Proyecto Fin de Máster

Máster Universitario en Ingeniería Industrial

Control automático de una piscina doméstica

Autor: Rafael Romero García

Tutores: José Ramón Domínguez Frejo y
Ramón Andrés García Rodríguez

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Máster
Ingeniería Industrial

Control automático de una piscina doméstica

Autor:
Rafael Romero García

Tutor:
José Ramón Domínguez Frejo
Profesor titular
Ramón Andrés García Rodríguez
Profesor

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Máster: Control automático de una piscina doméstica

Autor: Rafael Romero García

Tutor: José Ramón Domínguez Frejo y
Ramón Andrés García Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El Secretario del Tribunal

Agradecimientos

A mi familia, a mi pareja y a mi compañero Jesús.

Rafael Romero García

Sevilla, 2023

Resumen

La propuesta del presente trabajo surge principalmente de dos intereses. En primer lugar, surge del interés práctico en implementar un control automático de los niveles de pH y ORP en una piscina de uso doméstico. En segundo lugar, se tiene un interés académico por el control predictivo basado en modelo, buscando aprender acerca tanto de los aspectos más teóricos (fundamentos matemáticos, tipos, etc) como de los aspectos más prácticos (aplicaciones reales, ajuste del control, etc).

A partir de esta propuesta, se comenzó con la identificación del sistema (piscina), para posteriormente diseñar el control en el programa *Matlab/Simulink*®. Se realizaron varias simulaciones, ajustando los parámetros de control en el proceso, hasta dar con aquellos que lograban un seguimiento adecuado de las referencias sin producirse un gasto excesivo en las entradas del sistema.

Una vez satisfechos esos criterios, se dio paso a la conversión del código a lenguaje *Python* para su implementación en la *Raspberry Pi Pico*® presente en el sistema de la piscina.

Finalmente, se realizaron pruebas con distintas referencias para ambos niveles, con el objetivo de comprobar la efectividad del control en el caso real de la piscina. Dos primeras pruebas arrojaron resultados poco satisfactorios, pero que sirvieron para volver a realizar ajustes, acabando con una tercera prueba en la que se obtuvo un rendimiento adecuado del control.

El control se implementa en el seno de un código general ya presente en el microcontrolador del propietario de la piscina. Esto supone ciertas limitaciones, pues hay que amoldar el control a las necesidades del sistema real y del propietario (restricciones en la difusión de los líquidos actuadores, limitaciones en parámetros del control como el tiempo de muestreo, etc).

Abstract

This project arises mainly from two interests. Firstly, from the practical interest in implementing an automatic control for the pH and ORP levels in a domestic swimming pool. Secondly, from the academic interest in predictive control, aiming to learn about both the theoretical (mathematical principles, types, etc) and practical (real applications, control tuning, etc) aspects of it.

Following these interests, first step was to identify the model (swimming pool), followed by designing and tuning the control using *Matlab/Simulink*®. Several simulations were performed, adjusting the control parameters in the process until reaching a state where the setpoints were tracked properly and there was not an excessive use of the actuators.

Next step was converting the code to *Python* language to be able to implement it on the *Raspberry Pi Pico*® located in the pool system.

Finally, tests with different setpoints were carried out in order to verify the effectiveness of the control in the real case scenario of the pool. The first two tests gave unsatisfactory results, but served as basis for new adjustments, leading to a third and final test where an acceptable performance was achieved.

The control is implemented within a general code already present in the pool owner's microcontroller. This implies that some limitations must be taken into account, as the control has to be adapted to the needs of the real system and the owner's requirements (restrictions on the diffusion of the fluids, limitations on control parameters such as sampling time, etc).

Índice

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Piscina

1.1.1 pH

El pH es llamado así puesto que hace referencia a la potencia del hidrógeno. Fue el bioquímico danés S. P. L. Sørensen quien lo definió en 1909 [1] como el logaritmo negativo de la actividad de los iones de hidrógeno (a_{H+}), o lo que es lo mismo:

$$pH = -\log_{10} a_{H^+} \tag{1–1}$$

Despejando:

$$H^+ = 10^{-pH} \tag{1–2}$$

El agua se disocia en iones de hidrógeno (H^+) e iones de hidróxido (OH^-). A 25°C, la actividad (concentración molar) de ambos iones es la misma, 10^{-7} M. Por eso se dice que el pH neutro es igual a 7 (esta neutralidad varía con la temperatura ligeramente). Un pH menor del neutro significa que la potencia negativa es menor, luego la actividad del hidrógeno es mayor. Se tiene entonces lo que se conoce como una solución ácida. Por el contrario, un pH mayor que el neutro significa que la actividad del hidrógeno es menor, teniéndose una solución básica o alcalina [2].

En el agua de una piscina, es importante mantener el pH en unos niveles que típicamente rondan entre 7 y 8, dependiendo de los desinfectantes usados, pues estos dejan de ser efectivos fuera de sus respectivos rangos. Valores menores de estos límites harían que la acidez del agua produjese efectos perjudiciales, como irritación en piel y ojos o corrosión en los elementos de la piscina. Valores mayores también producirían irritación, además de favorecer la aparición de algas y de turbidez [3].

En el caso de este trabajo, el rango en el que se mueve el control es de [7,1-7.8], que es el adecuado para tener una buena desinfección por parte del bromo, el desinfectante presente en la piscina (en general, aunque más caro, el bromo ofrece mejor desinfección que el cloro).

1.1.2 ORP

La oxidación es el proceso por el cual un químico o agente cede electrones a otro. En las piscinas, el proceso de oxidación es el que verdaderamente produce la desinfección. El desinfectante (oxidante) "quita" electrones a los contaminantes (reductores), es decir, se reduce para oxidarlos, "inutilizándolos" y facilitando su eliminación o retirada.

ORP responde a *Oxidation-Reduction Potential* (también es comúnmente denominado *rédox*). Básicamente es un medidor del potencial de oxidación de la sustancia. Se mide con la diferencia de potencial (en mV) entre el oxidante y el reductor. Los valores típicos para un buen nivel de ORP están entre 650 mV y 800 mV. Valores menores al mínimo se traducen en un efecto reducido del desinfectante. Valores superiores al máximo no suponen un detrimento en la efectividad del desinfectante, pero pueden producir derivados químicos del mismo que ocasionen efectos indeseados.

En este trabajo se pretende mantener los valores cercanos a dicho rango, aunque, como se verá más adelante, se permitirán valores de ORP por encima del setpoint, cuando sea necesario para controlar el pH.

1.1.3 Caso particular del trabajo

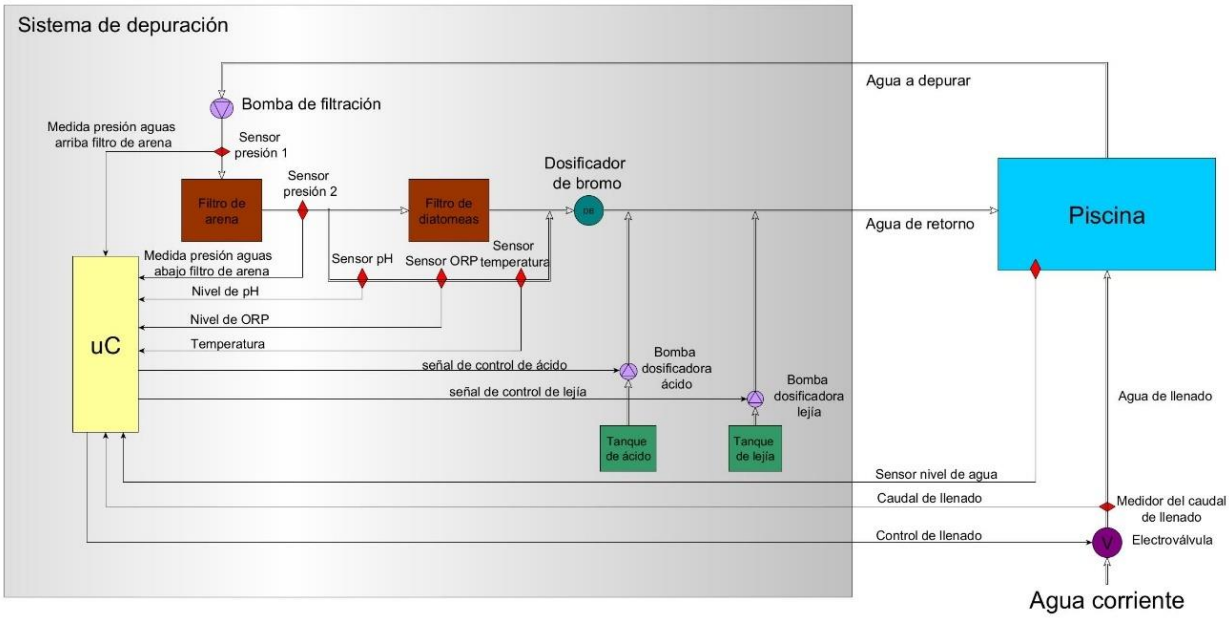

Figura 1-1. Esquema de la piscina.

Figura 1-2. Sistema de depuración y control de la piscina.

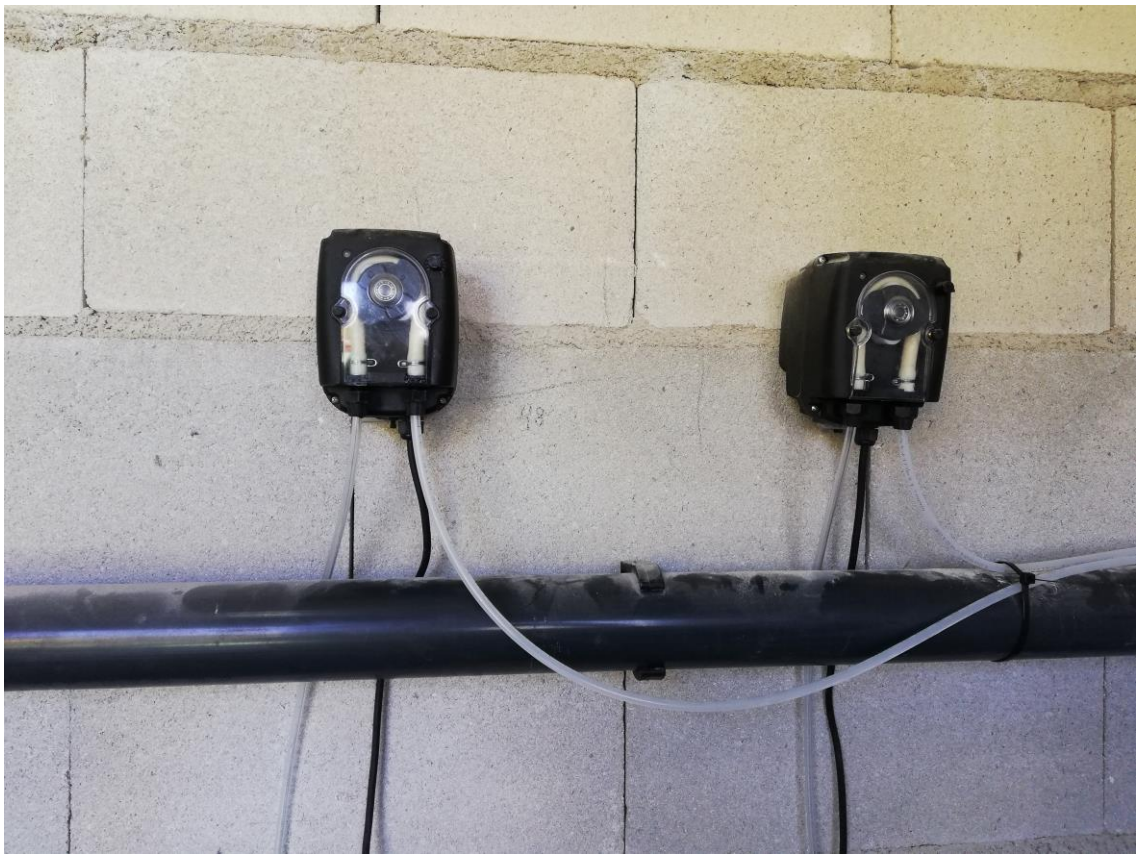

Figura 1-3. Bombas dosificadoras de ácido y lejía.

Figura 1-4. Filtros y depósitos.

Figura 1-5. Flujos de entrada y salida de agua.

Figura 1-6. Raspberry Pi Pico.

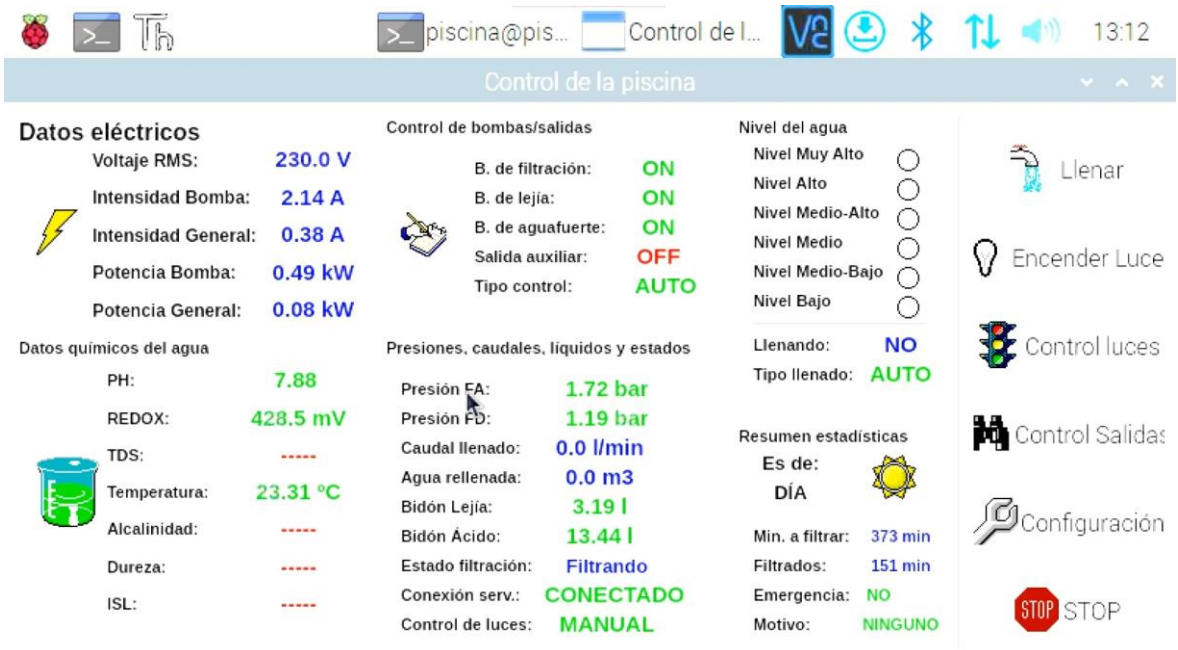

Figura 1-7. Interfaz gráfica de la Raspberry.

La piscina cuenta antes de la implementación del control predictivo con un control proporcional en bucle cerrado de los niveles de pH y ORP, en el que básicamente se mide, con un sensor para cada nivel, el error de estos niveles con respecto a sus setpoints y en función de este se calcula un tiempo de inyección por parte de los actuadores, teniendo solo en cuenta el efecto del ácido frente al pH y el de la lejía frente al ORP (no se considera acoplamiento).

El control predictivo GPC que se desarrolla en este trabajo, por tanto, formará parte de un código ya existente, usándolo como un módulo, quitando el antiguo e implementando en su lugar este nuevo. Se mantienen los mismos niveles a medir, así como los mismos líquidos a verter para modular dichos niveles.

Se tienen dos depósitos, uno para cada solución licuada que se vierte al agua para hacer fluctuar los niveles de pH y ORP, que son el ácido (tanque azul) y la lejía (tanque amarillo). El proceso normal en la piscina es que, debido a la luz solar, el viento, las personas bañándose, etc, el ORP vaya reduciéndose y, para mantenerlo en niveles aceptables, haya que añadir lejía para aumentarlo. Esto hará que el pH aumente ligeramente y para mantenerlo en valores adecuados haya que añadir ácido para disminuirlo. Además, el ácido también tiene efecto sobre el ORP, aumentándolo, aunque en mucha menor medida que la lejía.

El agua a depurar procedente de los fondos y de los skimmers entra al sistema de depuración mediante la bomba de filtración y pasa a través de los filtros de arena (el de la derecha en las figuras) y de diatomeas (izquierda), para después pasar por el dosificador de bromo y retornar a la piscina. Es a ese flujo de agua de retorno al que se le añade la lejía y el ácido procedente de los respectivos tanques, dependiendo de las señales de control que el microcontrolador envíe a las bombas dosificadoras de cada tanque.

Como puede verse en la imagen, también se controla el flujo de entrada del agua para el llenado de la piscina actuando sobre una electroválvula (en función de la medida del sensor del nivel del agua y del medidor del caudal de llenado), al igual que se monitorizan niveles de presión y temperatura. Esto último no es objeto de estudio en el presente trabajo.

1.1.4 Aspectos a tener en cuenta

- La actuación de los líquidos sobre los niveles es lenta. Esto se verá en más detalle más adelante, principalmente en el apartado de identificación del sistema y en los resultados del control. En el caso

particular del ORP, la subida de este requiere de cada vez más lejía conforme el nivel se va acercando al máximo.
- Ambos difusores funcionan como un ON/OFF, siendo el ON del valor de 0.001 L/s. Esto marcará en gran medida el diseño y funcionamiento del control.
- En la realización del trabajo, se ha llamado al ácido y a la lejía como entradas 1 y 2 respectivamente, así como salidas 1 y 2 al nivel de pH y ORP. Se hace así para separar por un lado los pares *i-i* (1-1 y 2-2), que se corresponden con las dinámicas principales, y los pares *i-j* (1-2 y 2-1), que se corresponden con las dinámicas acopladas.
- Los niveles se ven afectados por perturbaciones externas, principalmente la acción de los rayos solares y el baño de las personas (cremas, suciedad, fluidos, etc). Estas perturbaciones no se modelan en el trabajo, lo cual constituye una posible mejora de futuro.

1.2 Estado del arte

En la actualidad, hay un amplio abanico de empresas que ofrecen controladores automáticos para piscinas, tanto para uso doméstico como para uso profesional o de ocio. Prácticamente todos los dispositivos del mercado implementan un control proporcional [4] en el que periódicamente se mide el error y se vierte la cantidad calculada necesaria de líquido para llevar el nivel a la consigna dada (igual que se hacía en la piscina anteriormente) o un control difuso [5] que responda a unas determinadas reglas de control. En su gran mayoría cuentan con dosificación de caudal fijo ON/OFF, aunque también los hay con caudal variable.

Existen artículos científicos acerca del control de piscinas. En general, solo llegan a presentar los posibles esquemas y controles del sistema en cuestión, sin llegar a ahondar en el control propiamente dicho, más allá del tipo propuesto (proporcional o difuso principalmente) o del software utilizado. Existen tanto propuestas de control completamente automático [6] como otras que implementan un control manual [7], donde las medidas se vuelcan en alguna aplicación que permite al usuario utilizar los actuadores a distancia.

También existen proyectos universitarios sobre el tema, algunos de los cuales sí entran en aspectos más técnicos del control [8], aunque tampoco es común que el tipo de control elegido sea el predictivo.

Con todo lo anterior, el presente trabajo resulta novedoso, no por el tipo de control ni por el sistema tratado, sino por la conjunción de ambos.

1.3 Objetivos

Se tienen los siguientes objetivos en el trabajo:

- Diseñar un control predictivo que controle adecuadamente los niveles de pH y ORP presentes en la piscina. Esto se traduce en que el seguimiento de las referencias sea lo más rápido y ajustado posible a la par que no se produzca un gasto excesivo de los líquidos actuadores.
- Profundizar en los aspectos teóricos del control, aprendiendo a ajustar los parámetros de este para lograr los resultados deseados.
- Dejar al propietario un código manejable y que pueda implementar modularmente en el código general de la piscina.

Para lograrlos, los pasos que se han de seguir son los siguientes:

1. Identificar las dinámicas presentes en la piscina con un ajuste alto los cuatro modelos, especialmente las principales, G_{11} y G_{22}.
2. Diseñar el código del control en Matlab, donde se ajustarán los parámetros hasta que el rendimiento sea considerado adecuado.
3. Convertir el código a Python para que sea viable su implementación en la Raspberry.
4. Realizar pruebas en la piscina para comprobar el funcionamiento del control en una aplicación real.

2 FUNDAMENTOS TEÓRICOS DEL CONTROL PREDICTIVO

2.1 Control predictivo basado en modelo

El control predictivo basado en modelo (MPC o MBPC, por sus siglas en inglés de *Model Based Predictive Control*) es un tipo de control automático basado en dos principios básicos, predicción y optimización. Para la predicción se utiliza un modelo matemático del sistema para prever cómo evolucionará dicho sistema en el futuro según las entradas actuales. En función de esta predicción se selecciona una secuencia de entradas de control óptimas que minimizan una función objetivo, la cual se diseña teniendo en cuenta diversos aspectos, como puede ser el ajuste a los setpoints deseados, el consumo, la respuesta transitoria, etc.

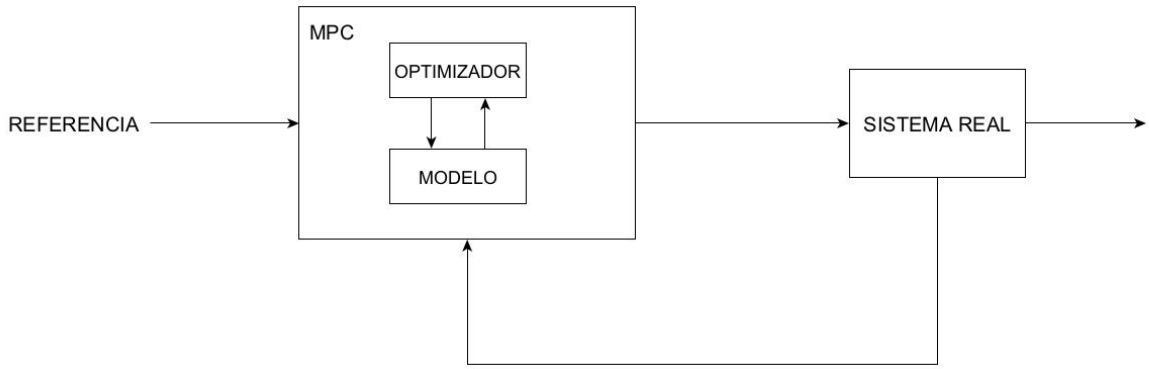

Figura 2-1. Esquema MPC.

El horizonte es un aspecto fundamental en la estrategia de los controles predictivos. Se tienen dos, el de predicción y el de control. El control se hace siempre "mirando" hasta un horizonte de predicción. Dicho horizonte se desplaza hacia adelante a cada instante conforme avanza el control en el tiempo, dando lugar a lo que se conoce como "estrategia de horizonte deslizante". De forma similar, el horizonte de control denota hasta qué punto del futuro se calculan las señales de control óptimas.

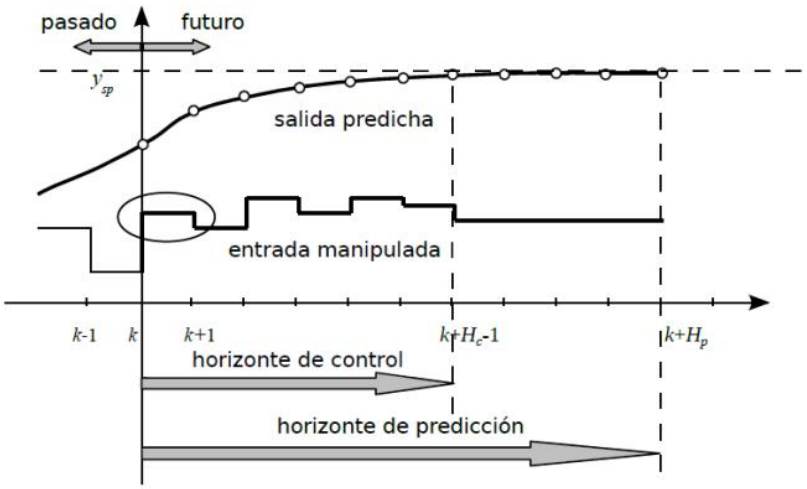

Figura 2-2. Estrategia de control [9].

El MPC cuenta con una serie de ventajas respecto a otros tipos de control [10] [11]:

- El elemento predictivo supone una clara diferencia con otros controles, como el PID o el LQR, en cuanto a que son controles que no "ven" a futuro, lo cual lo hace muy interesante en aplicaciones en las que se necesite que el control minimice o directamente evite ciertos estados. Un ejemplo es la ventaja del uso del control predictivo en la conducción automática de vehículos, en los cuales se necesita tener en cuenta estados futuros (trayectoria de otros coches, obstáculos, etc) para evitar daños o situaciones peligrosas, cosa que no se podría realizar con un control no predictivo, el cual actuaría cuando ya se ha dado dicha situación no deseada.
- El carácter predictivo y anticipativo aporta ventajas a la hora de corregir tiempos muertos y compensar perturbaciones.
- Permite tener en cuenta las limitaciones y restricciones del sistema, como los retrasos o límites físicos, lo cual hace que pueda ser utilizado en sistemas más complejos donde los métodos de control convencionales pueden resultar insuficientes.
- Permite trabajar fácilmente con sistemas multivariables, expandiéndose las ecuaciones del modelo matemático a las necesarias para representar todo el sistema, para lo cual normalmente se recurre a la notación matricial, como se verá en este trabajo.
- También se ha demostrado buen rendimiento (especialmente en algunos tipos dedicados a ello) en su uso en aplicaciones con modelos no lineales y otros procesos con comportamientos poco usuales, como son los procesos de fase no-mínima, oscilatorios o altamente inestables.
- Se puede decir que es un control relativamente moderno en comparación con otros como el PID y aún hoy en día sigue siendo objeto de estudio (implantación en nuevas aplicaciones, mejora de los tipos de control predictivo ya existentes, etc).

A pesar de estos puntos a favor, también cuenta con algunas desventajas:

- Es algo más complicado de implementar que otras contrapartes más antiguas, como el PID.
- Necesita de cierta formación y tiempo de comprensión para instaurarse en el flujo de trabajo de las empresas.
- Para cumplir con todos los aspectos que promete el control, en casos con modelos altamente no lineales, con un gran número de variables y/o con un tiempo de muestreo muy bajo, el coste computacional puede ser elevado (aunque conforme avanza la tecnología esto cada vez supone un menor problema). En el caso particular de este trabajo, donde se tiene un sistema lineal con solo dos entradas y dos salidas, el coste computacional no supone un aspecto a tener en cuenta.

En la actualidad, los sistemas de control industriales deben cumplir simultáneamente con diversos criterios económicos, de seguridad, calidad, medioambientales, etc; que en muchas ocasiones son cambiantes, al igual que los propios sistemas y plantas de la industria. Debido a esto, el MPC se postula como el control de referencia en este ambiente de restricciones y objetivos cambiantes, gracias a su naturaleza adaptativa [11]. Esto ha hecho que el MPC sea utilizado en una amplia gama de aplicaciones en la industria, como la automatización de procesos, la gestión de energía, la robótica o el control de sistemas de transporte (vehículos autónomos). La mayoría de las empresas simplemente implementan los MPC dentro de sus procesos productivos y no los comercializan. Sin embargo, existen algunos casos de empresas que sí los comercializan, como por ejemplo, el *3dMPC* de ABB [12].

