Proyecto Fin de Máster

Máster Universitario en Ingeniería Industrial

Control del sistema TCLab con técnicas de control predictivo.

Autor: Mario Fernández Rangel

Tutores: Pablo Krupa García, Daniel Limón Marruedo.

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

Sevilla, 2021

Trabajo Fin de Máster
Máster Universitario de Ingeniería Industrial

Control del sistema TCLab con técnicas de control predictivo.

Autor:
Mario Fernández Rangel

Tutores:
Pablo Krupa García
Daniel Limón Marruedo. Profesor titular

Dpto. de Ingeniería de Sistemas y Automática.
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Máster: Control del sistema TCLab con técnicas de control predictivo.

Autor: Mario Fernández Rangel

Tutor: Pablo Krupa García
Daniel Limón Marruedo

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

Agradecimientos

Mis agradecimientos van dirigidos a todas las personas que me han apoyado a lo largo de mi trayectoria académica, a mis dos tutores, Pablo y Daniel, por el apoyo que me han brindado para realizar este proyecto, sin su ayuda no hubiera sido posible.

Los amigos que han estado conmigo cuando el camino se hacía más cuesta arriba, por enseñarme a tener paciencia y poder continuar con todo hacia delante, con pequeños pasito, poco a poco. También por ayudarme a despejar la cabeza cuando era necesario. A Manu, que sin ti este maratón no hubiera sido fácil. A Curro que es muy grande y ha sido como un hermano durante este tiempo.

A mi familia, con especial dedicación a mis padres, que siempre me han estado apoyando en los momentos difíciles.

Ad Astra per Aspera.

Resumen

El objetivo de este trabajo es realizar el modelado, identificación y definición de conceptos, para diseñar el control del equipo TCLab (*Temperature Control Lab*), un equipo para enseñar los principios de control de forma práctica, montado en un Arduino compuesto de dos sensores de temperaturas y dos calentadores, de forma que se produce transferencia de calor entre los distintos dispositivos del TCLab. Se describe en detalle las características de este equipo. El objetivo del control es poder controlar las dos temperaturas según la referencia de temperatura deseada por el usuario.

Para controlar el sistema se van a obtener los modelos de espacios de estados del TCLab, mediante las ecuaciones de balance de calor y mediante la utilización de técnicas de identificación en Matlab. Por otro lado, se van a utilizar observadores para estimar los estados de los modelos, al no ser posible medir todos los estados de nuestro sistema, optimizando los parámetros del observador de forma que los estados estimados del sistema convergen de forma rápida al estado verdadero del sistema.

Se van a aplicar técnicas de control LQR (*Linear Cuadratic Regulator*) y MPC (*Model Predictive Control*), utilizando el toolbox SPCIES de Matlab para el control predictivo por modelo. De esta forma, haremos simulaciones del comportamiento y dinámicas del TCLab utilizando los modelos obtenidos.

Finalmente se realizarán las pruebas necesarias en el TCLab y se validará que se logra el objetivo de control, comparando los resultados obtenidos entre las dos técnicas de control.

Abstract

The objective of this paper is to introduce the modelling, identification, and definition the concepts, for the purpose of control development for the system TCLab (Temperature Control Lab), which is a system designed to teach the principles of control labs, it includes an Arduino, two temperature sensors and two heaters, mounted in a way that there is heat transfer between the devices. We describe in detail the system specifications. The objective of the controller is to follow the temperature setpoint introduced by the user.

To design the controller of the system, we need to obtain the TCLab states spaces models, using the thermal balance equations and system identification techniques with MATLAB. On the other hand, we are going to use an observer to estimate the models' states, because it isn't possible to measure all the states of our system, optimizing the observer's parameters in such a way, so the estimated states of the systems, converge fast to the true state of our system.

We are going to apply LQR (Linear Cuadratic Regulator) and MPC (Model Predictive Control) using the toolbox SPCIES in Matlab for the predictive control. We will use the models obtained of the system to simulate the performance of the TCLab.

Finally, we will test the TCLab, and we will validate if we achieve the control objective, comparing the results of the two control techniques.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
LQR	Control Lineal Cuadrático
A’	Matriz traspuesta.
TCL	Temperature Control Lab
SAE	Suma de Errores Absolutos
Ecs.	Ecuaciones
MPC	Model Predictive Control
MSE	Minimum square error o error cuadrático medio
TCLab	Temperature Control Lab.

1 INTRODUCCIÓN

En este Trabajo Fin de Máster se van a desarrollar y explicar los algoritmos de control utilizados para controlar la planta de control TCLab. Esta planta se caracteriza por disponer de dos sensores de temperaturas y dos calentadores, en un dispositivo Arduino.

El contexto en el que surge el trabajo es por el interés de aplicar los conceptos de control al sistema real, en este caso el TCLab, de forma que se ponga en práctica los conocimientos sobre control adquiridos durante el grado y el máster, además de la posibilidad de utilizar el toolbox de control predictivo, que se ha desarrollado recientemente en el departamento de Ingeniería de Sistemas y Automática, de la Universidad de Sevilla.

El objetivo del trabajo es lograr controlar la temperatura, la salida del sistema, mediante dos algoritmos de control: LQR y MPC. Aplicar el control mediante la estimación del espacio de estados y utilizar dos modelos para el control, un modelo obtenido del desarrollo de la dinámica del sistema y un modelo obtenido por identificación con las herramienta de Matlab, para finalmente comparar los resultados entre los modelos y entre los algoritmos de control. También se va a aplicar el toolbox SPECIES en Python, mediante la generación del controlador MPC en un fichero de C, que posteriormente va a ser compilado y ejecutado. Además, se va a utilizar un observador de estado para estimar los estados del sistema, al no ser posible medir todos los estados.

La técnica de control conocida como LQR, es una técnica de control óptima, de forma que se busca un control que provoque que el sistema dinámico siga una trayectoria, de acuerdo con una función de coste. Algunas aplicaciones recientes en la literatura son para una turbina de viento, una turbina de viento de velocidad variable o para un convertidos de energía para un generador doblemente alimentado [1].

El MPC surge en la década de los 70, como Richalet et al. [2] o Cutler y Ramakter, [3]. Originalmente el diseño del primer MPC tenía la funcionalidad para ser usado en refinerías de petróleo y centrales eléctricas.

En la industria hay diversas aplicaciones para el MPC. Según una encuesta [4], donde tiene mayor implantación es sobre todo en refinerías, también crece en petroquímicas, químicas, poliméricas o plantas de gas. También aparecen mencionados otras industrias como en la industria alimentaria, automovilística, aeroespacial o metalúrgica.

El MPC crece en importancia con el aumento de la potencia de cálculo de los microprocesadores, en distintas industrias como automatización, robótica, lo que se conoce como aplicaciones de “grandes volúmenes” donde a diferencia de la industria química, los sistemas a control son pequeños, las dinámicas más rápidas y con coste de planta menores, por lo que el control debe tener bajo coste. Todavía tiene limitaciones que resolver para el uso generalizado del MPC, pero también tiene muchas ventajas, como la capacidad de optimizar el rendimiento, forzar las restricciones, y la facilidad de diseño de sistemas multivariables [5].

El TCLab se ha utilizado ya que recientemente el departamento de Ingeniería de Sistemas y Automática ha adquirido copias para ser utilizadas por los alumnos en prácticas de control. El TCLab se caracteriza por su sencillez y robustez, lo que permite un fácil uso por parte de los alumnos. Es interesante por la posibilidad de aplicar distintos algoritmos de control a la placa y observar los resultados obtenidos de una forma fácil y muy clara, que facilita al alumno la comprensión de los conceptos.

La metodología utilizada para desarrollar el controlador es buscar en las referencias incluidas en el documento, toda la información posible sobre los controladores y trasladar las ecuaciones de los controladores a código, en cuanto a los ensayos realizados sobre el sistema TCLab, la intención es la de intentar reducir lo máximo posible

las perturbaciones que afecten a la medida de temperatura del sistema, realizando los ensayos en las condiciones lo más parecidas posible, teniendo en cuenta que las pruebas se han realizado en el espacio que se tiene disponible para realizar los ensayos, donde no se pueden controlar todas las perturbaciones. Con una muestra suficiente de pruebas se sacan conclusiones de forma cualitativa, según las gráficas de los resultados obtenidos y de forma cuantitativa, comparando los errores cometidos respecto a la referencia.

