XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0092

Desarrollo de un Controlador Predictivo para Autómatas programables basado en la normativa IEC 61131-3

Pablo Krupa, Daniel Limón, Teodoro Álamo

Departamento de Ingeniería de Sistemas y Automática. Universidad de Sevilla. España
E-mails: pabkrugar@alum.us.es, dlm@us.es, talamo@us.es

Resumen

En este trabajo se describe la implementación de un Controlador Predictivo (MPC) en un Autómata Programable (PLC) a través de una librería en MATLAB. La librería toma los datos del sistema y los parámetros del MPC y genera con ello el código del controlador de forma que se minimice el uso de memoria del PLC. El código generado se empaqueta en un archivo que puede ser directamente importado al software de control del PLC. El lenguaje de programación de dicho código sigue la normativa IEC 61131-3. En concreto, el controlador está programado con el lenguaje ST.

El controlador incluye un observador de estado y un estimador de perturbaciones, un Steady State Target Optimicer (SSTO), un predictor en bucle abierto y posibilidad de trabajar en modo manual.

Este trabajo proporciona una metodología, basada en el algoritmo FISTA, para resolver el problema de optimización requerido para la implementación del controlador MPC.

En este trabajo también se muestran los resultados de pruebas realizadas sobre el uso de memoria del PLC para sistemas de distintos tamaños, así como un ejemplo de uso del controlador aplicado en el PLC Modicon m340, de la empresa Schneider Electric. El software de control del PLC que ha sido usado, y para el que está programada la librería de MATLAB, es Unity Pro XL.

Palabras clave: Control Predictivo, Autómata Programable, FISTA, IEC 61131-3, Generación de código.

1. Introducción

El Control Predictivo (MPC) es una estrategia de control basada en optimización, que resulta de gran utilidad para el control de sistemas multivariables con restricciones [3]. Esta estrategia de control es usada en numerosos sectores industriales, tales como refinerías de petroleo, plantas eléctricas o la industria química. Sin embargo, su implementación ha estado enfocada a sistemas con muchos recursos, tales como un PC, dada su complejidad y gran demanda computacional.

En los últimos años la comunidad científica ha hecho avances en la programación de Controladores Predictivos en equipos con menos recursos que un PC, tales como Arduino, FPGA o PLC. Los Autómatas Programables (PLC) son los equipos más extendidos en la industria para la implementación de lazos de control. Se caracterizan por su robustez, fiabilidad, sistema de comunicación integrado y facilidad de programación. Sin embargo, son equipos con limitaciones en cuanto a capacidad computacional o de memoria en comparación con los de un PC. La estrategia de control PID es la que se suele implementar en estos equipos. En [7] y [2] se muestran casos de estudio realistas en los que se implementa un MPC en un PLC, y en [6] se muestran consideraciones prácticas sobre la implementación de un MPC en un PLC.

En [10] se muestra la implementación de un MPC en un PLC haciendo uso de la normativa IEC 61131-3, al igual que en este trabajo. Esta implementación hace uso de funciones que resuelven el problema de optimización del MPC en funcion de las matrices resultantes del mismo. Este enfoque, a pesar de estar orientado a implementación en un PLC, no aprovecha la estructura del problema para reducir la memoria necesaria, lo cual es de especial importancia en los PLC dadas sus limitaciones de memoria. Para ilustrar esto, mencionar que el tamaño de la memoria dedicada a datos del PLC usado en este trabajo tiene un tamaño de 4Mb.

Uno los principales avances en la implementación del MPC en sistemas con pocos recursos son los avances en algoritmos que resuelvan de forma rápida y eficiente problemas de programación cuadrática (QP), dado que el problema de optimización del MPC, cuya resolución proporciona la acción de control, es un problema de este tipo. Estos avances han dado lugar a la creación de diversas librerías de generación de código para la resolución eficiente de problemas QP, tales como: FiOrdOs [12], CVXGEN [9] o μAO-MPC [13]. Este enfoque proporciona un código más eficiente, tanto en memoria como en complejidad, que el enfoque del párrafo anterior.