2.2 Fundamentos matemáticos del MPC

En el MPC, las acciones de control son tales que se optimice una función objetivo. Esta función puede buscar optimizar tantos aspectos del sistema como se requiera, pero principalmente se busca acercarse a una referencia o *setpoint* a la par que se produce el mínimo gasto en las entradas, quedando la ecuación de la siguiente forma [13]:

$$J(u) = \sum_{k=1}^{N} \delta_k \cdot (y_k - r_k)^2 + \sum_{k=1}^{M} \beta_k \cdot {u_k}^2 + \sum_{k=1}^{M} \lambda_k \cdot \Delta {u_k}^2 \quad (2\text{–}1)$$

Donde:

- k es el instante discreto.
- y_k es la salida en cada instante k.
- r_k es la referencia en cada instante k.
- u_k es la acción de control (entrada) en cada instante k.
- λ, β y δ son los coeficientes de ponderación de cada término de la función.
- N y M son los horizontes de predicción y control, respectivamente.

El primer sumando de la función se corresponde con el seguimiento de la referencia. Minimizarlo supone conseguir que la diferencia entre la salida medida y la referencia sea lo menor posible. El segundo sumando se corresponde con el gasto, entendiendo que a mayor valor se de en la acción de control, mayor es dicho gasto. El último sumando se corresponde con la variación en las acciones de control, lo que se conoce como el esfuerzo de control.

Existen principalmente dos métodos de resolución del problema de optimización:

1. Resolución analítica: las ecuaciones se resuelven previamente y solo cambian los parámetros en cada iteración. Este método es solo posible en problemas lineales sin restricciones.
2. Resolución numérica o computacional: la solución no se obtiene de manera exacta a través de una ecuación matemática, si no que, a través de algoritmos computacionales, se realizan cálculos iterativos para dar con la solución óptima. Estos métodos son utilizados en problemas con restricciones. En sistemas lineales, se suele resolver por programación cuadrática. Para problemas no lineales, otras técnicas son más convenientes (optimización global, descomposición, etc).

La ley de control son las ecuaciones que permiten obtener las acciones de control óptimas a cada instante. En el caso de la resolución analítica, se calcula simplemente calculando la acción de control mínima a partir del gradiente de la función objetivo igualado a cero:

$$\frac{\partial J(u)}{\partial u} = 0 \rightarrow u_{min} = u(N, M, \lambda, \delta, \beta, r, f) \quad (2\text{–}2)$$

En el caso de la resolución mediante el problema cuadrático, la función objetivo se reorganiza para convertirla en una función cuadrática. En forma matricial:

$$J = \frac{1}{2} x^T \cdot H \cdot x + {f_0}^T \cdot x + l \quad (2\text{–}3)$$

Donde H, f y l solo dependen de parámetros del sistema y valores de salidas anteriores (la respuesta libre) y x depende de u.

Así, se resuelve el siguiente problema de optimización:

$$\min_x \frac{1}{2} x^T \cdot H \cdot x + {f_0}^T \cdot x$$

$$s.a. \begin{cases} A \cdot x \leq \mathrm{b} \\ A_{eq} \cdot x = b_{eq} \\ l_b \leq \mathrm{x} \leq u_b \end{cases} \quad (2\text{–}4)$$

Se minimiza la función objetivo previamente formulada sujeta a restricciones de desigualdad, igualdad y/o de límites. El algoritmo de optimización utilizado depende de cada caso, siendo algunos de ellos el *active-set*, el *interior-point-convex* o el *gradient-descent*.

Es importante destacar que de la secuencia de actuaciones de control calculada, solo se acaba usando la primera, puesto que en siguiente periodo de muestreo se calculará una nueva secuencia en función de la respuesta libre actualizada.

2.3 Tipos de MPC

En la actualidad existe gran variedad de controles predictivos, tantos como estrategias hay en cuanto a la optimización de la función objetivo y en cuanto a la forma de obtener y usar el modelo interno. A grandes rasgos, los principales tipos de MPC son los siguientes:

2.3.1 DMC

El control predictivo DMC (por sus siglas en inglés de *Dynamic Matrix Control*) se basa en un modelo compuesto por coeficientes de la respuesta ante escalón unitario de un sistema lineal.

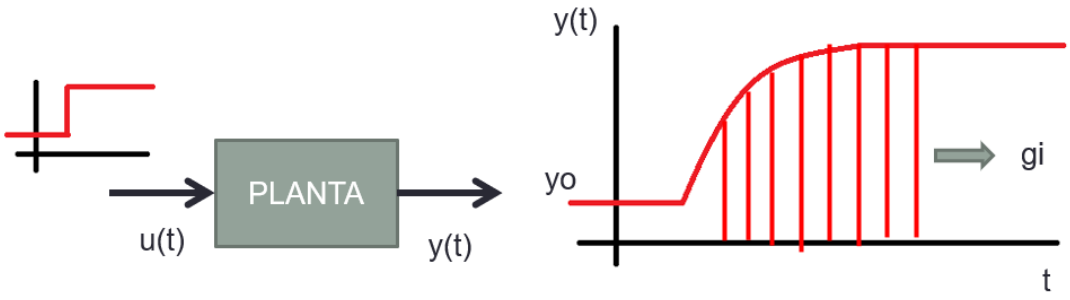

Figura 2-3. Entradas-salidas DMC [14].

El valor de la entrada se multiplica por el coeficiente que le corresponde en cada instante, siguiendo la ecuación:

$$y(t) = y_0 + \sum_{i=1}^{\infty} g_i * \Delta u(t-i) \tag{2–5}$$

Desarrollando estos cálculos en el horizonte de predicción y pasando a notación matricial, los coeficientes quedan almacenados en una "matriz dinámica" (de ahí su nombre), cuyas columnas se componen de los coeficientes en cada instante después del escalón, colocados desplazados hacia abajo una fila en cada columna de izquierda a derecha:

$$G = \begin{bmatrix} g_1 & 0 & \dots & 0 \\ g_2 & g_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_N & g_{N-1} & \dots & g_1 \end{bmatrix} \tag{2–6}$$

Con esa matriz se conforma la ecuación de la estimación a la salida, con la cual se resolverá la función objetivo.

2.3.2 GPC

Las siglas GPC se corresponden con *Generalized Predictive Control*. Se le llama generalizado porque utiliza como modelo directamente la función de transferencia, al contrario de otros tipos como el ya mencionado DMC. Este es el tipo de MPC que se ha utilizado en el presente trabajo.

El modelo parte de la ecuación CARIMA (del inglés *Controlled Auto Regresive Integrated Moving Average*), basada en la función de transferencia, tal que [15]:

$$y(k) = G(z^{-1})u(k) \tag{2–7}$$

$$G(z^{-1}) = \frac{B(z^{-1})z^{-1}}{A(z^{-1})} \tag{2–8}$$

$$yA(z^{-1})y(k) = B(z^{-1})z^{-1}u(k) = B(z^{-1})u(k-1) \tag{2–9}$$

Añadiendo el término del modelo de perturbaciones, que es:

$$n(k) = \frac{C(z^{-1})\varepsilon(k)}{1 - z^{-1}} \tag{2–10}$$

$$1 - z^{-1} = \Delta \tag{2–11}$$

Se tiene finalmente la CARIMA:

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + \frac{C(z^{-1})\varepsilon(k)}{\Delta} \tag{2–12}$$

Donde:

- $A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{na} z^{-na}$ (2–13)
- $B(z^{-1}) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{nb} z^{-nb}$ (2–14)
- $C(z^{-1}) = c_0 + c_1 z^{-1} + c_2 z^{-2} + \cdots + c_{nc} z^{-nc}$ (2–15)
- $y(k)$ son las salidas de la planta
- $u(k)$ son las entradas de la planta
- $\varepsilon(k)$ es el ruido blanco de media nula

2.3.3 NMPC

La N es de *Non-linear*, puesto que utiliza un modelo no lineal para describir la dinámica del proceso. Este tipo requiere de una resolución iterativa del problema basada en métodos de control óptimos de tipo Newton, normalmente. Se trata de un tipo de control más complejo, que está cobrando cada vez más importancia conforme el hardware y los algoritmos computacionales van mejorando [10].

2.4 Ecuaciones del GPC

A continuación se expone el proceso mediante el cual, partiendo de las premisas establecidas anteriormente para el control GPC, se llega a las ecuaciones de las salidas estimadas con el modelo interno del control y a la optimización de la función objetivo mediante la resolución del problema cuadrático.

Volviendo a la ecuación [2-12], donde quedó expresada la ecuación CARIMA, por simplicidad, se toma el polinomio C igual a 1, quedando lo que se conoce como la ecuación ARIMA:

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + \frac{\varepsilon(k)}{\Delta} \tag{2–16}$$

Multiplicando por Δ y tomando $A(z^{-1})\Delta = \tilde{A}(z^{-1})$:

$$\tilde{A}(z^{-1})y(k) = B(z^{-1})\Delta u(k-1) + \varepsilon(k) \tag{2–17}$$

Siendo:

$$\tilde{A}(z^{-1}) = (1-z^{-1}) + (1-z^{-1})a_1 z^{-1} + (1-z^{-1})a_2 z^{-2} + \cdots + (1-z^{-1})a_{na} z^{-na} \tag{2–18}$$

$$\tilde{A}(z^{-1}) = 1 + \tilde{a}_1 z^{-1} + \tilde{a}_2 z^{-2} + \cdots + \tilde{a}_{na} z^{-na} \tag{2–19}$$

Pasando el 1 a la izquierda:

$$\tilde{A}(z^{-1}) - 1 = \tilde{a}_1 z^{-1} + \tilde{a}_2 z^{-2} + \cdots + \tilde{a}_{na} z^{-na} \tag{2–20}$$

Teniendo en cuenta que $\tilde{A}y(k) = \left(1 + \left(\tilde{A}(z^{-1}) - 1\right)\right)y(k)$, se tiene que:

$$\left(1 + \left(\tilde{A}(z^{-1}) - 1\right)\right)y(k) = B(z^{-1})\Delta u(k-1) + \varepsilon(k) \tag{2–21}$$

$$y(k) = \left(1 - \tilde{A}(z^{-1})\right)y(k) + B(z^{-1})\Delta u(k-1) + \varepsilon(k) \tag{2–22}$$

Y, desglosando los polinomios:

$$\begin{aligned} y(k) = {} & (-\tilde{a}_1 z^{-1} - \tilde{a}_2 z^{-2} - \cdots - \tilde{a}_{na} z^{-na})y(k) \\ & + \left(b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{nb} z^{-nb}\right)\Delta u(k-1) + \varepsilon(k) \end{aligned} \tag{2–23}$$

El término z^{-n} actúa como un retraso de n instantes, tal que $x(k)\cdot z^{-n} = x(k-n)$. Por tanto:

$$\begin{aligned} y(k) = {} & -\tilde{a}_1 y(k-1) - \tilde{a}_2 y(k-2) - \cdots - \tilde{a}_{na} y(k-na) + b_0 \Delta u(k-1) + b_1 \Delta u(k-2) \\ & + b_2 \Delta u(k-3) + \cdots + b_{nb} \Delta u(k-nb-1) + \varepsilon(k) \end{aligned} \tag{2–24}$$

Finalmente, la estimación $\tilde{y}(k)$ sería igual que la ecuación de arriba sin el término de ruido blanco, ya que al tener media nula, su esperanza matemática es cero.

Queda así una expresión que denota que en cada instante *k* se tiene una salida definida por entradas y salidas pasadas y, por tanto, conocidas.

Es posible formular todas las ecuaciones de las salidas futuras desde *k+1* hasta *k+N* de esta forma. El problema es que en cada estimación *y(k+i)* se tiene el término *y(k+i-1)*, de modo que para horizontes de predicción

relativamente grandes los cálculos de las estimaciones futuras se hacen exponencialmente más complejos.

Es por esto que se usa la ecuación diofántica de la división [15]:

$$\begin{array}{r|l} 1 & \tilde{A} \\ \hline F_j z^{-j} & E_j \end{array}$$

$$1 = E_j(z^{-1})\tilde{A} + F_j(z^{-1})z^{-j} \tag{2–25}$$

Multiplicando por E_j a ambos lados de la ecuación [2-17] se tiene:

$$E_j(z^{-1})\tilde{A}(z^{-1})y(k) = E_j(z^{-1})B(z^{-1})\Delta u(k-1) + E_j(z^{-1})\varepsilon(k) \tag{2–26}$$

Sustituyendo la ecuación diofántica en la anterior y expresando la estimación para un instante futuro j:

$$\left(1 - F_j(z^{-1})z^{-j}\right)y(k+j) = E_j(z^{-1})B(z^{-1})\Delta u(k+j-1) + E_j(z^{-1})\varepsilon(k) \tag{2–27}$$

Llamando al término E_jB como G_j (E_jB da lugar a los polinomios de la matriz de coeficientes de respuesta frente a escalón) y aplicando el retardo z^{-j}, se tiene finalmente:

$$y(k+j) = G_j(z^{-1})\Delta u(k+j-1) + F_j(z^{-1})y(k) + E_j(z^{-1})\varepsilon(k) \tag{2–28}$$

Eliminando el término del ruido, se tiene la estimación:

$$\hat{y}(k+j|k) = G_j(z^{-1})\Delta u(k+j-1|k) + F_j(z^{-1})y(k|k) \tag{2–29}$$

Se llega así a una ecuación según la cual las estimaciones futuras dependen de las acciones de control y de la salida actual.

Introduciendo el término d para tener en cuenta el retraso de la planta, se tiene el siguiente conjunto de las N predicciones óptimas en un instante de tiempo k:

$$\begin{gathered} \hat{y}(k+d+1|k) = G_{d+1}(z^{-1})\Delta u(k|k) + F_{d+1}(z^{-1})y(k|k) \\ \hat{y}(k+d+2|k) = G_{d+2}(z^{-1})\Delta u(k+1|k) + F_{d+2}(z^{-1})y(k|k) \\ \vdots \\ \hat{y}(k+d+N|k) = G_{d+N}(z^{-1})\Delta u(k+N-1|k) + F_{d+N}(z^{-1})y(k|k) \end{gathered} \tag{2–30}$$

Separando las acciones de control posteriores al instante actual k (respuesta forzada, estimada) de las anteriores (respuesta libre, conocida) y agrupando las ecuaciones en matrices, se tiene:

$$\boldsymbol{y} = \boldsymbol{G}\boldsymbol{u} + \boldsymbol{F}(z^{-1})y(k|k) + \boldsymbol{P}(z^{-1})\Delta u(k-1|k) \tag{2–31}$$

Donde:

$$\boldsymbol{y} = \begin{bmatrix} \hat{y}(k+d+1|k) \\ \hat{y}(k+d+2|k) \\ \vdots \\ \hat{y}(k+d+N|k) \end{bmatrix} \tag{2–32}$$

$$\boldsymbol{u} = \begin{bmatrix} \Delta u(k|k) \\ \Delta u(k+1|k) \\ \vdots \\ \Delta u(k+N-1|k) \end{bmatrix} \tag{2–33}$$

$$\boldsymbol{G} = \begin{bmatrix} g_0 & 0 & \dots & 0 \\ g_1 & g_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_{N-1} & g_{N-2} & \dots & g_0 \end{bmatrix} \tag{2–34}$$

$$\boldsymbol{P} = \begin{bmatrix} z(G_{d+1}(z^{-1}) - g_0) \\ z^2(G_{d+2}(z^{-1}) - g_0 - g_1 z^{-1}) \\ \vdots \\ z^N\left(G_{d+N}(z^{-1}) - g_0 - g_1 z^{-1} - \dots - g_{N-1} z^{-(N-1)}\right) \end{bmatrix} \tag{2–35}$$

$$\boldsymbol{F} = \begin{bmatrix} F_{d+1}(z^{-1}) \\ F_{d+2}(z^{-1}) \\ \vdots \\ F_{d+N}(z^{-1}) \end{bmatrix} \tag{2–36}$$

Por último, agrupando todos los términos pasados en un vector $\boldsymbol{f}$, se tiene la ecuación:

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\boldsymbol{u} + \boldsymbol{f} \tag{2–37}$$

Volviendo a la función de coste expresada en [2-1], tomando solo el primer y tercer sumando (seguimiento de referencia y esfuerzo de control, respectivamente):

$$J(u) = \sum_{k=N_1}^{N_2} \delta(k) \cdot \left(\hat{y}(k+j) - r(k+j)\right)^2 + \sum_{k=1}^{Nu} \lambda(k) \cdot \Delta u(k+j-1)^2 \tag{2–38}$$

Donde ahora se tiene:

- $N_1 = 1 + d$
- $N_2 = d + N$
- Nu: horizonte de control, antes llamado M.

Pasando a notación matricial, se tiene:

$$\boldsymbol{J} = (\hat{\boldsymbol{y}} - \boldsymbol{r})^T \boldsymbol{\delta} (\hat{\boldsymbol{y}} - \boldsymbol{r}) + \boldsymbol{\Delta u}^T \boldsymbol{\lambda} \boldsymbol{\Delta u} \tag{2–39}$$

Nótese que ahora todos los elementos constituyen vectores o matrices, incluidos $\boldsymbol{\delta}$ y $\boldsymbol{\lambda}$, que son $\boldsymbol{\delta} = \delta \boldsymbol{I}$ y $\boldsymbol{\lambda} = \lambda \boldsymbol{I}$.

Sustituyendo la ecuación [2-37] en la [2-39]:

$$\boldsymbol{J = (Gu + f - r)^T \delta (Gu + f - r) + \Delta u^T \lambda \Delta u} \quad (2\text{–}40)$$

Separando los términos de las acciones de control:

$$\boldsymbol{J = u^T \left(G^T \delta G + \lambda\right) u + 2(f - r)^T \delta G u + (f - r)^T \delta (f - r)} \quad (2\text{–}41)$$

Como ya se expuso anteriormente, en caso de no trabajar con restricciones, el siguiente paso sería igualar el gradiente a cero y despejar la serie de acciones de control óptimas.

$$\boldsymbol{\frac{\partial J(u)}{\partial u} = 2\left(G^T \delta G + \lambda\right) u + 2(f - r)^T \delta G = 0} \quad (2\text{–}42)$$

Despejando la acción de control:

$$\boldsymbol{u_{min} = -\frac{(f - r)^T \delta G}{G^T \delta G + \lambda}} \quad (2\text{–}43)$$

Finalmente:

$$\boldsymbol{u_{min} = -H^{-1} {f_0}^T} \quad (2\text{–}44)$$

Siendo:

- $\boldsymbol{H = G^T \delta G + \lambda}$
- $\boldsymbol{{f_0}^T = (f - r)^T \, \delta G}$

La acción de control que se aplica en el instante *k* es el primer elemento del vector calculado en la ecuación [2-44].

En el caso de trabajar con restricciones, como ya se explicó, hay que resolver el problema cuadrático expresado en [2-4]. Comparando la ecuación [2-3] con la ecuación [2-41] se ven fácilmente las correspondencias entre los términos:

- $\boldsymbol{x = \Delta u}$
- $\boldsymbol{H = G^T \delta G + \lambda}$
- $\boldsymbol{{f_0}^T = (f - r)^T \, \delta G}$
- $\boldsymbol{l = (f - r)^T \delta (f - r)}$

Una vez identificados los términos, se aplica el algoritmo de resolución del problema cuadrático para obtener la secuencia de control óptima, de la cual, una vez más, solo se usa el primer elemento, puesto que la siguiente iteración se vuelve a resolver el problema de optimización.

3 IDENTIFICACIÓN DEL SISTEMA

El control predictivo es un control basado en un modelo del sistema real que se quiere controlar. Este modelo se puede representar de varias formas, siendo la más común la función de transferencia, que es la que se ha utilizado en este trabajo.

Hay dos formas de obtener el modelo:

1. Utilizando las ecuaciones de la dinámica del sistema: por ejemplo, la ecuación diferencial de segundo grado de la dinámica en un sistema amortiguado podría constituir el modelo para el control de la suspensión en un coche.
2. Si no se conoce o no es preferible utilizar algún modelo dinámico, se puede definir en función de unas entradas y salidas conocidas. Esta ha sido la vía utilizada para obtener las funciones de transferencia, aprovechando que se tenía un registro tanto de las inyecciones de los líquidos (entradas del sistema) como de la evolución en los niveles de pH y ORP (salidas del sistema).

Matlab cuenta con una toolbox llamada *System Identification* para esta tarea.

3.1 Obtención de entradas y salidas y preprocesado

Se hicieron dos pruebas, en dos días distintos, para obtener datos de cada inyector:

- El primer día se inyectó ácido durante un escalón de 45 minutos aproximadamente, para obtener la evolución de los niveles respecto a dicho líquido.
- El segundo día, se hizo lo mismo para la lejía y se dieron dos escalones, uno de 15 y otro de 10 minutos.
- En ambos casos se dejó bastante tiempo antes y después de los escalones, para ver el comportamiento en régimen permanente.
- Los niveles se registraron en tablas de *Excel*, de manera que en cada celda se representaba, separados por comas, el *datetime*, nivel del ORP y nivel del pH. Utilizando la función *readtable()* se pasan los datos a formato tabla en Matlab:

	A	B
1	datetime,id_value.orp sensor,id_value.ph sensor	
2	2022-10-08T08:31:55.059Z,,	
3	2022-10-08T08:31:55.687Z,,	
4	2022-10-08T08:31:55.967Z,,	
5	2022-10-08T08:31:56.290Z,,	
6	2022-10-08T08:31:56.398Z,,8.037535714565006	
7	2022-10-08T08:31:56.591Z,416.3929618768325,	
8	2022-10-08T08:31:56.905Z,,	
9	2022-10-08T08:31:56.988Z,,	
10	2022-10-08T08:31:56.997Z,,	
11	2022-10-08T08:31:57.812Z,,	
12	2022-10-08T08:31:57.820Z,,	
13	2022-10-08T08:31:57.833Z,,	
14	2022-10-08T08:31:57.838Z,,	
15	2022-10-08T08:31:57.843Z,,7.841560911045944	
16	2022-10-08T08:31:57.848Z,563.900293255132,	

Figura 3-1. Extracto de los datos en la tabla Excel.

data

308721x3 table

	1 datetime	2 id_value_orpSensor	3 id_value_phSensor
1	'2022-10-08T08:31:56.398Z'	NaN	8.0375
2	'2022-10-08T08:31:56.591Z'	416.3930	NaN
3	'2022-10-08T08:31:56.905Z'	NaN	NaN
4	'2022-10-08T08:31:56.988Z'	NaN	NaN
5	'2022-10-08T08:31:56.997Z'	NaN	NaN
6	'2022-10-08T08:31:57.812Z'	NaN	NaN
7	'2022-10-08T08:31:57.820Z'	NaN	NaN
8	'2022-10-08T08:31:57.833Z'	NaN	NaN
9	'2022-10-08T08:31:57.838Z'	NaN	NaN
10	'2022-10-08T08:31:57.843Z'	NaN	7.8416
11	'2022-10-08T08:31:57.848Z'	563.9003	NaN
12	'2022-10-08T08:31:57.853Z'	NaN	NaN
13	'2022-10-08T08:31:57.858Z'	NaN	NaN

Figura 3-2. Conversión de los datos de Excel a Matlab.

- En cuanto a la inyección de los líquidos, no se disponía de más datos más allá de un registro que indicaba cuándo se empezaba y terminaba de verter cada líquido, de modo que se formó manualmente una tabla en Excel con el estado de los difusores (0.001 L/s en caso de ON y 0 L/s en caso de OFF) en cada instante de tiempo.

	A	B	C
1	Time	bleach	acid
2758	9:17:56	0	0,001
2759	9:17:57	0	0,001
2760	9:17:58	0	0,001
2761	9:17:59	0	0,001
2762	9:18:00	0	0,001
2763	9:18:01	0	0,001
2764	9:18:02	0	0,001
2765	9:18:03	0	0,001
2766	9:18:04	0	0,001
2767	9:18:05	0	0,001
2768	9:18:06	0	0,001
2769	9:18:07	0	0,001
2770	9:18:08	0	0,001

Figura 3-3. Creación de las entradas (ON/OFF de ácido y lejía).

- A partir de estas dos tablas y los scripts creados para procesar los datos, se obtienen, para cada día, un vector de entrada *u1* y una matriz conformada por los dos vectores de salida *y1* e *y2*. Dentro del preprocesado para obtener estos datos, cabe destacar que los sensores con los que se contaba al principio aportaban medidas con mucho ruido, aunque siempre en torno al valor real. Esto se solventó utilizando la función *smoothdata()*, que suaviza los datos según el algoritmo y factor escogido:

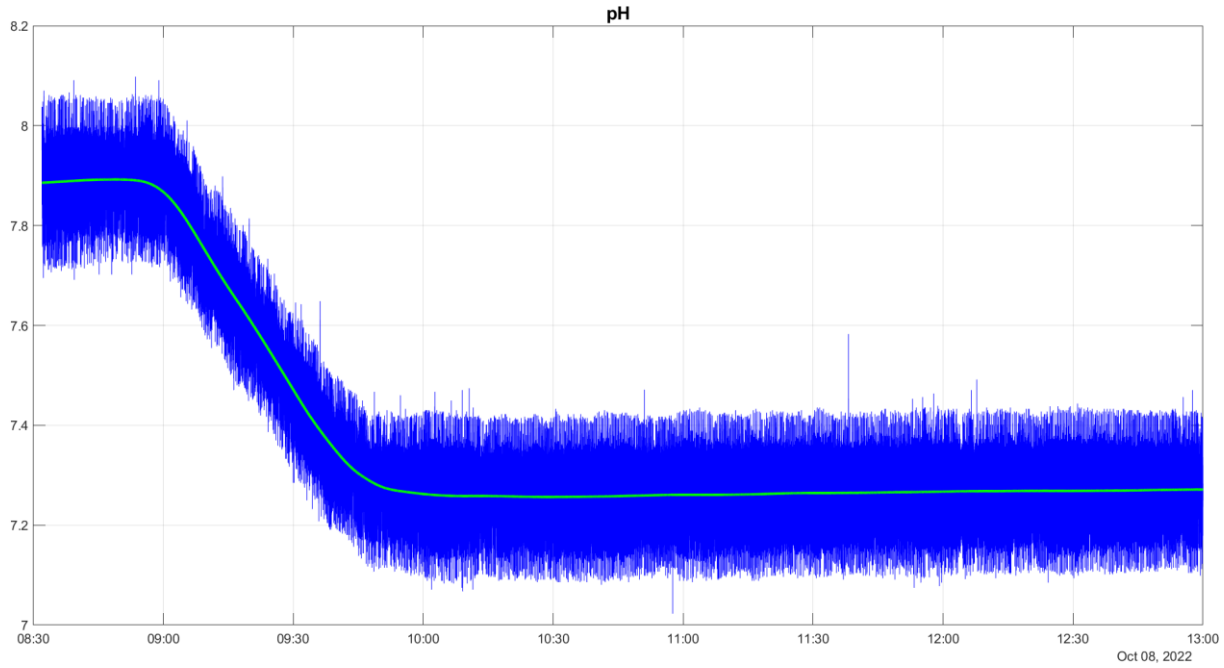

Figura 3-4. Ejemplo de suavización: nivel de pH.