2 PLANTEAMIENTO DEL PROBLEMA

En esta sección se va a proceder a detallar las parámetros característicos del problema de control de la placa Temperature Control Lab (TCLab), una descripción de las características de diseño y las consiguientes restricciones del TCLab, se desarrolla el objetivo de control que se ha impuesto y se describe la arquitectura de control que se va a utilizar para lograr los objetivos de control definidos.

2.1 Descripción del Temperature Control Lab

El sistema está formado por una placa montada sobre un Arduino modelo Leonardo, consta de un LED, dos calentadores, que son los actuadores del sistema y dos sensores de temperatura. El calor generado por los calentadores se transmite mediante convección, conducción y radiación a los sensores de temperatura.

La motivación por la que se ha creado el TCLab es para ayudar a trasladar a la práctica los conocimientos de control. Es un elemento práctico para ser usado en laboratorios de bajo coste y fácil uso, debido a su sencilla instalación, prácticamente plug-and-play y pequeño tamaño para reforzar los conocimientos de teoría de control adquiridos en la formación académica. Además, se disponen de bastantes recursos gratuitos en forma de video, tutoriales, aplicaciones, en diversos lenguajes, como Python, Matlab u Octane.

A continuación, se incluyen las figuras 2.1 y 2.2 con una fotos descriptivas para ayudar a visualizar el TCLab.

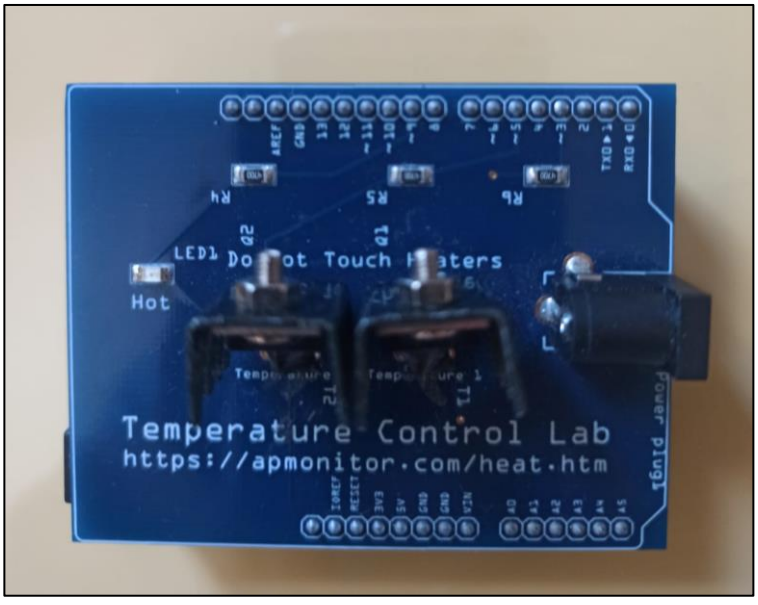

Figura 2-1. Vista superior del TCLab.

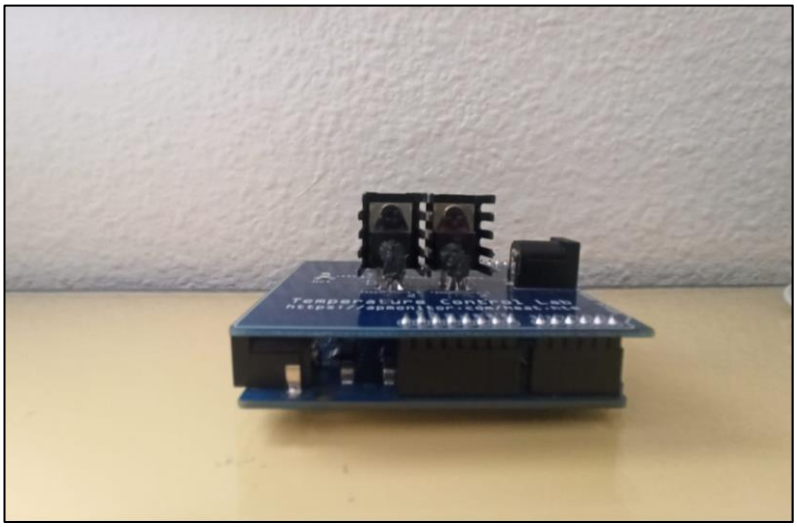

Figura 2-2. Vista frontal TCLab.

En la placa se dispone de un pequeño LED que se puede encender y apagar de forma manual.

Los calentadores se encuentran cubierto por una pintura termocrómica que cambia su color a morado mientras la temperatura sea alta, para tener una señal visual de que el dispositivo está en funcionamiento.

Los sensores de temperatura se encuentran situados delante de los calentadores conectados por una pasta térmica.

El dispositivo viene con dos cables de conexión, un cable USB con terminación Barrel Jack para alimentar a los calentadores y otro dispositivo USB para ser conectado directamente al ordenador y alimentar al Arduino.

Se incluye a continuación un esquema del diseño del TCLab.

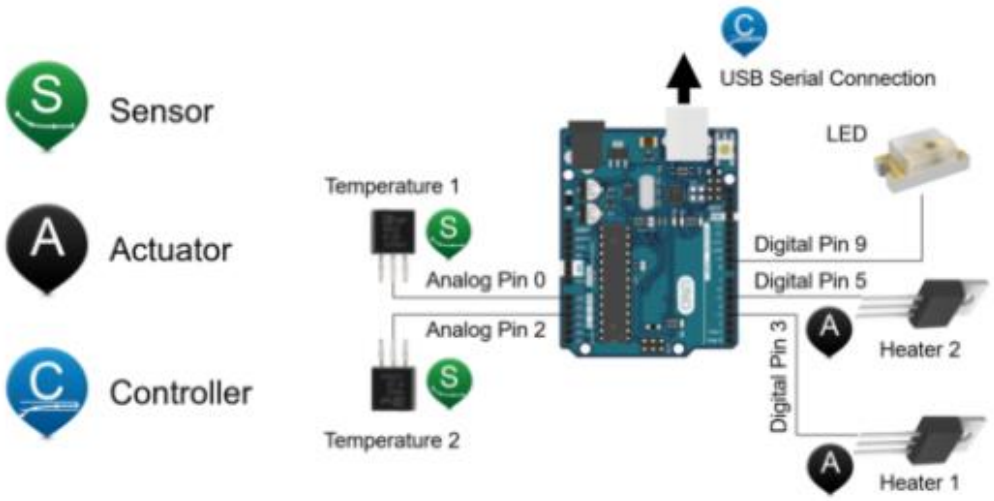

Figura 2-3. Esquema elementos TCLab [6].

El calentador 1 tiene una disipación máxima de 1 W, mientras que el 2º calentador tiene una potencia máxima de 0.75 W. El calentador 1 está conectado al pin digital 3 del Arduino, el calentador 2 se conecta al pin 5. Además, existe un pequeño LED que se conecta al pin digital 9. En cuanto a las conexiones de los sensores, se realizan en los pines analógicos 0 y 2. Además, se dispone de un puerto serie con conexión USB.

2.1.1 Funciones

Las funciones u operaciones básicas programas en el TCLab se encuentran en un fichero de Arduino suministrado en la página web. Para utilizar el TCLab en el ordenador en Python o Matlab es necesario realizar la instalación de los drivers del Arduino y, además, del firmware propio del TCLab disponible en la página web, llamado tclab_v2. Dentro de este fichero están programadas las siguientes funciones:

Cuando llega información por el puerto serie al que está conectado el Arduino, se almacenan los datos recibidos en la variable buffer. Esta información es tratada en la función *parseCommand* y adaptada para que la pueda leer el Arduino teniendo en cuenta las variaciones existentes en los inputs de distintos usuarios.

Entonces se definen dos variables:

- cmd: Contiene el nombre del comando.
- val: Valor numérico.

DispatchCommand: según el valor de cmd ejecuta cada una de las funciones de la placa. Devuelven una respuesta de tipo float.

Tabla 2-1. Funciones del TCLab según valor de cmd.

Q1	Escribe en el pin 3 el valor proporcionado para heater 1.
Q2	Escribe en el pin 5 el valor proporcionado para heater 2.
LED	Cambia la luminosidad del LED.
T1	Devuelve el valor de T1 por el puerto serie en formato ASCII.
T2	Devuelve el valor de T2 por el puerto serie en formato ASCII.

Existen algunas funciones más, pero se utilizan para devolver el resultado en bytes en vez de en formato float y para obtener parámetros como la versión del software que no son necesarias explicar. Además, es posible aumentar la potencia de los calentadores modificando el valor del duty cycle del PWM.

Una vez leída se reinicia la variable a una cadena de caracteres vacía y se quita del puerto serie el valor mandado.

