2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Implementation of Model Predictive Controllers in Programmable Logic Controllers using IEC 61131-3 standard

Pablo Krupa[1], Daniel Limon[2] and Teodoro Alamo[3]

Abstract— **This work presents a tool for implementing Model Predictive Controllers (MPC) in Programmable Logic Controllers (PLC). This tool is a Matlab library that automatically generates the controller's code using IEC 61131-3 standards so that it can be directly imported into the PLC's programming platform as an FBD block, which is designed to work using the cyclic mode of the PLC and has a limited maximum execution time. A hand-tailored optimization algorithm based on a fast gradient method named FISTA has been developed in order to reduce the necessary memory and computational resources. A complete architecture has been designed surrounding the MPC which provides the overall controller with a series of additional capabilities. The properties of the controller have been validated via a test-bed using a Modicom M340 PLC to control a quadruple-tank system. In addition, tests have been conducted in order to study the memory requirements.**

I. INTRODUCTION

Model Predictive Control (MPC) is an optimization based control strategy which is especially useful for controlling multi-variable systems with constraints [5]. This control strategy is used in numerous industrial areas, such as petrol refineries, power stations or the process industry. However, its implementation has been focused on systems with high computational and memory resources, such as PCs, due to the fact that an optimization problem must be solved at every sampling time. Its implementation in embedded systems, such as Programmable Logic Controllers (PLC), is complicated due to their limited computational resources.

Programmable Logic Controllers are the most extended embedded systems used to implement control loops in industry. They are characterized by their robustness, reliability, being easily programmable and by having dedicated I/O communication. However, they have limited computational and memory resources. In spite of this, previous studies have implemented MPCs in PLCs, as well as in other embedded systems, such as microcontrollers or FPGAs. [10] and [4] show implementations of MPC controllers in PLCs.

Most advances in the implementation of predictive controllers in embedded systems have arisen from the development of software tools that generate efficient code for solving generic convex optimization problems in embedded systems. Some of the best known ones are FiOrdOs [19], CVXGEN [12], qpOASES [8] or FORCES [7]. Generally, they create code in C/C++ programming languages.

These tools are used on many occasions to implement MPC controllers in embedded systems. However, another approach is also possible, such as with μAO-MPC [21] or in [6], where code generation tools have been developed that implement specific formulations of MPC controllers in embedded systems using C programming language. These tools, in contrast to the ones shown in the previous paragraph, do not aim to solve a generic convex optimization problem, but the specific one that arises from the MPC formulation that they propose. As such, they offer more efficient solutions and an easier and more streamlined way of implementing the controller in the embedded system.

As another example of an implementation of an MPC controller in a PLC, we would like to mention article [15], where a formulation very similar to the one we propose was implemented on a PLC using IEC 61131-3 standards. However, the structure of the optimization problem was not exploited in order to reduce memory consumption nor did it include any elements apart from the MPC itself. This approach proved to be viable only for problems with a number of decision variables in the dozens, since memory requirements grew quadratically with the prediction horizon.

Lastly, [9] shows practical considerations of implementing MPC controllers in PLCs and [20] shows another example where a Generalized Predictive Controller (GPC) was implemented in a PLC using IEC 61131-3 standards. However, it does not take restrictions into account nor does it exploit the problem structure to minimize memory.

The main contribution of this paper is the development of a tool, in the form of a Matlab library, that implements a stabilizing MPC controller by solving the QP problem following the latter approach. That is, it focuses on a specific MPC formulation in order to create a controller designed to be highly efficient, both in terms of memory and computational requirements. This tool generates controllers programmed in the standard IEC 61131-3 ST programming language, which is used in PLCs, instead of focusing on C, which we find to be the most common occurrence. An FBD block is generated, which has been designed to be implemented using the PLC's cyclic mode. The maximum execution time of the FBD block is limited in order to allow it to be used alongside other PLC tasks. Algorithm FISTA [3], [1], [16], which is used to solve generic QP problems, has been adapted in order to exploit the structure of the MPC formulation shown in Section III, resulting in a highly memory efficient hand-tailored algorithm.

