of equations with a low memory footprint. This problem is addressed in the next section.*

Remark 5. *We remark that the convergence rate of these methods are at most linear. However, for the purposes of the MPC optimization problems, we find that the control action returned by the algorithms converges very quickly to a value very close to the optimal one – which is enough for a practical setting. This is especially true for MPC formulation (9). As such, very small exit tolerances are not typically needed, and the number of iterations of the proposed algorithms are therefore reasonable. The use of one formulation over another and the selection of an appropriate exit tolerance will depend upon the particular system to be controlled. The linear convergence of FISTA algorithm could be guaranteed by the inclusion of a restart scheme [42], [43].*

IV. Embedded implementation of the Optimization Problems

This section describes how optimization problems (6), (8) and (9) are solved using Algorithms 2 or 3 in the PLC.

In order to achieve this, three conditions must be met: *(i)* the optimization problem must be strictly feasible, *(ii)* it must be expressed as an equivalent QP problem (10) in which Assumption 1 holds and *(iii)* a memory efficient way must be found for solving the W-system of the resulting QP.

A. Implementation of the MPC's optimization problem

The MPC's optimization problems (8) and (9) can be expressed as an equivalent QP problem (10) where Assumption 1 holds if $Q \in \mathcal{D}_+^n$, $R \in \mathcal{D}_+^m$ and $T \in \mathcal{D}_+^n$.

The simplest way to solve a generic W-system would be to compute $W^{-1} \in \mathbb{R}^{(N+1)n \times (N+1)n}$ offline and store it in the PLC. However, this would result in high memory consumption for average sized problems due to the quadratic memory growth with respect to N and n.

We propose using the Cholesky decomposition of W, denoted W_c,

$$W = W_c^\top W_c, \tag{32}$$

in order to efficiently find the solution of the W-system.

To illustrate this we show the structure of the W_H matrix that results from the QP problem of formulation (8) that is used in FISTA algorithm (Alg. 2), the structure of the Cholesky decomposition of W_H and how to compute $\Delta\lambda_k$. The following discussion would be very similar if we where considering MPC formulation (9) instead.

$$W_H = \begin{pmatrix} Q^{-1} & Q^{-1}A^\top & .. & .. & 0 & 0 \\ AQ^{-1} & \delta & -Q^{-1}A^\top & .. & .. & 0 \\ .. & -AQ^{-1} & \delta & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & \delta & -Q^{-1}A^\top \\ 0 & 0 & .. & .. & -AQ^{-1} & \delta - Q^{-1} \end{pmatrix}, \tag{33}$$

$$\delta = AQ^{-1}A^\top + BR^{-1}B^\top + Q^{-1}. \tag{34}$$

The Cholesky decomposition of W_H, denoted $W_{H,c}$, presents the upper-triangular block-diagonal structure,

$$W_{H,c} = \begin{pmatrix} \beta_0 & \alpha_1 & .. & .. & 0 & 0 \\ .. & \beta_1 & \alpha_2 & .. & .. & 0 \\ .. & .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & \beta_{N-1} & \alpha_N \\ 0 & 0 & .. & .. & .. & \beta_N \end{pmatrix}, \tag{35}$$

where we define the sets of matrices $\mathcal{A} = \{\alpha_1, \dots, \alpha_N\}$, $\alpha_i \in \mathbb{R}^{n \times n}$; and $\mathcal{B} = \{\beta_0, \dots, \beta_N\}$, $\beta_i \in \mathbb{R}^{n \times n}$.

Vector $\Delta\lambda$ can then be computed by solving two consecutive systems of equations,

$$W_{H,c}^\top \Delta\hat{\lambda}_k = -(Gz(\lambda_k) - b) \tag{36a}$$

$$W_{H,c} \Delta\lambda_k = \Delta\hat{\lambda}_k, \tag{36b}$$

which are easy to solve due to $W_{H,c}$ being an upper-triangular sparse matrix. Moreover, only matrices α_i and β_i need to be stored, eliminating the quadratic memory growth with respect to N.

Previous works have exploited the band structure of the MPC's optimization problem, such as in [26], where LDL$^\top$ and Cholesky factorizations are used in order to reduce the computational complexity of the algorithm.

Remark 6. *The same approach can be applied to solve the $W_{\hat{H}_\beta}$-system that arises in the ADMM algorithm when calculating μ_k, since $W_{\hat{H}_\beta}$ is equivalent to (33) (34) substituting Q for $\hat{Q}_\beta = Q/2 + \beta I_n$ and R for $\hat{R}_\beta = R/2 + \beta I_m$.*

B. Implementation of the SSTO's optimization problem

Optimization problem (6) cannot be transformed into an equivalent QP problem (10) in which Assumption 1 holds due to the decision variable x_r not being penalized in the cost function.

For this reason, we propose solving optimization problem (6) by using Algorithm 4 instead, which iteratively solves,

$$\begin{aligned}(x^*_{r,j}, u^*_{r,j}, h^*_j) = \arg \min_{x_r, u_r, h} & \|h\|^2_{T_h} + \|x_r - x^c_{j-1}\|^2_{Q_r} \quad (37)\\ & + \|u_r - u^c_{j-1}\|^2_{R_r} \\ s.t. \quad & x_r = Ax_r + Bu_r \quad (37a)\\ & h = r_f(k) - Cx_r(k) - \hat{w}(k) \quad (37b)\\ & \underline{x} \leq x_r \leq \overline{x} \quad (37c)\\ & \underline{u} \leq u_r \leq \overline{u}, \quad (37d)\end{aligned}$$

where $x^c_{j-1} \in \mathbb{R}^n$ and $u^c_{j-1} \in \mathbb{R}^m$ are the regularizing points for x_r and u_r, respectively, at iterate j.

Problem (37) can be expressed as a QP (10) that satisfies Assumption 1 by taking $Q_r \in \mathcal{D}^n_+$, $R_r \in \mathcal{D}^m_+$ and $T_h \in \mathcal{D}^p_+$.