Sin embargo, las librerías previamente mencionadas generan algoritmos genéricos para la resolución de problemas QP en código C/C++, que no son directamente trasladables a una implementación en PLC. La contribución de este trabajo es la creación de una librería que genera un código específicamente diseñado para optimizar la memoria de la formulación de MPC que se muestra en la Sección 2 y programada en el lenguaje de programación estándar de los autómatas programables ST. Este lenguaje está recogido bajo la normativa IEC 61131-3, que regula los estándares de los lenguajes de programación de los autómatas programables. La librería aprovecha la estructura del problema de optimización que se deriva del MPC y del algoritmo FISTA [1], que es el que se usa para obtener su solución.

Además, la librería genera un controlador que posee una serie de características necesarias para su implementación en el entorno industrial, tales como: control multivariable, modo manual y automático, cancelación de offset, optimización de consigna, restricciones en entrada y estado, realimentación de salida y compensación de retardo de la planta.

La salida de la librería es un archivo de extensión ".XDB"que se puede importar directamente al software de control de PLCs *Unity Pro XL*, de la empresa Schneider Electric, generándose un bloque FBD. Todas las pruebas que se han realizado en este trabajo han sido en este software o directamente en un PLC modelo Modicon m340, de la misma empresa.

Este trabajo está organizado de la siguiente manera: la formulación del MPC se muestra en la Sección 2, el algoritmo FISTA se muestra en la Sección 3, la arquitectura del controlador, incluyendo una breve descripción de cada uno de los elementos auxiliares, se muestra en la Sección 4, la explicación de la librería en MATLAB se encuentra en la Sección 5, un ejemplo de aplicación del controlador se muestra en la Sección 6, los resultados sobre uso de memoria en función del tamaño del problema se muestran el la Sección 7, y la conclusión y comentarios adicionales se muestran en la Sección 8.

2. Formulación del MPC

El MPC controla un sistema modelado por ecuaciones discretas en variables de estado obtenido a partir de la linalización del sistema en torno a un punto de funcionamiento (X_0, U_0, Y_0).

$$x(k+1) = Ax(k) + Bu(k) \quad (1a)$$
$$y(k) = Cx(k) \quad (1b)$$

Donde $x(k) \in \mathbb{R}^n$ es el vector de estado, $u(k) \in \mathbb{R}^m$ es el vector de acciones de control e $y(k) \in \mathbb{R}^p$ es el vector de salidas del sistema. $x(k)$, $u(k)$ e $y(k)$ son variables incrementales respecto al punto de funcionamiento (X_0, U_0, Y_0)

Se definen restricciones en caja para el estado y para las acciones de control de la siguiente forma.

$$LB_x \leq x(k) \leq UB_x \quad (2)$$
$$LB_u \leq u(k) \leq UB_u$$

La ley de control del MPC se deriva de la resolución del siguiente problema de optimización.

$$J^* = \min_u \quad J(x, u, x_r, u_r) \quad (3)$$
$$s.a. \quad x(k+1) = Ax(k) + Bu(k), \quad (3a)$$
$$LB_x \leq x(k) \leq UB_x, \quad (3b)$$
$$LB_u \leq u(k) \leq UB_u, k = 0, \ldots N-1 \quad (3c)$$
$$x(0) = X \quad (3d)$$
$$x(N) = x_r \quad (3e)$$

Siendo $J(x, u, x_r, u_r)$ la función de coste, cuya expresión es la siguiente.

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

donde N es el horizonte de predicción, u es la secuencia de N acciones de control futuras, x la secuencia de N estados futuros predichos del sistema, $x_r \in \mathbb{R}^n$ es el estado de referencia, $u_r \in \mathbb{R}^m$ es la acción de control de referencia, X es el estado leído en el periodo de muestreo actual, y $Q \in \mathbb{R}^{n \times n} \succ 0$ y $R \in \mathbb{R}^{m \times m} \succ 0$ son matrices diagonales que penalizan la discrepancia entre x y x_r, y entre u y u_r, respectivamente.