Nota: Q1 y Q2 llaman a otra función para asignar el valor que se escribe en el pin.

SetHeater1 : Con un valor máximo entre 0 y 100.

SetHeater2 : Con un valor máximo entre 0 y 100.

Las variables T_{H1}y T_{H2}, son las temperaturas a las que se encuentran los calentadores 1 y 2, pero no se pueden conocer. Las temperaturas conocidas son las de los sensores, T_{C1} y T_{C2} respectivamente. Esta nomenclatura se va a utilizar en secciones posteriores del documento.

2.1.2 Restricciones

El sistema TCLab tiene unos límites de operación, siendo estos:

$$EQ \leq e$$

Donde Q: Es el valor que recibe el calentador, en las funciones descritas anteriores

$$E = \begin{bmatrix} I \\ -I \end{bmatrix} \quad e = \begin{bmatrix} \overline{Q} \\ \underline{Q} \end{bmatrix}$$

Valor máximo de $\overline{Q}$ = 100

Valor mínimo de $\underline{Q}$ = 0

El valor de la entrada elegida cumple la siguiente inecuación:

$$\underline{Q} \leq Q \leq \overline{Q}$$

Esto se debe a restricciones operativas de diseño del TCLab, ya que los propios fabricantes solo admiten en su diseño de la comunicación con el microcontrolador, en un fichero Arduino, entradas comprendidas entre esos valores.

Las restricciones de temperatura según el modelo de termistores escogido son:

Valor máximo de T, $\overline{T}$ = 150 ºC.

Valor inferior de T, $\underline{T}$ = -40ºC

Aunque están son las restricciones de diseño en la práctica no se van a alcanzar, ya que a máxima potencia de los calentadores es difícil superar los 80 $^{\circ}C$ y la temperatura ambiente va a ser superior a los -40 $^{\circ}C$.

2.1.3 Ecuaciones de transmisión de calor del TCLab

Las ecuaciones de transmisión de calor del TCLab se pueden aproximar por las siguientes cuatro ecuaciones, obtenidas mediante los balances de energía e incluye el calor transmitido entre los dos calentadores, entre cada calentador y el sensor al que está unido, incluyendo el calor transmitido por radiación, conducción y convección, ver [7] y [8].

La transmisión de calor por convección y radiación entre calentador y sensor termistor se suponen despreciable frente a la conducción.

$$mc_p \frac{dT_{H1}}{dt} = UA(T_\infty - T_{H1}) + \epsilon\sigma A(T_\infty^4 - T_{H1}^4) + Q_{c12} + Q_{R12} + \alpha_1 Q_1 \quad (2\text{-}1)$$

$$mc_p \frac{dT_{H2}}{dt} = UA(T_\infty - T_{H2}) + \epsilon\sigma A(T_\infty^4 - T_{H2}^4) - Q_{c12} - Q_{R12} + \alpha_2 Q_2 \quad (2\text{-}2)$$

$$\tau \frac{dT_{C1}}{dt} = T_{H1} - T_{C1} \quad (2\text{-}3)$$

$$\tau \frac{dT_{C2}}{dt} = T_{H2} - T_{C2} \quad (2\text{-}4)$$

Donde desarrollando los términos de cada ecuación:

Q_{C12} es el calor transferido por convección entre los dos calentadores:

$$Q_{C12} = U_s A_s (T_{H2} - T_{H1})$$

Q_{R12} es el calor transferido por radiación entre los dos elementos calentadores:

$$Q_{R12} = \epsilon \sigma A (T_{H2}^4 - T_{H1}^4)$$

$$\tau = \frac{m_s c_{p,s} \Delta x}{k A_c} \; [seg]$$

T_{Hi} es la temperatura del calentador i, T_{Ci} es la temperatura del sensor i.

A continuación, se organizan en una tabla los distintos parámetros que describen las ecuaciones del modelo de balance de calor del sistema con los calentadores.

Tabla 2-2. Parámetros ecuaciones balances de calor del TCLab.

Parámetro	Descripción
m	Masa [kg]
m_s	Masa del sensor [kg]
c_p	Capacidad calorífica $[\frac{J}{Kg*K}]$
$c_{p,s}$	Capacidad calorífica del sensor $[\frac{J}{Kg*K}]$
T_∞	Temperatura ambiente [K]
U	Coeficiente general de transferencia de calor $[\frac{W}{m^2K}]$
U_s	Coeficiente de transferencia de calor entre los calentadores, es superior que con el ambiente $[\frac{W}{m^2K}]$
A_s	Área de superficie entre calentadores [m^2]
A	Área de superficie que no está entre calentadores [m^2]
ϵ	Emisividad
σ	Constante de Boltzmann $[\frac{W}{m^2K^4}]$
α_i	Factor de calentamiento $[\frac{W}{\%heater}]$
Δx	Distancia entre calentado y sensor [m]
k	Coeficiente de conducción térmica $[\frac{W}{mK}]$
A_c	Área de contacto sensor y calentador [m^2]
Q_i	Salida del calentador [0-100]

2.2 Objetivo de control

El objetivo de control es controlar las temperatura T1 y T2 a las consignas Tr1 y Tr2 dadas por el usuario, es

decir una temperaturas de referencia, respetando en todo momento las restricciones operativas de temperatura y potencias térmicas.

2.3 Arquitectura de control

Para controlar el sistema se va a utilizar algoritmos de control que requieren la utilización de un diseño basado en variables de estado. Además, incluye un observador cuya función es estimar los estados del sistema. También se detalla la conexión del sistema al controlador, la forma en la que se transmite la acción de control al TCLab.

2.3.1 Espacio de estados

El concepto de estado de un sistema dinámico se refiere al mínimo número de un grupo de variables, conocidas como variables de estado, que describen totalmente el sistema y su respuesta a cualquier valor de las entradas del sistema [9].

Si un sistema requiere de n variables para describir su comportamiento, entonces las n variables de estado son consideradas las n componente del vector x. Este vector se conoce como vector de estado.

El espacio de estado es el espacio n-dimensional cuyas coordenadas $x_1, x_2, \ldots\ x_n$ son las variables de estado.

El estado de un sistema es descrito por un grupo de ecuaciones diferenciales de primer orden en los términos de las variables de estado $x_1, x_2, \ldots\ x_n$.

$$\frac{dX(t)}{dt} = F(X(t), U(t), t)$$

$$X(t_0) = X_0 \qquad (2\text{-}5)$$

$$Y(t) = H(X(t), U(t), t)$$

En el cual $X \in \mathbb{R}^n$ es el estado, $U \in \mathbb{R}^m$ es la entrada, $Y \in \mathbb{R}^P$ es la salida y $t \in \mathbb{R}$ es el tiempo. La condición inicial del sistema para $t = t_0$ es el valor $X = X(t = 0)$. Conocido el estado en un momento determinado, en $t = t_0$ y las entradas aplicadas al sistema, es posible determinar la evolución de cualquier variable del sistema para cualquier instante de tiempo $t \geq t_0$.

Se va a trabajar en torno a un punto de funcionamiento, que cumple la condición de ser un punto de equilibrio de nuestro sistema.

$$0 = F(X_0, U_0)$$
$$Y_0 = G(X_0, U_0)$$

Se definen las variables incrementales en torno al punto de funcionamiento, de la siguiente forma:

$$x(t) = X - X_0$$
$$u(t) = U - U_0$$
$$y(t) = Y - Y_0$$

X, U,Y son las variables absolutas, con los valores reales de las variables. En todo el trabajo se va a trabajar con las variables incrementales.

Nos interesa utilizar un modelo lineal para expresar el comportamiento de nuestro sistema en torno a nuestro punto de funcionamiento. A medida que nos alejamos de ese punto de funcionamiento el modelo lineal va perdiendo precisión. Para obtener el modelo lineal, se utiliza el desarrollo en serie de Taylor de las ecuaciones descritas (2-5).

$$\begin{aligned} F(X,U) &\cong F(X_0,U_0) + \nabla_x F(X_0,U_0)(X-X_0) + \nabla_u F(X_0,U_0)(U-U_0) + O(2) \\ G(X,U) &\cong G(X_0,U_0) + \nabla_x G(X_0,U_0)(X-X_0) + \nabla_u G(X_0,U_0)(U-U_0) + O(2) \end{aligned} \quad (2\text{-}6)$$

Donde ∇_x es la laplaciana de la función respecto a X y ∇_u es la laplaciana de la función respecto a U.