Algorithm 4 provides a suboptimal solution of the original optimization problem (6), where the suboptimality is characterized by the value of the accuracy parameter $\epsilon_r > 0$.

If the conditions of Remark 1 hold, term $||u_r||^2_{T_u}$ can be included by taking $R_r = T_u$ and eliminating Step 7, so that $u_{c,j} = 0 \, \forall j$.

Remark 7. *Instead of using Algorithm 4 in order to obtain the solution of (6), the cost function of (6) could have been chosen as $||x_r||^2_{Q_r} + ||u_r||^2_{R_r} + ||h||^2_{T_h}$, with diagonal positive definite cost function matrices. Using this cost function, the problem can be transformed into an equivalent QP problem (10) in which Assumption 1 holds. However, the solution of this problem would not minimize the distance $\|r_f(k) - y_r(k)\|$ due to the penalization of x_r.*

Algorithm 4: SSTO

```
  Require: $r(k)$, $\hat{w}(k)$, $x^c_0$, $u^c_0$
1 Compute $r_f(k)$ from (5)
2 $j = 0$
3 repeat
4 |   $j = j + 1$
5 |   Obtain $x^*_{r,j}$ and $u^*_{r,j}$ by solving (37)
6 |   $x^c_j = x^*_{r,j}$
7 |   $u^c_j = u^*_{r,j}$
8 until $||x^c_j - x^c_{j-1}|| \leq \epsilon_r$
  Output: $x_r(k) = x^*_{r,j}$, $u_r(k) = u^*_{r,j}$
```

Due to the size and structure of optimization problem (37), the use of the Chollesky decomposition of W provides no memory benefits. Therefore, the inverse of W is used to solve the W-system in one step. Additionally, the SSTO has a warmstart procedure in which the unconstrained version is solved, which is an easier problem. The SSTO in only called if the result of the unconstrained problem violates constraints.

V. Controller implementation in the PLC

Previous sections have described the individual ingredients of the MPC based controller. This section addresses the implementation of the overall controller in the PLC. The following discussion considers the use of FISTA algorithm (Alg. 2) and MPC formulation (8). The results would be very similar for any other combination of gradient method (GM) and MPC formulation.

A. Controller generation and implementation

In order to provide an efficient and low memory footprint program, an automatic code generating tool, that particularizes the code for the given system, has been developed in Matlab. The tool generates code in the programming languages defined in the IEC 61131-3 standard. Specifically, the controller is programmed as an FBD block that contains the controller's algorithm written in *Structured Text*.

The tool receives the model matrices, the system bounds and the controller's parameters, such as the cost function matrices or the accuracy parameters, and generates the code of the controller and the variable declaration, both of which are packaged into a file that can be directly imported into the PLC's IDE software. Currently, controllers can be generated for Schneider Electric PLCs, which use software *Unity Pro XL*, and for the PLC programming software *CODESYS*.

Figure 3 shows the FBD block of the controller that appears once imported, where U is the control action, Y is the measured system output, R is the reference, U_m is the manual control action, *Manual* is the operation mode boolean, *NewST* is a boolean that signals the arrival of a new sample time and *Clock* is the elapsed time since the start of the sample time.

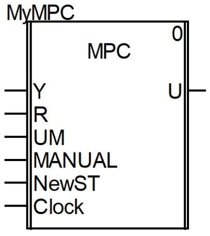


Fig. 3: Controller FBD block in Unity Pro XL.

B. Controller algorithm

This section describes the controller's general algorithm, which is shown in Algorithm 5.

The control objective of the controller is to steer the system output $Y_m \in \mathbb{R}^p$ to the desired reference $R \in \mathbb{R}^p$. Both of these signals, as well as signal U_m, are expressed in engineering units.

The controller can work in one of two operating modes: *manual* mode, where the control action applied to the system is provided by signal U_m; and *automatic* mode, where the control action is obtained by solving the MPC optimization problem. The controller's input boolean signal *Manual* determines the mode of operation. Henceforth, we assume the controller to be in *automatic* mode unless specified otherwise.

We assume that the controller may be implemented alongside other tasks which may also be time sensitive (e.g. monitoring or security tasks). For this reason, the number of successive iterations of the Gradient Method (GM) used to solve the MPC's QP problem are limited to a given number N_{GM} within each call to the controller FBD block. If N_{GM} iterations are performed and the GM algorithm has not converged, then the algorithm is paused and the controller FBD block exits, freeing the PLC's processor for the next task. The GM algorithm will continue from where it stopped the next time the controller block is called within the current sample time. This process will continue until either the GM algorithm converges, or a maximum allowed time T_{max} to converge is exceeded – at which point an auxiliary control law is applied. As such, the controller may be called on multiple occasions within a single sample time before returning the value of $u(k)$, and will then be called multiple other times before a new sample time arrives.

In order for this setup to work it is assumed that the PLC's main task is working cyclically (See Figure 1b) and that the other tasks have low execution times compared to the sample time. The estimator and predictor are run once the current sample time's control action $u(k)$ has been computed, whether it be the one obtained from solving the MPC's QP problem, the auxiliary controller or the manual control action U_m.

The auxiliary control law takes the following steps:

1) Take the current candidate solution provided by the GM algorithm, i.e. either $z(\lambda_k)$ for FISTA algorithm, or z_k for the ADMM algorithm.
2) Compute the cost of the candidate solution.

Algorithm 5: Controller

```
Require: $Y_m$, $R$, $U_m$, $Manual$, $NewST$, $Clock$
Compute $y_m$ and $r$ from $Y_m$ and $R$
if Manual = True then                 // Manual mode
    $U = U_m$
    if NewST then
        Update $u$ record
        Run Estimator
        Run Open loop predictor$^{(1)}$
    end if
else                                  // Automatic mode
    if NewST then
        GM_converged = False
        Reset GM variables
        Run SSTO
    end if
    if Not GM_converged then
        Perform $N_{\text{GM}}$ iterations of the GM algorithm
        Compute GM residual $\Gamma^{(2)}$
        if $||\Gamma|| < \epsilon$ then
            $U = N_u^{-1}u(k) + U_0$
            GM_converged = True
        else if $Clock > T_{max}$ then
            $U =$ Auxiliary control law
            GM_converged = True
        end if
    else
        if Estimator not done this Sample Time then
            Update $u$ record
            Run Estimator
            Run Open loop predictor$^{(1)}$
        end if
    end if
end if
Output: U
```

(1) The open loop predictor is only included if $d > 0$.
(2) Γ is either step 9 of FISTA algorithm or step 8 of ADMM algorithm.