[1]Pablo Krupa is a PhD student at the Systems Engineering and Automation department, University of Seville, Spain. `pabkrugar@alum.us.es`

[2]Daniel Limon is a professor of the Systems Engineering and Automation department, University of Seville, Spain. `dlm@.us.es`

[3]Teodoro Alamo is a professor of the Systems Engineering and Automation department, University of Seville, Spain. `talamo@.us.es`

Furthermore, the proposed tool differs from the other MPC implementations referenced in this paper in that it does not merely generate an MPC, but rather an MPC based controller that includes a series of additional elements that are not part of the MPC formulation. These elements are introduced in order to generate a controller that possesses a series of properties that facilitate its implementation in realistic industrial environments, such as a state observer and disturbance estimator; a predictor, used to control systems with delays efficiently; and manual and automatic modes of operation.

The output of the Matlab library is a file that contains the variable declaration and code of the controller in a format that can be directly imported to the programming platform of the PLC. Currently, the tool supports the creation of MPC controllers for Unity Pro, which is the platform used to monitor, control and program Schneider Electric PLCs; and for CODESYS. All experimental results shown in this paper have been produced using a Modicon M340 PLC by Schneider Electric programmed using Unity Pro programming platform.

This paper is structured as follows. Section II defines the control objective and the system model. Section III shows the MPC formulation. Section IV presents FISTA algorithm, giving a brief explanation of the underlying theory behind it. Section V describes the architecture of the overall controller. Section VI describes in further detail the Matlab library. In section VII we present an example of implementation of the controller on a quadruple-tank system via a test-bed, and in Section VIII we show the results of tests on the memory requirements of the controller. Finally, conclusions and future work are presented in Section IX.

II. CONTROL OBJECTIVE

Consider a plant with state $X \in \mathbb{R}^n$, input $U \in \mathbb{R}^m$ and output $Y \in \mathbb{R}^p$. The control objective is to steer the output Y, which is measured at each sample time, to the desired reference Y_r given by the user. We assume that the state and system input are subject to the following box constraints

$$\begin{aligned} X_{min} \leq X \leq X_{max} \\ U_{min} \leq U \leq U_{max} \end{aligned} \tag{1}$$

We obtain the following linear, time-invariant, discrete-time model of the plant at the equilibrium point (X_0, U_0, Y_0).

$$x(k+1) = Ax(k) + Bu(k-d) \tag{2a}$$

$$y(k) = Cx(k) \tag{2b}$$

where $x(k) = X(k) - X_0$ is the state vector, $u(k) = U(k) - U_0$ is the input vector and $y(k) = Y(k) - Y_0$ is the output vector, at sample time k. Variable d is the delay of the system, measured in number of sample times. We define the following box restrictions for the incremental variables x and u.

$$\begin{aligned} LB_x \leq x(k) \leq UB_x \\ LB_u \leq u(k) \leq UB_u \end{aligned} \tag{3}$$

III. MPC FORMULATION

The control law of the MPC is derived from the following optimization problem, which is stabilizing by design.

$$J^* = \min_u \sum_{i=0}^{N-1} ||x(i) - x_r||_Q^2 + \sum_{i=0}^{N-1} ||u(i) - u_r||_R^2 \tag{4}$$

$$s.t. \quad x(k+1) = Ax(k) + Bu(k), \tag{4a}$$

$$LB_x \leq x(k) \leq UB_x, \tag{4b}$$

$$LB_u \leq u(k) \leq UB_u, k = 0, \ldots N-1 \tag{4c}$$

$$x(0) = x_p \tag{4d}$$

$$x(N) = x_r \tag{4e}$$

where N is the control horizon, $u = \{u_0, ..., u_{N-1}\}$ the input sequence, $x = \{x_0, ..., x_{N-1}\}$ the predicted state sequence, $x_r \in \mathbb{R}^n$ the state reference, $u_r \in \mathbb{R}^m$ the input reference, x_p is the predicted state of the system (See section V-C), and $Q \in \mathbb{R}^{n \times n} \succ 0$ and $R \in \mathbb{R}^{m \times m} \succ 0$ are the state and input weight matrices[1], respectively. Both Q and R are assumed to be diagonal.