Una vez suavizadas las medidas, tenemos las entradas y salidas necesarias para identificar los modelos:

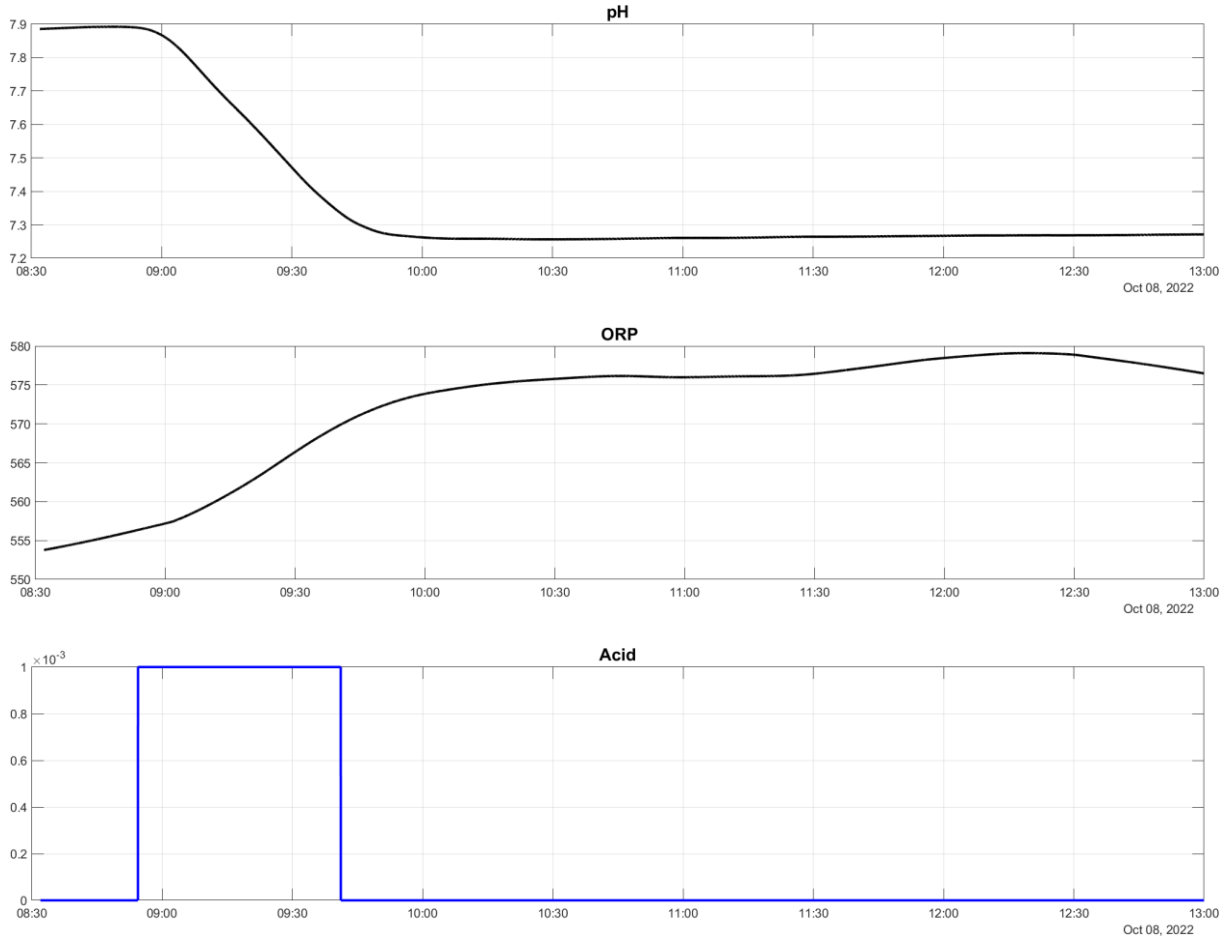

Figura 3-5. Respuestas frente a ácido.

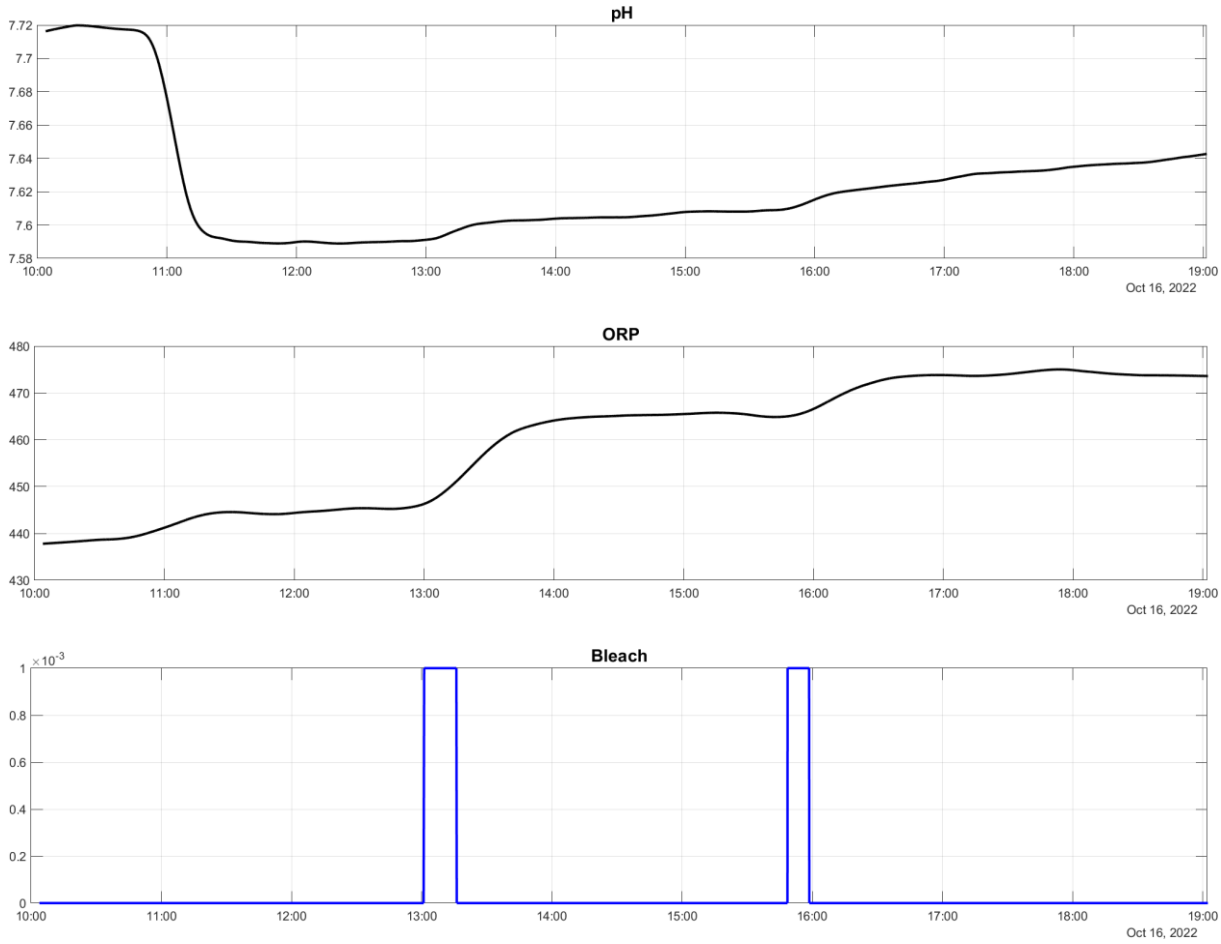

Figura 3-6. Respuestas frente a lejía.

- Las entradas y salidas se importan en la toolbox para conseguir el *working data* con el que posteriormente generar las funciones de transferencia. El último preprocesado se realiza ya en la toolbox, reduciéndose el rango temporal para un ajuste mayor en la posterior generación de los modelos.

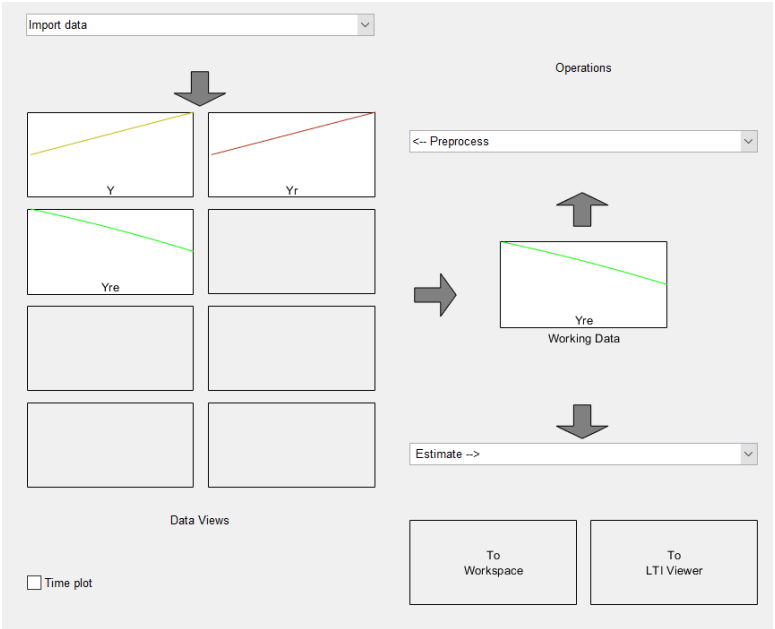

Figura 3-7. Importación de los datos a la *System Identifiation* toolbox.

3.2 Generación de los modelos

Una vez se tiene el working data, el siguiente paso es estimar los modelos. La estimación escogida es la de un modelo de proceso (más conocido por su nombre en inglés *process model*), a pesar de existir la opción de estimar una *transfer function*. Esto se debe a que en la estimación de funciones de transferencia solo se permite seleccionar el número de ceros y polos, mientras que en la de modelos de procesos se permite seleccionar más parámetros. De igual forma, el resultado es una función de transferencia con un número de ceros y polos determinado en función de los parámetros escogidos.

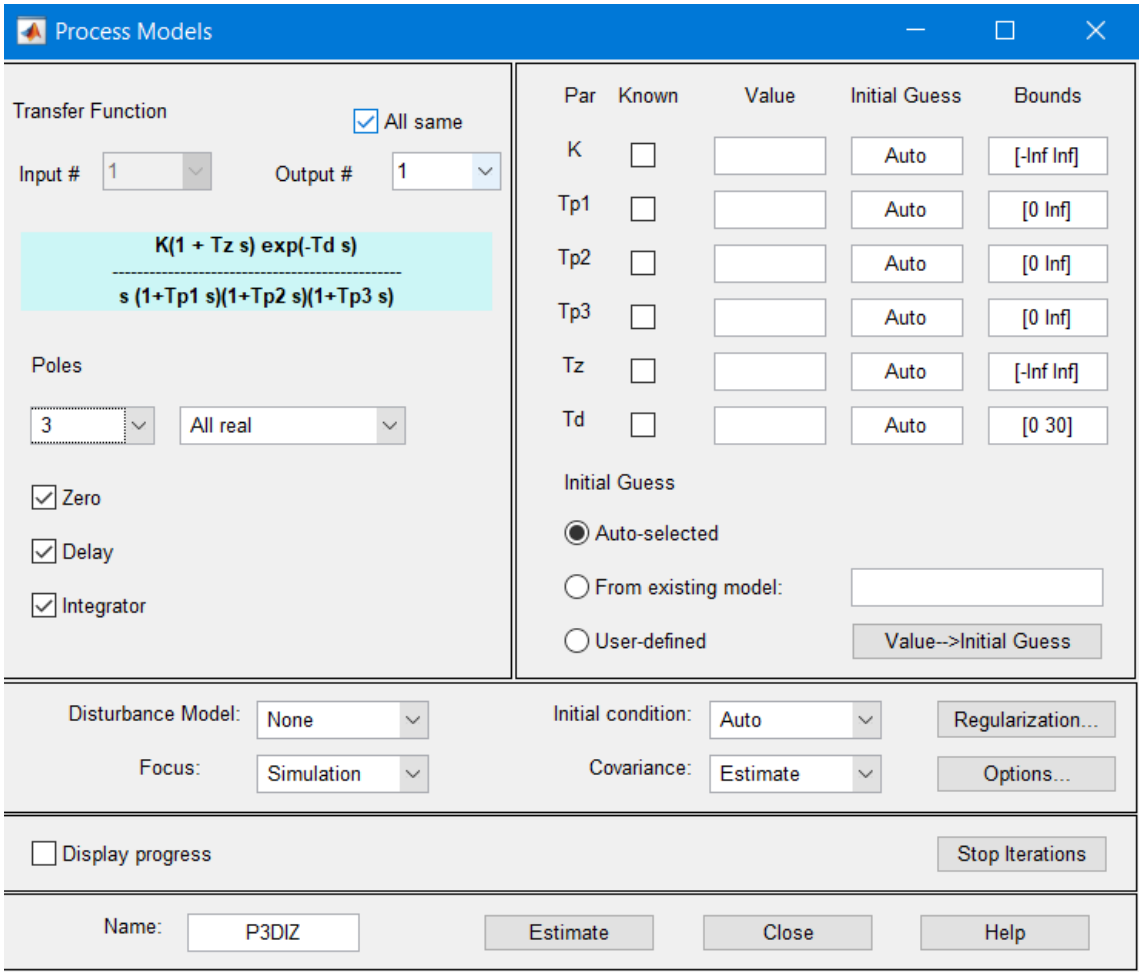

Figura 3-8. Ejemplo de estimación de modelo con la *System Identification* toolbox.

Las estimaciones quedaron de la siguiente forma:

- Acción del ácido en pH y ORP:

```
proc_acid =
Process model with 2 outputs: y_k = Gk(s)u
  From input "u1" to output "y1":
                  1+Tz*s
  G1(s) = Kp * ---------
               s(1+Tp1*s)

        Kp = -0.22338
       Tp1 = 264.63
        Tz = 27.863

  From input "u1" to output "y2":
                  1+Tz*s
  G1(s) = Kp * ----------
               s(1+Tp1*s)

        Kp = 6.6656
       Tp1 = 1318.2
        Tz = 243.44
```

Figura 3-9. Modelos matemáticos de respuesta del pH frente a ácido y lejía.

- Acción de la lejía en pH y ORP:

```
proc_bleach =
Process model with 2 outputs: y_k = Gk(s)u
  From input "u1" to output "y1":
                Kp
  G1(s) = ------------
           s(1+Tp1*s)

        Kp = 0.03
       Tp1 = 1400

  From input "u1" to output "y2":
                Kp
  G1(s) = ------------
           s(1+Tp1*s)

        Kp = 15
       Tp1 = 1300
```

Figura 3-10. Modelos matemáticos de respuesta del ORP frente a ácido y lejía.

En ninguna de las cuatro dinámicas hay punto de funcionamiento, pues tras aplicar un escalón, la salida se queda en el valor final obtenido antes de que la entrada vuelva a valer cero. Esto, sumado a que las dinámicas presentan cierta suavidad al comienzo y al final de la respuesta (véase figura 3-5 cerca de las 09.00 y las 10:00), lleva a que la identificación de los modelos de lugar a funciones de transferencia de segundo orden con un integrador.

La estimación final se logró después de algunas pruebas con distintos parámetros, comparando los modelos obtenidos entre sí, atendiendo principalmente a dos factores: el ajuste (tomando de referencia para validar el *working data*) y la respuesta transitoria. Es importante atender a los dos factores, ya que para unos datos y un rango de tiempo determinados se puede obtener un modelo con mejor ajuste pero que carece de sentido una vez que se observa la respuesta frente a escalón o impulso (especialmente en el caso de este trabajo, donde los actuadores actúan como un ON/OFF, de modo que las inyecciones siempre son escalones).

A continuación se expone, para cada par de entrada-salida, el ajuste del modelo obtenido y la respuesta transitoria (frente a escalón de 0.001) de este:

- Par ácido-pH (G_{11}): se obtiene un ajuste casi perfecto, de un 96% aproximadamente, además de que la respuesta frente a escalón tiene sentido, pues manteniendo la inyección, se baja el nivel.

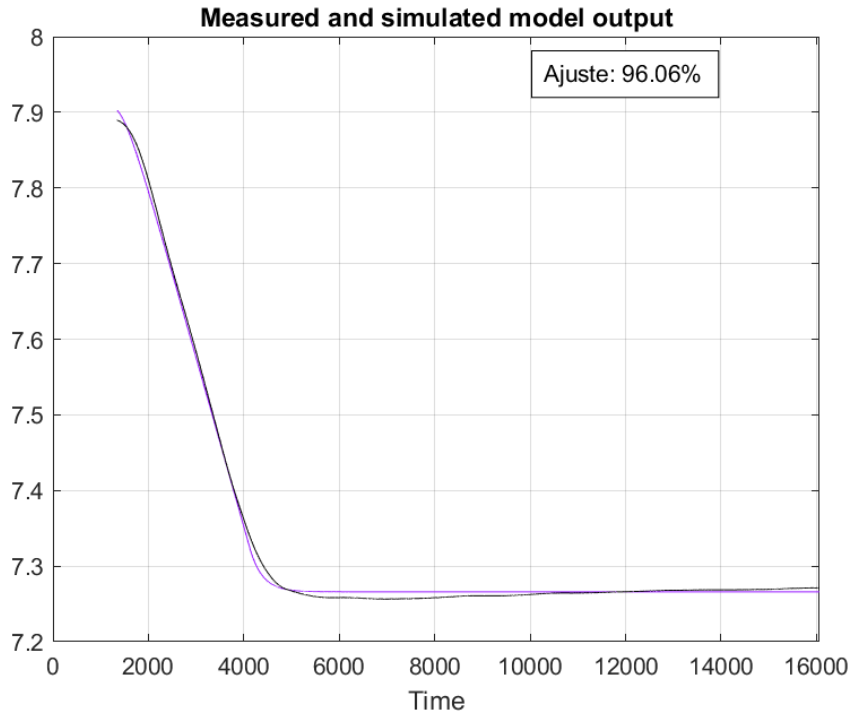

Figura 3-11. Ajuste del modelo para la respuesta del pH frente al ácido.

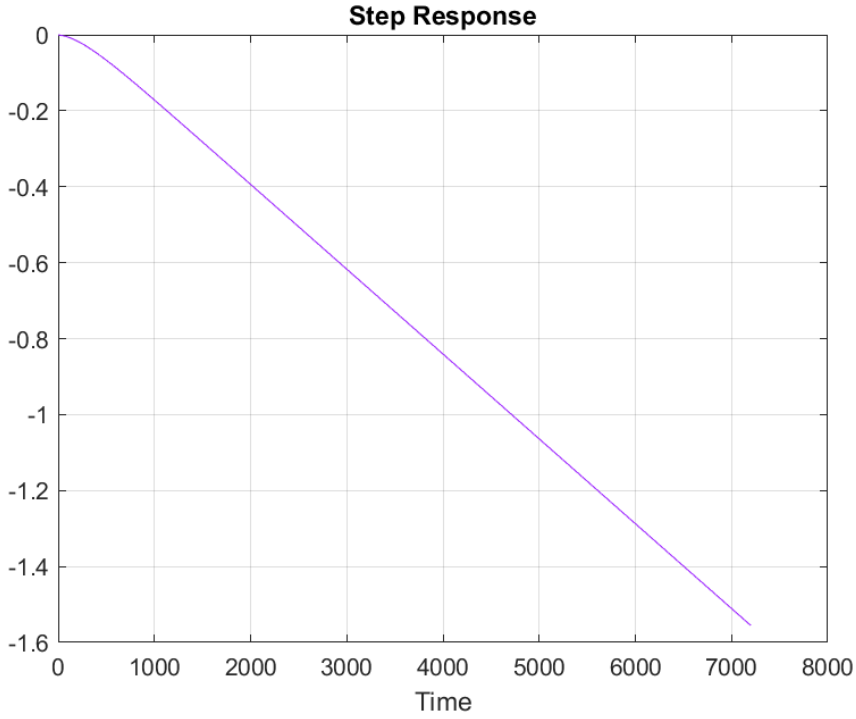

Figura 3-12. Respuesta ante escalón del modelo para la respuesta del pH frente al ácido.

- Par ácido-ORP (G_{21}): se consigue también un ajuste alto, del 85.36% y el transitorio va en consonancia con la subida del nivel de pH a consecuencia de la lejía.

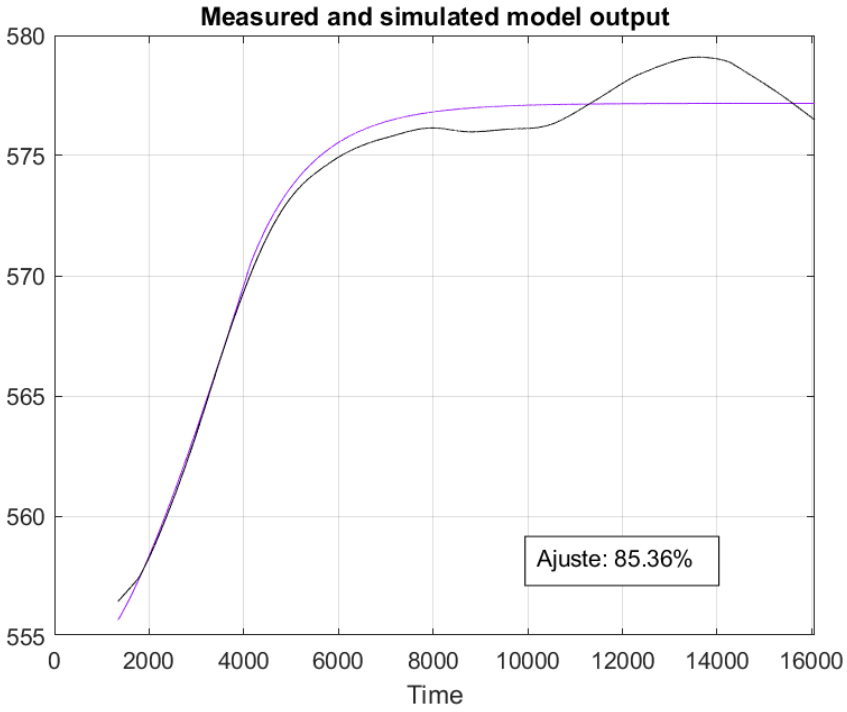

Figura 3-13. Ajuste del modelo para la respuesta del ORP frente al ácido.

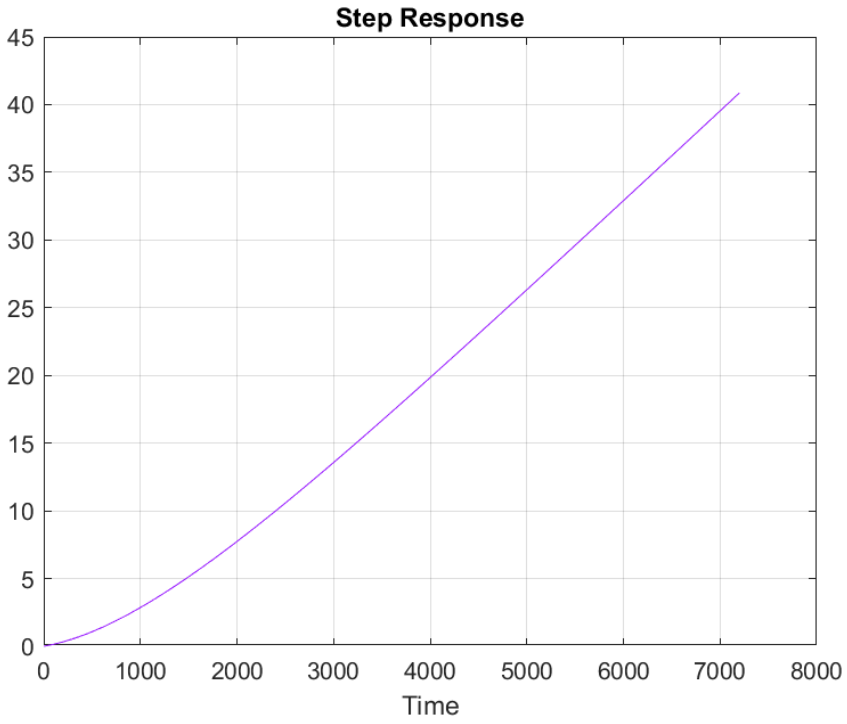

Figura 3-14. Respuesta ante escalón del modelo para la respuesta del ORP frente al ácido.

- Par lejía-pH (G_{12}): en este par se consigue el menor ajuste, solo de un 64.87%. Sin embargo, se considera un valor aceptable, puesto que se trata del modelo de una respuesta acoplada (el objetivo principal del uso de la lejía es subir el ORP). El transitorio concuerda con el comportamiento esperado, que es que el pH suba ligeramente.

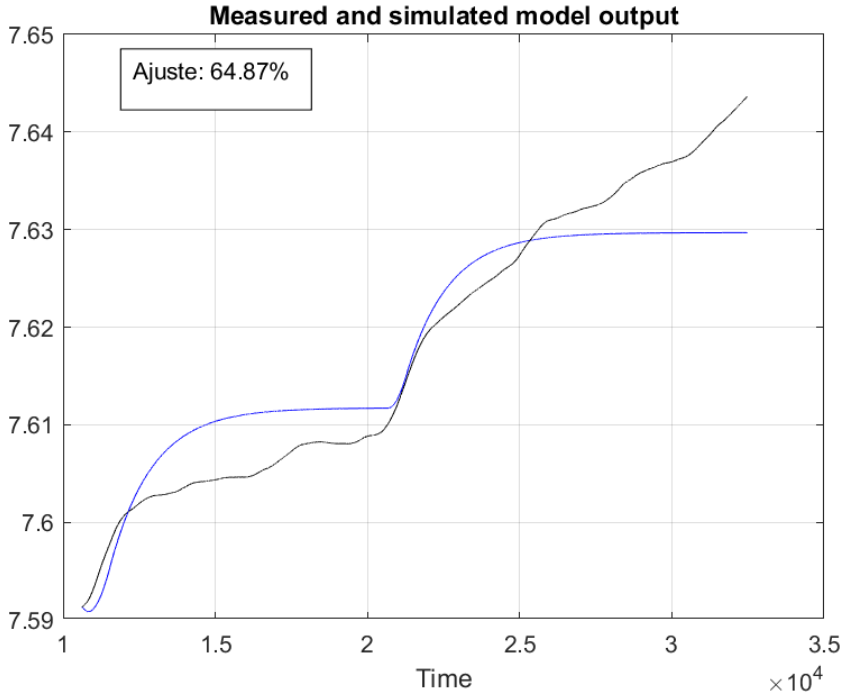

Figura 3-15. Ajuste del modelo para la respuesta del pH frente a la lejía.