3. Algoritmo FISTA

Considérese el problema QP siguiente.

$$\min_z \ \frac{1}{2} z' H z + f' z \quad (5)$$
$$s.a. \ z \in \mathcal{Z} \quad (5a)$$
$$Az = b \quad (5b)$$

donde $z \in \mathbb{R}^{n_z}$ son las variables de decisión, el conjunto $\mathcal{Z} = \{z \in \mathbb{R}^{n_z} : LB \leq z \leq UB\}$, H es una matriz diagonal definida positiva, $A \in \mathbb{R}^{m_z x n_z}$, con $n_z > m_z$, y $[A\ b]$ es de rango m_z.

Nótese que el problema de optimización del MPC (3) se puede reescribir de esta forma. Tomando,

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

La solución en linea de este problema QP se ha realizado mediante el algoritmo FISTA [1], que es un algoritmo basado en gradiente que resuelve el problema mediante teoría de dualidad. A continuación se muestran los conceptos básicos de este algoritmo y su formulación.

Defínase,

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

$$L(z,\lambda) = J(z) - \lambda'(Az - b) \quad (7)$$

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

$$f(\lambda) = L(z(\lambda),\lambda) \quad (9)$$

La funcion $L(z,\lambda)$ se denomina la función Lagrangiana del problema, λ es el multiplicador de Lagrange y $f(\lambda)$ es la función dual.

Sea z^* la solución óptima del problema original y J^* el coste óptimo, entonces

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

para todo λ, pues $Az^* - b = 0$. Por lo tanto la función dual es una cota inferior del problema de optimización. El objetivo es pues obtener la mejor cota inferior, maximizando la función dual, es decir, resolviendo el problema

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

Definiendo $f^* = f(\lambda^*)$, se puede demostrar que el problema anterior posee la propiedad de dualidad fuerte, es decir, $J^* = f^*$ y $z^* = z(\lambda^*)$.

Para el problema QP (5), en el que H es diagonal y definido positivo, el cálculo de $z(\lambda)$ tiene una solución explícita directa, puesto que es equivalente a la resolución de n_z problemas de optimización desacoplados de una sola variable.

$$z(\lambda) = arg\min_{z\in\mathcal{Z}} \tfrac{1}{2}z'Hz + (f - A'\lambda)'z \quad (12)$$

El objetivo es obtener la mejor cota inferior de $z(\lambda)$, maximizando la función dual, cuya obtención se basa en la siguiente desigualdad. Esta desigualdad se cumple si $H \succ 0$.

$$f(\lambda + \Delta\lambda) \geq f(\lambda) - \Delta\lambda'(Az(\lambda) - b) - \frac{1}{2}\Delta\lambda' W \Delta\lambda \quad (13)$$

Con

$$W = A'H^{-1}A \quad (14)$$

La obtención de la desigualdad (13) así como su demostración no se detallan en este trabajo. Como referencia para la obtención de esta desigualdad se citan los trabajos de Rockafellar [11] y Fletcher [5].

Nótese que

$$arg\max_{\Delta\lambda} -\Delta\lambda'(Az(\lambda) - b) - \frac{1}{2}\Delta\lambda' W \Delta\lambda = arg\min_{\Delta\lambda} \Delta\lambda'(Az(\lambda) - b) + \frac{1}{2}\Delta\lambda' W \Delta\lambda \quad (15)$$

Y, dado que W es invertible, por ser H diagonal y A de rango completo por filas, el óptimo del problema (15) se alcanza en el punto en el que se anula el gradiente, es decir,

$$Az(\lambda) - b + W\Delta\lambda = 0 \quad (16)$$

Llegando finalmente a,

$$\Delta\lambda = -W^{-1}(Az(\lambda) - b) \quad (17)$$

Los pasos del algoritmo FISTA se detallan en Algoritmo 1, donde TOL es un parámetro de entrada del algoritmo.