Note that the MPC does not include the delay d in the model it uses to predict the system's evolution. The reason for this is that delay compensation is achieved using a predictor (See section V-C).

IV. FISTA ALGORITHM

Consider the following quadratic programming (QP) problem

$$\min_z \frac{1}{2} z' H z + f' z \tag{5}$$

$$s.t. \; z \in \mathcal{Z} \tag{5a}$$

$$A_z z = b \tag{5b}$$

where $z \in \mathbb{R}^{n_z}$ are the decision variables, set $\mathcal{Z} = \{z \in \mathbb{R}^{n_z} : LB \leq z \leq UB\}$, $A_z \in \mathbb{R}^{m_z \times n_z}$, with $n_z > m_z$, and $rank([A_z \; b]) = m_z$.

Note that optimization problem (4) can be reformulated as (5) by taking

$$z = [x(0)', u(0)', \; \ldots \; , x(N-1)', u(N-1)']'$$

In this work we use FISTA algorithm [3], which is a gradient based algorithm based on Nesterov's fast gradient method [13], to find the solution of the QP problem at each sample time.

We will now give a brief overview of the basic concepts of this algorithm and its formulation. Consider the following definitions,

$$J(z) = \frac{1}{2} z' H z + f' z \tag{6}$$

$$L(z, \lambda) = J(z) - \lambda'(A_z z - b) \tag{7}$$

$$z(\lambda) = arg \min_{z \in \mathcal{Z}} L(z, \lambda) \tag{8}$$

$$f(\lambda) = L(z(\lambda), \lambda) \tag{9}$$

Where function $L(z, \lambda)$ is the Lagrange function, λ the Lagrange multiplier and $f(\lambda)$ the dual function.

[1]$M \succ 0$ denotes that matrix M is positive definite

Let z^* be the optimal solution of problem (5) and $J^* = J(z^*)$ the optimal cost. We have

$$J^* = \quad L(z^*, \lambda) \geq f(\lambda) \tag{10}$$

for all λ, as $A_z z^* - b = 0$. As such, the dual function is a lower bound of the optimization problem. The objective is to find the best lower bound by solving the problem

$$\lambda^* = \quad arg \max_{\lambda} f(\lambda) \tag{11}$$

Assuming that Slater's condition is satisfied, strong duality holds for the previous problem. That is, we have that $J^* = f(\lambda^*)$.

Given our MPC formulation, the Hessian H of problem (5) is diagonal and positive definite. As such, each element in $z(\lambda)$ has a direct and explicit solution, since problem (12) can be cast as n_z uncoupled single-variable problems.

$$z(\lambda) = arg \min_{z \in \mathcal{Z}} \tfrac{1}{2} z' H z + (f - A_z' \lambda)' z \tag{12}$$

The solution of (11) is achieved by making use of the following inequality, obtained from [17] - see also [14] - and which holds if $H \succ 0$.

$$\begin{aligned} f(\lambda + \Delta\lambda) \geq f(\lambda) - \Delta\lambda'(A_z z(\lambda) - b) \\ - \frac{1}{2} \Delta\lambda' W \Delta\lambda \end{aligned} \tag{13}$$

with

$$W = A_z H^{-1} A_z' \tag{14}$$

Note that

$$\begin{aligned} arg \max_{\Delta\lambda} -\Delta\lambda'(A_z z(\lambda) - b) - \frac{1}{2}\Delta\lambda' W \Delta\lambda = \\ arg \min_{\Delta\lambda} \Delta\lambda'(A_z z(\lambda) - b) + \frac{1}{2}\Delta\lambda' W \Delta\lambda \end{aligned} \tag{15}$$

The solution to this problem is found at the point where the gradient is null.

$$A_z z(\lambda) - b + W \Delta\lambda = 0 \tag{16}$$

Given that H is positive definite and A is a full-row rank matrix, we have that W is invertible. As such, we can calculate $\Delta\lambda$ as,

$$\Delta\lambda = -W^{-1}(A_z z(\lambda) - b) \tag{17}$$

The steps of FISTA are shown in Algorithm 1, where TOL is an input parameter of the algorithm.