3) If this cost is lower than the cost of the solution of the previous sample time, apply the control action provided by the candidate solution.
4) Otherwise, apply the control action provided by an LQR control law.

We remark that this auxiliary control law is the one currently implemented, but there are other possible control laws which could be applied. The application of the current candidate solution if it is feasible and provides a lower cost than the one of the previous sample time provides stability of the closed loop system [44].

Algorithm 5 is the general outline of the controller's algorithm, where some minor details have been omitted due to space considerations. Boolean flag variables are used to determine if different elements of the controller have been executed in the current sample time, e.g. *GM_converged* indicates whether the GM algorithm has been completed in the current sample time or not. A record of the last d control actions is needed in systems with delay $d \geq 0$ in order to compute $x_p(k)$ (Section II-D). We note that the open loop predictor's code and variables are only generated if $d > 0$. If $d = 0$, $\hat{x}(k)$ is used instead of $x_p(k)$ as the initial condition (8d) or (9d) of the MPC.

Remark 8. *The implementation of the GM algorithm for solving the SSTO's QP problem does not use a maximum number of iterations per execution of the FBD block because it is assumed that it is a much simpler problem to solve than the MPC's QP problem.*

C. Memory requirements

This section describes the memory requirements for a controller that uses FISTA algorithm (Alg. 2) and MPC formulation (8). The following results would be very similar for any other combination of gradient method and MPC formulation.

Table I shows the variables that are stored in memory. As can be observed, only the basic data is stored, i.e. the complete matrices of the QP problems are not stored, but rather, only the inner matrices that form them. All diagonal matrices are stored as vectors.

TABLE I: Stored variables for FISTA algorithm

Category	**Variables**
Inputs	Y_m, R, U_m, Manual, NewST, Clock
Outputs	U
System	A, B, C, U^0, Y_m^0, N_u, N_u^{-1}, N_m, $\underline{x}$, $\overline{x}$, $\underline{u}$, $\overline{u}$
MPC	Q, Q^{-1}, R, R^{-1}, ϵ, N_{GM}, $\mathcal{A}^{(1)}$, $\hat{\mathcal{B}}$, z_{MPC}, λ_{MPC}, η_{MPC}, η_{MPC}^+, Γ_{MPC}, $\|\Gamma\|$, t_{MPC}, t_{MPC}^+, $\text{T}_{\max}$
SSTO	Q_r, Q_r^{-1}, R_r, R_r^{-1}, T_h, T_h^{-1}, ϵ_r, η_r, r_f, x_r, u_r, x_c, u_c, z_{SSTO}, λ_{SSTO}, η_{SSTO}, η_{SSTO}^+, Γ_{SSTO}, t_{SSTO}, t_{SSTO}^+, W_{SSTO}^{-1}
Estimator	$\hat{x}$, $\hat{x}^+$, $\hat{w}$, $\hat{w}^+$, Aux_w, L_x, L_d
Predictor(2)	H_u(3), x_p, x_p^+
Others	Various Boolean variables and counters

(1) Recall that $\mathcal{A}$ and $\mathcal{B}$ store the α_i and β_i matrices (See (35)).
(2) x_p and x_p^+ are only included if $d > 0$.
(3) Vector H_u contains $u(k) \dots u(k-d)$.

We note that, in order to avoid performing divisions, we store the inverse of N_u, Q, R, Q_r, R_r and T_h, as well as $\hat{\mathcal{B}} = \{\hat{\beta}_0, \dots, \hat{\beta}_N\}$, where $(\hat{\beta}_i)_{j,k} = (\beta_i)_{j,k}$ if $j \neq k$, and $(\hat{\beta}_i)_{j,k} = 1/(\beta_i)_{j,k}$ if $j = k$.

The code of the algorithms of two different MPC implementations only differ in some sporadic numbers that appear throughout the code, due to all the operations being made using conditional loops. As such, the amount of memory required to store the code of the algorithm should be approximately equal regardless of the problem dimensions. The exception is between having a value of $d = 0$ or $d > 0$, since the open loop predictor and certain variables are not declared if $d = 0$.

The memory required to store the problem data is given by,

$$\begin{aligned}\text{Mem} = \theta\big[&(2N+3)n^2 + (5N+19+m+4p)n + \\ &(N+14)m + 2p^2 + 15p + 11\big] + 6\tau + 6\psi + \\ &\delta_{d>0}\big[\theta\left((d+1)m + 2n\right) + \psi\big],\end{aligned}$$

where θ, τ and ψ are the memory required to store a real number, integer and boolean, respectively; and $\delta_{d>0} = 1$ if $d > 0$ and 0 otherwise. Note that the memory grows linearly with N, m and d, and quadratically with n and p.

VI. Case Study

A. Double reactor and separator system

The plant under consideration is a 12 state, 4 input and 4 output double reactor and separator system inspired from [45] and depicted in Figure 4. Reactant A is converted into reactant B by a first-order reaction. Similarly, reactant B is converted into reactant C by a first-order reaction. Reactions occur in reactors 1 and 2, which are fed by flows F_{f1} and F_{f2}, respectively. The reactants are distilled in the separator and the distillate is redirected to reactor 1 through flow F_R. Heat can be added to the reactors and separator (Q_1, Q_2, Q_3).