Algoritmo 1 FISTA

1: $k = 1, \lambda_1 = \eta_1 = 0, t_1 = 1$
2: Repetir
   a: Obtener $z(\lambda_k)$ resolviendo (12)
   b: $\Delta\lambda = -W^{-1}(Az(\lambda_k) - b)$
   c: $\eta_k = \lambda_k + \Delta\lambda$
   d: $t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$
   e: $\lambda_{k+1} = \eta_k + \frac{t_k - 1}{t_{k+1}}(\eta_k - \eta_{k-1})$
   f: $k = k + 1$
3: Hasta que $|Az(\lambda_k) - b| \leq TOL$

4. Arquitectura del controlador

El controlador que se muestra en este trabajo obtiene la acción de control a partir del MPC descrito en la Sección 2, pero también incluye una serie de elementos auxiliares que proporcionan al controlador una serie de características adicionales: cancelación de offset, estimación de estado, control de sistemas con retardo, SSTO y modo manual de funcionamiento.

La arquitectura del controlador se muestra en la Figura 1, donde el objetivo de control es llevar la salida del sistema y a la referencia dada y_r.

A continuación se explican cada uno de los bloques auxiliares.

4.1. Observador de estado y estimador de perturbaciones

Con el objetivo de eliminar el error en régimen permanente ante referencias asintóticas a una

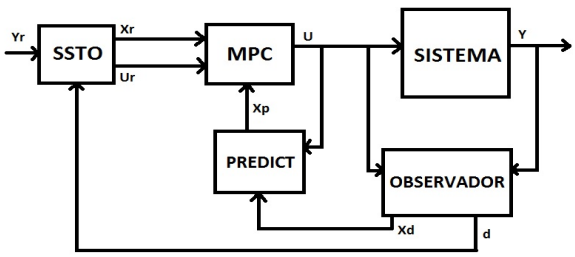

Figura 1: Arquitectura del controlador

constante, se ha incluido un observador de estado de Luenberger y un estimador de perturbaciones, que se usa para estimar y predecir la discrepancia entre la salida medida del sistema y la predicha. De este bloque se obtiene, en cada tiempo de muestreo, un estado estimado $x_d \in \mathbb{R}^n$ y una perturbación estimada $d \in \mathbb{R}^p$, que se usan para iniciar el problema del MPC (3) y el del SSTO (20).

La formulación de este bloque es la siguiente.

$$x_d^+ = Ax_d + Bu + L_x(y - Cx_d - D_d d) \quad (18)$$
$$d^+ = d + L_d(y - Cx_d - D_d d) \quad (19)$$

Donde A, B y C son las matrices del modelo del sistema (1), L_x, L_d y D_d matrices que se eligen para que el estimador sea estable, u la acción de control actual e y la salida medida del sistema.

La demostración sobre la convergencia de la salida del sistema y a la referencia dada y_r se puede ver en [8], donde en este caso se está considerando el caso $n_d = p$ y B_d matriz nula.

En la práctica, se toma $D_d = I_p$ y L_x, L_d de forma que el estado estimado x_d converja en un tiempo significativamente inferior al tiempo característico del sistema.

4.2. Predictor en bucle abierto

Con el objetivo de poder controlar sistemas con grandes retrasos de forma eficiente, es decir, sin tener que aumentar el estado del sistema, se añade un bloque que toma el estado estimado x_d, proporcionado por el observador de estado, y el historial de acciones de control pasadas - en un horizonte igual al retraso del sistema ret, dado en término del número de tiempos de muestreo de retraso - y devuelve el estado predicho x_p en ret tiempos de muestreo.

Por lo tanto, la ecuación (3d) se sustituye por $x(0) = x_p$.

4.3. Steady State Target Optimizer

El Steady State Target Optimizer (SSTO) calcula un punto de funcionamiento de referencia (x_r, u_r) a partir de una referencia en salida y_r, de tal forma que se minimice un cierto criterio. El SSTO transforma una referencia en términos de salida del sistema, que es la más habitual en la práctica, a la mejor referencia alcanzable en términos de estado x_r y acción de control u_r, de tal forma que el conjunto (x_r, u_r) cumple las restricciones del sistema (2).