Desarrollando los términos se obtiene la ecuación en espacio de estados del sistema.

$$\begin{aligned} \frac{dx}{dt} &= A_c x + B_c u \\ y &= C_c x + D_c u \end{aligned} \quad (2\text{-}7)$$

En el cual, $A_c \in \mathbb{R}^{nxn}$ es la matriz de estado, $B_c \in \mathbb{R}^{nxm}$ es la matriz de entrada, $C_c \in \mathbb{R}^{pxn}$ es la matriz de salidas y $D_c \in \mathbb{R}^{pxm}$, es la matriz feedforward que estable una relación entre u e y. En nuestro caso D_c es una matriz cero. Estas matrices son invariantes en el tiempo. El subíndice de las matrices hace referencia a que están en tiempo continuo.

La ventaja de utilizar estas matrices lineales es simplificar mucho la resolución de los problemas de optimización de los controladores predictivos, obteniendo una aproximación fiable del sistema en torno al punto de funcionamiento definido.

En este trabajo se va a utilizar el modelo de espacio de estado en forma discreta, ya que los ordenadores trabajan con tiempos de muestreo de forma discreta. La forma de obtener la ecuación discreta desde la ecuación (2-7) se hace seleccionando un tiempo de muestreo T_S. Este paso es estándar en control.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \quad (2\text{-}8)$$

Donde k es cada instante de tiempo en el que se realiza el muestreo.

Se van a obtener dos modelos del sistema TCLab de la forma (2-8) mediante dos técnicas, uno mediante las ecuaciones (2-1)-(2-4) identificando los parámetros de las ecuaciones dinámicas del sistema y otro mediante técnicas de identificación.

2.3.2 Observador

En realidad, para el sistema TCLab, no es posible medir todos los estados del sistema, solo es posible medir el estado de las temperaturas de los sensores, porque nos proporcionan las medidas de temperatura, que también es la salida de nuestro sistema, como se explica anteriormente.

Esto hace necesario la inclusión de un observador de estado en nuestra arquitectura de control, que es un

elemento que estima las variables de estado de nuestro sistema [10].

Se ha realizado el diseño de un observador de estado de orden completo, esto quiere decir, que estima todas las variables de estado. En la siguiente figura, se muestra el diagrama del sistema con el observador de orden completo.

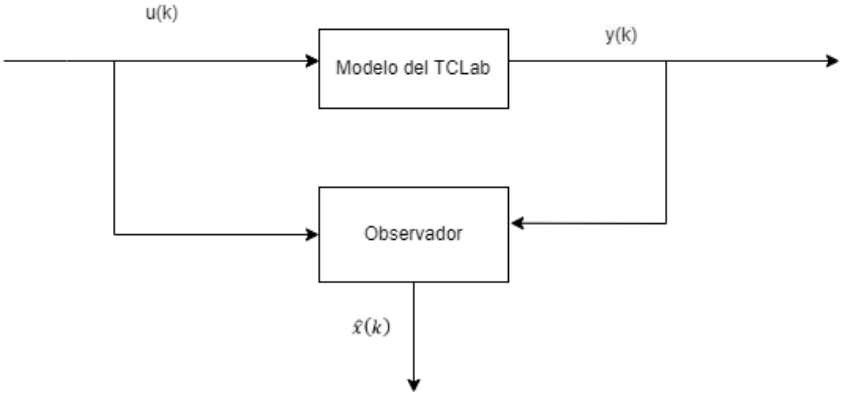

Figura 2-4. Esquema sistema con modelo y observador de orden completo.

El modelo matemático del observador es muy parecido al de nuestro sistema (2-8) pero se añade un término adicional que contiene el error de estimación, el error de estimación es la diferencia entra la variable de salida medida y la salida estimada por el modelo de espacio de estados.

$$\begin{aligned}\hat{x}(k+1) &= A\hat{x}(k) + Bu(k) + \mathrm{L}\big(y(k) - C\hat{x}(k)\big) = \\ &= (A - LC)\hat{x}(k) + Bu(k) + Ly(k)\end{aligned} \qquad (2\text{-}9)$$

Donde $\hat{x}$ representa el estado estimado, L es la matriz de ganancia del observador, este término corrige la salida del modelo y mejora el comportamiento del observador.

Si definimos:

$$A_{obs} = (A - LC) \qquad B_{obs} = [\,B\ L]$$

Es posible agrupar los términos y quedarnos con la siguiente ecuación.

$$\hat{x}(k+1) = A_{obs}\hat{x}(k) + B_{obs}[u(k), \mathrm{y(k)}] \qquad (2\text{-}10)$$

El error cometido por el observador se puede definir con e.

$$e(k) = x(k) - \hat{x}(\mathrm{k})$$

Es posible su cálculo mediante la resta de las ecuaciones (2-8) y (2-9)

$$\frac{de}{dk} = (A - LC)e(k) \qquad (2\text{-}11)$$

Si la matriz $A - LC$ es estable, el error va a converger a cero para cualquier error inicial $e(0)$. Si el sistema es observable, esto quiere decir que el rango, es demostrable, que se puede seleccionar la matriz L de forma que

$A - LC$ tenga los polos deseados [10].

Para seleccionar la matriz L se ha usado el comando *dlqr* de Matlab, de la siguiente forma:

$$K = -dlqr(A, B, Q, R)$$
$$polos = \ eig(A + BK)$$
$$Polos_{Obs} = polos.\hat{}ratio$$
$$L = place(A', C', Polos_{obs})$$

Donde K es la ganancia LQR del sistema. Q y R son matrices definidas positivas de dimensiones: $Q \in \mathbb{R}^{nxn}$ y $R \in \mathbb{R}^{mxm}$

Se elevan los polos para que el sistema sea más rápido que el sistema original, elevando al valor de ratio los polos del sistema en bucle cerrado y se asignan mediante place para que la matriz tenga los polos deseados. Los observadores se van a ajustar modificando el valor al que se elevan los polos del sistema con la ganancia LQR, el valor del ratio.

2.3.3 Controlador

Se incluye una breve descripción del diseño de los dos controladores que se van a utilizar para llevar a cabo los objetivos de control del sistema TCLab.

Controlador LQR (Linear Cuadratic Regulator). La técnica de control LQR, es control óptimo para un sistema lineal con índice de desempeño cuadrático, obtiene una ley de control muy sencilla $u = -K_{LQR}x$ [11].

Controlador MPC (Model Predictive Control) es un método de control de procesos avanzado. Agrupa una serie de técnicas similares. Utiliza el modelo para predecir la salida del proceso en un intervalo de tiempos del futuro. Se calcula la secuencia futura de las señales de control y minimiza la función de coste del sistema, con una estrategia de horizonte deslizante en el que, en cada instante de tiempo, el horizonte se desplaza ese mismo instante hacia el futuro [12].

Ambas técnicas hacen uso del modelo de espacio de estados y del observador.

En las secciones 5.1 y 5.2 se detalla en más profundidad los controladores.

2.3.4 Conexión del sistema real y el controlador

El controlador, tanto el MPC, como el LQR se van a ejecutar en un ordenador, en código de Matlab o Python.

La comunicación entre el ordenador y el TCLab se realiza mediante el puerto serie asignado, cuya conexión se realiza mediante el USB conectado al Arduino y al ordenador desde el que se ejecuta el programa con el archivo de control. El tiempo de muestreo T_s elegido para nuestra planta es de 10 segundos.

Una vez se envían los comandos por el puerto serie, el sketch cargado en el Arduino realiza la interpretación y puede actuar sobre los pines o devolver información por el puerto serie, de la forma explicada en la sección 2.1.1

3 OBTENCIÓN DEL MODELO EN ESPACIO DE ESTADO

En este capítulo se va a describir la forma en la que se han obtenido los parámetros necesarios para los controladores LQR y MPC, de forma concisa, que son de suma importancia para que el control funcione de forma correcta. Se han obtenido dos modelos de formas distintas, un modelo de primeros principios, obtenido a través de las ecuaciones de balance de energía del TCLab y otro modelo de espacio de estado obtenido mediante el uso del toolbox de Matlab para identificar sistemas, el *System Identification Toolbox.*

Con el fin de comparar y observar las diferencias entre los modelos se han obtenido dos modelos de nuestro sistema TCLab, uno obtenido mediante las ecuaciones que describen el comportamiento dinámico de nuestro sistema, en ecuaciones diferenciales de primer orden, denominado comúnmente modelo de primeros principios y otro modelo, obtenido utilizando técnicas de identificación

3.1 Identificación de los parámetros del modelo no lineal

3.1.1 Modelo de primeros principios

Este modelo se obtiene mediante la linealización en torno al punto de funcionamiento de los modelos dinámicos del TCLab, que son las ecuaciones de balance (2-1) a (2-4), definidas en el apartado 2.1.3.