Algorithm 1 FISTA

```
1: $k = 1, \lambda_1 = \eta_1 = 0, t_1 = 1$
2: Repeat
    a: Obtain $z(\lambda_k)$ solving (12)
    b: $\Delta\lambda = -W^{-1}(A_z z(\lambda_k) - b)$
    c: $\eta_k = \lambda_k + \Delta\lambda$
    d: $t_{k+1} = \frac{1}{2}(1 + \sqrt{1 + 4t_k^2})$
    e: $\lambda_{k+1} = \eta_k + \frac{t_k - 1}{t_{k+1}}(\eta_k - \eta_{k-1})$
    f: $k = k + 1$
3: Until $|A_z z(\lambda_k) - b| \leq TOL$
```

V. CONTROLLER ARCHITECTURE

The controller designed in this work uses the MPC described in Section III to obtain the control action at each sample time. However, it also includes a series of auxiliary elements that provide the controller with additional characteristics that are necessary to implement the controller on a real plant: offset cancellation, state estimation, Steady state target optimizer (SSTO), delay compensation, and manual and automatic modes.

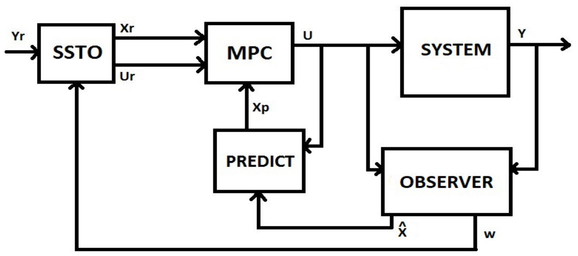

Fig. 1. Controller architecture

A. State observer and disturbance estimator

A Luenberger state observer and a disturbance estimator, which is used to estimate and predict the discrepancy between the measured output of the system and the predicted one according to the model, is included in the controller. An objective of this element is to achieve an offset-free MPC when an asymptotic reference to a steady state is provided.

The observer provides, at each sample time, an estimated state $\hat{x} \in \mathbb{R}^n$ and an estimated disturbance $\hat{w} \in \mathbb{R}^p$. To do so it requires the measured system output y and the system input $u^{-d} = u(k - d)$.

$$\hat{x}^+ = A\hat{x} + Bu^{-d} + L_x(y - C\hat{x} - \hat{w}) \tag{18}$$

$$\hat{w}^+ = \hat{w} + L_w(y - C\hat{x} - \hat{w}) \tag{19}$$

We assume that the system outputs that we wish to control are also the system outputs that we measure, which in reality is not always the case. The demonstration of the system's convergence to the desired output using this observer is shown in [11] - where in our case we are selecting $n_d = p$ and B_d as a null matrix.

B. Steady State Target Optimizer

The Steady State and Target Optimizer (SSTO) provides, at each sample time, the best feasible steady state (x_r, u_r) for the reference y_r provided by the user.

The SSTO optimization problem is shown in (20), where matrices Q_r, R_r and T_h are diagonal and positive definite. Therefore, this problem can be transformed into a QP problem (5) with a diagonal and positive definite Hessian, meaning that algorithm FISTA can be used to solve it. The SSTO uses the disturbance $\hat{w}$ estimated by the observer (19) in order to provide a steady state reference (x_r, u_r) such that an offset-free controller is achieved. This is possible as long as the provided reference y_r converges asymptomatically to a certain value and assuming that the disturbance is bounded.

$$(x_r, u_r, h) = arg \min_{x_r,u_r,h} ||x_r||^2_{Q_r} + ||u_r||^2_{R_r} + ||h||^2_{T_h} \quad (20)$$

$$s.t. \quad x_r = Ax_r + Bu_r \quad (20a)$$

$$y_r - \hat{w} = Cx_r + h \quad (20b)$$

$$LB_x \leq x_r \leq UB_x \quad (20c)$$

$$LB_u \leq u_r \leq UB_u \quad (20d)$$

Vector $h \in \mathbb{R}^p$, which penalizes the discrepancy between y_r and x_r, is the slack variable. Matrix T_h is usually selected to be at least an order of magnitude greater than Q_r and R_r, since the main objective is to obtain the (x_r, u_r) whose output is as close as possible to the given reference y_r.