El conjunto (x_r, u_r) se obtiene en cada tiempo de muestreo de la resolución del problema de optimización (20), que se ha diseñado de tal forma que el hessiano del problema QP que se deriva de él sea diagonal y definido positivo. De esta forma se puede aplicar el algoritmo FISTA (Sección 3) y las mismas técnicas de ahorro de memoria que se han usado para el MPC (Sección 2). EL SSTO hace uso de la perturbación estimada por el estimador de perturbaciones (18) y (19). De esta forma la referencia (x_r, u_r) proporcionada por el SSTO será tal que la salida del sistema converja a la referencia dada en régimen permanente, siempre y cuando la referencia converja a un valor constante de forma asintótica.

$$(x_r, u_r) = \underset{x_r, u_r}{arg\,mín} \quad S(y_r, d) \quad (20)$$
$$s.a. \quad x_r = Ax_r + Bu_r \quad (20a)$$
$$y_r - D_d d = Cx_r + h \quad (20b)$$
$$LB_x \leq x_r \leq UB_x \quad (20c)$$
$$LB_u \leq u_r \leq UB_u \quad (20d)$$

Con

$$S(y_r, d) = ||x_r||^2_{Q_r} + ||u_r||^2_{R_r} + ||h||^2_{T_h} \quad (21)$$

Donde las matrices Q_r, R_r y T_h ponderan el estado de referencia, la acción de control de referencia y la variable de holgura h, respectivamente. Las tres matrices son diagonales y definidas positivas.

4.4. Modo de funcionamiento manual

El controlador posee un modo de funcionamiento manual, en el que la acción de control aplicada sobre el sistema es directamente la proporcionada por el usuario. En caso de estar activo el modo manual, el controlador solo ejecuta el observador de estado y estimador de perturbaciones.

En general, el controlador será iniciado en modo manual, se aplicará una acción de control que lleve al sistema a un punto de equilibrio y una vez que está en dicho punto de equilibrio y el observador de estado haya convergido se activará en modo de control automático.

5. Librería en MATLAB

Como se ha mencionado en la introducción, hay dos enfoques a la hora de implementar un MPC en un PLC. La primea, [10], implementa una función genérica de resolución de problemas QP a la que se le pasan las matrices de dicho problema. Esta solución no optimiza el uso de memoria, por lo que resulta inviable para problemas de gran tamaño. Por otro lado, existen aplicaciones orientadas a resolver problemas QP de forma eficiente a través de generación de código ([12], [9] o [13], por ejemplo). Sin embargo, estas aplicaciones generan código en C/C++ y no están orientadas a ningún problema QP en particular. Además, únicamente resuelven el problema QP, pero no proporcionan un controlador propiamente dicho.

En este trabajo se presenta una librería en MATLAB que genera controladores predictivos aprovechando la estructura que presenta el problema para la formulación del MPC mostrada en la Sección 2 y para el algoritmo FISTA (Sección 3). La especialización de la librería a una formulación concreta permite la optimización del problema QP resultante en mayor medida. Se busca minimizar la cantidad de memoria necesaria así como la cantidad de operaciones que se deben realizar en cada iteración del algoritmo FISTA. Además, se genera un controlador con todos los elementos auxiliares descritos en la sección 4 y programado en el lenguaje ST, según la normativa IEC 61131-3, que es un lenguaje estándar de los PLC.

La librería toma como datos las matrices del modelo del sistema y los parámetros del controlador, realiza una serie de cálculos fuera de línea para la generación de ciertas matrices y parámetros del problema y finalmente genera el controlador en forma de un archivo de extensión ".XDB". Este archivo contiene la declaración de variables y el código del controlador y puede ser directamente importado a Unity Pro XL. Al ser importado se crea un bloque FBD (Figura 2). Sus entradas son la salida leída del sistema *Ysys*, la referencia de salida *YR*, el valor de la acción de control en modo manual *UM* - en valor absoluto, el Booleano *MANUAL* para indicar si trabajar en modo manual o automático y el valor del tiempo de muestreo *ST*. El bloque devuelve el valor de la acción de control del periodo de muestreo actual *UK*, que se obtiene de la resolución del problema (5).