La ecuación de Taylor (2-6) se aplica a las ecuaciones de balance.

Se decidió tomar como punto de equilibrio el valor de T = 23ºC. Esto es de suma importancia tenerlo en cuenta en el algoritmo de control. A continuación, se incluye el valor de las constantes de las ecuaciones:

Parámetro	Descripción
m	0.004 kg
C_p	$500\,\frac{J}{Kg*K}$
T_∞	Temperatura ambiente $23ºC = 296.15\,K$
A_s	0.0002
A	0.001
ϵ	0.9
σ	$5.67x10^{-8}\frac{W}{m^2 - K^4}$

Tabla 3-1. Valores de las constantes.

Es importante mencionar que no se conocen el valor de todos los parámetros de las ecuaciones, en concreto de la Tabla 2-2, no se tiene el valor de $\alpha_1, \alpha_2, \tau, U\ y\ U_s$ de forma que se va a hacer pruebas empíricas con el TCLab, para ajustar los datos experimentales obtenidos mediante la entrada con las ecuaciones de balance de energía, de forma que se busca minimizar el error cuadrático medio (MSE) entre los valores predichos con las ecuaciones de balance y los valores medidos de temperatura, mediante el ajuste de los parámetros desconocidos mencionados anteriormente.

Esto se ha implementado en un fichero Python de la siguiente forma:

Mediante la aplicación de escalones de altura variable a las dos entradas del sistema, Q1 y Q2. Estos escalones

se aplican al TCLab durante un periodo de 10 minutos de prueba y se almacenan los valores obtenidos en un fichero de datos.

Entonces se implementa las ecuaciones de balance de energía (2-1) - (2-4) mediante el uso del package Gekko. Se crea el modelo Gekko (*m = GEKKO()*) y se definen como valores las variables de las ecuaciones según si son constantes, parámetros o variables. Estos valores están relacionados por las ecuaciones. Los parámetros se han estimado y se obtienen los valores aproximados. El objetivo del cálculo es minimizar el error entre las mediciones obtenidas del sistema real, almacenadas en un fichero de datos, con las predicciones del sistema de ecuaciones.

Para ello se definen los valores de las constante según la Tabla 2-2. Parámetros ecuaciones balances de calor del TCLab., se definen las temperaturas de los calentadores como variables de estado (SV), variables no medidas y con capacidad de ajustarse en simulaciones dinámicas y las temperaturas de los sensores como variables controladas (CV). Una vez definidas las variables de las ecuaciones, se procede a formular las ecuaciones de balance de energía utilizando la llamada *.Equation* del modelo Gekko, en un lado de la igualdad, la derivada primera de la temperatura de la ecuación de primer orden y al otro lado, el resto de los elementos de la ecuación. Se le indica en la opciones al solver de gekko los límites máximos y mínimos de los parámetros a estimar y los valores iniciales y se asigna el valor inicial con el que iterar. A continuación, se asignan los valores medidos durante la prueba con TCLab a las temperaturas de los sensores (todas las medidas) y de los calentadores (solo el valor inicial, la temperatura ambiente), de forma que se añade el objetivo de minimizar el error cuadrático medio de la temperatura de los sensores (Tsensor.FSTATUS = 1 y m.options.EV_Type=2) entre el modelo estimado y las medidas obtenidas.

De donde se obtuvieron los siguientes valores de los parámetros:

Tabla 3-2. Parámetros obtenidos por regresión.

Parámetro	**Valor**
U	13,698 $\frac{W}{m^2K}$
U_s	28,850 $\frac{W}{m^2K}$
τ	14,095 seg
α_1	0,011308 $[\frac{W}{\%\ heater}]$
α_2	0,0051104 $[\frac{W}{\%\ heater}]$

En la siguiente figura, se puede comprobar el grado de ajuste de las ecuaciones de balance de la energía con los parámetros obtenidos (en líneas discontinuas), con las temperaturas reales medidas en el TCLab (línea continua), para las entradas de la figura inferior.

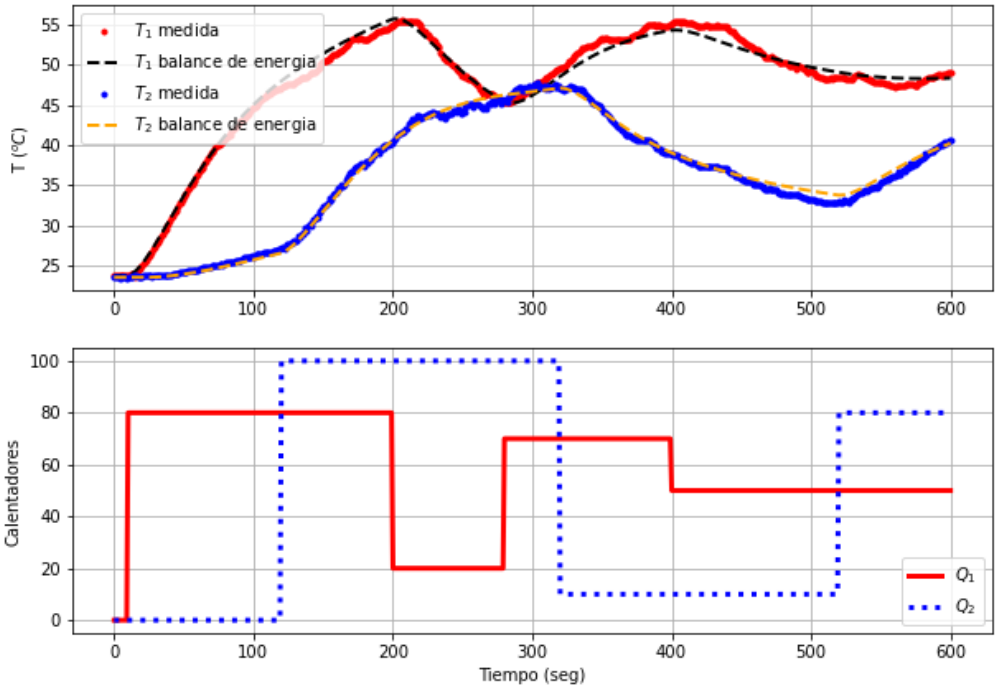

Figura 3-1. Gráfica balance de energía con los parámetros obtenidos por regresión.

Donde la suma de errores absolutos (SAE) obtenido entre el modelo y el balance de energía es de 678,65 unidades.

Una vez obtenido los parámetros se procede a linealizar el modelo mediante la ecuación (2-6), de donde es importante señalar que nuestro punto de equilibrio es el siguiente:

$$X_0 = \begin{bmatrix} 296{,}15 \\ 296{,}15 \\ 296{,}15 \\ 296{,}15 \end{bmatrix} [K] \qquad U_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} [\%\ heater]$$

La temperatura ambiente era de 23 ºC, por lo que se puede aproximar que a esa temperatura se encuentran todas las variables de estado, que son las temperaturas de los calentadores y de los sensores y se considera el punto de equilibrio con los calentadores apagados. Una vez elegido el punto de equilibrio del sistema, es posible calcular los valores de las matrices de nuestro espacio de estados.

$$A = \begin{bmatrix} -0.0129 & 0.00341 & 0 & 0 \\ 0.00341 & -0.0129 & 0 & 0 \\ 0.0709 & 0 & -0.0709 & 0 \\ 0 & 0.0709 & 0 & -0.0709 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.00565 & 0 \\ 0 & 0.00255 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

En la siguiente figura se observa el comportamiento de modelo respecto a las entradas anteriores del sistema y se va a validar frente a la temperatura medida y frente a la temperatura obtenida por las ecuaciones de balance, para ello simulamos el sistema en Python mediante el paquete *gekko.*

Se pueden examinar los resultados obtenidos en la siguiente figura:

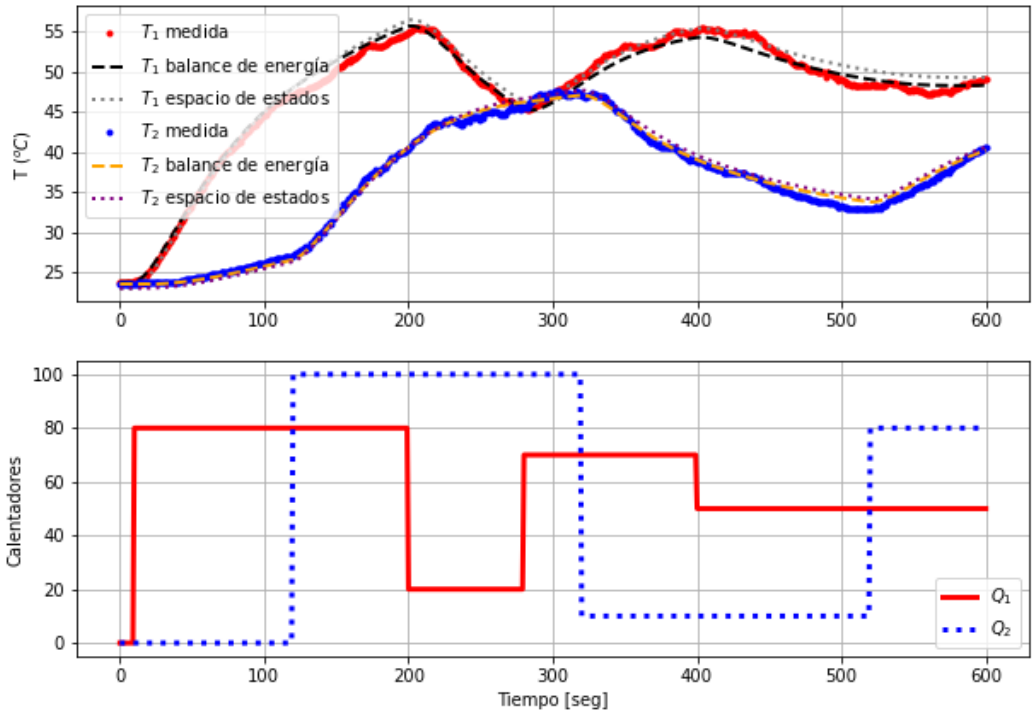

Figura 3-2. Comparación del modelo de espacio de estados con el valor real y las ecs. de balance.

Donde la línea discontinuas de puntos es el valor de temperatura obtenido por nuestro modelo de espacio de estados. Es posible darse cuenta de que cuanto mayor se aleja de nuestro punto de linealización mayor es la diferencia y el error que se comete con el modelo, esto se debe a que el error del modelo aumenta conforme nos alejemos del punto de funcionamiento definido y a que al transcurrir más tiempo el error cometido se acumula en el futuro. Pero en general el modelo obtenido se ajusta de forma adecuada al sistema real.

Si comparamos el modelo con los datos obtenidos del sistema real se obtiene:

- MSE = 12,25
- SAE = 978,7

Se obtienen unos errores superiores que con las ecuaciones del balance de energía del TCLab con los parámetros ajustados, esto es debido a que cuanto más se aleja del punto de linealización, mayor error comete el modelo linealizado respecto a las ecuaciones.

Pero hay que recordar que nuestro sistema es de tiempo discreto, por lo que es necesario convertir estas matrices a tiempo discreto. Se puede hacer en Python mediante la función *cont2discrete* incluida en SciPy.

$$A = \begin{bmatrix} 0.8794 & 0.0300 & 0 & 0 \\ 0.0300 & 0.08794 & 0 & 0 \\ 0.4732 & 0.0089 & 0.4919 & 0 \\ 0.0089 & 0.4732 & 0 & 0.4919 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0531 & 0.000401 \\ 0.000886 & 0.240 \\ 0.0153 & 0.0000816 \\ 0.000181 & 0.00693 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Estas son las matrices que van a ser utilizadas para el control LQR y MPC y para el diseño del observador, este modelo va a ser referido en las siguientes secciones del trabajo como modelo A.

3.2 Identificación mediante el toolbox de Matlab

Este modelo se basa en el uso de la herramienta *System Identification Toolbox* de Matlab, a diferencia del anterior que se generó en Python, mediante una prueba en bucle abierto con el TCLab conectado al ordenador, introduciendo escalones a ambas entradas, Q1 y Q2, de duración 40 segundos y amplitud aleatoria durante un periodo de tiempo de 30 minutos, después se apagan los calentadores y se ponen a cero, para enfriarse y volver al punto de funcionamiento.

Se ha programado el código utilizando Matlab para generar escalones aleatorios para el sistema.

Una vez ejecutado el código *generar_respuesta_escalon.m* se obtuvieron los escalones y la evolución de la temperatura medida por los sensores durante los 30 min de prueba. Es necesario para obtener el modelo, restar la temperatura ambiente a la temperatura medida para tener la temperatura en variables incrementales.

Entonces, se introducen al propio TCLab los escalones para las entradas, los niveles de los calentadores y se observa en la siguiente figura la temperatura medida en variables incrementales, restando la temperatura ambiente de ese momento que estaba en torno a los 27ºC. Pero este modelo se va a considerar para cálculos futuros como linealizado respecto a la temperatura ambiente que se de en ese instante.

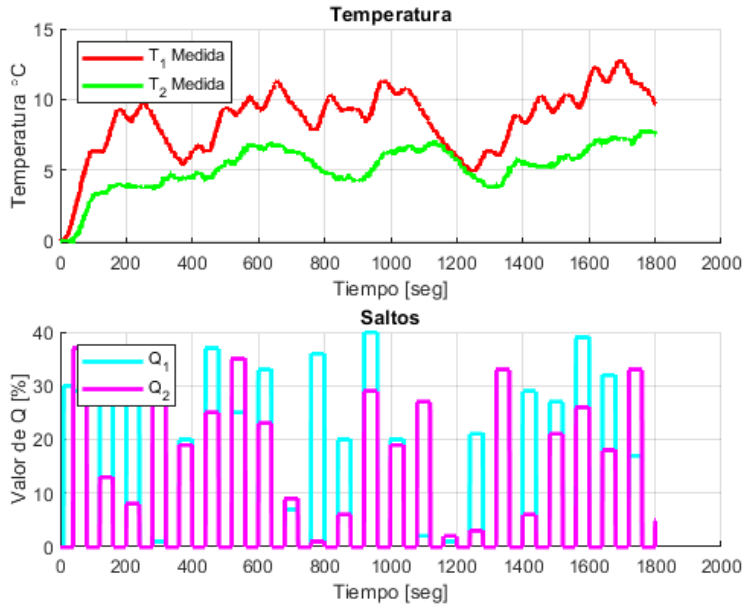

Figura 3-3. Escalones y temperatura.

Una vez obtenido los datos y almacenados en un fichero de texto, se sigue a continuación, con *lectura_de_datos.m,* script de Matlab en el cual se hace la preparación de los datos, para utilizarlos en la identificación.

Se van a dividir los datos en dos bloques un 66% del total (20 minutos) van a ser utilizados para hacer la estimación y el 33 % restante (10 minutos) se van a utilizar para validar el modelo obtenido.

Los datos se han filtrado para mejorar el resultado de la identificación, eliminando el posible ruido de la medida, mediante la ecuación:

$$y(k) = \alpha * y(k-1) + (1-\alpha) * s(k)$$

Donde $\alpha = 0.85$ y s es el valor muestreado por los sensores de la temperatura sin filtrar.

Es un filtro sencillo que consiste en sumar de forma ponderada el valor realmente leído de la señal con el anterior filtrado. De esta forma se elimina el posible ruido de la medida de temperatura, como se puede apreciar en la siguiente figura:

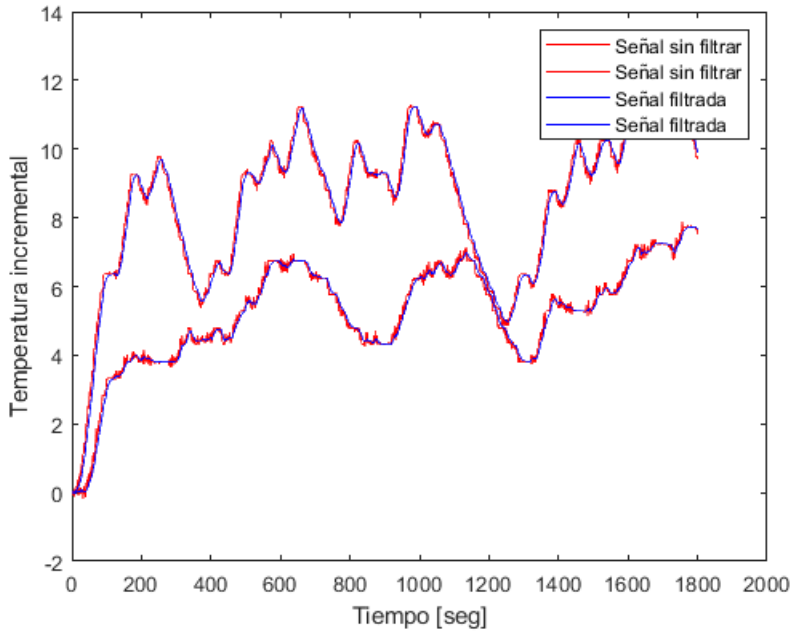

Figura 3-4. Comparación filtro medida.