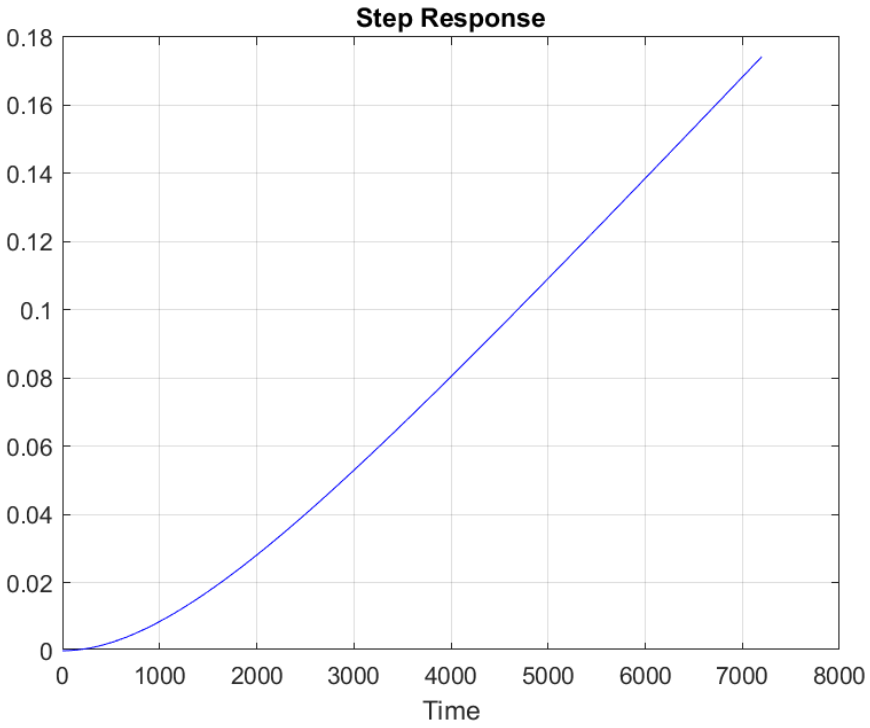

Figura 3-16. Respuesta ante escalón del modelo para la respuesta del pH frente a la lejía.

- Par lejía-ORP (G_{22}): nuevamente se consigue un ajuste bastante alto, con un 92.71%, y un transitorio en consonancia con la respuesta esperada.

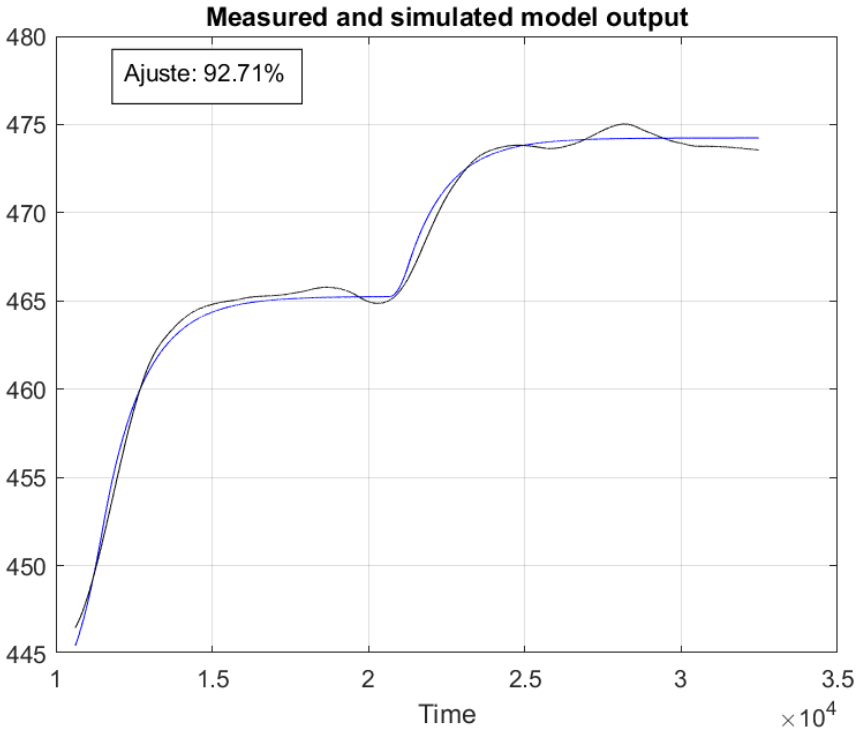

Figura 3-17. Ajuste del modelo para la respuesta del ORP frente a la lejía.

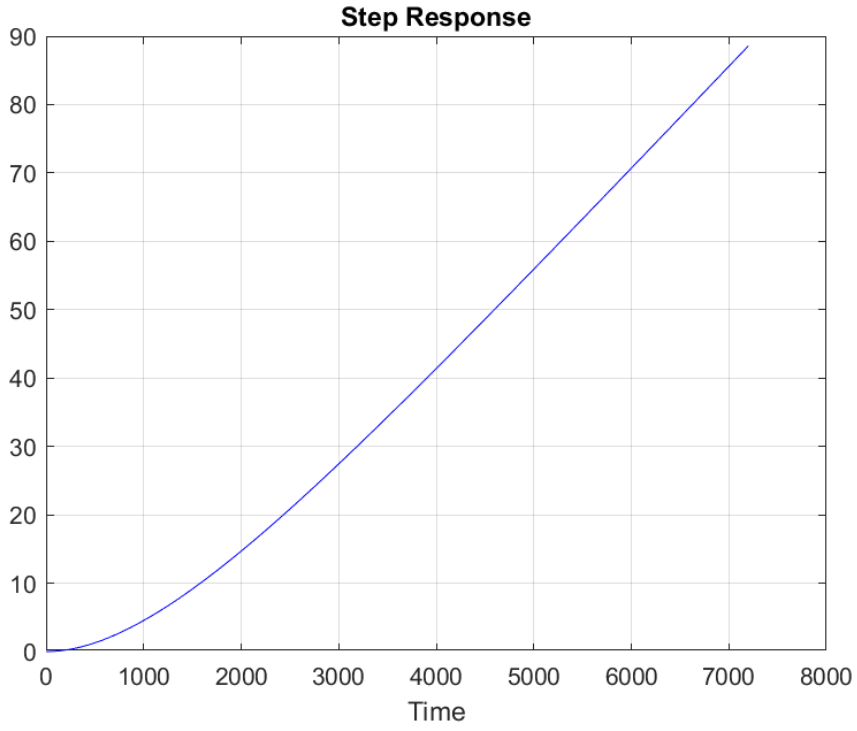

Figura 3-18. Respuesta ante escalón del modelo para la respuesta del ORP frente a la lejía.

4 DISEÑO DEL CONTROL

El código[1][2] desarrollado en Matlab consta principalmente de dos partes:

1. Una primera parte en la que, a partir de las funciones de transferencia obtenidas tras identificar el sistema, se inicializan todas las variables necesarias para la función que ejecuta el control. Todo esto se incluye en el script *init.m*, que a su vez cuenta con diversas funciones con las que realizar todos los cálculos matemáticos previos.
2. Una segunda parte en la que se implementa el control propiamente dicho. Esta parte se incluye en el script *calculo_gpc.m* e incluye el algoritmo de control utilizado para obtener las acciones de control óptimas.

4.1 Init.m

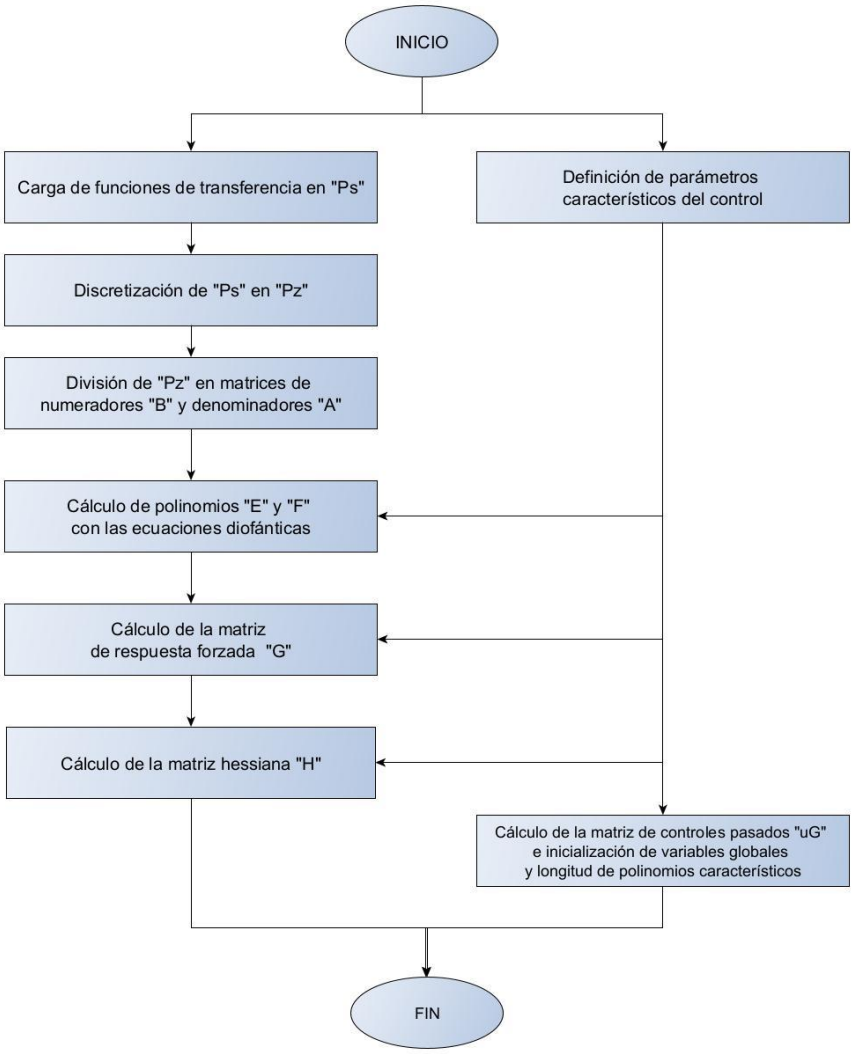

Figura 4-1. Diagrama de flujo de *init.m*.

[1] El código se desarrolla en base a los archivos disponibles en la página web de Sergio Andrés Castaño Giraldo [18].
[2] El código del control en Matlab se encuentra en el anexo A.

Primero, se cargan los modelos que se exportaron de la toolbox *System Identification* al espacio de trabajo de Matlab y se agrupan en una matriz de matrices *Ps*, para trabajar con las 4 funciones de transferencia. Esto es lo que se conoce como un control MIMO, de *multiple inputs, multiple outputs* (en este caso, al ser 2, se le conoce como TITO, de *two inputs, two outputs*). De esta manera, se tiene en cuenta en el control el acoplamiento, sin necesidad de implementar un *feed-forward* o similar para convertir el sistema en desacoplado.

Tras esto, dado que el GPC trabaja con funciones discretas, se discretiza *Ps*, con un tiempo de muestreo de 180 segundos. El tiempo de muestreo escogido es de 3 minutos porque tras distintas pruebas (se muestran en el apartado 5) se llegó a la conclusión de que el rendimiento era adecuado y un tiempo menor o mayor no lo mejoraba.

Se definen los parámetros característicos del control predictivo:

- Horizontes de predicción y de control: en el código, se definen como *N* y *Nu* respectivamente. Se trata de vectores de dos elementos, por ser un GPC TITO. Los valores escogidos son 5 para ambos *N* y 3 para ambos *Nu*. El valor escogido para el horizonte de predicción se obtiene de ir aumentándolo hasta que el rendimiento del control deje de mejorar o empeore. En cuanto al de control, generalmente se usa un $Nu < N$, de modo que, tras probar con distintos valores, se dejó en 3 para ambas entradas.
- Parámetros λ y δ: λ se deja como 1 para ambas acciones de control, puesto que no suponía prácticamente ningún cambio utilizar otros valores debido a la naturaleza ON/OFF del control del trabajo. En cuanto a δ, se toman los valores de 1 para el pH y 10^{-5} para el ORP. El motivo es que la ponderación afecta al uso de los líquidos en función del error respecto a la referencia, estando ese término elevado al cuadrado en la función objetivo, de modo que cuando $error_{ORP}^2 \times \delta_{ORP}$ empieza a ser mayor que $error_{pH}^2 \times \delta_{pH}$ el control deja de usar lejía para aumentar el pH, y viceversa para el uso del ácido para elevar el ORP. La elección de la relación $1/10^{-5}$ hace que se puedan utilizar los líquidos para las dinámicas acopladas hasta proporciones de aproximadamente $300 \cdot error_{pH} = error_{ORP}$ (por ejemplo, hasta un error de 0.1 en el pH se permite el uso del ácido para aumentar más rápidamente el ORP). Como se verá en las simulaciones del siguiente apartado, ponderaciones mayores del ORP acortan el rango del uso de las dinámicas acopladas y mayores lo amplifican demasiado, motivo por el cual el valor de 10^{-5} resulta el óptimo para los errores considerados como aceptables en el proyecto.

Después, de la nueva matriz de funciones de transferencia discretizadas, *Pz*, se extraen las matrices de numeradores y denominadores para realizar el cálculo de los polinomios *E* y *F* resolviendo las ecuaciones diofánticas. Tras esto, se utilizan dichos polinomios para conformar la matriz de la respuesta forzada, que se utilizará posteriormente para el cálculo de la Hessiana.

Por último, se calcula la matriz de controles pasados y se inicializan las variables globales, valores iniciales y longitudes de polinomios.

4.2 Calculo_gpc.m

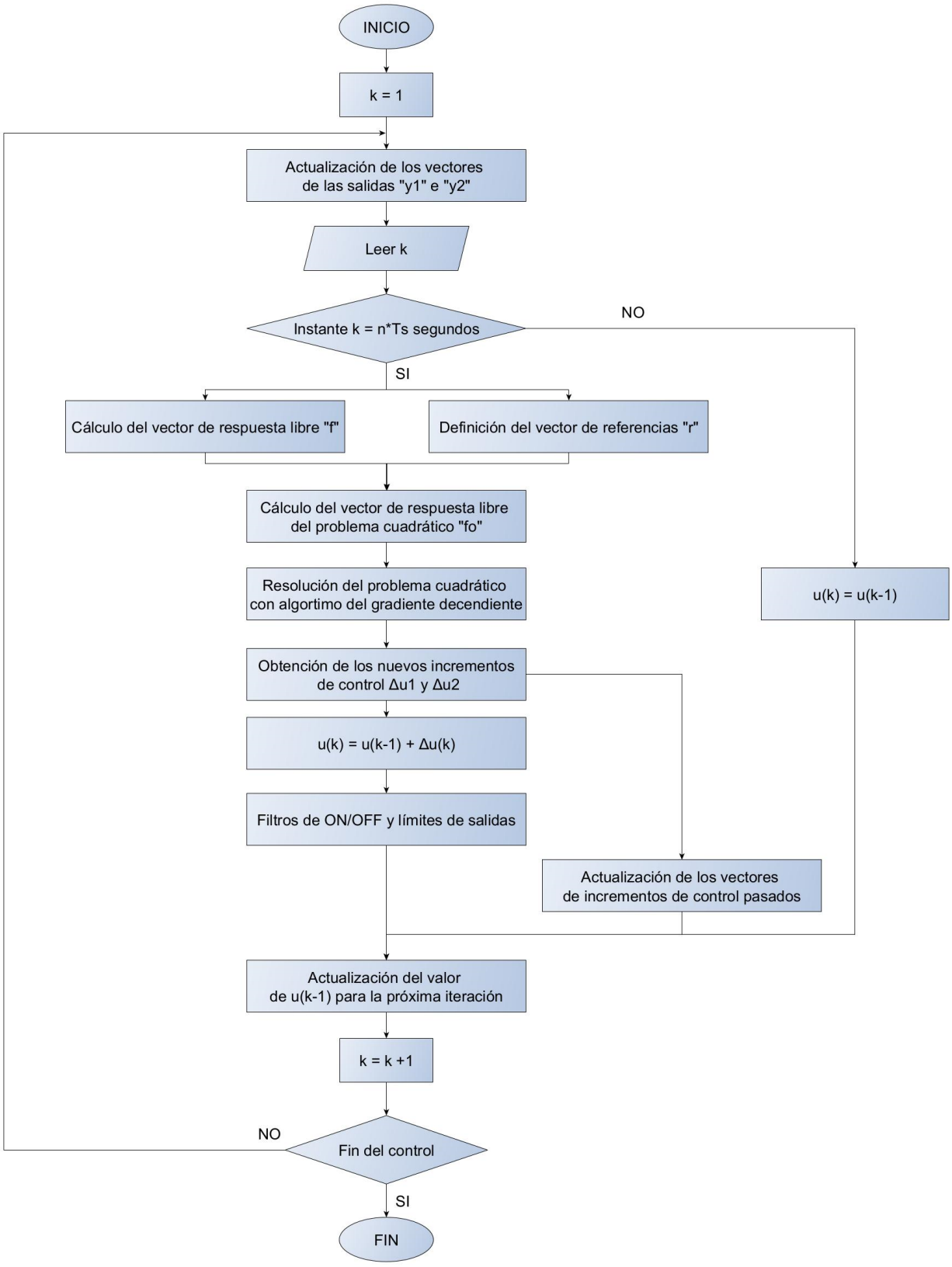

Figura 4-2. Diagrama de flujo de *calculo_gpc.m*.

Una vez inicializados todos los vectores, matrices y demás parámetros, el control se lleva a cabo con la ejecución de la función *calculo_gpc()* en cada instante de la simulación del modelo en Simulink.

Consta básicamente de 3 partes. En primer lugar, se actualizan, con las medidas actuales, los vectores de las salidas del sistema, es decir, los vectores y_1 e y_2 correspondientes a los niveles de pH y ORP. Esto es necesario para posteriormente realizar el cálculo del vector de la respuesta libre. Idealmente funcionan como una pila, de modo que en el último elemento se encuentra el último valor medido y los anteriores hasta el primero son las medidas sucesivas anteriores. No obstante, se opta por igualar todos los términos de cada vector a la última medida correspondiente, para no introducir el error de los sensores en el control.

El resto del código está todo englobado en un *if-else*:

- Cuando el instante de ejecución de la función es un múltiplo del tiempo de muestreo, se ejecuta el código dentro del *if*. Se calcula el vector de la respuesta libre *f* con la función *calcula_free()*, se forma el vector de referencias *r* (con las referencias definidas en *init.m* o con las auxiliares que se tienen como entrada de la función si estas últimas están dentro de los límites) y se resuelve el problema cuadrático. Para esto último es necesario definir los límites inferior y superior de ambos inyectores y el vector final libre *fo*.
  Una vez resuelto el problema cuadrático con la función *quad_grad()* y obtenidos los incrementos de las acciones de control (el resultado del problema cuadrático es Δu), se calculan u_1 y u_2 simplemente sumando dichos incrementos a las acciones pasadas $uant_1$ y $uant_2$.

  Estas nuevas acciones de control pasan por dos filtros. Uno para asegurarse de que no se devuelve ningún valor de *u* entre los límites inferiores y superiores previamente definidos (son límites para las *u* pero no para los Δu, por lo que el control puede calcular valores intermedios para los incrementos de las acciones de control, lo cual no tiene sentido ya que los inyectores funcionan como un ON/OFF y no vierten un caudal intermedio entre 0 y 0.001 L/s). El otro, para obviar el control en caso de que se supere cualquiera de los límites en las salidas del sistema (previamente establecidos en el *init.m*) y forzar las acciones de control a ON/OFF dependiendo del límite superado.

  Por último, se actualizan los vectores de incrementos de control pasados para futuras iteraciones.

- El resto del tiempo, cuando el instante de ejecución no sea un múltiplo del tiempo de muestreo, se ejecuta el *else*, en el cual simplemente se mantienen las acciones de control calculadas en la última ejecución del control.

4.3 Calcula_free.m

Se implementan los pasos para obtener la respuesta libre expresada en la ecuación [2-31].

Ir al anexo A para ver el código de la función.

4.4 Quad_grad.m

Matlab cuenta con una función llamada *quadprog()* para resolver problemas cuadráticos, pudiendo configurar límites, restricciones de igualdad, de desigualdad, algoritmo a utilizar, etc. Esta fue la función utilizada en el desarrollo del trabajo para tunear el control. Sin embargo, dado que el código hay que pasarlo a MicroPython para poder implementarlo en la Raspberry de la piscina, se buscó una función distinta que sí se pudiera utilizar. Se creó la función *quad_grad()*, que debe el nombre al algoritmo escogido para resolver el problema, el conocido como "método del gradiente descendiente".

Básicamente, consiste en calcular el gradiente de la ecuación del problema cuadrático e ir iterando, calculando la nueva solución *x* de la siguiente forma [16]:

$$x_{new} = x - step * grad \tag{4–1}$$

Siendo[3]:

$$grad = H * x + f_0 \tag{4–2}$$

$$\text{step} = \frac{grad^T * grad}{grad^T * H * grad} \tag{4–3}$$

Con la nueva *x* se calcula en cada iteración el nuevo valor de la función objetivo, hasta que se alcance el máximo de iteraciones o la diferencia entre dicho valor y el de la iteración anterior sea menor que la tolerancia establecida.

Ir al anexo A para ver el código de la función.

[3] *x* y *grad* vectores columna.

5 SIMULACIONES EN SIMULINK

Las simulaciones del funcionamiento del control se realizan en Simulink. Para ello, se crea un modelo que contiene un bloque *Matlab function* que ejecuta la función *calculo_gpc()*, al cual se conectan como entradas los valores de los niveles (Y_1 e Y_2), un bloque *clock* para el tiempo y dos bloques *step* para las referencias auxiliares. Las salidas de la función son las acciones de control, que llegan como entradas a unos bloques *transfer Fcn* que representan cada una de las funciones de transferencia que se identificaron. De este modo, se simula la acción de la piscina y se obtienen Y_1 e Y_2.

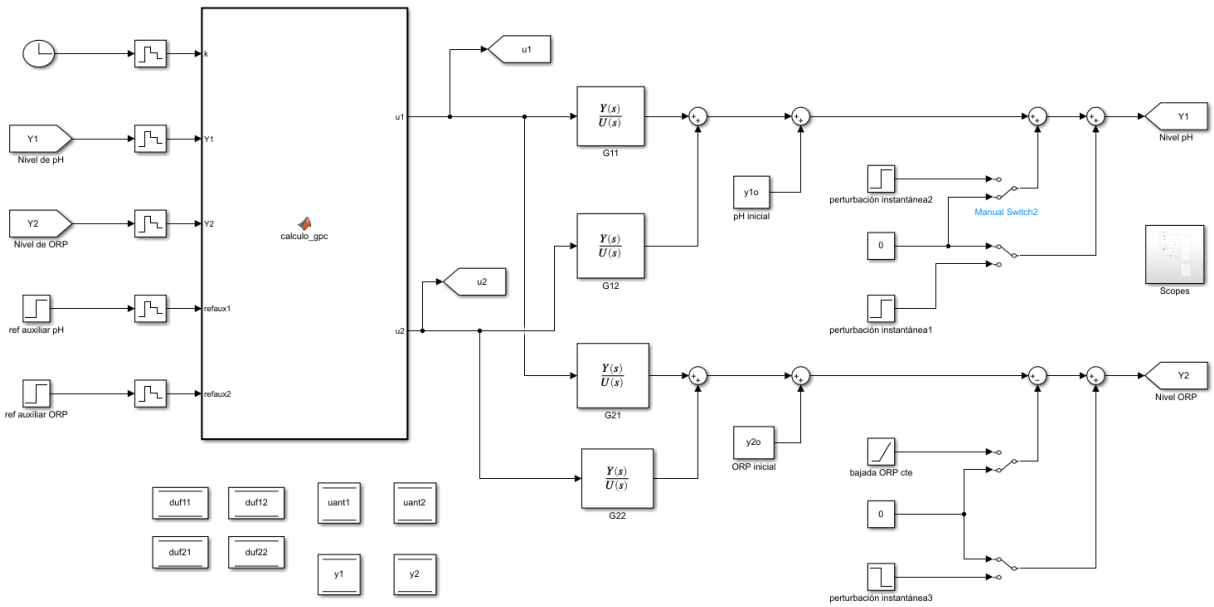

Figura 5-1. Modelo del control en Simulink.

Se realizaron varias simulaciones para ir ajustando el control y comprobar el correcto funcionamiento del código. No solo se comprobó este para un seguimiento simple de los setpoints, también se sumaron a las salidas Y_1 e Y_2 perturbaciones, modeladas como escalones para las perturbaciones instantáneas (por ejemplo, personas que entran a bañarse o cambios bruscos de temperatura) o como rampas para el caso de perturbaciones más prolongadas en el tiempo (por ejemplo, la acción de los rayos solares sobre el ORP).

Con el objetivo de mostrar el proceso de ajuste con el que se llegó a los parámetros finales, primero se representa la simulación para estos y después distintas simulaciones con cambios en dichos parámetros. Se dejan las simulaciones con cambios en las referencias para el apartado de las pruebas en el sistema real, donde se realizarán con los mismos valores iniciales y setpoints que se tienen en estas para comprobar el grado de similitud que existe entre la teoría y la práctica.

5.1 Simulación sin perturbaciones

El seguimiento conseguido es adecuado en ambos niveles. El error presente en régimen permanente depende principalmente del tiempo de muestreo, del ajuste limitado por la restricción ON/OFF de los actuadores y de la ponderación del error escogidos. En las pruebas posteriores se explica por qué, para las restricciones del sistema con las que se cuenta, los valores finales son aquellos con los que se obtienen los mejores resultados posibles.

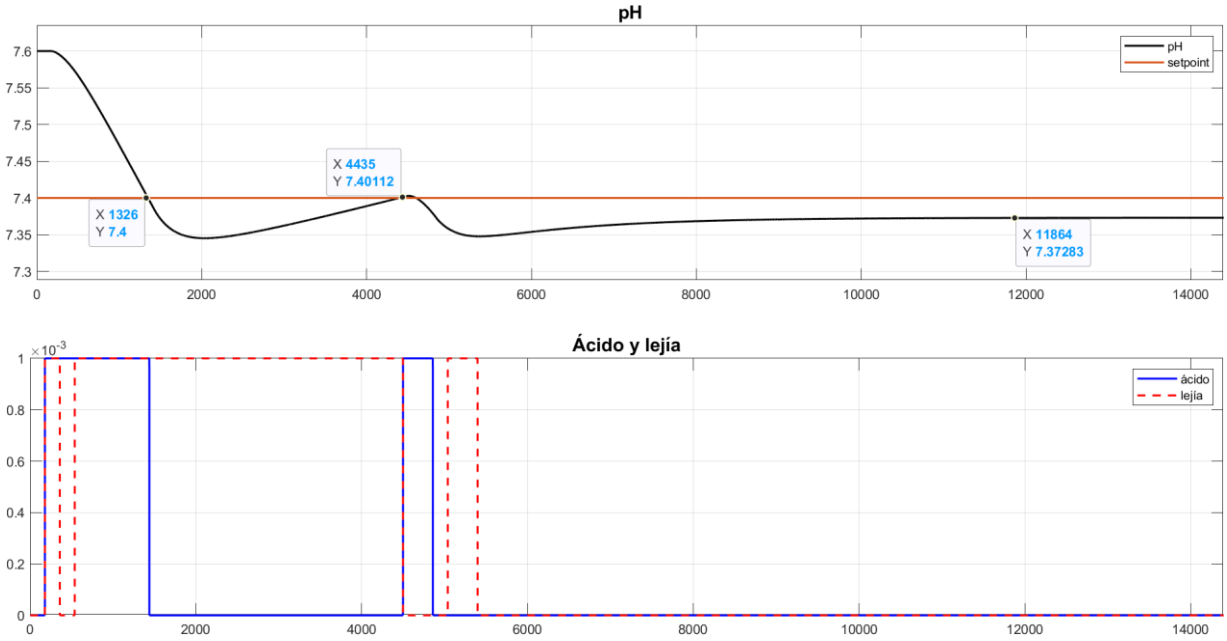

Figura 5-2. Simulación sin perturbaciones: pH.