El ahorro en memoria se obtiene mediante el aprovechamiento de la estructura que presentan las matrices requeridas para la resolución del algoritmo FISTA. Por ejemplo, la matriz A de (5b) presenta una estructura en banda compuesta únicamente por las matrices del modelo del sistema

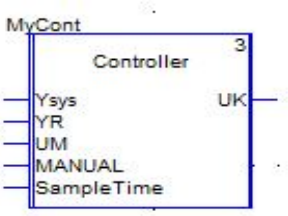

Figura 2: Bloque FBD en Unity Pro XL

(1). Por tanto, no se guarda en memoria la matriz A completa, sino únicamente las matrices del modelo. La multiplicación de A con otras matrices o vectores se realiza mediante bucles que recorren su estructura. Este mismo planteamiento se realiza para el resto de matrices y vectores del algoritmo FISTA: W (14), H , f y b (5).

Como ejemplo del orden de ahorro de memoria véase la matriz A, cuyo número de variables es igual a $N(N+1)n(n+m)$, mientras que el número de variables necesarias usando la librería es $n(n+m)$, ya que únicamente hay que almacenar las matrices A y B del modelo del sistema (1). Esto mismo ocurre de forma muy similar con el resto de matrices y vectores del problema (si bien el contraste no es tan pronunciado en el caso de los vectores).

6. Ejemplo de aplicación

En esta sección se muestra un ejemplo de uso del controlador en *Unity Pro XL*. El sistema a controlar consta de cuatro tanques de agua, dos de ellos situados encima de los dos restantes, de tal forma que los tanques superiores evacuan agua a los inferiores por gravedad. Hay dos fuentes de alimentación de agua, cuyos caudales se pueden controlar con sendas válvulas. Cada fuente alimenta uno de los tanques inferiores y uno de los superiores, tal y como se muestra en la Figura 3. El modelo de la planta consta de cuatro estados, siendo cada uno de ellos el nivel de agua de cada uno de los cuatro tanques. La salida del sistema es la altura de ambos tanques inferiores. El objetivo de control es regular la altura de los dos tanques inferiores. La acción de control es la regulación de las válvulas que controlan el caudal de cada una de las fuentes de agua. Tanto la altura de los tanques como el valor de las acciones de control deben estar contenidas entre un valor superior e inferior, es decir, hay restricciones en caja tanto en el estado como en la acción de control.

Para este sistema $n = 4$, $m = 2$ y $p = 2$. Se toma un horizonte de predicción $N = 30$. Por lo tanto, el número de variables de decisión del problema QP (5) es 180.

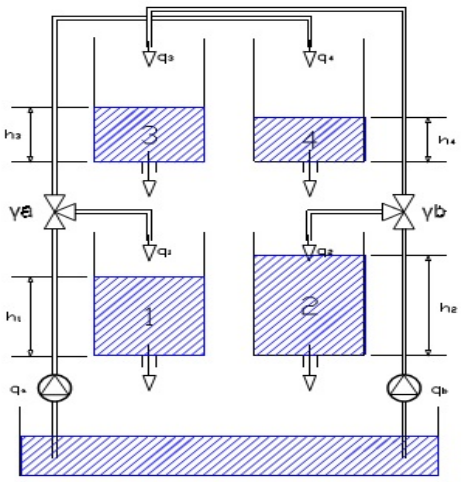

Figura 3: Planta de cuatro tanques de agua

Se han introducido las matrices del modelo del sistema y los parámetros del controlador en la librería de MATLAB, se ha importado en controlador generado en Unity Pro XL y se ha programado en un PLC Modicon m340.

Las figuras 4, 5 y 6 muestran la evolución de la salida del sistema, el estado del sistema y la acción de control, respectivamente. Todas ellas en valor absoluto. Como se puede observar en Figura 4, la salida del sistema converge a la referencia - en líneas discontinuas - a pesar de que se está regulando al planta a un punto distinto del de linealización. Por otro lado, en Figura 5 se puede observar cómo el estado real del sistema no converge al estado de referencia generado por el SSTO, x_r. Sin embargo, el estado predicho x_p sí lo hace.

En las figuras se representan 10 periodos de muestreo en modo manual, tras los cuales se activa el modo automático del controlado.