Una vez filtrados la salida del TCLab, se introducen los datos en el *System Identification Toolbox* de Matlab.

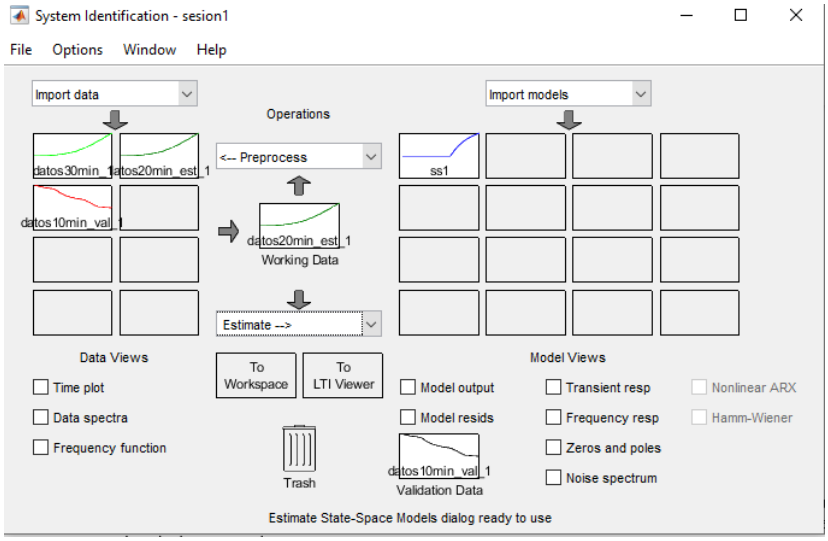

Figura 3-5. System Identification Toolbox.

En la figura se puede apreciar que se ha utilizado de datos de estimación 20 minutos para estimar el modelo en espacio de estados ss1, y se ha validado con datos de 10 minutos. Se ha estimado el modelo de espacio de estados con un orden 4 en forma discreta, con tiempo de muestreo de 1 segundo, una vez obtenido se modifican con la función d2d para obtener el espacio de estados en el tiempo de muestreo asignado de 10 segundos.

Si abrimos ss1, que es la estimación del espacio de estados obtenida, se obtiene que el ajuste del modelo estimado a los datos es de 99,44% y 99,16 % respectivamente, con un MSE de 0,0003613, por lo que parece que el modelo obtenido se ajusta muy bien a los datos obtenidos en el ensayo de escalones.

A continuación, vamos a validar el modelo de espacio de estados obtenidos comparándolo con el sistema real, el TCLab, en una prueba de bucle abierto (8.1). Se ha diseñado una prueba con una duración de 30 minutos, en la que se introducen escalones de distinta amplitud a las dos entradas del sistema Q1 y Q2. Se ha diseñado un bucle con la temporización del T_s, 10 segundos. De forma que entre cada iteración transcurra ese tiempo de muestreo, teniendo en cuenta el tiempo de cálculo de las instrucciones que conforman el bucle.

A continuación, en la siguiente figura se muestran los resultados obtenidos en la prueba.

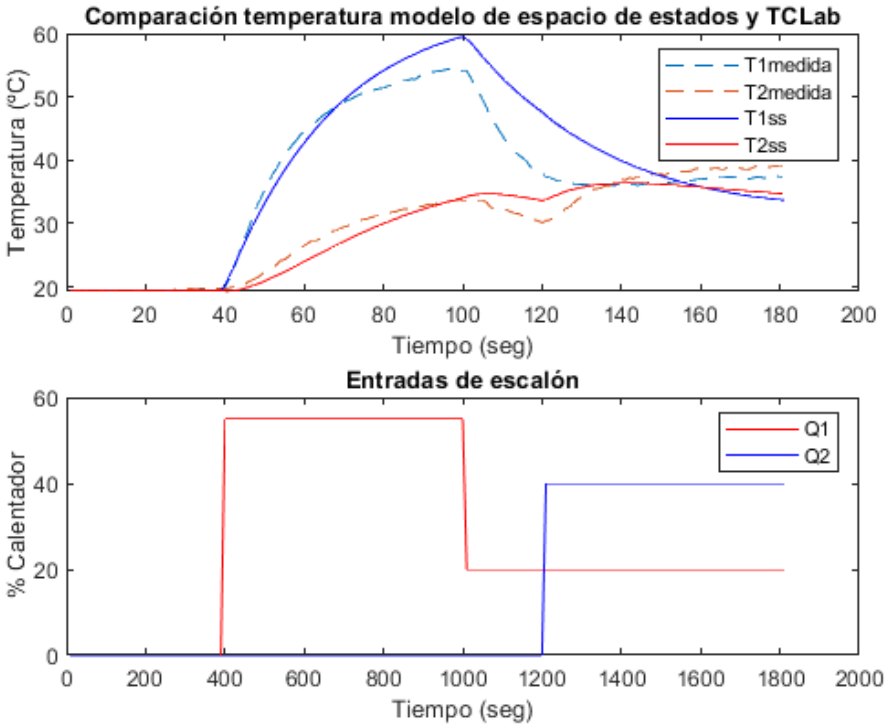

Figura 3-6. Comparación entre TCLab y el modelo de identificación.

Donde T1medida y T2medida, las líneas discontinuas, representan las temperaturas medidas por los sensores del TCLab, mientras que por el otro lado T1ss y T2ss representan las temperaturas obtenidas por el modelo obtenido con la identificación de la respuesta del sistema ante entradas en escalón.

Se puede observar en la figura que a medida que nos alejamos el punto de linealización, aumenta el error que comete el modelo de espacio de estados en la salida respecto a la temperatura real del sistema.

El RMS obtenido para esta prueba del modelo de espacio de estados es de 21,0727 y el SAE es de 802 unidades.

Mediante este procedimiento de identificación se obtiene las siguientes matrices para la ecuación (2-8) de nuestro modelo de espacio de estados.

$$A = \begin{bmatrix} 0{,}974 & 0{,}0610 & 0{,}0201 & 0{,}00290 \\ -0{,}0685 & 0{,}956 & 0{,}0189 & -0{,}0714 \\ -0{,}167 & -0{,}12 & 0{,}377 & 0{,}121 \\ 0{,}00536 & 0{,}0915 & 0{,}0273 & 0{,}0949 \end{bmatrix}$$

$$B = \begin{bmatrix} 4{,}10 * 10^{-5} & 2{,}50 * 10^{-6} \\ 2{,}98 * 10^{-4} & -2{,}37 * 10^{-4} \\ 1{,}98 * 10^{-3} & 1{,}04 * 10^{-4} \\ -1{,}60 * 10^{-3} & 1{,}68 * 10^{-4} \end{bmatrix}$$

$$C = \begin{bmatrix} 226 & 13{,}3 & 0{,}300 & -0{,}0804 \\ 137 & -26{,}2 & 0{,}18 & 0{,}157 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

El estado inicial del modelo se ha obtenido mediante el comando *idssdata(sys),* que permite devolver el punto de funcionamiento del sistema y se ha comprobado que es una matriz de ceros. Este es utilizado como punto de funcionamiento para el sistema en el control. Hay que tener en cuenta que los estados de este modelo no se corresponden con las temperaturas reales, son los ajustes que Matlab ha calculado para el modelo con mayor precisión para el sistema real.

Este modelo de espacio de estados va a ser referido en los siguientes apartados como modelo B. Estas matrices han sido guardadas en un fichero de Matlab para preservarlas sin perder la información por causas externas, de forma que siempre se han ejecutado el controlador con los mismos valores para los modelos de ecuaciones de espacios de estados.

4 DISEÑO Y VALIDACIÓN DEL OBSERVADOR DE ESTADO

En este apartado del trabajo se va a realizar una descripción del observador, especificando la forma en la que se ha diseñado y la justificación de este. Por otro lado, finalmente se realizará la validación del observador comparando el estado observado con el estado medido y la forma en la que converge el observador hacia el estado del modelo del sistema.

4.1 Diseño del observador

Para el diseño del observador se han utilizado las funciones de Matlab.