C. Open-loop predictor

This element is added to compensate the effect of the delay [18]. It provides the predicted system state $x_p \in \mathbb{R}^n$, which is the predicted state at sampling time $k + d$ estimated from the data available at sampling time k using the model of the system as follows.

$$x_p = A^d\hat{x} + \sum_{i=1}^{d} A^{d-i}Bu(k - d - 1 + i) \quad (21)$$

The predicted state is used as the initial state of the MPC (4d). If the system has no delay, the predictor is not included and the estimated state $\hat{x}$ is used as the initial state of the MPC.

D. Manual and Automatic Modes

The controller provides a manual mode of operation, in which instead of calling the MPC at each sample time and applying the control action provided by it, an external control action U_m is applied. When in Manual mode, the controller still runs the observer at every sample time. However, the SSTO and the predictor are not called, since they are only necessary if the MPC is also called.

VI. MATLAB LIBRARY

The main contribution of this paper is the development of a tool that automatically creates the code of the presented MPC controller so that is can be directly imported into a PLC.

From our previous experience [15] we have noticed that the bottleneck that limits the size of the MPC problem to be solved in the PLC is the required memory. Therefore, the main focus when creating the tool has been to minimize the memory requirements. To do so, the structure of the matrices of the QP problem that arises from the MPC formulation (4) has been exploited. Since this tool doesn't aim to solve generic QP problems, but to solve a particular formulation of one, the matrices have a specific pattern that extends with the prediction horizon N. Instead of storing these matrices, which are sparse, we only store the basic matrices of the MPC (2) (3), and the matrices of the cost function of the MPC (4). Memory is also allocated to store the variables and vectors of FISTA algorithm and for internal boolean or counter variables.

The result of this approach is that the memory requirements do not grow quadratically with the prediction horizon N in spite of the fact that the matrices of the QP problem do so. Instead, the memory requirement grows linearly with N. A quadratic growth of memory in relation to the dimensions of n, m or p is inevitable, since the system matrices must be stored. The exact same approach has been used for the QP problem that arises from the SSTO (20).

As the matrices of the QP problem are not stored in their entirety they cannot be simply multiplied with other matrices or vectors directly. Instead, the structure of the matrices has been studied and a hand-tailored algorithm has been developed in Matlab that writes the code of FISTA algorithm for a generic system and cost matrices so that only the necessary multiplications are made. This approach is not only more efficient in terms of memory, but also in terms of computation time since the matrices of the QP problem are sparse.

The Matlab library creates the code that implements the proposed controller with the architecture shown in the previous section. However, the user may chose which auxiliary elements are included, if any. The predictor is always included if the system has delay. The following table shows the variables that are required in order to create the controller. Obviously, if an auxiliary element is not selected to be included in the generated controller, its associated variables are not required.

	Mandatory	Other
System	A, B, C, X_{min}, X_{max}, U_{min}, U_{max}, ST*	d
MPC + FISTA	Q, R, N, X_0, U_0, Y_0	U_m, TOL
Observer	-	D_d, L_x, L_w, $\hat{x}_0$

*ST stands for *Sample Time*

The library generates a file that contains the variable declaration and code of the controller. This file can be directly imported into the programming platform of the PLC, generating a FBD block. Figure 2 shows the appearance of the FBD block in Unity Pro. The inputs of the block are subject to the configuration chosen by the user, such as the omission of auxiliary elements. Variables *YR*, *Ysys*, *UM*, and

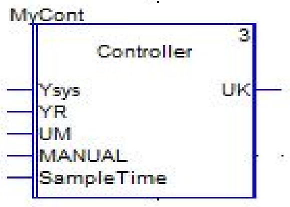

Fig. 2. FBD block in Unity Pro

UK are absolute variables, i.e. the values directly measured from the plant. The controller transforms these variables into incremental variables relative to (X_0, U_0, Y_0) for its internal use.