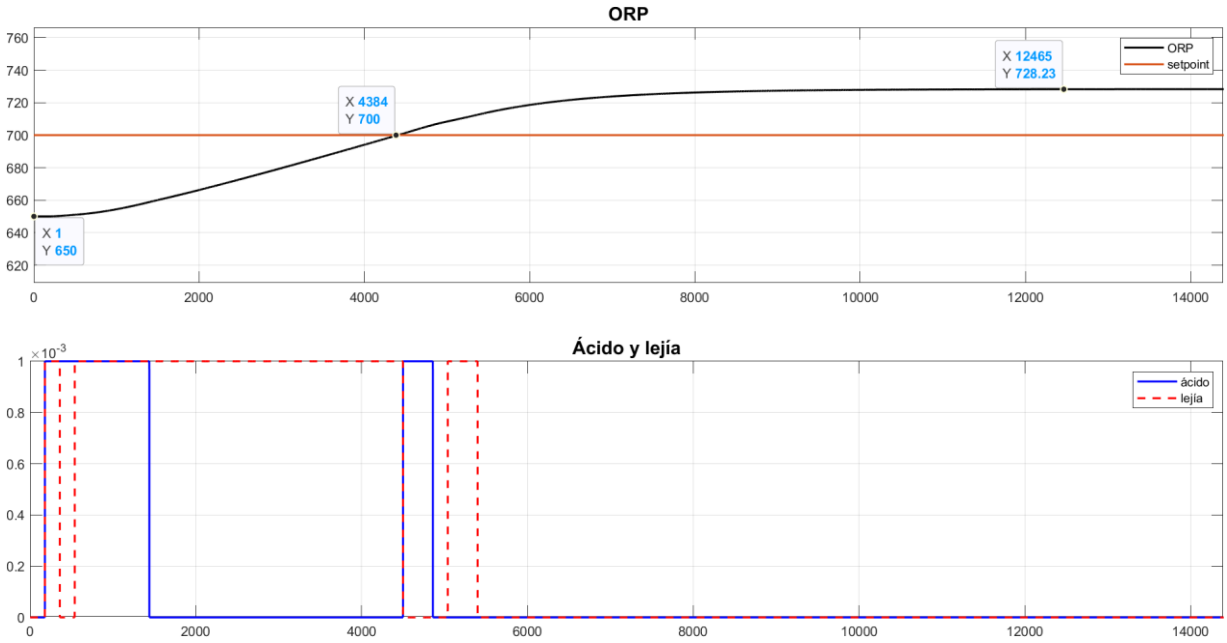

Figura 5-3. Simulación sin perturbaciones: ORP.

5.2 Simulación con perturbaciones

Se introdujeron perturbaciones instantáneas en ambos niveles y una bajada constante del ORP (10 mV a la hora). Se responde rápido a las perturbaciones y se vuelve a la referencias, lo cual da seguridad frente a la aplicación sobre el sistema real.

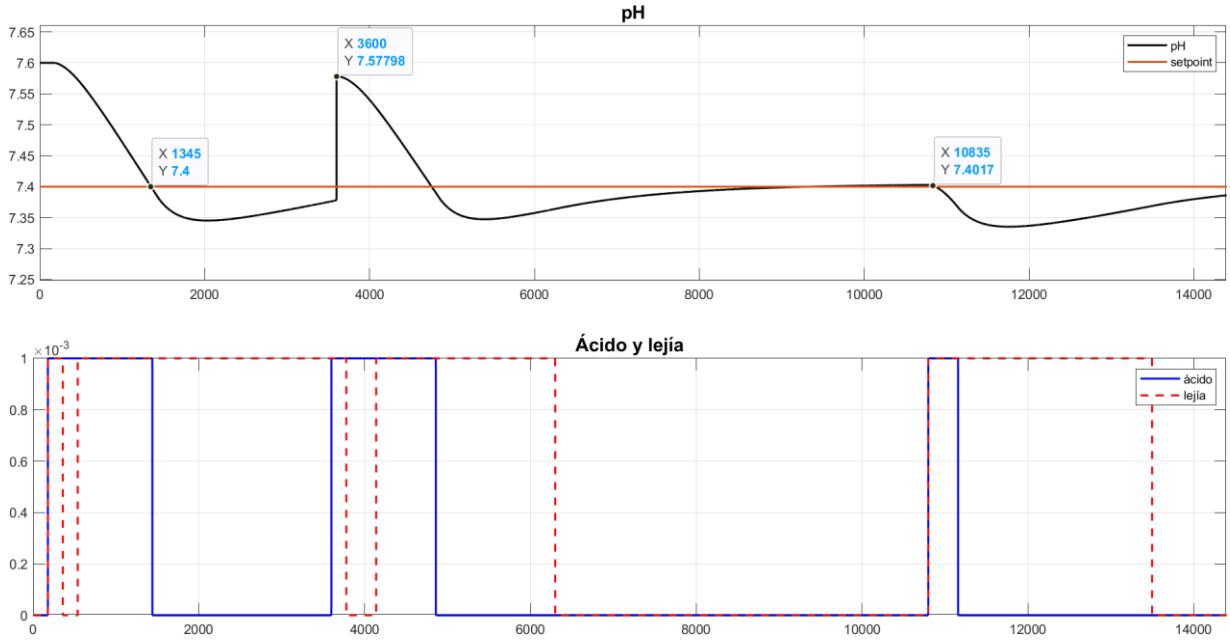

Figura 5-4. Simulación con perturbaciones: pH.

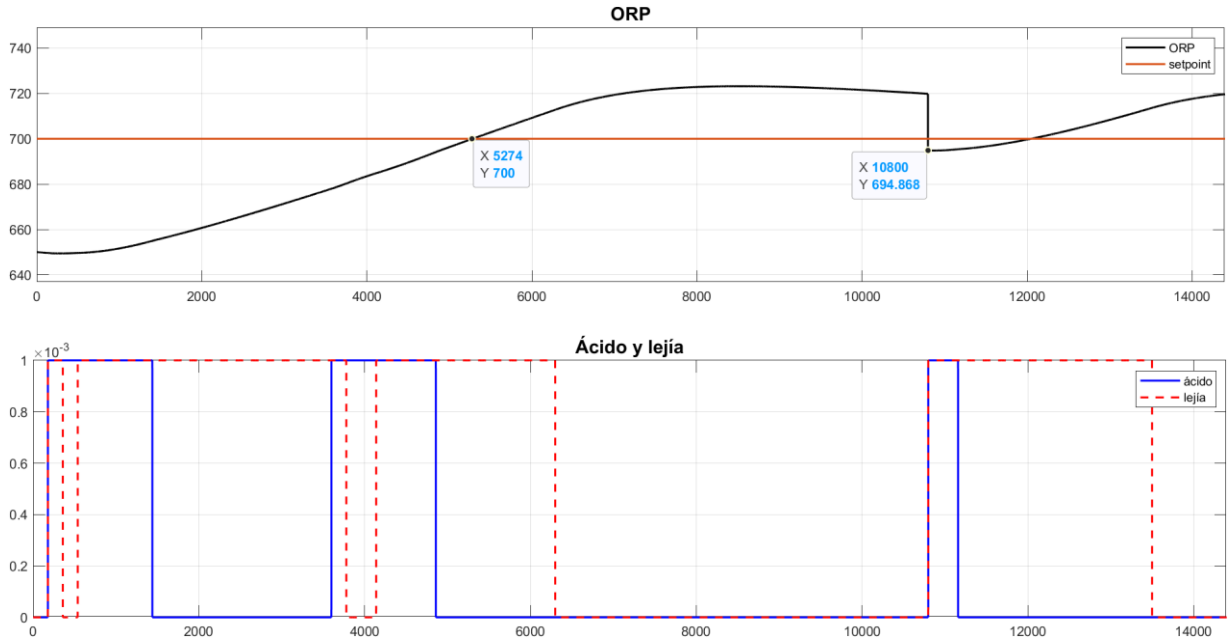

Figura 5-5. Simulación con perturbaciones: ORP.

5.3 Simulación con tiempo de muestreo mayor

En el caso de un tiempo de muestreo mayor (5 minutos), se aprecia un ajuste peor, puesto que el control prolonga más las acciones de control y no cuenta con la misma rapidez para realizar un cambio en los actuadores cuando los valores están cerca de los setpoints.

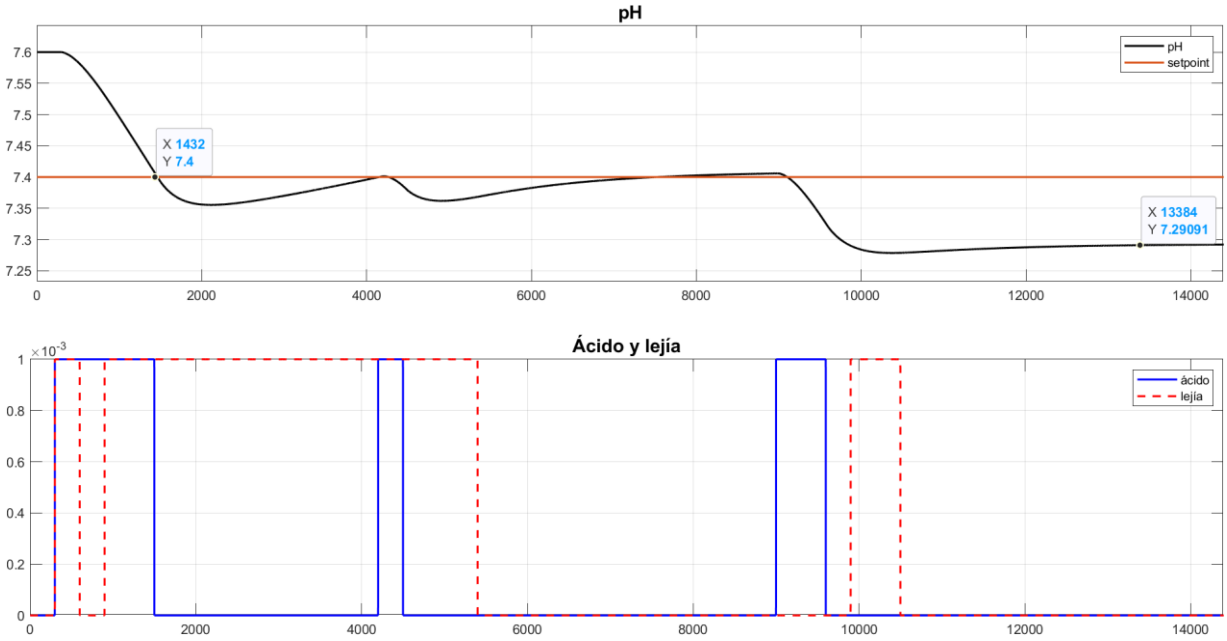

Figura 5-6. Simulación con Ts mayor: pH.

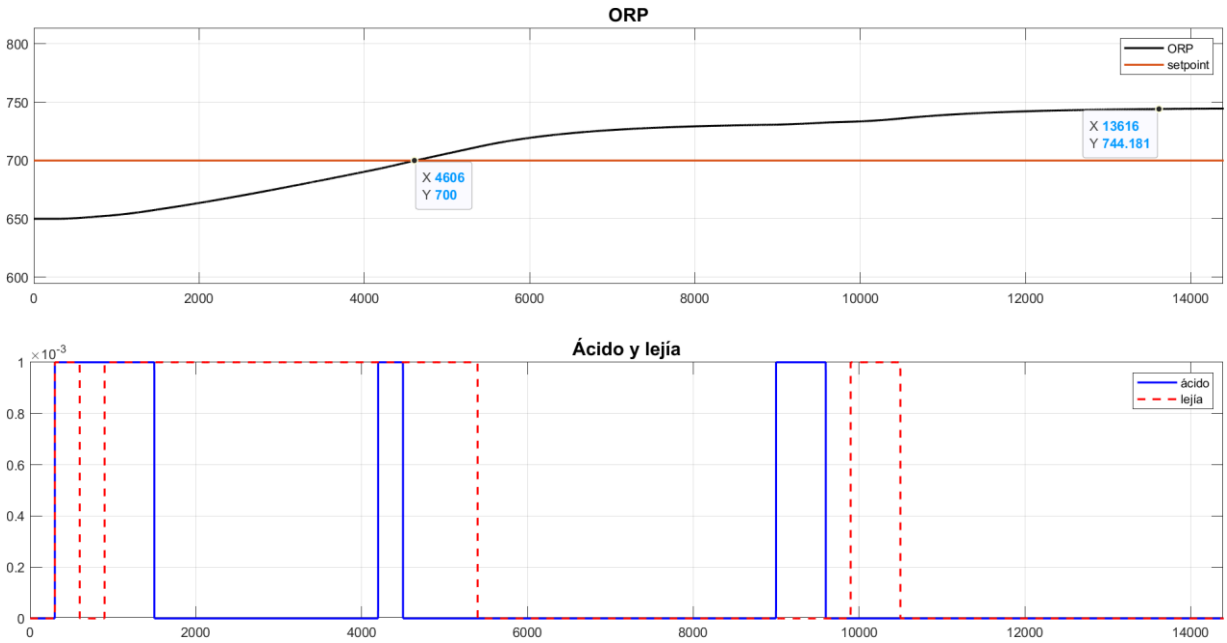

Figura 5-7. Simulación con Ts mayor: ORP.

5.4 Simulación con tiempo de muestreo menor

En el caso de un tiempo de muestreo menor (se escoge 1 minuto porque menos no es compatible con el código general, puesto que las medidas de los niveles se realizan cada 30 segundos, de modo que se prefiere contar con al menos dos medidas entre cada cálculo del control), a pesar de contar con escalones más pequeños, no se consigue un ajuste menor, con errores en régimen permanente más acentuados.

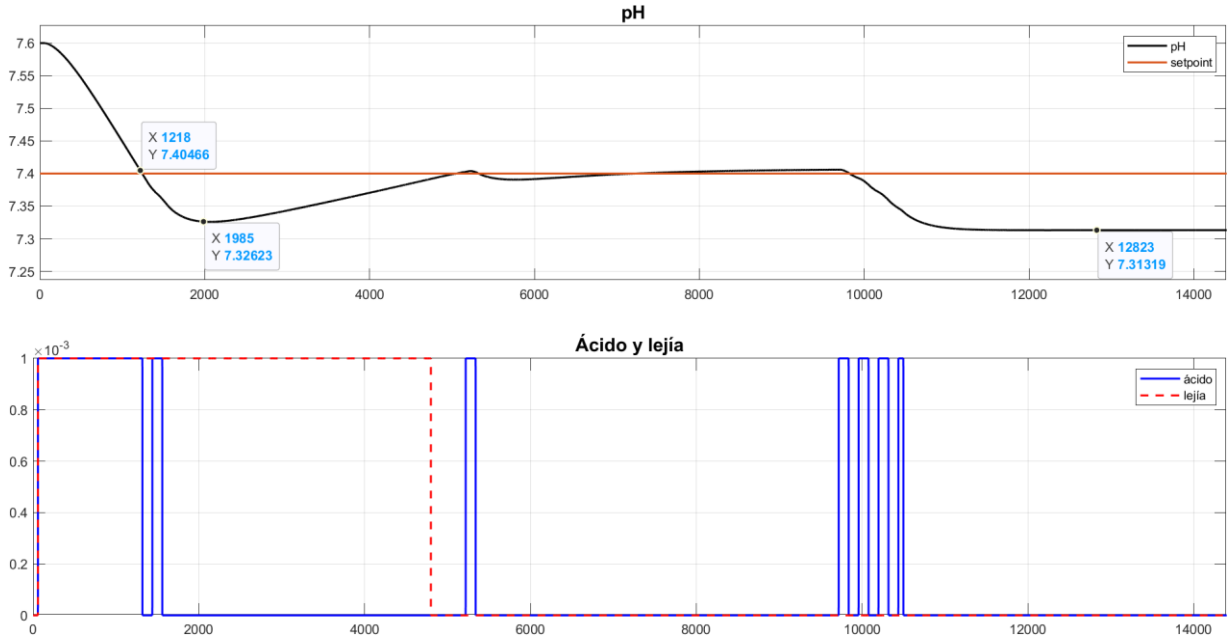

Figura 5-8. Simulación con Ts menor: pH.

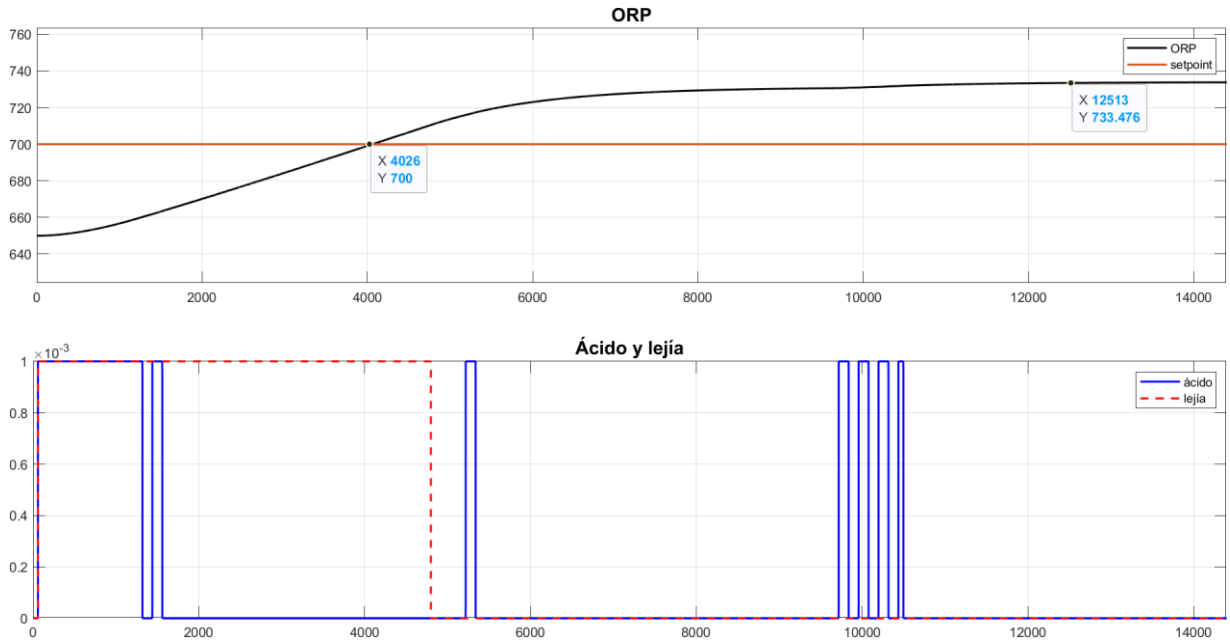

Figura 5-9. Simulación con Ts menor: ORP.

5.5 Simulación con mayor ponderación para el seguimiento del ORP

Al dar mayor importancia al seguimiento del ORP ($\delta_{ORP} = 10^{-4}$) se consigue un mayor ajuste (aunque solo de unos pocos mV) en este a costa de uno peor en el pH. Atendiendo a los objetivos del trabajo, en el que un valor de ORP mayor no es perjudicial, mientras que uno más alejado del setpoint de pH sí que lo es, no se trata de una opción conveniente.

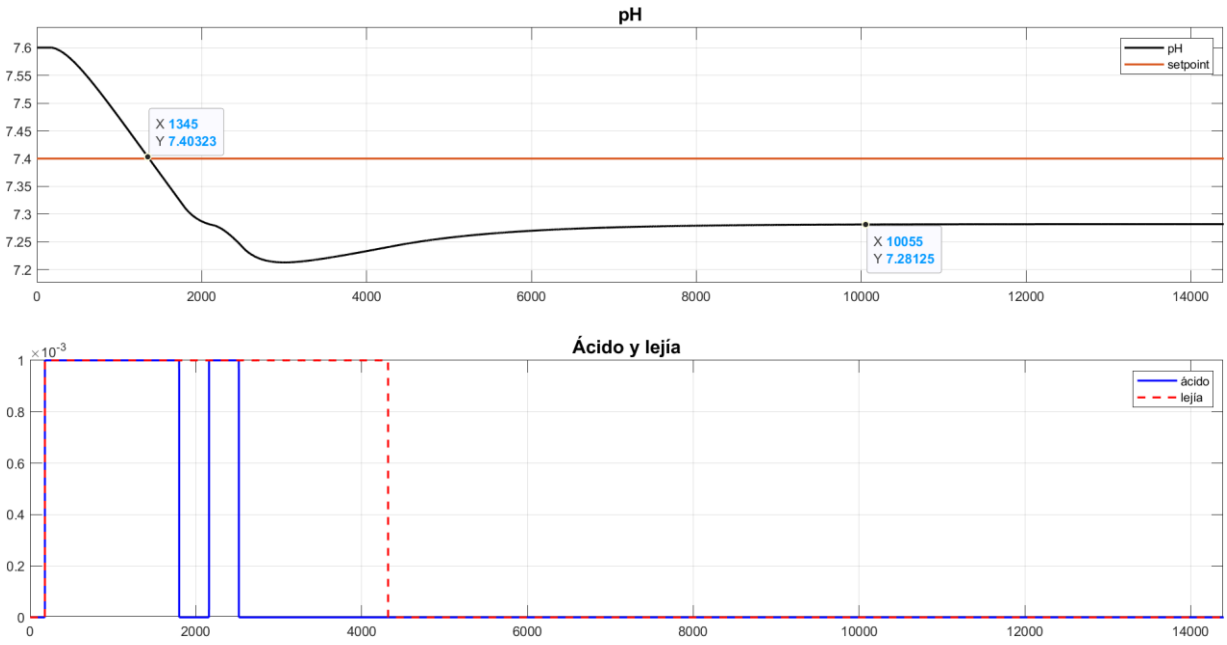

Figura 5-10. Simulación con δ_{ORP} mayor: pH.

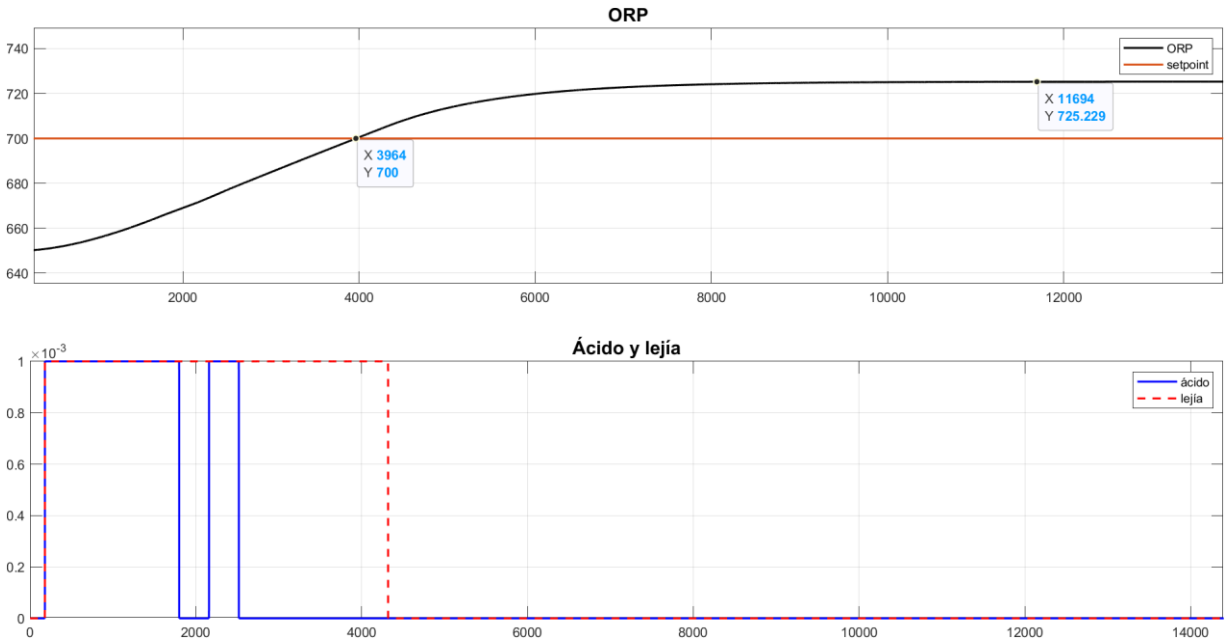

Figura 5-11. Simulación con δ_{ORP} mayor: ORP.

5.6 Simulación con menor ponderación para el seguimiento del ORP

En el caso de una ponderación menor en el ORP ($\delta_{ORP} = 10^{-6}$), atendiendo a lo explicado en el apartado 4 acerca del efecto sobre el término del error, era de esperar que el control usase más la lejía en busca de un mayor ajuste en el pH. Esto, unido a que el control no consigue dejar el nivel en el setpoint y siempre lo sobrepasa, hace que se tengan más escalones en las entradas, consiguiendo un mayor gasto sin mejora en el seguimiento.

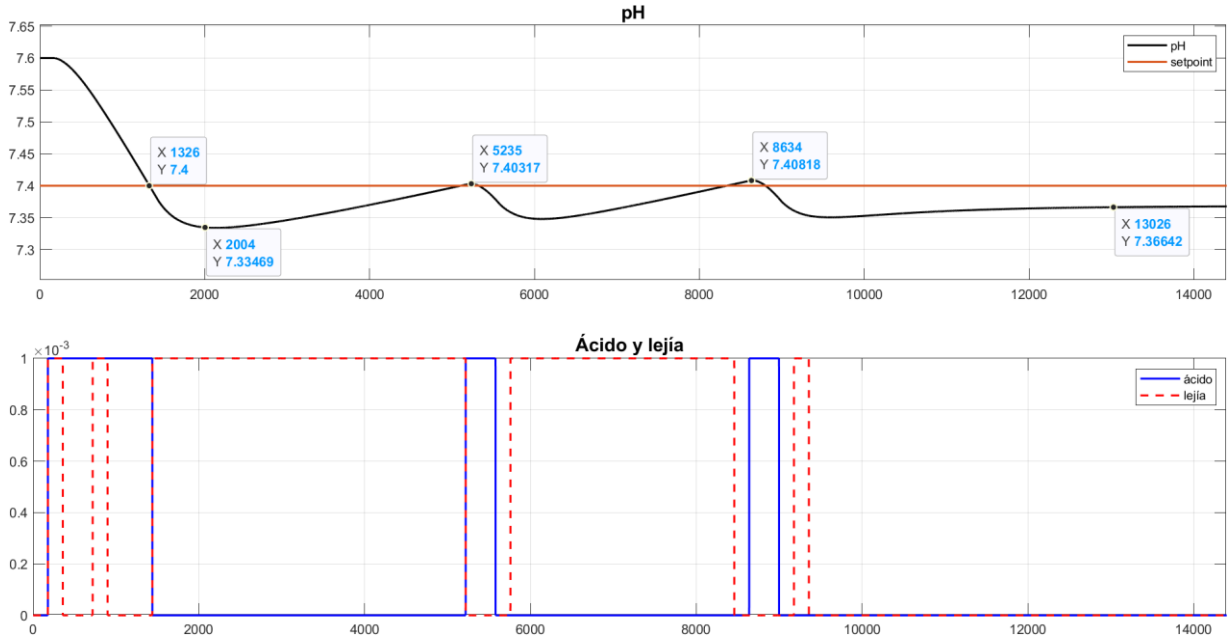

Figura 5-12. Simulación con δ_{ORP} menor: pH.