The non-linear model of the plant is,

$$\begin{aligned}
\frac{\mathrm{d}H_1}{\mathrm{d}t} &= \frac{1}{\rho A_1}(F_{f1} + F_R - F_1) \\
\frac{\mathrm{d}x_{A1}}{\mathrm{d}t} &= \frac{1}{\rho A_1 H_1}\left(F_{f1}(x_{A0} - x_{A1}) + F_R(x_{AR} - x_{A1})\right) - k_{A1}x_{A1} \\
\frac{\mathrm{d}x_{B1}}{\mathrm{d}t} &= \frac{1}{\rho A_1 H_1}\left(F_{f1}(x_{B0} - x_{B1}) + F_R(x_{BR} - x_{B1})\right) \\
&\quad - k_{B1}x_{B1} + k_{A1}x_{A1} \\
\frac{\mathrm{d}T_1}{\mathrm{d}t} &= \frac{1}{\rho A_1 H_1}\left(F_{f1}(T_0 - T_1) + F_R(T_R - T_1)\right) + \frac{Q_1}{\rho A_1 H_1 C_p} \\
&\quad - \frac{1}{C_p}\left(k_{A1}x_{A1}\Delta H_A + k_{B1}x_{B1}\Delta H_B\right) \\
\frac{\mathrm{d}H_2}{\mathrm{d}t} &= \frac{1}{\rho A_2}(F_{f2} + F_1 - F_2) \\
\frac{\mathrm{d}x_{A2}}{\mathrm{d}t} &= \frac{1}{\rho A_2 H_2}\left(F_{f2}(x_{A0} - x_{A2}) + F_1(x_{A1} - x_{A2})\right) - k_{A2}x_{A2} \\
\frac{\mathrm{d}x_{B2}}{\mathrm{d}t} &= \frac{1}{\rho A_2 H_2}\left(F_{f2}(x_{B0} - x_{B2}) + F_1(x_{B1} - x_{B2})\right) \\
&\quad - k_{B2}x_{B2} + k_{A2}x_{A2} \\
\frac{\mathrm{d}T_2}{\mathrm{d}t} &= \frac{1}{\rho A_2 H_2}\left(F_{f2}(T_0 - T_2) + F_1(T_1 - T_2)\right) + \frac{Q_2}{\rho A_2 H_2 C_p} \\
&\quad - \frac{1}{C_p}\left(k_{A2}x_{A2}\Delta H_A + k_{B2}x_{B2}\Delta H_B\right) \\
\frac{\mathrm{d}H_3}{\mathrm{d}t} &= \frac{1}{\rho A_3}(F_2 - F_D - F_R - F_3) \\
\frac{\mathrm{d}x_{A3}}{\mathrm{d}t} &= \frac{1}{\rho A_3 H_3}\left(F_2(x_{A2} - x_{A3}) - (F_D + F_R)(x_{AR} - x_{A3})\right) \\
\frac{\mathrm{d}x_{B3}}{\mathrm{d}t} &= \frac{1}{\rho A_3 H_3}\left(F_2(x_{B2} - x_{B3}) - (F_D + F_R)(x_{BR} - x_{B3})\right) \\
\frac{\mathrm{d}T_3}{\mathrm{d}t} &= \frac{1}{\rho A_3 H_3}F_2(T_2 - T_3) + \frac{Q_3}{\rho A_3 H_3 C_p}
\end{aligned}$$

where, denoting Euler's number by e and for $i \in \{1, 2, 3\}$, we have that,

$$\begin{aligned}
F_i &= k_{vi}H_i & k_{Ai} &= k_A e^{-\frac{E_A}{RT_i}} & k_{Bi} &= k_B e^{-\frac{E_B}{RT_i}} \\
F_D &= \alpha_D F_R & x_{AR} &= \frac{\alpha_A x_{A3}}{\overline{x}_3} & x_{BR} &= \frac{\alpha_B x_{B3}}{\overline{x}_3}
\end{aligned}$$

$$\overline{x}_3 = \alpha_A x_{A3} + \alpha_B x_{B3} + \alpha_C x_{C3} \quad x_{C3} = 1 - x_{A3} - x_{B3}$$

The states X, inputs U and outputs Y_m of the system are,

$$\begin{gathered}
X = (H_1, x_{A1}, x_{B1}, T_1, H_2, x_{A2}, x_{B2}, T_2, H_3, x_{A3}, x_{B3}, T_3) \\
U = (Q_1, Q_2, F_{f2}, F_R) \quad Y_m = (T_1, T_2, T_3, x_{B3}).
\end{gathered}$$

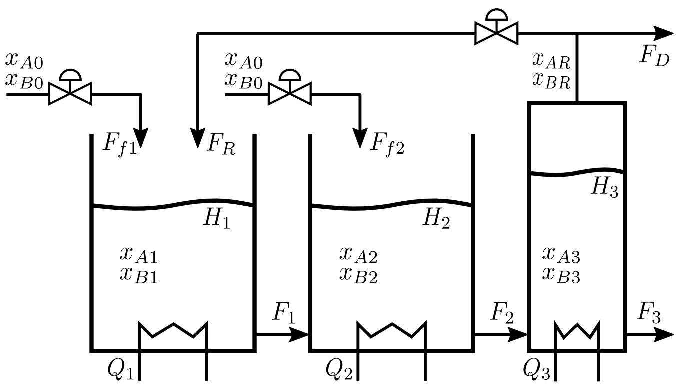


Fig. 4: Double reactor and separator system.

B. Hardware in the loop framework Setup

The PLC used is a Modicon M340 with processor module BMXP3420302, from Schneider Electric. The PLC is equipped with an input module AMI0410, which offers 4 analog input channels, and two output modules AMO0210, each of which offer 2 analog output channels. The processor module has 256Kb of internal memory for data and 3840Kb for user defined programs, constants, system and symbols. This PLC has an overhead of around 6 to 9ms at each cycle.

The non-linear system is simulated in real time using software QUARC [46] in Simulink. A discrete linear time-invariant state space model (1) of the system is obtained for the steady state and parameters defined in Table II and a sample time of 3s. The resulting linear model has no delay, i.e. $d = 0$. Additionally, the system is subject to the constraints shown in Table III.