The controller is designed to run using the cyclic mode of operation of the PLC, meaning that the controller will be constantly being called. Each time it is called, a certain number of FISTA iterations are performed. This process is repeated until the algorithm converges. Subsequent calls to the controller do nothing until the next sampling time arrives. The auxiliary elements are called once the MPC has converged. We also limit the time we give the controller to calculate the control action. If FISTA algorithm does not converge within the given calculation time an auxiliary controller is utilized. This implementation approach is dependent on the controller FBD being called frequently. As such, the addition of other FBD blocks in the PLC may hinder the performance of the controller.

VII. EXAMPLE

We tested the Matlab library and resulting controller on a quadruple water tank system [2]. An FBD block was created for Unity Pro and programmed in a Modicom M340 PLC. The system was simulated in the PLC with another FBD block that contained the system's differential equations solved using Euler method.

A. System description

The system consists of four water tanks, two of them located over the other two. The two upper water tanks feed water to the lower ones. There are two sources of water, whose flow can be controlled with two valves. Each source feeds water to one of the upper water tanks and to one of the lower ones as shown in Figure 3. The control objective is to regulate the water level of the two lower tanks. The model of the system has four states, each one being the water level of one of the tanks. The system input is the amount of water flow in each of the sources and the system output is the water level of the two lower tanks. The sample time is 5 seconds.

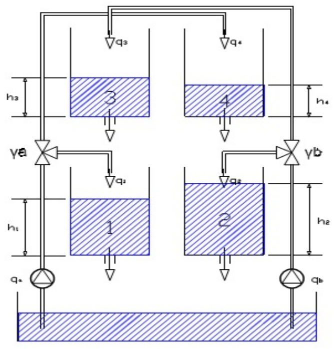

Fig. 3. Quadruple-tank system

The water flow in each of the sources is restricted between a lower and an upper level. The same can be said for the water level of the tanks.

This system has $n = 4$, $m = 2$ and $p = 2$. We chose a prediction horizon $N = 30$. As such, the number of decision variables of the resulting QP problem (5) is 180.

The generated controller includes all additional elements shown in Section V, except for the predictor - since the system has no delay.

B. Results

We controlled the system for 250 sample times. Starting at the linearization point (X_0, U_0, Y_0) and in manual mode, with the manual control action being equal to U_0. At sample time 10 we activated the automatic mode and steered the system to a new reference. At sample time 120 the reference was changed.

Figures 4 and 5 show the evolution of the absolute values of the system output and input, respectively. As can be observed in Figure 4, the system converges to the given reference - shown in dashed lines - in spite of the MPC using a linear model to control a non-linear system.

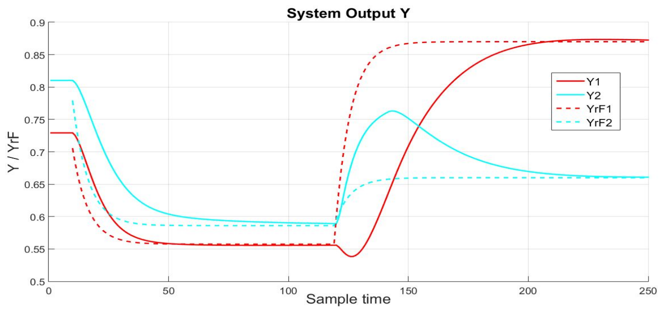

Fig. 4. System output (reference is represented in dashed lines)

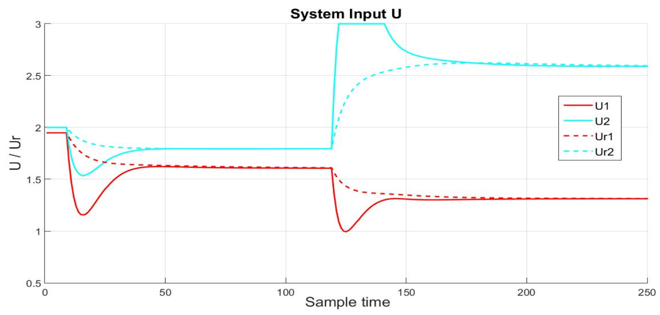

Fig. 5. Cotrol action (reference is represented in dached lines)

VIII. ANALYSIS OF MEMORY REQUIREMENTS

In order to analyse the memory requirements of the controller, tests were performed for generic systems of different sizes and with different prediction horizons, since these parameters are the ones that affect memory consumption the most.