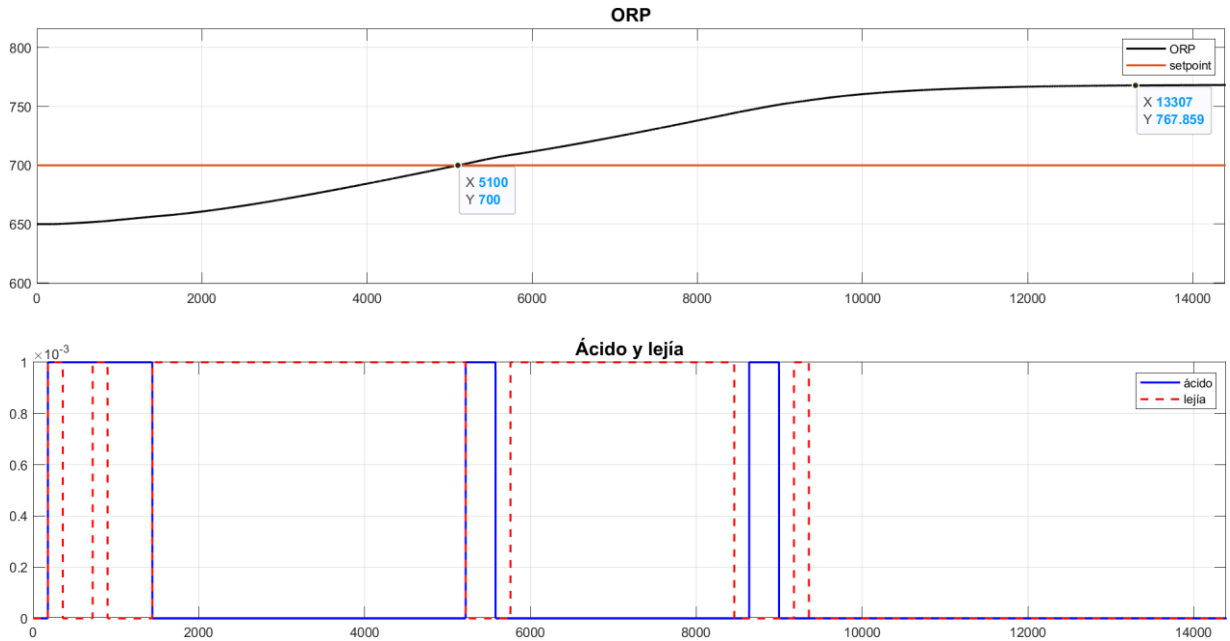

Figura 5-13. Simulación con δ_{ORP} menor: ORP.

5.7 Simulación con horizontes mayores

Con valores mayores en los horizontes (N = 10 y Nu = 5) no se obtiene ninguna diferencia significativa, por lo que se opta por valores menores, que disminuyen el coste computacional.

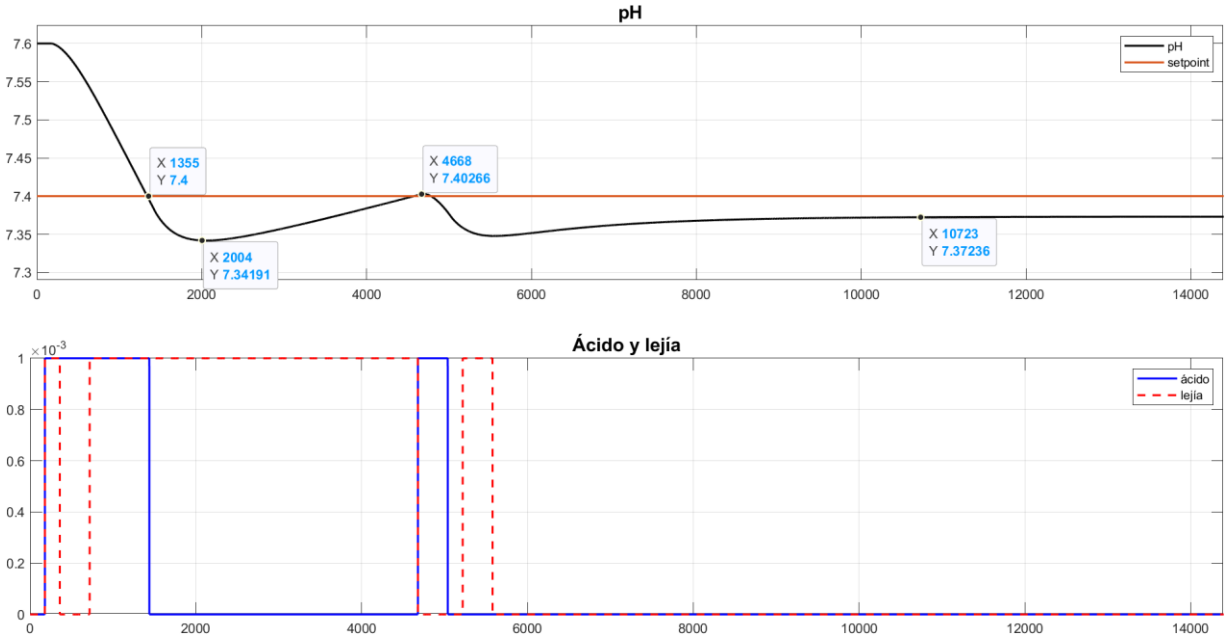

Figura 5-14. Simulación con horizontes mayores: pH.

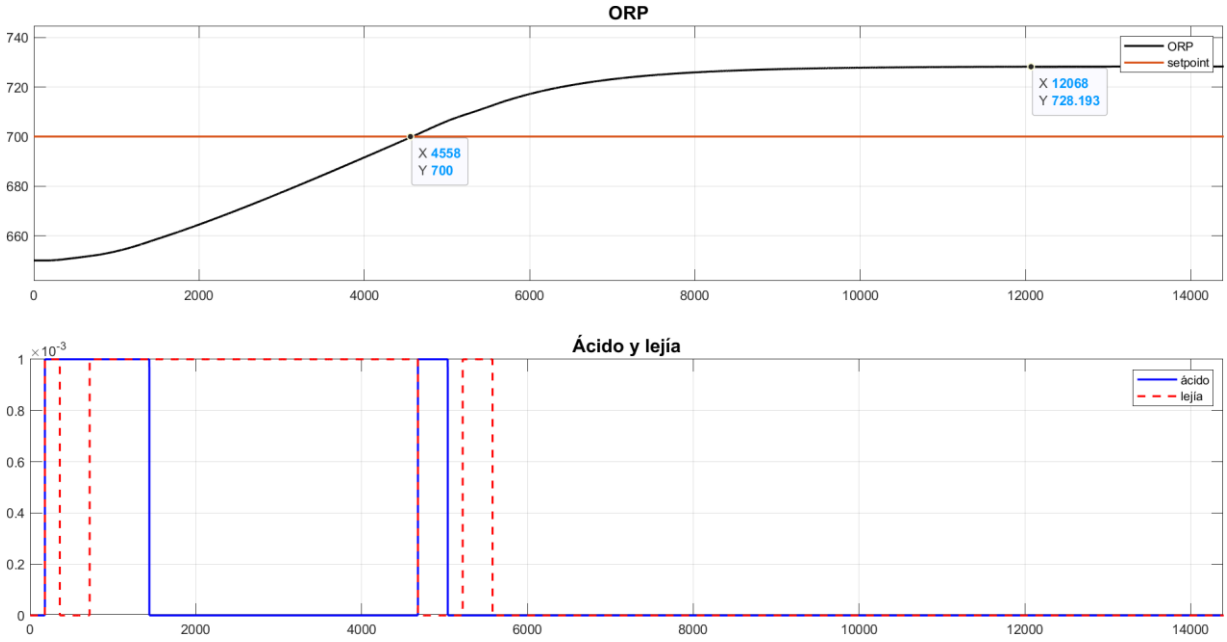

Figura 5-15. Simulación con horizontes mayores: ORP.

5.8 Simulación con horizontes menores

Para valores menores en los horizontes (N = 3 y Nu = 2) tampoco se obtiene una mejora significativa, presentando un gasto mayor de lejía en el último escalón y un ajuste ligeramente mejor en el pH. Sin embargo, se opta por conservar valores más altos en estos parámetros, para mayor seguridad en situaciones con cambios en las referencias (a mayores horizontes, mayor capacidad de anticipación del control). Es por eso que se escogen los valores de N = 5 y Nu = 3, algo mayores pero sin derivar en un gasto computacional innecesario.

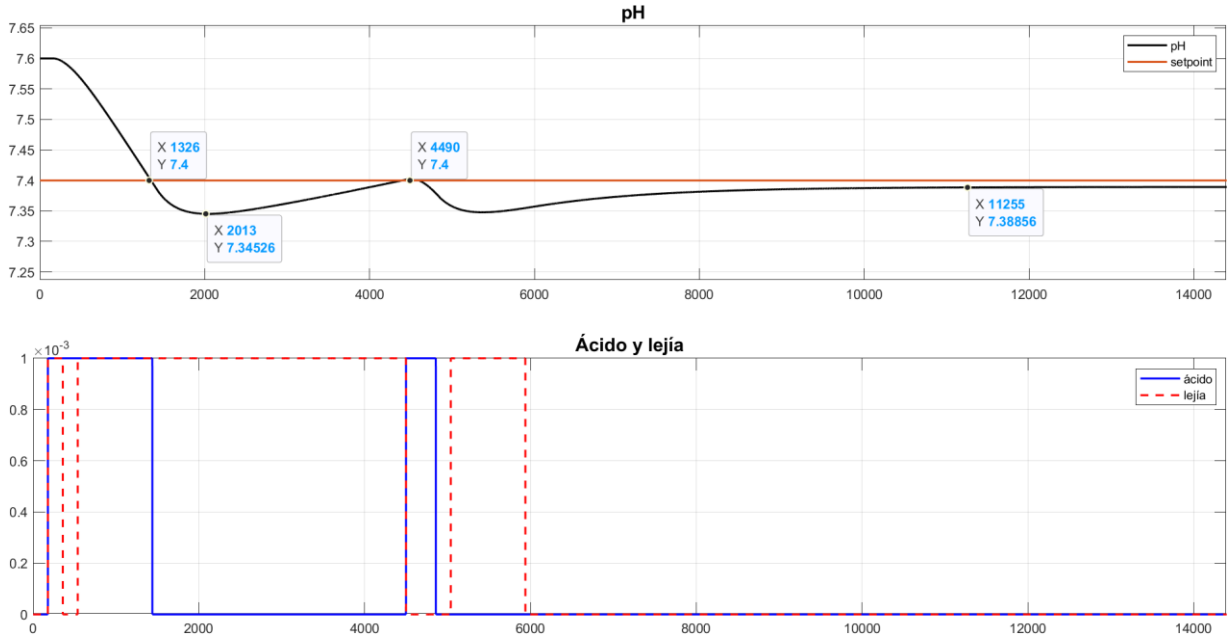

Figura 5-16. Simulación con horizontes menores: pH.

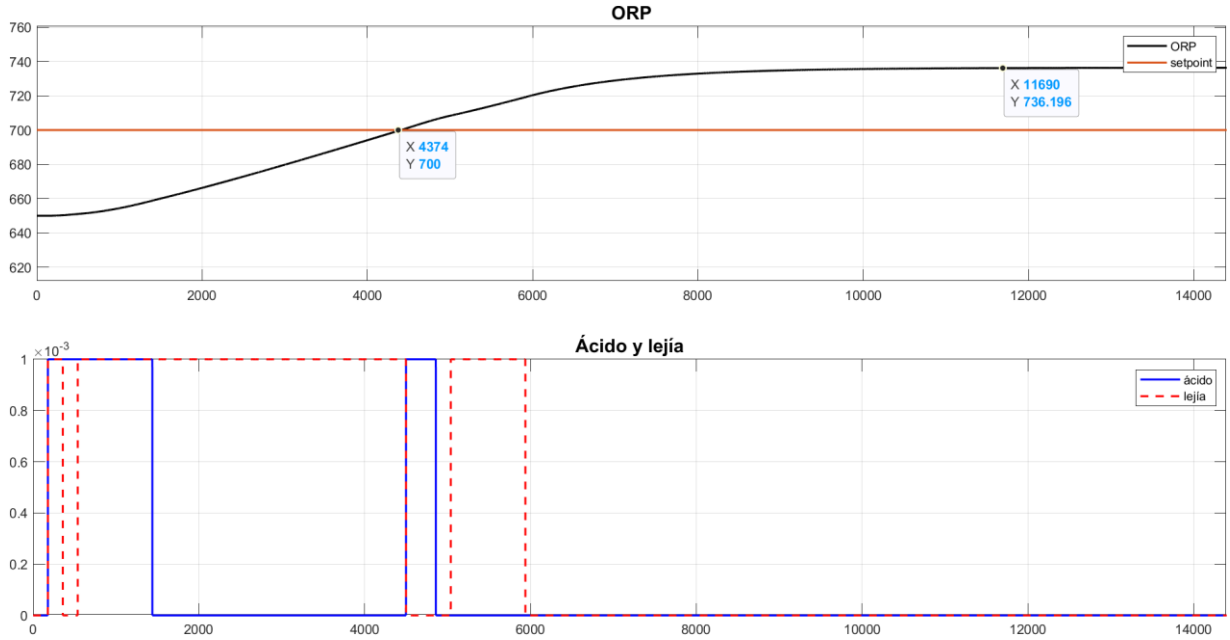

Figura 5-17. Simulación con horizontes menores: ORP.

6 CONVERSIÓN A PYTHON Y PRUEBAS EN SISTEMA REAL

Para implementar el control en la Raspberry, se necesita en lenguaje Python. Para realizar la conversión, se ha hecho uso de la herramienta *Copilot*, una extensión para *Visual Studio Code* que genera sugerencias en el código a partir de una inteligencia artificial.

Dado que en el *init.m*, todas las variables generadas conforman parámetros que no cambian a lo largo del control (además de la declaración de las variables globales), estos se inicializan en un archivo a partir de los valores guardados en el workspace de Matlab, en lugar de volver a implementar todas las funciones que contenía el script.

Para el código de *calculo_gpc.m* sí se utilizó Copilot. Es importante destacar que no bastaba con programar en Python, si no que había que habilitarlo para funcionar en Micropython, un lenguaje Python reducido para microcontroladores, que conlleva algunas limitaciones, principalmente a la hora de disponer de ciertas funciones. Sin embargo, aprovechando el proyecto de github *ulab* [17], fue posible importar la librería *numpy*, sobre la cual se han basado prácticamente todos los cálculos.

Se producen cambios notables al pasar de un lenguaje a otro, principalmente:

- Las variables declaradas como *global* ahora se manejan dentro del objeto *self*.
- Los vectores y matrices están indexados con base 0 en Python, al contrario que en Matlab, que utiliza base 1. Por tanto, hay diferencias a la hora de trabajar con arrays.
- *Numpy* permite realizar operaciones similares a lo que se tenía en Matlab, como el uso de las funciones *size()* para obtener el tamaño de los vectores o la función *zeros()* para crear vectores de ceros. Sin embargo, hay algunos cambios más notables, como el uso de la función *numpy.dot()* para multiplicar matrices, puesto que el operador (*) con *numpy.arrays* se utiliza para multiplicar los elementos uno a uno en lugar de realizar la multiplicación matricial.
- No es necesario el uso de (;) al final de cada línea (en Matlab se usaban para evitar mostrar por la consola el resultado de cada línea).
- No se utilizan ({}) para establecer los bloques de código, en Python se usa la indentación para ello.

Una vez convertido el código, se realizaron pruebas simultáneamente en el entorno de la Raspberry y en Matlab para comprobar que, con los mismos valores de entrada, las funciones devolvían los mismos resultados.

Tras esto, el código del trabajo[4] está listo para implementarse dentro del de la piscina, sustituyendo al control que se utilizaba hasta entonces.

Se llevan a cabo tres días de pruebas en la piscina, puesto que en los dos primeros los resultados no fueron del todo satisfactorios. A continuación, se detallan las pruebas en cada uno de esos días, justificando los resultados y explicando qué cambios se implementan entre las mismas. En los tres casos se trata de comprobar el rendimiento del control dando distintas referencias para cubrir los distintos estados posibles dependiendo del valor de cada nivel respecto a su setpoint. Los datos se extraen de un archivo *json* que se genera durante el transcurso de las pruebas.

[4] El código final del control en Python se encuentra en el anexo B.

6.1 Primera prueba

En esta primera prueba se tuvieron algunos inconvenientes. Por un lado, era necesario parar el control cada vez que se quería cambiar de referencia. Por otro lado, en el paso del código de Matlab a Python no se percibió un bug en la parte donde se actualizan los vectores de las variaciones de control pasadas (los duf_{ij}), de modo que el control no actuó como debía en algunos puntos, como puede observarse en que sigue vertiendo ácido a pesar de estar el nivel de pH por debajo del setpoint.

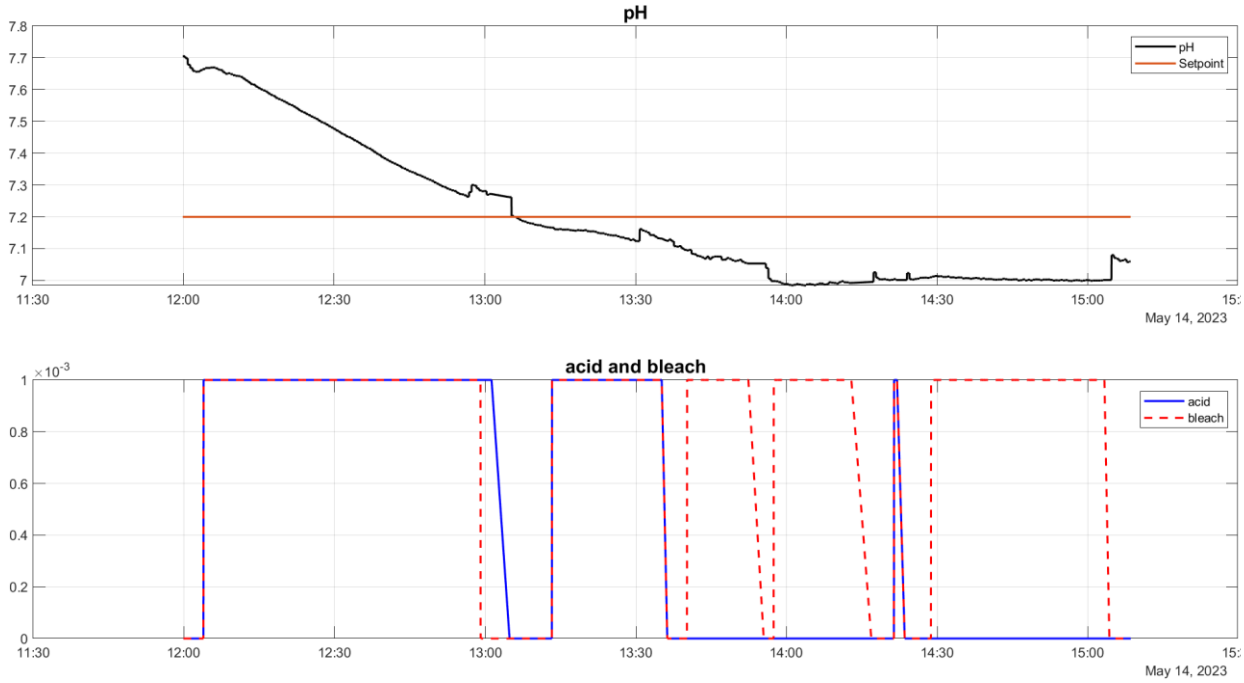

Figura 6-1. Resultado pH primera prueba.

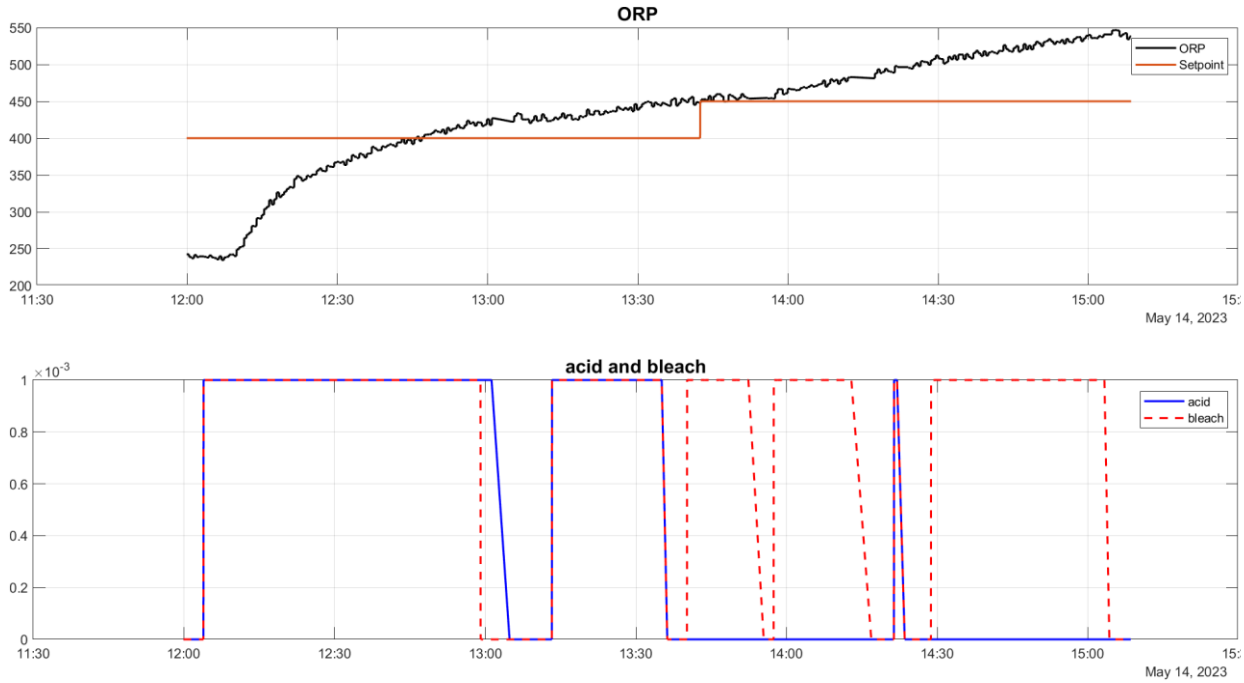

Figura 6-2. Resultado ORP primera prueba.

Por tanto, esta prueba solo sirvió para concluir que:

- El pH y el ORP responden como se esperaba frente a sus actuadores principales, es decir, el ácido y la lejía, respectivamente, lo cual indica una buena identificación de G_{11} y G_{22}.
- No puede extraerse un resultado claro de la dinámica acoplada del ORP respecto al ácido, puesto que no se dio un solo escalón de ácido aislado.
- En el escalón de lejía aislado al final, se ve que la subida del ORP es menor que en el del principio con los dos líquidos actuando de forma simultánea, pero esto también puede deberse a que cuanto más alto es el nivel de ORP, más difícil es seguir subiéndolo, como se comentó en la introducción.
- Respecto a la dinámica del pH respecto a la lejía, esta prueba hace ver que respuesta es de ganancia prácticamente nula, cosa que se confirmará en las siguientes pruebas.

6.2 Segunda prueba

En esta prueba se obtuvieron resultados más satisfactorios, aunque no finales. Esta vez, se contaba con la capacidad de cambiar de referencia sin tener que parar el control. Sin embargo, como puede verse fácilmente en la gráfica primera, alrededor de las 12:00 hubo un fallo en el código general de la Raspberry, lo cual ocasionó que la depuradora dejase de bombear agua y, por consiguiente, el valor del pH se vio distorsionado hasta las 12:30 aproximadamente, que se volvió a estabilizar. En ese periodo no es factible sacar conclusiones de la acción del control, solo de la respuesta del ORP, que no sufrió distorsión.

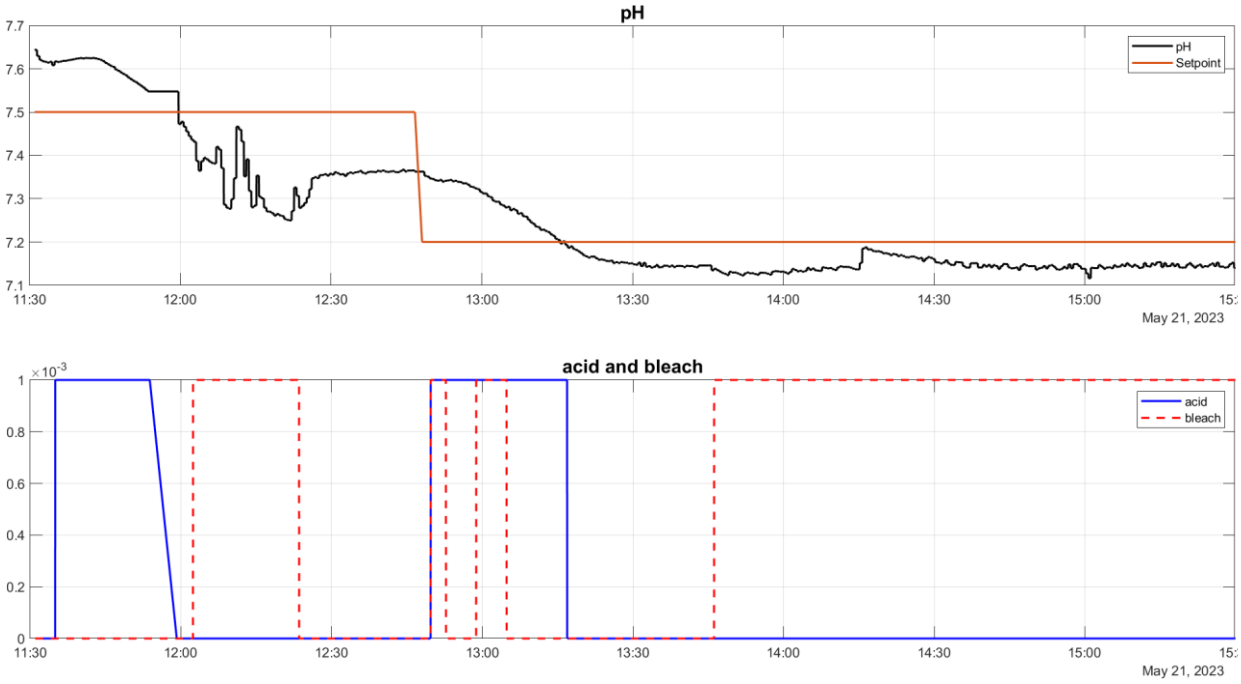

Figura 6-3. Resultado pH segunda prueba.