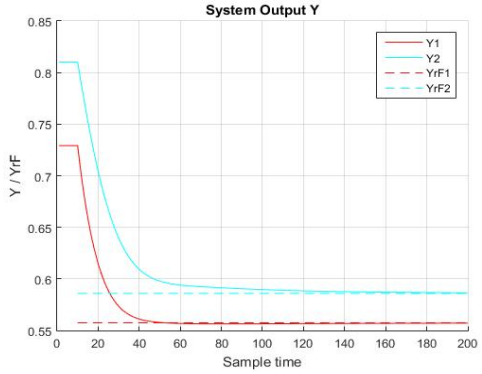

Figura 4: Salida del sistema y referencia en salida

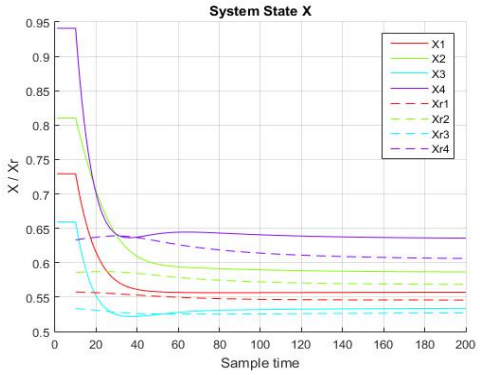

Figura 5: Estado real del sistema y referencia en estado

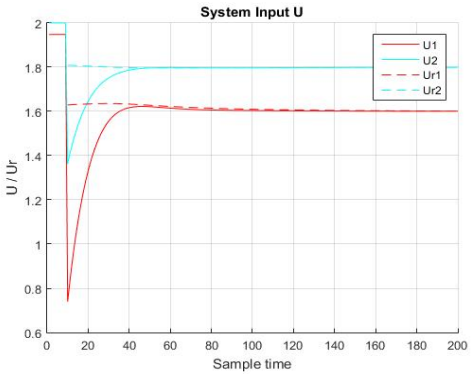

Figura 6: Accion de control aplicada y de referencia

7. Uso de memoria

Se han realizado dos conjuntos de pruebas para estudiar el uso de memoria de los controladores generados en función del tamaño del problema. En concreto, se han realizado dos conjuntos de pruebas, ambos sobre el PLC Modicom m340. En el primer conjunto (Figuras 7 y 8) se han mantenido constantes las matrices del modelo del sistema y se incrementado el valor del horizonte de predicción (N). Los valores de las dimensiones del modelo del sistema son $n = 4$, $m = 2$ y $p = 2$. En el segundo conjunto (Figuras 9 y 10) se han mantenido constantes los parámetros $N = 10$, $m = 2$ y $p = 2$, y se ha incrementado el valor de n.

Se ha medido tanto la memoria usada para almacenar los datos del problema como la memoria del código del programa. Los valores de memoria están expresados en kb.

Como se puede observar, la cantidad de memoria crece de forma lineal con el horizonte de predicción N y de forma cuadrática con el número de estados n. A pesar de que no se representa en este trabajo la evolución de la memoria en función de m o p, dejando el resto constantes, mencionar que

su comportamiento es lineal con m y cuadrático con p. Sin embargo, el parámetro que mayor efecto tiene sobre la memoria es n.

El conjunto de pruebas en las que se ha incrementado N se ha realizado usando el modelo de la planta de cuatro tanques de agua descrito en Sección 6.