Using the code generation tool, an MPC controller is generated using the linear model and the parameters described in Table IV. Both the MPC and the SSTO are solved using FISTA algorithm (Alg. 2).

The connection between the PC and the PLC is done using National Instrument acquisition card USB-6211. The input and output voltage signals are converted to and from engineering units by linear interpolation using FBD blocks.

A project that only contains the generated MPC controller, an internal timer that generates signals *NewST* and *Clock*, and the voltage conversion FBD blocks, is loaded into the PLC. This project, which contains an 80 decision variable MPC and a 20 decision variable SSTO, consumes 274.08Kb of out of the 3840Kb (7.14%) dedicated to programs and 36.47Kb out of the 256Kb (14.25%) dedicated to data, both of which include the memory consumed by the PLC by default (which can be seen in Figure 8).

C. Results

This section shows the results of two tests. In both tests the system is started at the operating point with the controller engaged in manual mode. The controller is then switched to automatic mode and the reference is changed after a few sample times.

TABLE II: Parameters and Operating Point

Parameter	Value	Units	Parameter	Value	Units
H_1^0	0.7	m	A_1	1	m^2
x_{A1}^0	0.4155	wt(%)	A_2	1	m^2
x_{B1}^0	0.5480	wt(%)	A_3	1	m^2
T_1^0	329	K	ρ	1100	kg/m^3
H_2^0	0.9	m	C_p	4	kJ/kg K
x_{A2}^0	0.2581	wt(%)	k_{v1}	50	kg/m s
x_{B2}^0	0.6755	wt(%)	k_{v2}	50	kg/m s
T_2^0	333	K	k_{v3}	30	kg/m s
H_3^0	1.3332	m	T_0	313	K
x_{A3}^0	0.2282	wt(%)	k_A	10^{-5}	1/s
x_{B3}^0	0.7	wt(%)	k_B	$5 \cdot 10^{-6}$	1/s
T_3^0	333	K	E_A/R	-2840	K
Q_1^0	0	kJ/s	E_B/R	-2077	K
Q_2^0	0	kJ/s	ΔH_A	-100	kJ/kg
F_{f2}^0	10	kg/s	ΔH_B	-39	kJ/kg
F_R^0	5	kg/s	α_A	3.5	-
F_{f1}	30	kg/s	α_B	1.1	-
Q_3	0	kJ/s	α_C	0.5	-
x_{A0}	1	wt(%)	α_D	0.001	-
x_{B0}	0	wt(%)			

TABLE III: Constraints

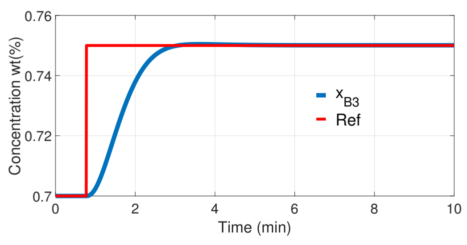
Input	$\overline{U}$	$\underline{U}$	Output	$\overline{Y}_c$	$\underline{Y}_c$
Q_1	1000	-1000	T_1	373	0
Q_2	1000	-1000	T_2	373	0
F_{f2}	30	0	T_3	373	0
F_R	20	0	x_{B3}	1	0.4

We show the evolution of the systems inputs and outputs, as well as the number of iterations of FISTA algorithm and the total computation time of the controller at each sample time.

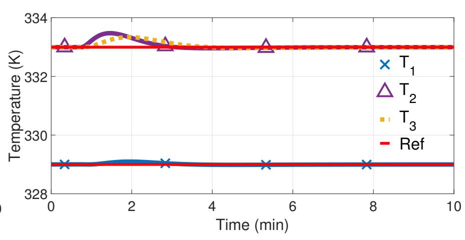
Test A. No active constraints. The reference for x_{B3} was changed from x_{B3}^0 to 0.75. The reference for the temperatures remained unchanged. Figure 5 shows the results for this test. As can be seen, no constraints are active during this test. As such, the number of FISTA iterations of the SSTO algorithm is 0, due to its warmstarting procedure, and the number of iterations requires by the MPC is very low. The maximum iteration time of this test if 47ms.

Test B. Active constraints. The reference for x_{B3} was changed from x_{B3}^0 to 0.6. The reference for the temperatures remained unchanged. Figure 6 shows the results for this test. In this case, the reference is not admissible. Therefore, the SSTO is engaged, providing an admissible reference. Since T_h has a higher penalty on the diagonal term corresponding to x_{B3} (see Table IV), the SSTO returns an admissible reference which steers the system close to the reference for x_{B3} at the expense of the reference for the temperatures. Even though this test requires more iterations than Test A due to the presence of active constraints, the computational time is still reasonable. The maximum iteration time of this test is 129ms.

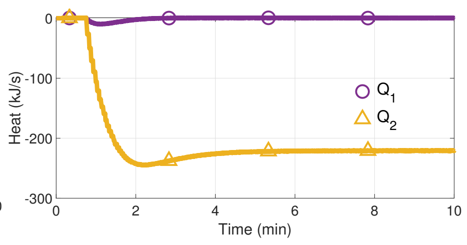
The results show that each iteration of FISTA algorithm takes $\sim$20ms for the MPC, whose QP problem has 80 decision variables, and $\sim$3.3ms for the SSTO, whose QP problem



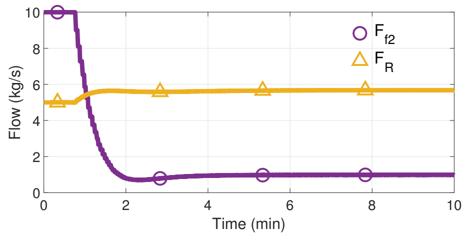
(a) System output: x_{B3}.



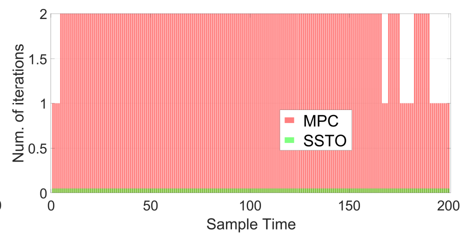
(b) System output: Temperatures T_i.