Two series of tests were performed. In one of them, controllers for the quadruple-tank system shown in the previous section were generated, each of them with a different value of N. In the second one, controllers were generated for random systems with the same values of m, p and N, but different values of n. The resulting controllers were programmed into a Modicom M340 PLC and their code memory consumption and variable storage memory consumption were measured.

Figures 6 and 7 shows the results. As can be observed, the required memory grows linearly with N and quadratically with n. It is worth mentioning that the memory consumption grows linearly with m and quadratically with p. However, the two main contributors to memory growth are N and, more importantly, n.

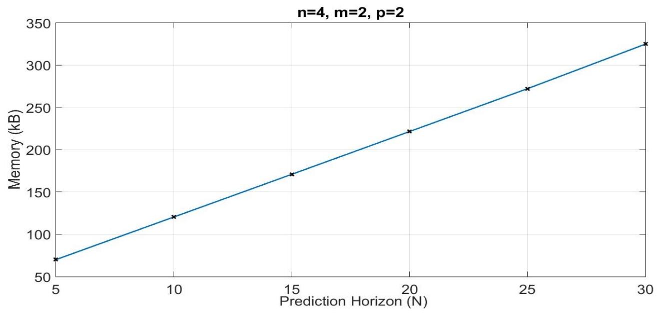

Fig. 6. Memory in relation to the prediction horizon N.

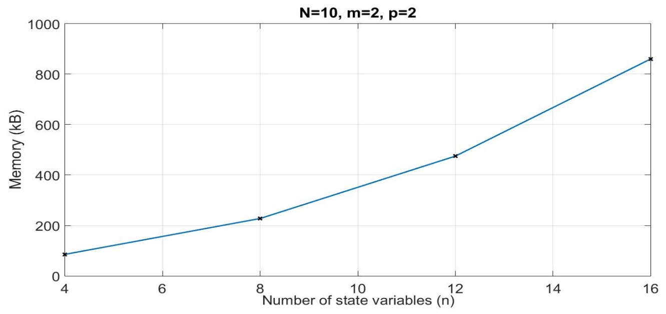

Fig. 7. Memory in relation to the number of state variables n

IX. CONCLUSIONS

The developed library generates the code of a MPC based controller for PCs by exploiting the structure of the MPC formulation and using a hand tailored version of FISTA algorithm. This very same algorithm and code generation procedure can be altered to generate controllers in other programming languages, such as C, and for other embedded systems. In fact, this is one of the main short term objectives of future work. One of the main focuses when developing the library was the creation of a tool that facilitated the implementation of predictive controllers in PLCs and other embedded systems. We aim to make it easy to install and use. We are also working to provide a controller that includes all guaranties and takes into account all practical considerations related to its implementation in realistic industrial environments by including additional elements to the MPC.

Future work includes expanding the guaranties and practical considerations of the controller, such as providing a robust control formulation or the possibility of changing controller parameters on-line; and to generate controllers for other embedded systems and programming languages.

ACKNOWLEDGMENT

The authors would like to thank MEyC and FEDER funds for financing projects DPI2013-48243-C2-2-R and DPI2016-76493-C3-1-R.

REFERENCES

[1] M. Alamir, “Monitoring control updating period in fast gradient based NMPC”, in European Control Conference, 2013.

[2] I. Alvarado, D. Limon, W. García-Gabín, T. Alamo, E. F. Camacho, “An Educational Plant Based on the Quadruple-Tank Process”, in IFAC Proceeding Volumes, vol. 39, issue 6, pp. 82-87, 2006.