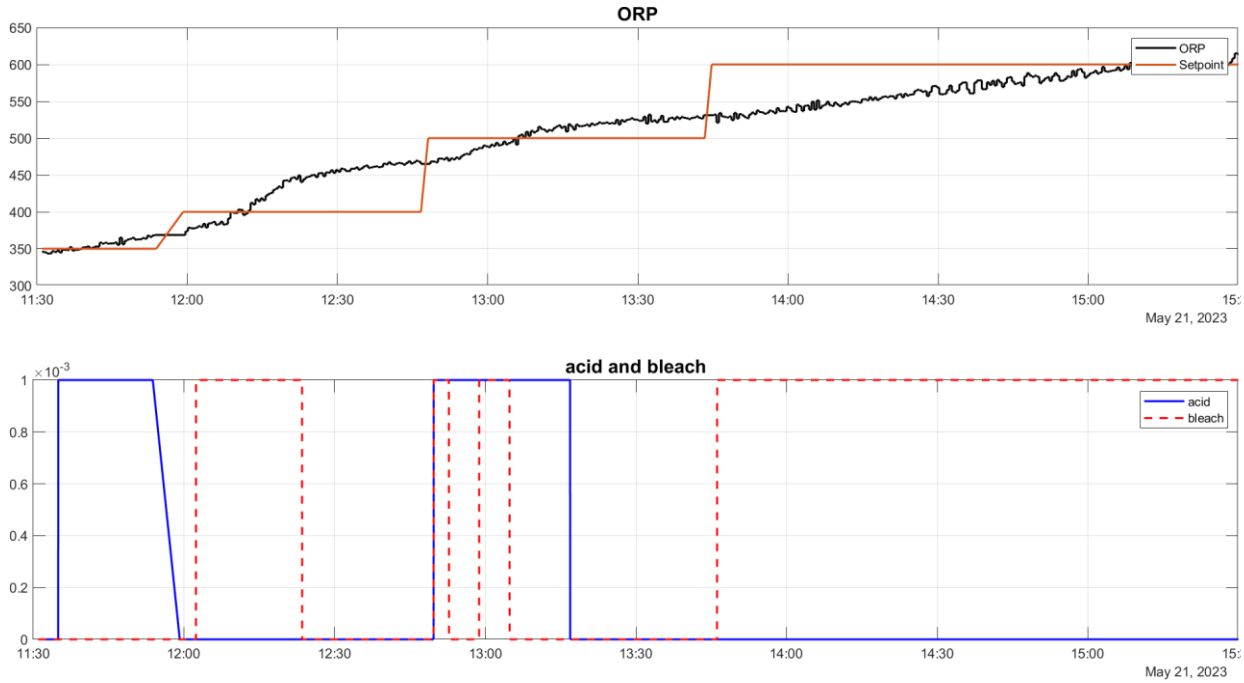

Figura 6-4. Resultado ORP segunda prueba.

Las conclusiones extraídas son las siguientes:

- El seguimiento del ORP es satisfactorio. Los escalones empiezan inmediatamente después de detectar el error respecto al setpoint. En los dos primeros escalones, se aprecia que tanto el ácido como la lejía lo suben, en mayor medida la lejía, como era de esperar puesto que la ganancia de esa respuesta se había identificado mayor. El segundo y tercer escalón de lejía son de menor amplitud, puesto que hay menor error que cubrir. El último, a pesar de ser de gran amplitud, demuestra que conforme aumenta el nivel, más lejía es necesaria para seguir con la tendencia.
- El seguimiento del pH solo es satisfactorio en la respuesta frente al ácido. El segundo escalón empieza justo después de detectar el error respecto a la referencia y cesa cuando se alcanza este. Esto provoca que el pH no quede exactamente en el valor del setpoint, cosa que ocurre también en las simulaciones, debido principalmente al tiempo de muestreo escogido y a la naturaleza ON/OFF de los actuadores, que impide un ajuste más exacto.
- En cuanto a la respuesta frente a la lejía del pH, como puede verse en la segunda mitad de la prueba, vuelve a demostrarse que la ganancia es prácticamente nula. La lejía, al ser una solución básica, debe contribuir a aumentar el pH, pero se trata de una subida tan pequeña en los rangos de tiempo de las pruebas que la mejor solución pasa por considerarla nula. Al no hacerlo así en esta prueba, el primer escalón de lejía no cesa al alcanzar el setpoint de ORP y eso es debido a que el control considera que el error del pH frente a su referencia es suficientemente grande como para utilizar la lejía para subirlo. Esto supone un problema de gasto de dicho líquido, pues ante valores de pH considerablemente por debajo del setpoint, el control va a intentar subirlo vertiendo lejía sin efecto real en la piscina (hasta que el nivel de ORP suba tanto como para considerar el error en el pH menos importante que el del ORP o hasta que el código general detecte la llegada al límite de litros que se pueden verter en un día).

6.3 Tercera prueba

Llegado a este punto, se decide, sirviéndose de la prueba anterior, volver a identificar los modelos, principalmente para corregir la G_{12}:

- Acción de los líquidos sobre el pH:

```
From input "u1" to output "y1":
                1+Tz*s
G11(s) = Kp * ----------
              s(1+Tp1*s)

      Kp = -0.22338
     Tp1 = 264.63
      Tz = 27.863

From input "u2" to output "y1":
             Kp
G12(s) = ------------
          s(1+Tp1*s)

      Kp = 0
     Tp1 = 1142.3
```

Figura 6-5. Nueva identificación de los modelos de las respuestas del pH.

- Acción de los líquidos sobre el ORP:

```
From input "u1" to output "y2":
                1+Tz*s
G21(s) = Kp * ----------
              s(1+Tp1*s)

      Kp = 6.6656
     Tp1 = 1318.2
      Tz = 243.44

From input "u2" to output "y2":
             Kp
G22(s) = ------------
          s(1+Tp1*s)

      Kp = 15
     Tp1 = 1300
```

Figura 6-6. Nueva identificación de los modelos de las respuestas del ORP.

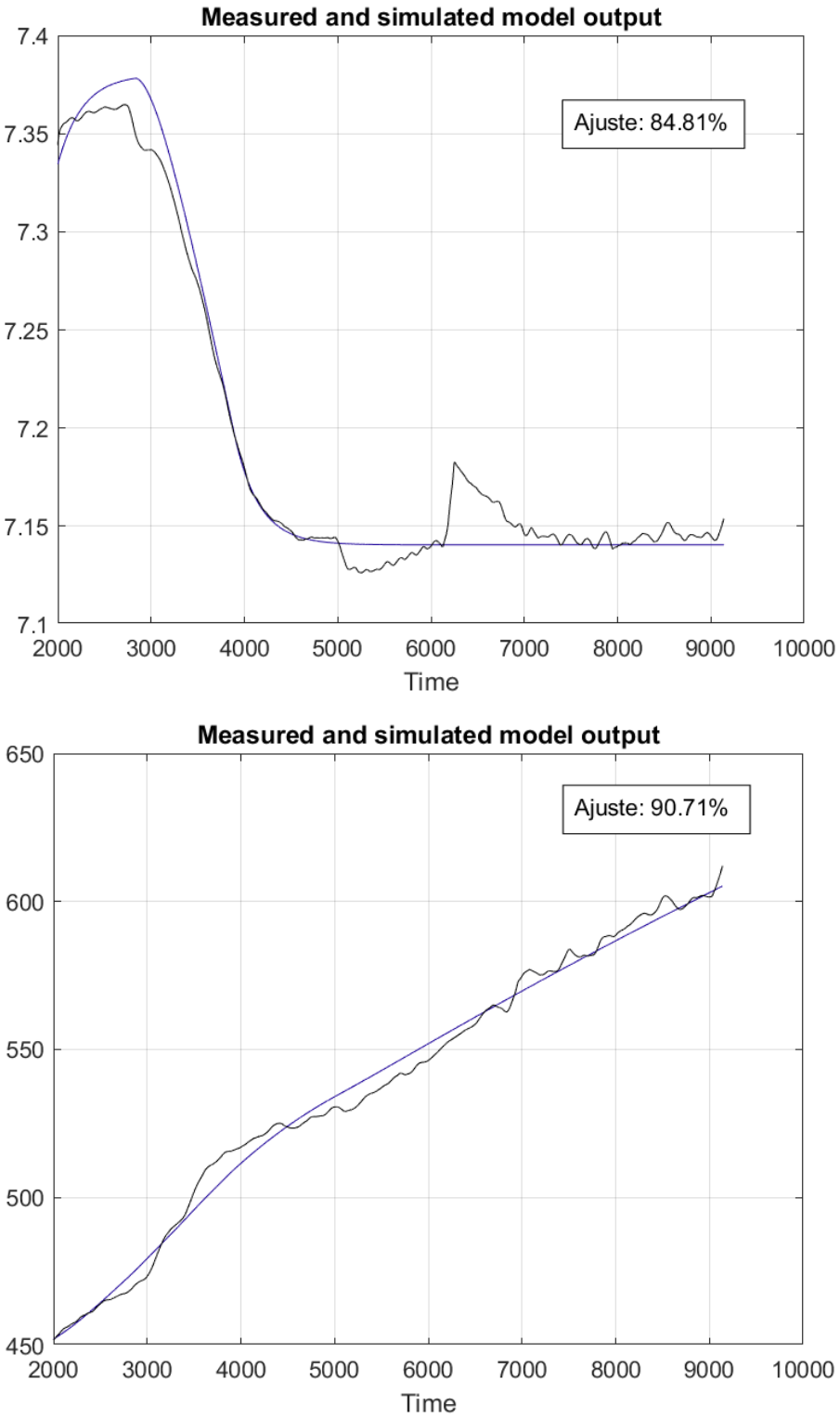

Figura 6-7. Nuevos ajustes de los modelos identificados.

El resultado es que, para el mejor ajuste, las funciones G_{11}, G_{21} y G_{22} quedan igual y la G_{12} se identifica con ganancia nula, como se esperaba.

En esta última prueba se dejó un régimen permanente sin controlar al principio (véase que no hay setpoints hasta unos minutos después de las 13:00) para permitir que la depuradora estuviese moviendo agua un tiempo sin la acción de los líquidos, permitiendo que los niveles se estabilizasen antes de empezar el control y demostrando que la variación en el valor de los mismos es prácticamente nula cuando no hay actuación por parte de los difusores, más allá del ruido que presenten los sensores.

Los resultados son los siguientes:

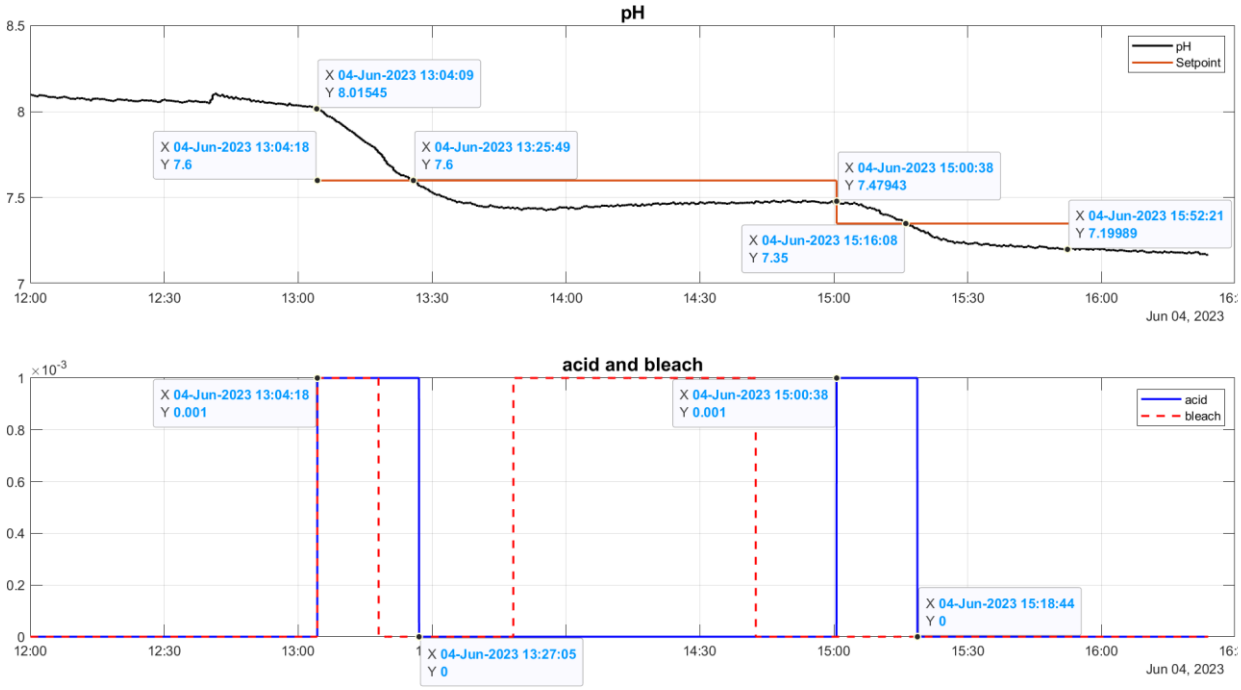

Figura 6-8. Resultado pH tercera prueba.

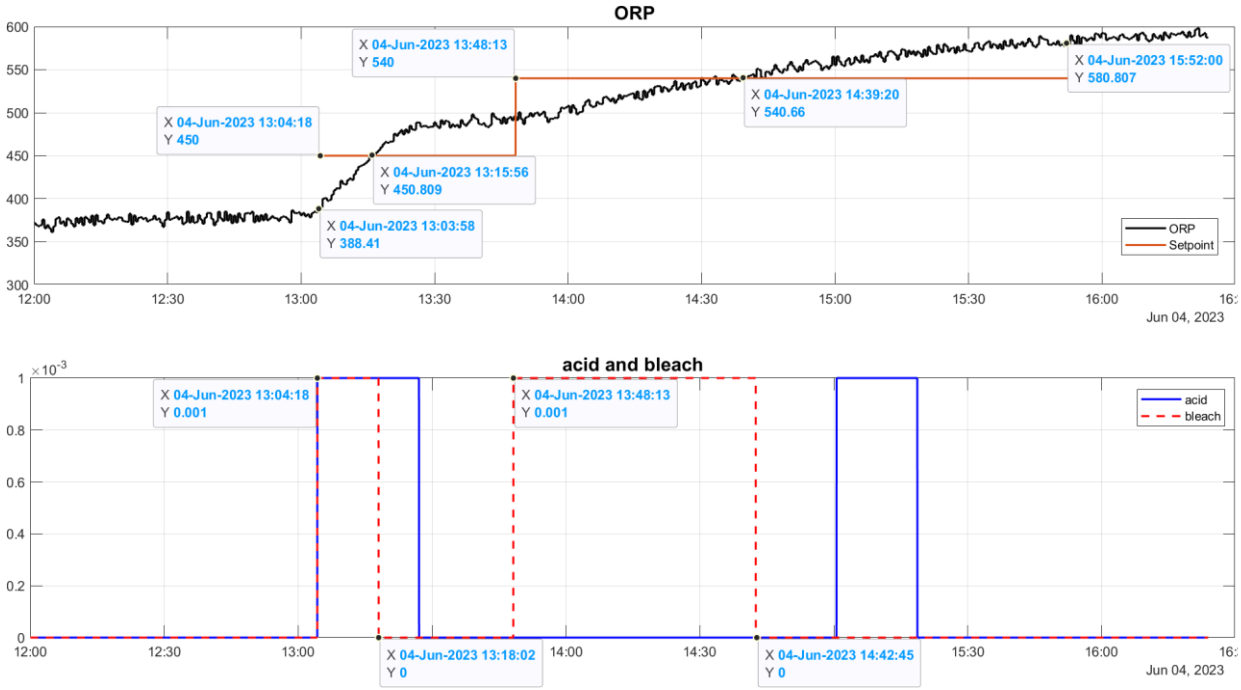

Figura 6-9. Resultado ORP tercera prueba.

Se concluye que:

- En cuanto al pH, esta vez el seguimiento es adecuado. Como en la prueba anterior, al detectar el cambio en la referencia comienza a verter ácido y el escalón se mantiene hasta que se produce un nuevo cálculo en el control tras haber llevado el nivel al setpoint, como puede verse en el primer escalón. Se tiene un cierto error en régimen permanente de 0.15 aproximadamente, por motivos explicados antes en esta memoria. Tras eso, el escalón de lejía parece subir muy ligeramente el nivel, aunque esta vez es por la acción del control para el ORP y no porque el control decida echar lejía (sin éxito, como se vio en las pruebas anteriores) para solventar el error en el pH. En cualquier caso, esta subida, de ser prolongada en el tiempo y llevar el pH por encima del setpoint de nuevo, se solucionaría con un escalón de solo unos pocos ciclos de ácido. El segundo escalón es más corto, debido al menor error respecto a la referencia nueva, y deja el nivel otra vez 0.15 aproximadamente por debajo de esta.
- Nuevamente, el control del ORP es satisfactorio, con escalones del ancho exacto necesario. El error en régimen permanente al final es debido a que, gracias a la ponderación menor para el ORP, el término del error es pequeño comparado con el del pH y el control permite utilizar el ácido, sacrificando un poco el ajuste en el ORP.

Resulta de interés comprobar cómo de acertada es la simulación en Matlab/Simulink para los mismos valores iniciales y referencias dadas:

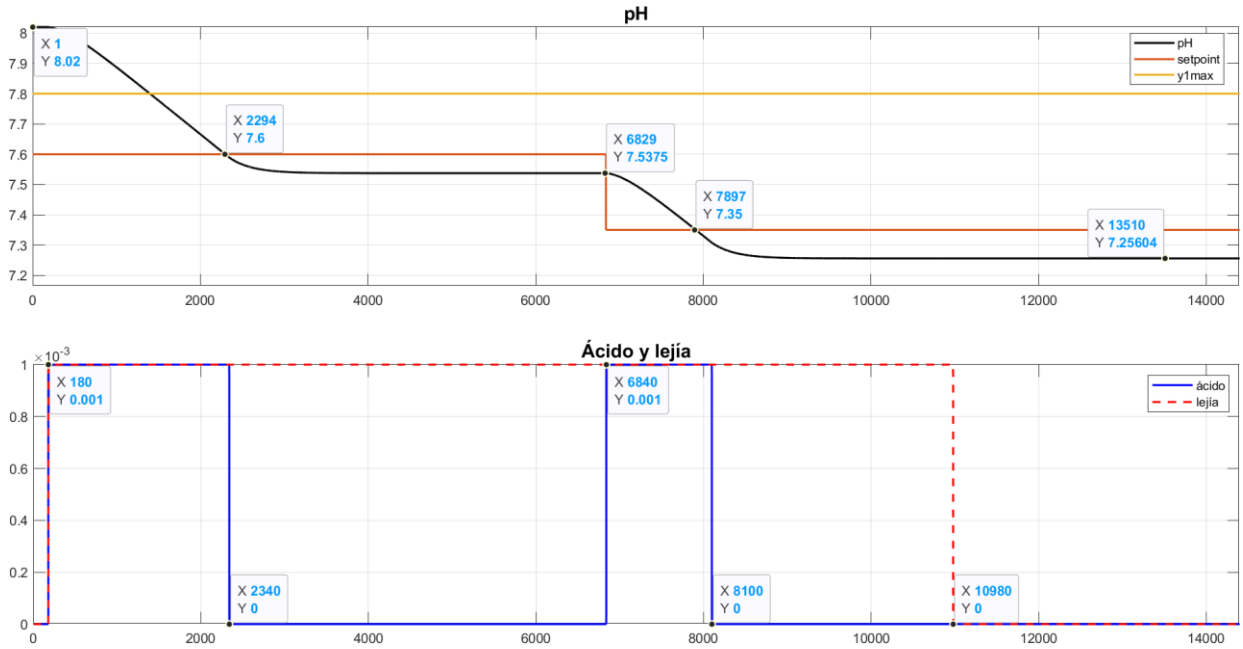

Figura 6-10. Resultado pH tercera prueba simulada.

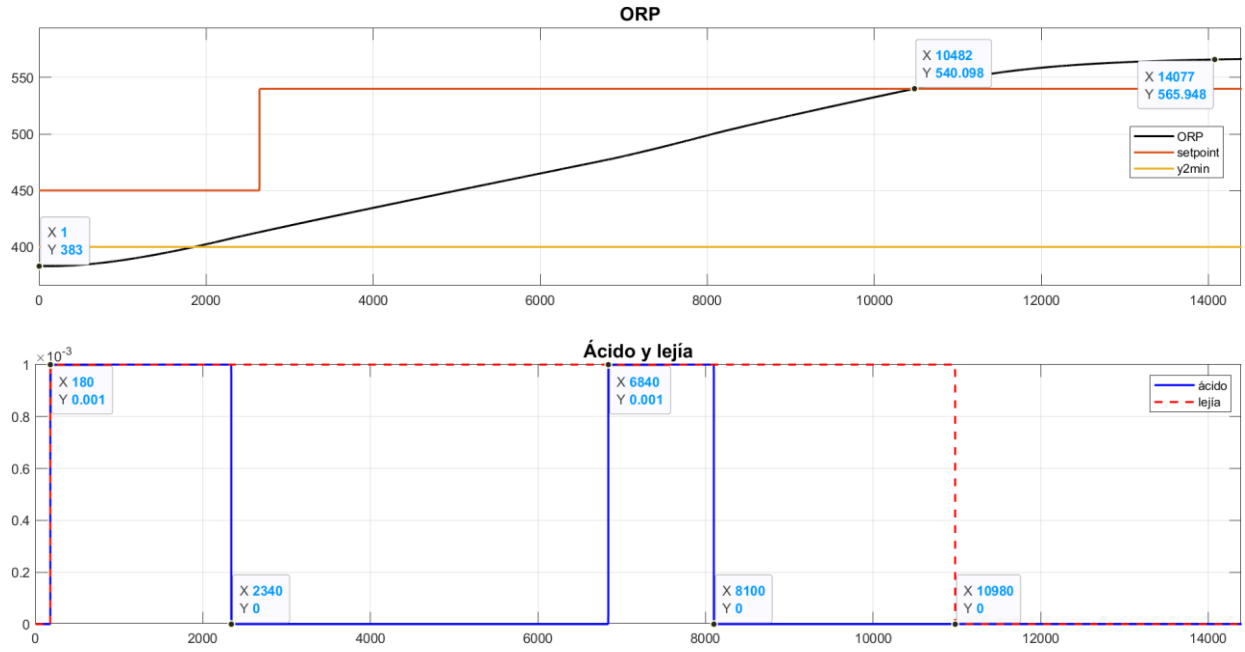

Figura 6-11. Resultado ORP tercera prueba simulada.

Comparando estos resultados con los reales, se comprueba que:

- En el caso del pH, la simulación muestra en gran medida lo que ocurrirá en el caso real. El setpoint se alcanza algo más rápido en la prueba real (23 minutos frente a 38), aunque el error en régimen permanente es ligeramente mayor en esta.
- En el caso del ORP, no se observa a priori el mismo grado de similitud entre el caso real y el simulado, con un escalón mucho mayor en este último, debido una vez más a que la cantidad de lejía que es necesario verter para aumentar el nivel de ORP es mayor cuanto mayor es este último. Esto es claramente visible en que las subidas de ORP en los primeros escalones de lejía en las tres pruebas son más acentuadas que en los escalones posteriores. Esta disminución de la ganancia parece darse a partir de los 450-500 mV aproximadamente, valores sobre los cuales se realizó la identificación de los modelos. Si se realiza una nueva simulación solo con la parte de la prueba correspondiente al último setpoint de ORP, sí que se obtiene un resultado muy similar al real (se alcanza el setpoint en 55 minutos aproximadamente en ambos casos y se da un error en régimen permanente parecido):

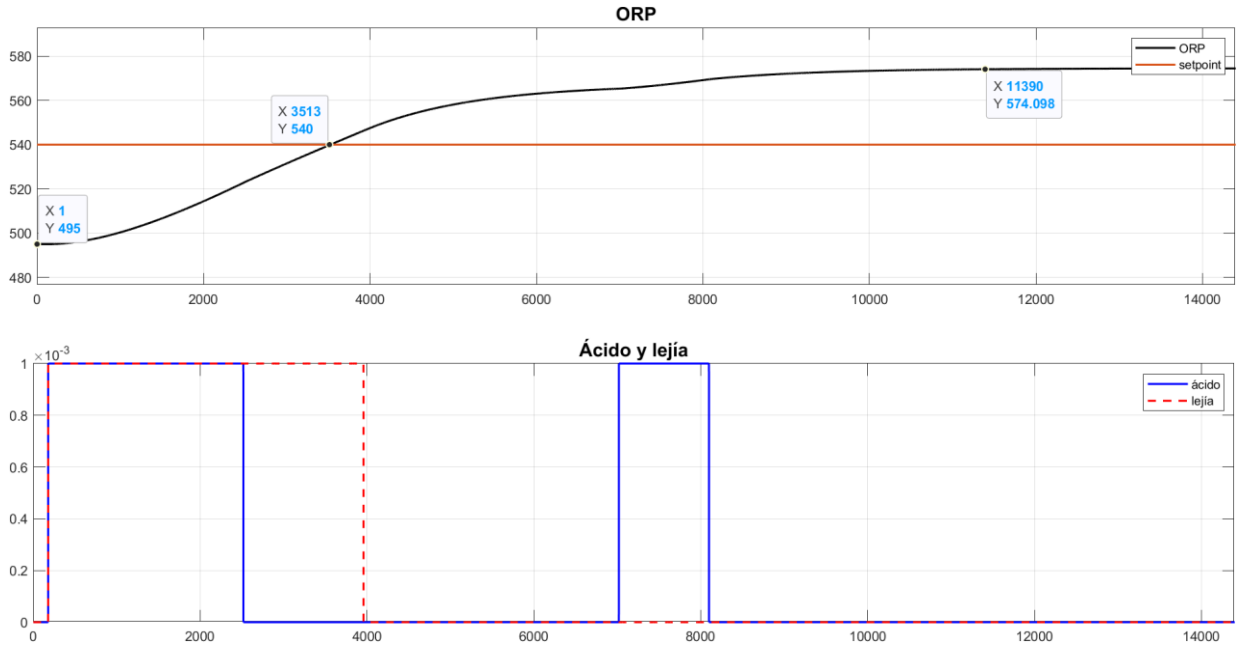

Figura 6-12. Resultado ORP tercera prueba simulada (último setpoint).

7 CONCLUSIONES Y MEJORAS PROPUESTAS

Una vez realizadas las pruebas, se pueden extraer las siguientes conclusiones:

- El control, en líneas generales, es satisfactorio. Las dinámicas principales fueron identificadas bien en un primer momento, y durante el control se obtienen valores similares a los conseguidos previamente con las simulaciones (algo peores, como era de esperar en la aplicación real), tanto en cuanto a la cantidad de líquido vertida como al seguimiento de la referencia. En cuanto a las dinámicas acopladas, la del ORP respecto al ácido también se identificó bien desde un primer momento. No fue así la identificación de respuesta del pH frente a la lejía, por la necesidad final de volver a identificarla, cosa que ha sido solventada como se ha explicado en la tercera prueba, evitando un gasto excesivo e innecesario.
- El control predictivo, respecto al control proporcional anterior, supone una mejora tanto en seguimiento de la referencia como en gasto de los líquidos. Esta mejoría es notada por parte del propietario (no se cuenta con datos de calidad anteriores para hacer una comprobación directa, más allá de las pruebas realizadas para identificar los modelos) tras observar el funcionamiento en la prueba final, principalmente porque se ha acabado con el rizado (el control P trataba de corregir constantemente en torno al setpoint, produciendo un gasto muy elevado) y porque la identificación de las dinámicas acopladas ha permitido tener en cuenta el efecto del ácido sobre el ORP, cosa con la que no se contaba antes. Esto último se traduce en un mejor ajuste del ORP y en un uso menor de la lejía, puesto que, mientras que antes se calculaba la cantidad de lejía necesaria para llevar el ORP a la referencia sin aprovechar el efecto del ácido, ahora sí se tiene en cuenta.