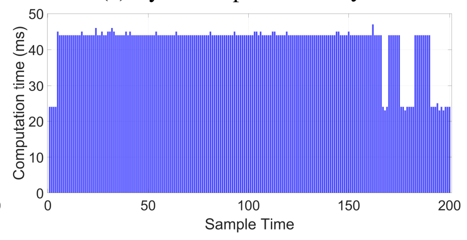
(c) System input: Heats Q_i



(d) System input: Flows F_i

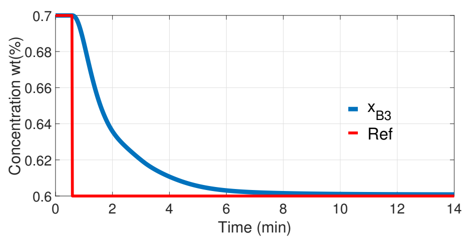


(e) Iterations of FISTA algorithm

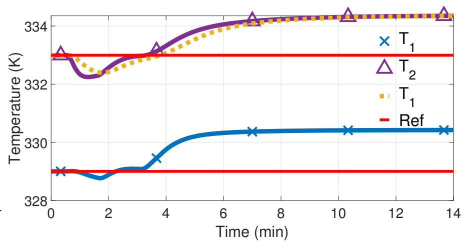


(f) Computation time

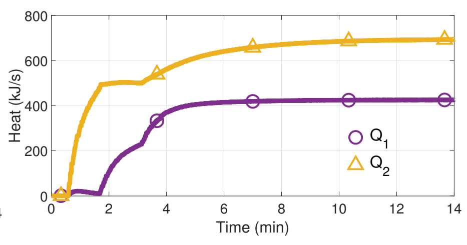
Fig. 5: Results of Test A.



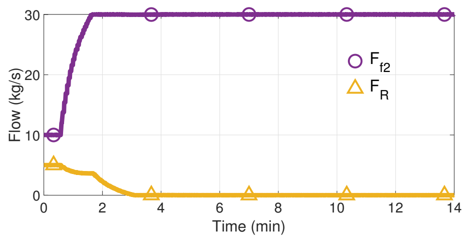
(a) System output: x_{B3}.



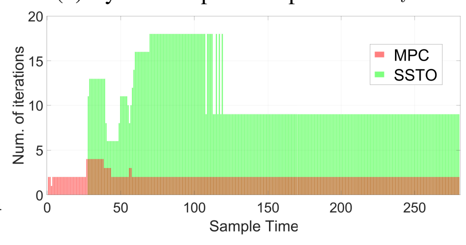
(b) System output: Temperatures T_i.



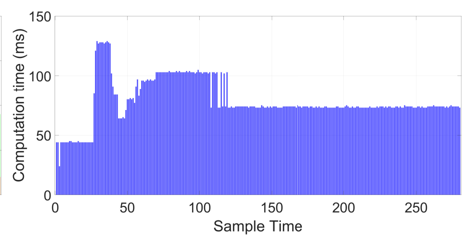
(c) System input: Heats Q_i



(d) System input: Flows F_i



(e) Iterations of FISTA algorithm



(f) Computation time

Fig. 6: Results of Test B.

TABLE IV: Parameters of MPC controller

Parameter	Value	Parameter	Value
Q	$20I_{12}$	Q_r	$0.1I_{12}$
R	$10I_4$	R_r	$0.1I_4$
T	$40I_{12}$	T_h	$\mathrm{diag}(50, 50, 50, 500)$
N	5	η_r	0.912
ϵ	0.001	ϵ_r	0.001
N_{GDM}	5	N_u	$\mathit{diag}(0.001, 0.001, 0.1, 0.1)$
$\text{T}_{\max}$	0.3	N_y	$\mathit{diag}(0.033, 0.033, 0.033, 10)$
U_m	U^0		

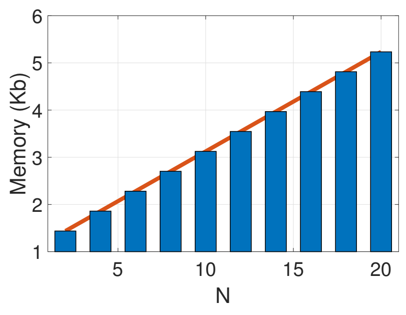
diag indicates a diagonal matrix

has 20 decision variables. The auxiliary control law was not applied in any of the two tests due to the algorithm converging before the allowed $T_{\max} = 300$ms at every sample time.

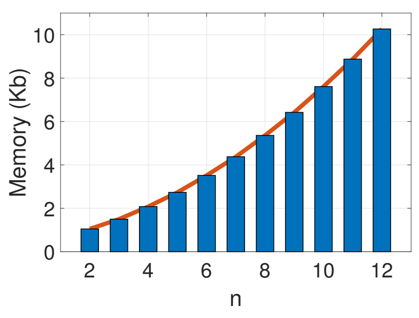
VII. Memory Study

This section presents the results of memory tests performed for the proposed controller using FISTA algorithm (Alg. 2) and for the MPC formulation (8). The results are very similar for any other combination of gradient method and MPC formulation.

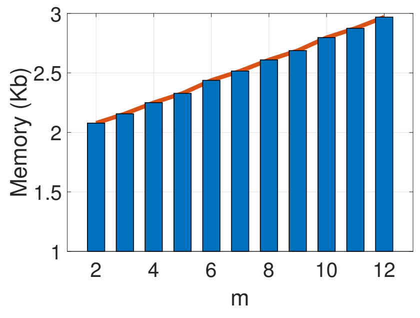
Controllers are generated for random systems with different values of the prediction horizon N, the system delay d, and the dimensions of the state n, control input m and system output p. The controller is programmed on the PLC described in Section VI-B. Memory measurements are obtained using the built in tool of Unity Pro XL depicted in Figure 8, which is showing the memory consumption of an empty project, i.e. with no user-declared variables nor user-declared tasks. The fields whose value changes when an MPC controller is implemented are *Declared Data*, *Executable code*, *Upload information* and *System*. The memory consumption results shown here are



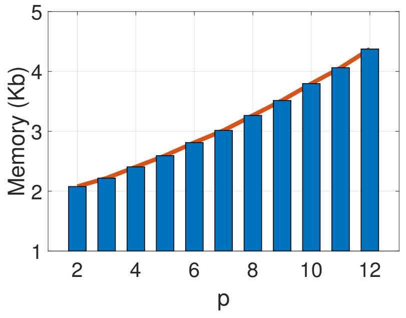
(a) *Declared data* with respect to N.