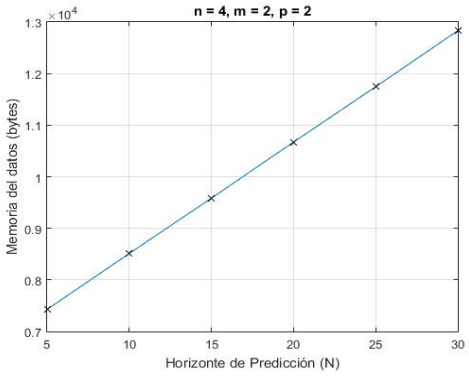

Figura 7: Memoria de datos en función de N

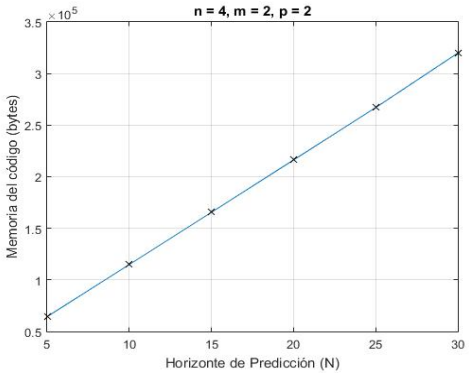

Figura 8: Memoria del código en función de N

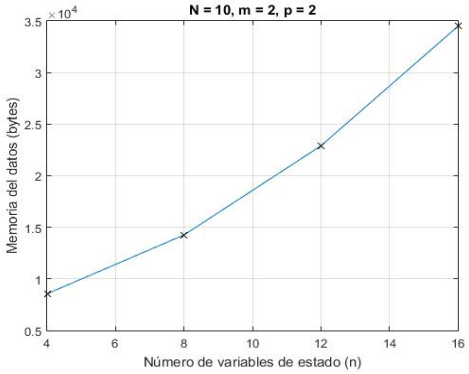

Figura 9: Memoria de datos en función de n

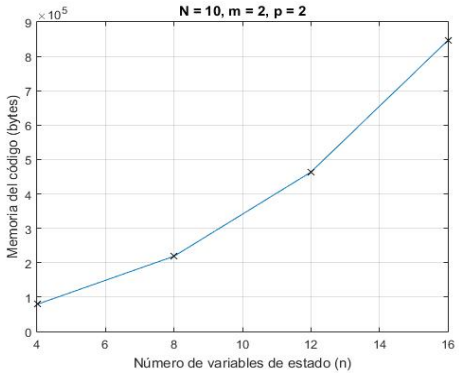

Figura 10: Memoria del código en función de n

8. Conclusiones

La librería propuesta genera el código un controlador predictivo orientado a la implementación en un PLC que aprovecha la estructura de una formulación concreta de MPC y un algoritmo concreto de resolución de problemas QP, FISTA. Esto permite obtener resultados de uso de memoria inferiores a los de otras soluciones ([10]), lo cual es de especial importancia considerando las grandes limitaciones de memoria de los PLC. El código generado está escrito en lenguaje estándar de los PLC, recogido en la norma IEC 61131-3, y puede ser directamente importado a su software de control.

Un objetivo del desarrollo de esta librería es la creación de una herramienta que facilite la implementación de controladores predictivos en autómatas programables. De ahí que se busque que sea fácil de instalar, usar y de introducir el controlador en el autómata. También se busca que el controlador posea todas las garantías y tenga en cuenta todas las consideraciones prácticas que se esperan de un controlador en el ámbito industrial.

Como posibles trabajos futuros se destacan la ampliación de estas garantías y consideraciones prácticas, así como la inclusión de características adicionales que resulten de interés. Como ejemplo se destacan las siguientes: formulación de control robusto, posibilidad de cambios de los parámetros del controlador y de las matrices del modelo del sistema en línea, o generación de código para otros equipos embebidos.

Agradecimientos

A los autores les gustaría agradecer al Mineco y los fondos FEDER por la financiación de los proyectos DPI2013-48243-C2-2-R y DPI2016-76493-C3-1-R que han dado lugar a este trabajo.

Referencias

[1] 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.

[2] 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.

[3] E. Camacho, C. Bordons, "Model Predictive Control", Springer Science & Business Media, 2013.

[4] A. Ferramosca, D. Limón, I. Alvarado, T. Alamo, E. Fernández, "MPC for Tracking With Optimal Closed-Loop Performance", in Automatica (Oxford), vol. 45, pp. 1975-1978, 2009.

[5] R. Fletcher, "Practical methods of optimization", 2nd ed. Chichester: New York, 1987.

[6] B. Huyck, H. J. Ferreau, M. Diehl, J. D. Brabanter, J. F. M. V. 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.

[7] D. K. M. 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 Proc. 22nd IEEE Mediterranean Conf. Control and Automation (MED 2014), Palermo, Italy, 2014.

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

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

[11] R. T. Rockafellar, "Convex analysis", Princeton, NJ: Princeton University Press, 1970.

[12] F. Ullmann, "A Matlab toolbox for C-code generation for first order methods", Master's thesis, Eidgenössische Technische Hochschule Zürich, 2011.

[13] 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.