[3] A. Beck, M. Teboulle, “A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems”, in SIAM J. Imaging Sciences, vol. 2, No.1, pp. 183-202, 2009.

[4] B. J. T. Binder, D. K. M. Kufoalor, A. Pavlov, and T. A. Johansen, “Embedded Model Predictive Control for an Electric Submersible Pump on a Programmable Logic Controller”, in 2014 IEEE Conference on Control Applications (CCA), 2014.

[5] E. Camacho, C. Bordons, “Model Predictive Control”, Springer Science & Business Media, 2013.

[6] J. Currie, A. Prince-Pike, D. I. Wilson, “Auto-Code Generation for Fast Embedded Model Predictive Controllers”, Mechatronics and Machine Vision in Practice (M2VIP), 19th International Conference, pp. 116-122, 2012.

[7] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, C. Jones, “Efficient interior point methods for multistage problems arising in receding horizon control”, Proc. 51st IEEE Conf. Decision and Control (CDC 2012), pp. 668-674, Maui, HI, USA, Dec. 2012.

[8] H. J. Ferreau, H. G. Block, M. Diehl, “An online active set strategy to overcome the limitations of explicit MPC”, Int. J. Robust Nonlinear Control, vol. 18, pp. 816-830, July 2008.

[9] B. Huyck, H. J. Ferreau, M. Diehl, J. D. Brabanter, J. Van Impe, B. D. Moor, and F. Logist, “Towards Online Model Predictive Control on a Programmable Logic Controller: Practical Considerations”, Mathematical Problems in Engineering, vol. 2012, pp. 1-20, 2012.

[10] D. K. M. Kufoalor, S. Richter, L. Imsland, T. A. Johansen, M. Morari, G. O. Eikrem, “Embedded model predictive control on a PLC using a primal-dual first-order method for a subsea separation process”, in Proc. 22nd IEEE Mediterranean Conf. Control and Automation (MED 2014), Palermo, Italy, 2014.

[11] U. Maeder, F. Borrelli, M. Morari, “Linear offset-free Model Predictive Control”, in Automatica, vol. 45, pp. 2214-2222, 2009.

[12] J. Mattingley and S. Boyd, “CVXGEN: A Code Generator for Embedded Convex Optimization”, Optimization and Engineering, vol. 13, no. 1, pp. 1-27, 2012.

[13] Y. Nesterov, “Introductory lectures on convex optimization”, Springer, 2010.

[14] Y. Nesterov, “Smooth minimization of non-smooth functions”, Mathematical Programming, vol. 103, no. 1, pp. 127-152, Springer, 2005.

[15] M. Pereira, D. Limon, D. Muñoz de la Peña, T. Alamo, “MPC implementation in a PLC based on Nesterov’s fast gradient method”, in 23rd Mediterranean Conf. Control and Automation (MED), Torremolinos, Spain, 2015.

[16] S. Richter, C. N. Jones, M. Morari, “Computational Complexity Certification for Real-Time MPC With Input Constraints Based on the Fast Gradient Method”, in IEEE Transactions on Automatic Control, vol. 57, no. 6, pp. 1391-1403, 2011.

[17] R. T. Rockafellar, “Convex analysis”, Princeton, NJ: Princeton University Press, 1970.

[18] L. M. Santos, D. Limon, J. E. Normey-Rico, T. Alamo, “On the explicit dead-time compensation of robust model predictive control”, in Journal of Process Control, vol. 22, pp. 236-246, 2012.

[19] F. Ullmann, “A Matlab toolbox for C-code generation for first order methods”, Master’s thesis, Eidgenssische Technische Hochschule Zürich, 2011.

[20] G. Valencia-Palomo, K. R. Hilton, J. A. Rossiter, “Predictive Control implementation in a PLC using the IEC 1131.3 programming standard”, in European Control Conference (ECC), pp. 1317-1322, 2009.

[21] P. Zometa, M. Kögel, and R. Findeisen, “μAO-MPC: A Free Code Generation Tool for Embedded Real-Time Linear Model Predictive Control”, in 2013 American Control Conference (ACC), Washington, DC, USA, June 2013, pp. 5320-5325.