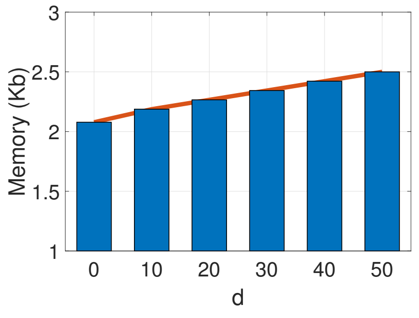
(b) *Declared data* with respect to n.



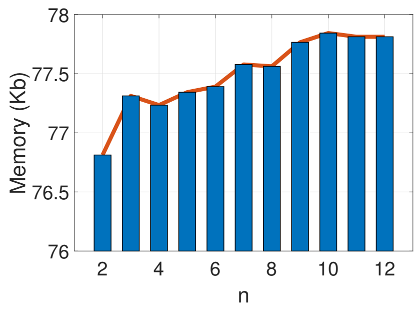
(c) *Declared data* with respect to m.



(d) *Declared data* with respect to p.



(e) *Declared data* with respect to d.



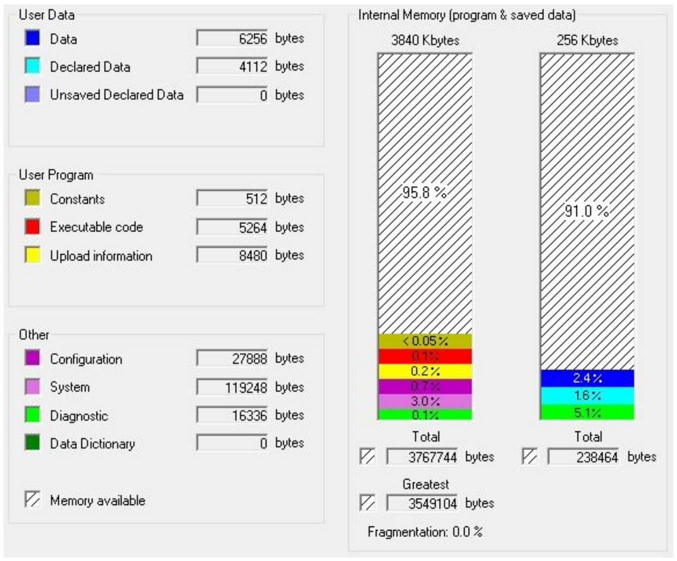
(f) *Executable code* with respect to n.

Fig. 7: MPC memory requirements in Modicom M340 PLC. Results shown are increments with respect to an empty project.

calculated as the difference between the memory consumption of an empty project (See Figure 8) and a project that only contains the MPC controller.

Figures 7a to 7e show the increment of *Declared Data* in kilobytes (1 Kb $=$ 1024 bytes) of MPC controllers with increasing values of each one of the aforementioned parameters. In each figure, the remaining parameters are set to $N = 5$, $n = 4$, $m = 2$, $p = 2$ and $d = 0$. Note that the memory grows linearly with N, m and d, and quadratically with n and p.

Fields *Executable code*, *Upload information* and *System* show a slight increasing trend. However, there is no clear relation between the value of these fields and the value of the parameters, as illustrated in Figure 7f, which shows the increment of the value of field *Executable code* for the MPC controllers whose *Declared Data* is shown in figure 7b. In spite of this, the sum of the three fields is monotone increasing in all the tests that have been conducted. The *Executable code* field increases by about 7.75Kb if $d > 0$ with respect to $d = 0$, due to the inclusion of the open loop predictor.


User Data
Data 6256 bytes
Declared Data 4112 bytes
Unsaved Declared Data 0 bytes

User Program
Constants 512 bytes
Executable code 5264 bytes
Upload information 8480 bytes

Other
Configuration 27888 bytes
System 119248 bytes
Diagnostic 16336 bytes
Data Dictionary 0 bytes

Memory available

Internal Memory (program & saved data)
3840 Kbytes 256 Kbytes
95.8 % 91.0 %
< 0.05 % 2.4 %
0.1 % 1.6 %
0.2 % 5.1 %
0.7 %
3.0 %
0.1 %
Total 3767744 bytes Total 238464 bytes
Greatest 3549104 bytes
Fragmentation: 0.0 %

Fig. 8: Memory consumption of an empty project.

VIII. Conclusions

This paper presents an implementation of an MPC based controller in PLCs using an automatic code generation tool, showing that it it possible to implement predictive controllers in low-end PLCs alongside other tasks. The implementation is performed using a code generation tool that allows the MPC's optimization problem to be solved online using a low memory footprint gradient method, allowing for the possibility of controlling medium sized systems, as shown by the hardware-in-the-loop results.

One of the main results of the proposed controller is its linear memory growth with respect to the prediction horizon, which is achieved by exploiting the structure of the proposed MPC formulation's QP problem by means of a banded Cholesky factorization of the main memory intensive ingredient of the optimization method.

References

[1] E. F. Camacho and C. B. Alba, *Model Predictive Control*. Springer Science & Business Media, 2013.

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

[3] G. Pannocchia, N. Laachi, and J. B. Rawlings, "A candidate to replace PID control: SISO-constrained LQ control," *AIChE Journal*, vol. 51, no. 4, pp. 1178–1189, 2005.

[4] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal*, vol. 11, no. 2, pp. 1208–1219, 2014.

[5] 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.