A pesar de esto, del control se pueden extraer también algunos aspectos negativos, que se exponen a continuación, acompañados de posibles mejoras que se podrían implementar en un futuro:

- A pesar del adecuado funcionamiento del control, el ajuste podría ser mejor. En el pH se observa que siempre se acaba con un error en régimen permanente de 0.1-0.15 aproximadamente. Una posible solución sería disminuir el tiempo de muestro (habría que hacer posible usar tiempos de menos de 1 minuto, aumentando la frecuencia con la que los sensores toman medidas) o añadir a la función de transferencia un delay que tuviera en cuenta la acción prolongada del ácido que sigue presente en la piscina después de haber dejado de verterlo con el difusor.
- A pesar de que en la identificación del sistema se obtienen porcentajes de ajuste elevados, de en torno al 90% para ambos niveles, el sistema real es muy susceptible a perturbaciones. La solución pasa por identificar estas perturbaciones e incluirlas en el modelo de predicción del MPC.
- En un futuro, el propietario podría alternar días con el control nuevo y el antiguo, para obtener una visión más precisa de la mejora respecto al gasto de los líquidos. Según el caso, se puede tomar alguna medida al respecto, como modificar algún parámetro que reduzca dicho gasto aunque en consecuencia se empeore el ajuste.
- En general, el control predictivo cuenta con un rendimiento superior a otros controles clásicos, pero necesita de un modelo de predicción muy fiel al sistema real para aprovecharse bien. La falta de identificación de las perturbaciones, unida a las limitaciones en los actuadores, hace que el MPC no sea la opción óptima en este caso. Por tanto, teniendo en cuenta esta dificultad en la identificación completa del sistema y la difusión de tipo ON/OFF de los actuadores, un control difuso sería la opción más recomendable, implementando unas reglas de control similares a las que se han usado en este trabajo para los casos en que los niveles sobrepasaban los límites establecidos. Esto daría lugar, posiblemente, a un control con una rendimiento similar, más rápido de diseñar y de mantener que el MPC.

REFERENCIAS

[1] Carlsberg Group, [En línea]. Available: https://www.carlsberg.com/en-sg/pioneers/soeren-pl-soerensen/.

[2] Wikipedia, «pH,» [En línea]. Available: https://es.wikipedia.org/wiki/PH.

[3] Gre, [En línea]. Available: https://blog.grepool.com/en/articles/water-care/how-to-raise-or-lower-the-ph-of-the-pool-water/.

[4] Seko, «Quimipool, Bomba dosificadora Seko PoolMatch,» [En línea]. Available: https://www.quimipool.com/13159-bomba-dosificadora-seko-poolmatch-redox-15-lh-con-modbus.html.

[5] AstralPool, «Piscinayspa, AstralPool pH Pure,» [En línea]. Available: https://www.piscinayspa.com/ficheros/AstralPool-pH-Pure-regulador-automatico-pH_pdf_925.pdf.

[6] P. Duffy, G. Woods, S. O'Hogain, J. Walsh y C. Caplier, «On-Line Realtime Water Quality Monitoring and Control for Swimming Pools,» [En línea]. Available: https://www.researchgate.net/publication/254584975_On-Line_Realtime_Water_Quality_Monitoring_and_Control_for_Swimming_Pools.

[7] G. Simões, C. Dionísio, A. Glória, P. Sebatião y N. Souto, «Smart System for Monitoring and Control of Swimming Pools,» [En línea]. Available: https://www.researchgate.net/publication/334634003_Smart_System_for_Monitoring_and_Control_of_Swimming_Pools.

[8] V. Gutiérrez García, «CONTROL DIGITAL BASADO EN EL MICROPROCESADOR 80C32 DEL NIVEL DE PH Y CLORO DE UNA PISCINA,» [En línea]. Available: https://biblus.us.es.us.debiblio.com/bibing/proyectos/use/abreproy/2876/.

[9] J. Olidén-Semino, «Desarrollo de un controlador predictivo basado en modelo para plataforma industrial,» 2016. [En línea]. Available: https://pirhua.udep.edu.pe/bitstream/handle/11042/5449/MAS_IME_AUT_006.pdf?sequence=2&isAllowed=y.

[10] Wikipedia, «Model Predictive Control,» [En línea]. Available: https://en.wikipedia.org/wiki/Model_predictive_control.

[11] D. F. Sendoya, «¿Qué es el control predictivo y hacia dónde se proyecta?,» 2012. [En línea]. Available: https://oaji.net/articles/2017/5082-1501183303.pdf.

[12] ABB, «Process Improvement Packages. Multivariable control for the process industries - 3dMPC,» [En línea]. Available: https://library.e.abb.com/public/b788e3a9870e9a1ec1256a34007129d6/3BUS025043R0001_-_en_3dMPC_Multivariable_Controller.pdf.

[13] J. A. Yanes Melús, «Proyecto de Fin de Carrera,» 2004. [En línea]. Available: https://biblus.us.es/bibing/proyectos/abreproy/3746.

[14] S. A. Castaño Giraldo, «Control predictivo basado en modelo - DMC,» [En línea]. Available: https://controlautomaticoeducacion.com/control-predictivo/dmc/.

[15] S. A. Castaño Giraldo, «Control GPC,» [En línea]. Available: https://controlautomaticoeducacion.com/control-predictivo/control-gpc/.

[16] Wikipedia, «Gradient descent,» [En línea]. Available: https://en.wikipedia.org/wiki/Gradient_descent.

[17] Z. Vörös, «v923z/micropython-ulab,» [En línea]. Available: https://github.com/v923z/micropython-ulab.

[18] S. A. Castaño Giraldo, «Control automático educación, MIMO MPC,» [En línea]. Available: https://controlautomaticoeducacion.com/control-predictivo/mimo-mpc/.

ANEXO A: CÓDIGO DE MATLAB

init.m

```
%% MIMO GPC

clc;
close all;
clear all;

%% Definicion de la matriz de funciones de transferencia

descomposicion;             %se cargan las funciones de transferencia de los modelos
identificados

Ts=60*3;                     %Tiempo de Muestreo
Pz=c2d(Ps,Ts);              %Processo discreto

[Bp,Ap,dp]=descompMPC(Pz);     %se descompone num, den y atraso de Pz en celdas

%% Parametros del GPC
%Encuentro cual es el retardo minimo en mi función de transferencia
dmin=[100 100]';
for i=1:2
    dmin(i)=min(dp(i,:));
end

N=[5;5];                                %Ventana de Predicción
N1=[dmin(1)+1 dmin(2)+1];                 %Horizonte Inicial
N2=[dmin(1)+N(1) dmin(2)+N(2)];           %Horizonte final
Nu=[3;3];                                 %Horizonte de control
lambda=[1 1];                           %Ponderación de la acción de control
delta=[1 1e-5];                         %Ponderación del seguimiento de referencia

%Matrices en bloque de los parametros de poderación
Ql=blkdiag(lambda(1)*eye(Nu(1)),lambda(2)*eye(Nu(2)));  %(lambda -> Acción de
Control)
Qd=blkdiag(delta(1)*eye(N(1)),delta(2)*eye(N(2)));      %(delta  -> Seguimiento de
Referencia)

%% Calculo de la ecuacion Diofantina
%Se obtiene la matriz B y A (Numerador y denominador sistema MIMO)
%(la matriz A es el minimo común denominador del sistema)
[B,A] = BA_MIMO(Bp,Ap);
%Función que Calcula Los Polinomios de la ecuación Diofántica
[E,En,F] = diophantineMIMO(A,N,dmin);

%% Matriz de la Respuesta Forzada G y Hessiana H

[G] = MatrizG(E,B,N,Nu,dp);

%Cálculo de H (cte)
H=2*(G'*Qd*G + Ql);

%% Matriz de controles pasados
uG = deltaUFree(B,En,N,dp);
```

```
uG11=uG{1,1};
uG12=uG{1,2};
uG21=uG{2,1};
uG22=uG{2,2};

%% Definición de variables globales y longitud de polinomios para cálculos
posteriores

%Acciones de Control
global uant1
uant1=0;
global uant2
uant2=0;

% Determino el tamaño de los polinomios F
lduf1=size(F{1}); %Averiguo la longitud del polinomio F
lduf2=size(F{2}); %Averiguo la longitud del polinomio F
lduf1=lduf1(1,2);
lduf2=lduf2(1,2);
F1=F{1};
F2=F{2};

%Vectores de las salidas
%Se inicializan de longitud lduf para usarlos en el cálculo de las respuestas libres
y1o=7.6;
y2o=650;
global y1;
y1=zeros(lduf1,1);
global y2;
y2=zeros(lduf2,1);

%Referencias
ref1 = 7.4;
ref2 = 700;

%Restricciones en las salidas
y1max = 7.8;
y1min = 7.1;
y2max = 1000;
y2min = 400;

%Cálculo de vectores de incrementos de control pasados
luG11=size(uG{1,1});
luG12=size(uG{1,2});
luG21=size(uG{2,1});
luG22=size(uG{2,2});
luG11=luG11(1,2);
luG12=luG12(1,2);
luG21=luG21(1,2);
luG22=luG22(1,2);

global duf11;
global duf12;
global duf21;
global duf22;
duf11=zeros(luG11,1);
duf12=zeros(luG12,1);
duf21=zeros(luG21,1);
duf22=zeros(luG22,1);
```

calculo_gpc.m

```
function [u1, u2] = calculo_gpc(k,Ts,N,Nu,H,Y1,Y2,F1,uG11, uG12, uG21, uG22, F2,
lduf1, lduf2, refaux1, refaux2,Qd, G, ref1, ref2, y1max, y2max, y2min, y1min)

    global y1;
    global y2;

    global uant1;
    global uant2;

    global duf11;
    global duf12;
    global duf21;
    global duf22;

    %En lugar de almacenar los valores medidos como una pila, se llenan los
    %vectores con el último valor medido, para evitar introducir error en
    %el control como consecuencia del error en la medida de los sensores

    y1(1:end)=Y1*ones(lduf1,1);
    y2(1:end)=Y2*ones(lduf2,1);

    resto=rem(k,Ts);

    if resto==0 %cálculo de control cada vez que el tiempo sea múltiplo de Ts

        %Cálculo de la respuesta libre
        [f1,f2] =
calcula_free(F1,F2,y1,y2,uG11,uG12,uG21,uG22,duf11,duf12,duf21,duf22,lduf1,lduf2);
        f=[f1;f2];

        %Cálculo de vector de referencias r
        r1=ref1*ones(N(1),1);
        r2=ref2*ones(N(2),1);
        if refaux1>=7.1 && refaux1<=y1max %si se pasa referencia manual para pH
            r1=refaux1*ones(N(1),1);
        end
        if refaux2>=y2min && refaux2<=y2max %si se pasa referencia manual para ORP
            r2=refaux2*ones(N(1),1);
        end
        r=[r1;r2];

        %Límites
        lb=[(0-uant1)*ones(Nu(1),1);(0-uant2)*ones(Nu(2),1)];
        ub=[(0.001-uant1)*ones(Nu(1),1);(0.001-uant2)*ones(Nu(2),1)];

        %Cálculo del incremento de control. Se aplica algoritmo del
        %gradiente descendiente.
        fo=2*(f-r)'*Qd*G;
        fo=fo';

        deltau=quad_grad(H,fo,lb,ub, zeros((Nu(1)+Nu(2)),1), 100, 1e-5);

        deltaU1=deltau(1);
        deltaU2=deltau(1+Nu(1));

        %Calculo de la ley de control
        u1 = uant1 + deltaU1;
        u2 = uant2 + deltaU2;
```

```
        %Restricciones de u1 y u2
        if ((1e-6)<u1) && (u1<=0.001)
            if ((u1-uant1)>=0)
                u1=0.001;
            else
                u1=0;
            end
        end
        if ((1e-6)<u2) && (u2<=0.001)
            if ((u2-uant2)>=0)
                u2=0.001;
            else
                u2=0;
            end
        end

        %Comprobación de límites
        if Y1>=y1max
            u1=0.001;
        end
        if Y2>=y2max
            u1=0;
            u2=0;
        end
        %No se comprueba el límite inferior para el pH, para evitar
        %problemas de gasto de lejía (G12 se consideró finalmente nula)
        if Y2<=y2min
            u2=0.001;
        end

        %Actualizo vectores de incrementos de control pasados
        aux_2=duf11(1:end-1);
        duf11=[deltaU1; aux_2];

        aux_2=duf12(1:end-1);
        duf12=[deltaU2; aux_2];

        aux_2=duf21(1:end-1);
        duf21=[deltaU1; aux_2];

        aux_2=duf22(1:end-1);
        duf22=[deltaU2; aux_2];

    else %resto del tiempo se mantienen las últimas acciones de control

        u1=uant1;
        u2=uant2;

    end

    %Actualizo entradas anteriores
    uant1 = u1;
    uant2 = u2;

end
```

calcula_free.m

```
function [free1,free2] =
calcula_free(F1,F2,y1,y2,uG11,uG12,uG21,uG22,duf11,duf12,duf21,duf22,lduf1,lduf2)

    rows_duf1=size(F1);
    rows_duf2=size(F2);
    rows_duf1=rows_duf1(1,1);
    rows_duf2=rows_duf2(1,1);

    free1=zeros(rows_duf1,1);
    free2=zeros(rows_duf2,1);

    %F(z^-1)*y(k): retrasos en F hace que aparezcan y(k-i)
    for i=1:lduf1
        free1=free1+y1(lduf1+1-i)*F1(:,i);
    end
    for i=1:lduf2
        free2=free2+y2(lduf2+1-i)*F2(:,i);
    end

    %uG(z^-1)*?u(k-1): sumo vectores de controles pasados
    free1=free1+uG11*duf11+uG12*duf12;
    free2=free2+uG21*duf21+uG22*duf22;

end
```

quad_grad.m

```
% Método del gradiente descendiente
function [x,i] = quad_grad(hessian, objective, lower, upper, x0, max_iter, tol)

% Inicialización
    x = x0;
    x_old = x0;
    x_new = x0;
    f_old = 0;
    f_new = 0;
    f = 0;

    % iteración
    for i = 1:max_iter
        % cálculo del gradiente
        grad = hessian*x + objective;
        % cálculo del step
        step = (grad'*grad)/(grad'*hessian*grad);
        % nueva x
        x_new = x - step * grad;
        % se aplican límites
        x_new = max(x_new, lower);
        x_new = min(x_new, upper);

        % nueva f
        f_new = 0.5*(x_new'*hessian*x_new) + objective'*x_new;
        % se comprueba convergencia
        if abs(f_new - f_old) < tol
            break
        end
        % se actualizan x y f
```

```
        x_old = x;
        x = x_new;
        f_old = f;
        f = f_new;
    end

end
```

ANEXO B: CÓDIGO DE PYTHON

```
from ulab import numpy as np


class PredictiveAlgo:
    """
    This class implements the predictive control chemical injection algorithm.
    """

    def __init__(self, lduf1=3, lduf2=5, y1min=7.1, y1max=7.8, y2min=400, y2max=1000,
ref1=7.4, ref2=700):
        """
        This is the constructor of the Predictive Algorithm class.
        """

        # Save attributes
        self.lduf1 = lduf1
        self.lduf2 = lduf2
        self.y1min = y1min
        self.y1max = y1max
        self.y2min = y2min
        self.y2max = y2max
        self.ref1 = ref1
        self.ref2 = ref2

        self.nu = np.array([[3], [3]])

        self.qd = np.array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                            [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                            [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                            [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                            [0, 0, 0, 0, 0, 1e-5, 0, 0, 0, 0],
                            [0, 0, 0, 0, 0, 0, 1e-5, 0, 0, 0],
                            [0, 0, 0, 0, 0, 0, 0, 1e-5, 0, 0],
                            [0, 0, 0, 0, 0, 0, 0, 0, 1e-5, 0],
                            [0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-5]])

        self.g = np.array([
                            [-14.1086358374596, 0, 0, 0, 0, 0],
                            [-41.0970298261121, -14.1086358374596, 0, 0, 0, 0],
                            [-74.6092559506830, -41.0970298261121, -14.1086358374596,
0, 0, 0],
                            [-111.425921897651, -74.6092559506830, -41.0970298261121,
0, 0, 0],
```

```
                            [-149.916346812101, -111.425921897651, -74.6092559506830,
0, 0, 0],
                            [285.425564119266, 0, 0, 178.586409744024, 0, 0],
                            [687.560305500371, 285.425564119266, 0, 683.199819135613,
178.586409744024, 0],
                            [1191.50779621268, 687.560305500371, 285.425564119266,
1471.68387393365, 683.199819135613, 178.586409744024],
                            [1784.27294608064, 1191.50779621268, 687.560305500371,
2507.33317397391, 1471.68387393365, 683.199819135613],
                            [2454.51932133254, 1784.27294608064, 1191.50779621268,
3758.18844703111, 2507.33317397391, 1471.68387393365]
                        ])

        self.h = np.array([
                            [84916.0591589225, 57478.4480107708, 33723.9419157565,
319.450998360782, 194.340205585859, 100.881571140443],
                            [57478.4480107708, 39845.7437739746, 23981.7230602246,
218.000302090956, 134.960075231850, 71.2542511751192],
                            [33723.9419157565, 23981.7230602246, 14950.5990335696,
132.438195945709, 83.8876226447470, 45.4847402492065],
                            [319.450998360782, 218.000302090956, 132.438195945709,
463.504171442335, 284.809934506726, 150.133952859902],
                            [194.340205585859, 134.960075231850, 83.8876226447470,
284.809934506726, 181.024563374372, 96.3493231609948],
                            [100.881571140443, 71.2542511751192, 45.4847402492065,
150.133952859902, 96.3493231609948, 55.2901704681711]
                        ])

        self.f1 = np.array([
                            [2.50651821713858, -2.01303643427717, 0.506518217138584],
                            [4.26959713857042, -4.53919427714084, 1.26959713857042],
                            [6.16262873052869, -7.32525746105738, 2.16262873052869],
                            [8.12148371747441, -10.2429674349488, 3.12148371747441],
                            [10.1136794530952, -13.2273589061904, 4.11367945309518]
                        ])

        self.f2 = np.array([
                            [4.74305957896983, -8.98874221689593, 8.50786917686881, -
4.02175001892916, 0.759563479986443],
                            [13.5078719527615, -34.1262706978699, 36.3315803770406, -
18.3158364715176, 3.60265483958536],
                            [29.9423707591735, -85.0871985051712, 96.6073709604728, -
50.7226294421259, 10.2600862276508],
                            [56.9312499411926, -172.536881156460, 204.023143822224, -
110.160643939839, 22.7431313328821],
                            [97.4914292198390, -307.717185984827, 374.202982635448, -
206.220124195770, 43.2428983253102]
                        ])
```

```
        self.uG11 = np.array([
                            [-5.73347708054537],
                            [-14.3710647499335],
                            [-24.4796373371556],
                            [-35.3332905823967],
                            [-46.5643407541620]
                        ])

        self.uG12 = np.array([
                        [0, 0],
                        [0, 0],
                        [0, 0],
                        [0, 0],
                        [0, 0]
                    ])

        self.uG21 = np.array([
                        [-666.230150478380, 495.985121069906, -115.180534710792],
                        [-2663.98417595509, 2237.30644480634, -546.308138470887],
                        [-6762.04511892466, 6153.39536741632, -1555.84391432398],
                        [-13795.1148091473, 13295.1264717847, -3448.77827455037],
                        [-24634.1887434583, 24788.2746201931, -6557.37180998028]
                    ])

        self.uG22 = np.array([
                        [-163.846162274609, -163.506371937991, 148.766124468576],
                        [-940.638481392020, -626.754339174516, 705.606591686899],
                        [-2839.96731913132, -1503.01654391208, 2009.51376023011],
                        [-6408.95908220612, -2886.25464982453, 4454.41045524346],
                        [-12214.2214661855, -4854.21167253590, 8469.44141490308]
                    ])

        n = np.array([[5], [5]])
        self.n = n
        self.r1 = np.ones((int(n[0][0]), 1)) * ref1
        self.r2 = np.ones((int(n[1][0]), 1)) * ref2
        self.y1 = np.zeros((lduf1, 1))
        self.y2 = np.zeros((lduf2, 1))
        self.uant1 = False
        self.uant2 = False
        self.duf11 = np.zeros((1, 1))
        self.duf12 = np.zeros((1, 2))
        self.duf21 = np.zeros((1, 3))
        self.duf22 = np.zeros((1, 3))
        self.last_output = None

    def calculate_free_response(self, y1, y2):
        """
        This functions calculates the free response of the given inputs.
```

```
        :param y1: PH input.
        :param y2: ORP input.
        :return: Free response of each input.
        :rtype: tuple
        """

        rows_duf1 = np.size(self.f1, axis=0)
        rows_duf2 = np.size(self.f2, axis=0)

        free1 = np.zeros((rows_duf1, 1))
        free2 = np.zeros((rows_duf2, 1))

        for i in range(self.lduf1):
            free1 = free1 + (y1[self.lduf1 - i - 1] * (self.f1[:,
i])).reshape((rows_duf1, 1))

        for i in range(self.lduf2):
            free2 = free2 + (y2[self.lduf2 - i - 1] * (self.f2[:,
i])).reshape((rows_duf2, 1))

        free1 = free1 + np.dot(self.uG11, self.duf11.T) + np.dot(self.uG12,
self.duf12.T)
        free2 = free2 + np.dot(self.uG21, self.duf21.T) + np.dot(self.uG22,
self.duf22.T)

        return free1, free2

    def calculate_gpc(self, ph, orp, new_ph_ref=None, new_orp_ref=None):
        """
        This function implements the pool's predictive chemical injection algorithm.

        :param ph: Current ph value
        :param orp: Current orp value
        :param new_ph_ref: Indicates a new ph reference.
        :param new_orp_ref: Indicates a new orp reference.
        :return: Tuple with two booleans that indicates if the ph and orp needs to be
turned on or off.
        :rtype: tuple
        """
        for i in range(len(self.y1)):
            self.y1[i] = ph

        for i in range(len(self.y2)):
            self.y2[i] = orp

        if self.uant1:
            self.uant1 = 0.001
        else:
```

```
            self.uant1 = 0

        if self.uant2:
            self.uant2 = 0.001
        else:
            self.uant2 = 0

        # Calculate free response
        f1, f2 = self.calculate_free_response(self.y1, self.y2)
        f = np.zeros(((int(self.n[0][0]) + int(self.n[1][0])), 1))
        f[:int(self.n[0][0]), 0] = f1[:, 0]
        f[int(self.n[0][0]):, 0] = f2[:, 0]

        # Change references if needed
        if new_ph_ref is not None and new_ph_ref != self.ref1:
            if self.y1min <= new_ph_ref <= self.y1max:
                self.r1 = np.ones(int(self.n[0][0])) * new_ph_ref
                self.ref1 = new_ph_ref
        if new_orp_ref is not None and new_orp_ref != self.ref2:
            if self.y2min <= new_orp_ref <= self.y2max:
                self.r2 = np.ones(int(self.n[1][0])) * new_orp_ref
                self.ref2 = new_orp_ref

        # Reference vector
        r = np.zeros(((int(self.n[0][0] + self.n[1][0])), 1))
        r[:int(self.n[0][0]), 0] = self.r1
        r[int(self.n[0][0]):, 0] = self.r2

        # Limits
        lb = np.zeros(((int(self.nu[0][0] + self.nu[1][0])), 1))
        ub = np.zeros(((int(self.nu[0][0] + self.nu[1][0])), 1))
        lb[:int(self.nu[0][0]), 0] = (0 - self.uant1) * np.ones(int(self.nu[0][0]))
        lb[int(self.nu[0][0]):, 0] = (0 - self.uant2) * np.ones(int(self.nu[1][0]))
        ub[:int(self.nu[0][0]), 0] = (0.001 - self.uant1) *
np.ones(int(self.nu[0][0]))
        ub[int(self.nu[0][0]):, 0] = (0.001 - self.uant2) *
np.ones(int(self.nu[1][0]))

        # Objective function
        fo = (2 * np.dot((f - r).T, np.dot(self.qd, self.g))).T

        # Resolve quadratic problem
        deltau = self.quadratic_problem_resolver(self.h, fo, lb, ub,
                                                 np.zeros((int(self.nu[0][0] +
self.nu[1][0]), 1)), 100, 1e-5)
        delta_u1 = deltau[0][0]
        delta_u2 = deltau[int(self.nu[0][0])][0]

        # Calculate control law
```

```
u1 = self.uant1 + delta_u1
u2 = self.uant2 + delta_u2

# u1 and u2 constraints
if (1e-6 < u1) and (u1 <= 0.001):
    if (u1 - self.uant1) >= 0:
        u1 = 0.001
    else:
        u1 = 0
if (1e-6 < u2) and (u2 <= 0.001):
    if (u2 - self.uant2) >= 0:
        u2 = 0.001
    else:
        u2 = 0

# Check limits
if ph >= self.y1max:
    u1 = 0.001
if orp >= self.y2max:
    u1 = 0
    u2 = 0
if orp <= self.y2min:
    u2 = 0.001

# Update Past Controls Vectors
self.duf11[0][0] = delta_u1
self.duf12 = np.roll(self.duf12, 1)
self.duf12[0][0] = delta_u2
self.duf21 = np.roll(self.duf21, 1)
self.duf21[0][0] = delta_u1
self.duf22 = np.roll(self.duf22, 1)
self.duf22[0][0] = delta_u2

if u1 > 0.0005:
    u1 = True
else:
    u1 = False

if u2 > 0.0005:
    u2 = True
else:
    u2 = False

self.uant1 = u1
self.uant2 = u2

# Return if ph pump and/or orp pump needs to be turned on
self.last_output = u1, u2
return u1, u2
```

```
    @staticmethod
    def quadratic_problem_resolver(hessian, objective, lower, upper, x0,
max_iter=100, tol=1e-6):
        """
        This functions resolves a quadratic problem.

        :param hessian: Hessian matrix.
        :param objective: Objective to resolve.
        :param lower: Lower matrix.
        :param upper: Upper matrix.
        :param x0: Initial condition.
        :param max_iter: Number of max iterations allowed for the algorithm.
        :param tol: Minimum tolerance for the algorithm.
        :return: Quadratic problem result.
        """
        # Initialize variables
        x = x0
        f_old = 0

        for i in range(max_iter):
            # Compute gradient
            grad = np.dot(hessian, x) + objective
            # Compute step size
            step = (np.dot(grad.T, grad)) / (np.dot(grad.T, np.dot(hessian, grad)))
            # Compute new x
            x_new = x - step * grad

            # Apply lower and upper limits
            x_new = np.maximum(x_new, lower)
            x_new = np.minimum(x_new, upper)

            # Compute new f
            f_new = 0.5 * np.dot(x_new.T, (np.dot(hessian, x_new))) +
np.dot(objective.T, x_new)

            # Check convergence
            if abs(f_new - f_old)[0][0] < tol:
                break

            # Update x and f
            x = x_new
            f_old = f_new

        return x
```