[6] G. Valencia-Palomo and J. Rossiter, "Novel programmable logic controller implementation of a predictive controller based on Laguerre functions and multiparametric solutions," *IET control theory & applications*, vol. 6, no. 8, pp. 1003–1014, 2012.

[7] J. Velagić and B. Šabić, "Design, implementation and experimental validation of explicit MPC in programmable logic controller," in *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*. IEEE, 2014, pp. 93–98.

[8] F. Ullmann, "FiOrdOs: A Matlab toolbox for C-code generation for first order methods," *MS thesis*, 2011.

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

[10] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 668–674.

[11] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

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

[13] B. Binder, D. K. M. Kufoalor, and T. A. Johansen, "Scalability of QP solvers for embedded model predictive control applied to a subsea petroleum production system," in *2015 IEEE Conference on Control Applications (CCA)*. IEEE, 2015, pp. 1173–1178.

[14] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive control using an FPGA with application to aircraft control," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 1006–1017, 2014.

[15] D. Kufoalor, S. Richter, L. Imsland, T. A. Johansen, M. Morari, and G. O. Eikrem, "Embedded model predictive control on a PLC using a primal-dual first-order method for a subsea separation process," in *22nd Mediterranean Conference on Control and Automation*. IEEE, 2014, pp. 368–373.

[16] D. Kouzoupis, A. Zanelli, H. Peyrl, and H. J. Ferreau, "Towards proper assessment of QP algorithms for embedded model predictive control," in *2015 European Control Conference (ECC)*. IEEE, 2015, pp. 2609–2616.

[17] I. Necoara and D. Clipici, "Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: Application to distributed MPC," *Journal of Process Control*, vol. 23, no. 3, pp. 243–253, 2013.

[18] M. Pereira, D. Limon, D. Muñoz de la Peña, and T. Alamo, "MPC implementation in a PLC based on Nesterov's fast gradient method," in *2015 23rd Mediterranean Conference on Control and Automation (MED)*. IEEE, 2015, pp. 371–376.

[19] R. M. Levenson, Z. E. Nelson, and A. A. Adegbege, "Programmable logic controller for embedded implementation of input-constrained systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 412–14 417, 2017.

[20] S. Lucia, D. Navarro, Ó. Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE transactions on industrial informatics*, vol. 14, no. 1, pp. 137–145, 2018.

[21] P. Zometa, M. Kögel, and R. Findeisen, "μAO-MPC: a free code generation tool for embedded real-time linear model predictive control," in *American Control Conference (ACC), 2013*. IEEE, 2013, pp. 5320–5325.

[22] J. R. Sabo and A. A. Adegbege, "A primal-dual architecture for embedded implementation of linear model predictive control," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 1827–1832.

[23] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones, "Software and hardware code generation for predictive control using splitting methods," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 386–14 391, 2017.

[24] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[25] J. Currie, A. Prince-Pike, and D. I. Wilson, "Auto-code generation for fast embedded model predictive controllers," in *Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference*. IEEE, 2012, pp. 116–122.

[26] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on control systems technology*, vol. 18, no. 2, p. 267, 2010.

[27] P. Krupa, D. Limon, and T. Alamo, "Implementation of model predictive controllers in programmable logic controllers using IEC 61131-3 standard," in *European Control Conference (ECC)*. IEEE, 2018, pp. 288–293.

[28] U. Maeder, F. Borrelli, and M. Morari, "Linear offset-free model predictive control," *Automatica*, vol. 45, no. 10, pp. 2214 – 2222, 2009.

[29] T. L. Santos, D. Limon, J. E. Normey-Rico, and G. V. Raffo, "Dead-time compensation of constrained linear systems with bounded disturbances: output feedback case," *IET Control Theory & Applications*, vol. 7, no. 1, pp. 52–59, 2013.

[30] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Nob Hill Publishing, 2017.

[31] D. Limon, T. Alamo, F. Salas, and E. F. Camacho, "On the stability of constrained MPC without terminal constraint," *IEEE transactions on automatic control*, vol. 51, no. 5, pp. 832–836, 2006.

[32] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[33] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk, "Fast alternating direction optimization methods," *SIAM Journal on Imaging Sciences*, vol. 7, no. 3, pp. 1588–1623, 2014.

[34] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for large convex problems," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 1951–1956.

[35] Y. Nesterov, *Introductory Lectures on Convex Optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[36] D. P. Bertsekas, *Convex Optimization Theory*. Athena Scientific Belmont, 2009.

[37] A. Beck and M. Teboulle, "A fast dual proximal gradient algorithm for convex minimization and applications," *Operations Research Letters*, vol. 42, no. 1, pp. 1–6, 2014.

[38] I. Necoara, "Computational complexity certification for dual gradient method: Application to embedded MPC," *Systems & Control Letters*, vol. 81, pp. 49–56, 2015.

[39] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2011.

[40] S. Boyd, *Convex Optimization*, 7th ed. Cambridge University Press, 2009.

[41] P. Giselsson and S. Boyd, "Linear convergence and metric selection for Douglas-Rachford splitting and ADMM," *IEEE Transactions on Automatic Control*, vol. 62, no. 2, pp. 532–544, 2017.

[42] T. Alamo, P. Krupa, and D. Limon, "Restart FISTA with global linear convergence," in *Proceedings of the European Control Conference (ECC)*. IEEE, 2019, pp. 1969–1974.

[43] ——, "Gradient based restart FISTA," in *Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 3936–3941.

[44] P. O. Scokaert, D. Q. Mayne, and J. B. Rawlings, "Suboptimal model predictive control (feasibility implies stability)," *IEEE Transactions on Automatic Control*, vol. 44, no. 3, pp. 648–654, 1999.

[45] P. D. Christofides, J. Liu, and D. Muñoz de la Peña, *Networked and distributed predictive control: Methods and nonlinear process network applications*. Springer Science & Business Media, 2011.

[46] "QUARC," https://www.quanser.com/products/quarc-real-time-control-software/, accessed: 2020-05-03.