Debido al hecho de haber obtenido dos modelos de espacio de estados distintos, y, en consecuencia, se diferencian los sistemas en los polos, se han diseñado dos observadores con distintas características. De forma que el observador sea más rápido que el respectivo modelo de espacio de estados para que converja rápidamente el estado estimado por el observador con el estado del modelo.

4.2 Validación del observador

La validación se ha hecho mediante la observación de los estados estimados respecto al estado del modelo, de qué forma se ajusta el observador al estado real de la planta, y con la ayuda del MSE para averiguar el ratio al que se elevan los polos de cada observador.

4.2.1 Modelo A

El modelo A hace referencia al modelo de espacio de estados de primeros principios.

Para validar el observador se han calculado los resultado de un bucle abierto del TCLab, durante un periodo de simulación de 2500 segundos, iniciando con un valor de las entradas de cero, de forma que el sistema se encuentra en el punto de funcionamiento. En el instante $t = 620$ segundos se introduce en el bucle el observador que hasta entonces no estaba actualizándose. Cuando $t = 1250$ segundos se introduce una entrada u_1, cuyo valor es:

$$u_1 = \begin{bmatrix} 40 \\ 30 \end{bmatrix} [\%\ calentador]$$

El estado inicial del observador se ha definido a 30 $ºC$.

Si se representan gráficamente los distintos observadores con los distintos valores de los polos elegidos según el valor al que se elevan los polos del sistema completo, de forma que el observador sea más rápido que el sistema.

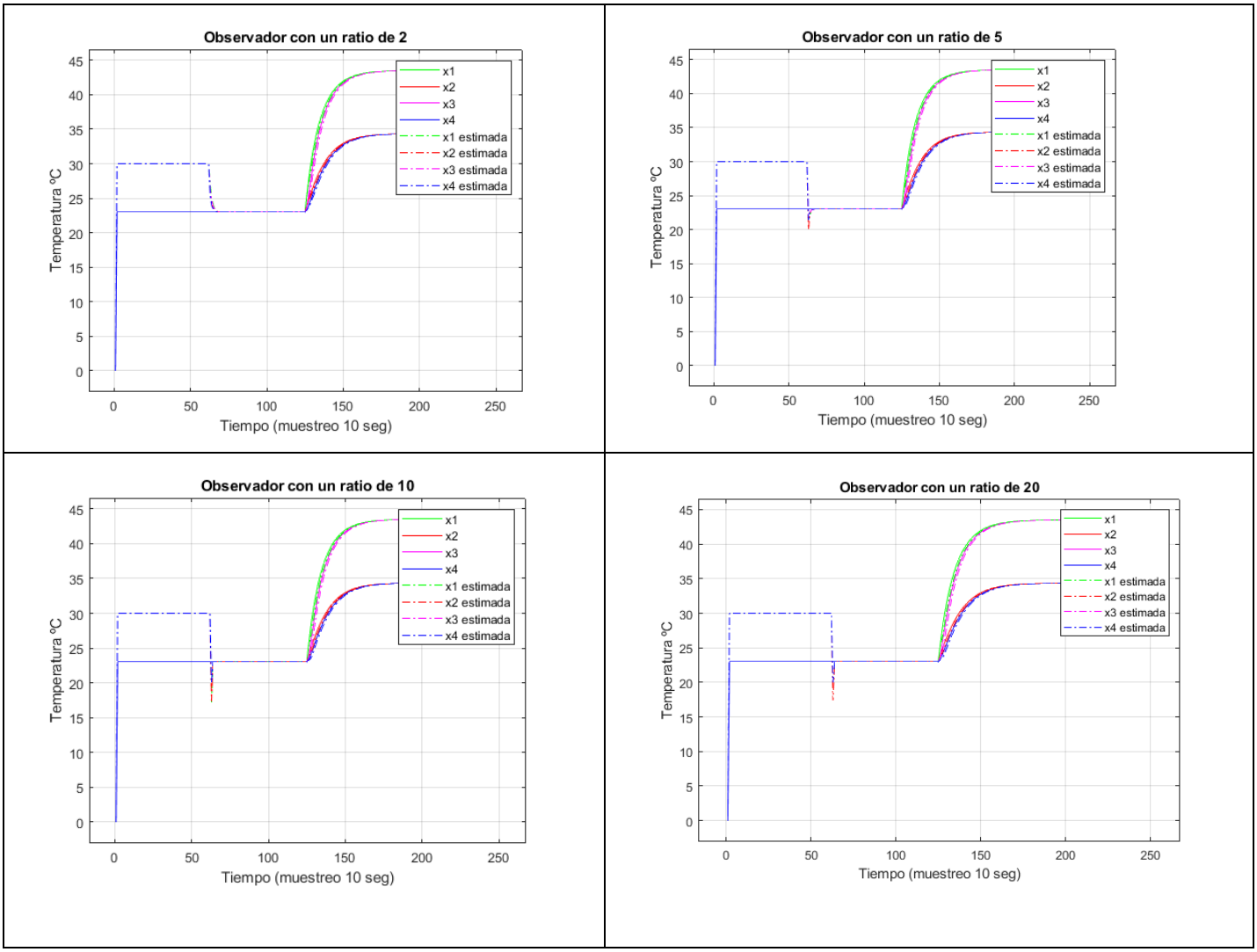

Figura 4-1. Observadores modelo A.

Donde las líneas punteadas son el estado estimado por el observador $\hat{x}$, y las líneas continuas son los estados del modelo de espacio, x. Recordar que el tiempo de muestreo es de 10 segundos por lo que entre cada dos puntos de la gráfica han trascurrido 10 segundos de forma simulada. Observar que durante las primeras 62 iteraciones (620 segundos), el observador no está conectado al sistema y mantiene los estados estimados constante, pero en cuanto se activa, rápidamente se ajusta a los estados del modelo.

Además, tras probar con distintos saltos de temperatura la tendencia se mantiene, por lo que resulta correcto elegir el ratio de 3 para el observador al ser el valor que tiene el mejor seguimiento del sistema, al no producirse un pico en las temperaturas como si existen en la figura 4-1, cuyo comportamiento se incluye a parte, en la siguiente figura:

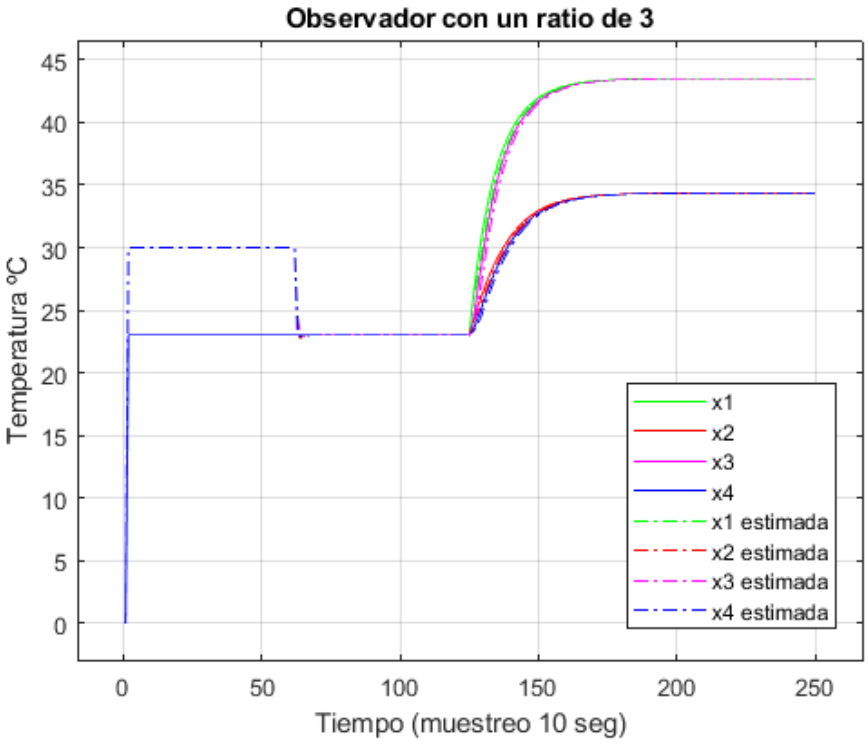

Figura 4-2. Observador modelo A.

4.2.2 Modelo B

Modelo B es el obtenido mediante el *System Identification Tool* de Matlab. La metodología es la misma que para el modelo A, por lo que no se explica aquí.

El estado inicial del observador se ha definido cerca del punto de funcionamiento. Pero en este modelo ese estado no representa ninguna variable real, ya que se ha obtenido mediante técnicas de identificación el modelo y no tiene el estado una correlación directa con alguna variable de las ecuaciones de balance.

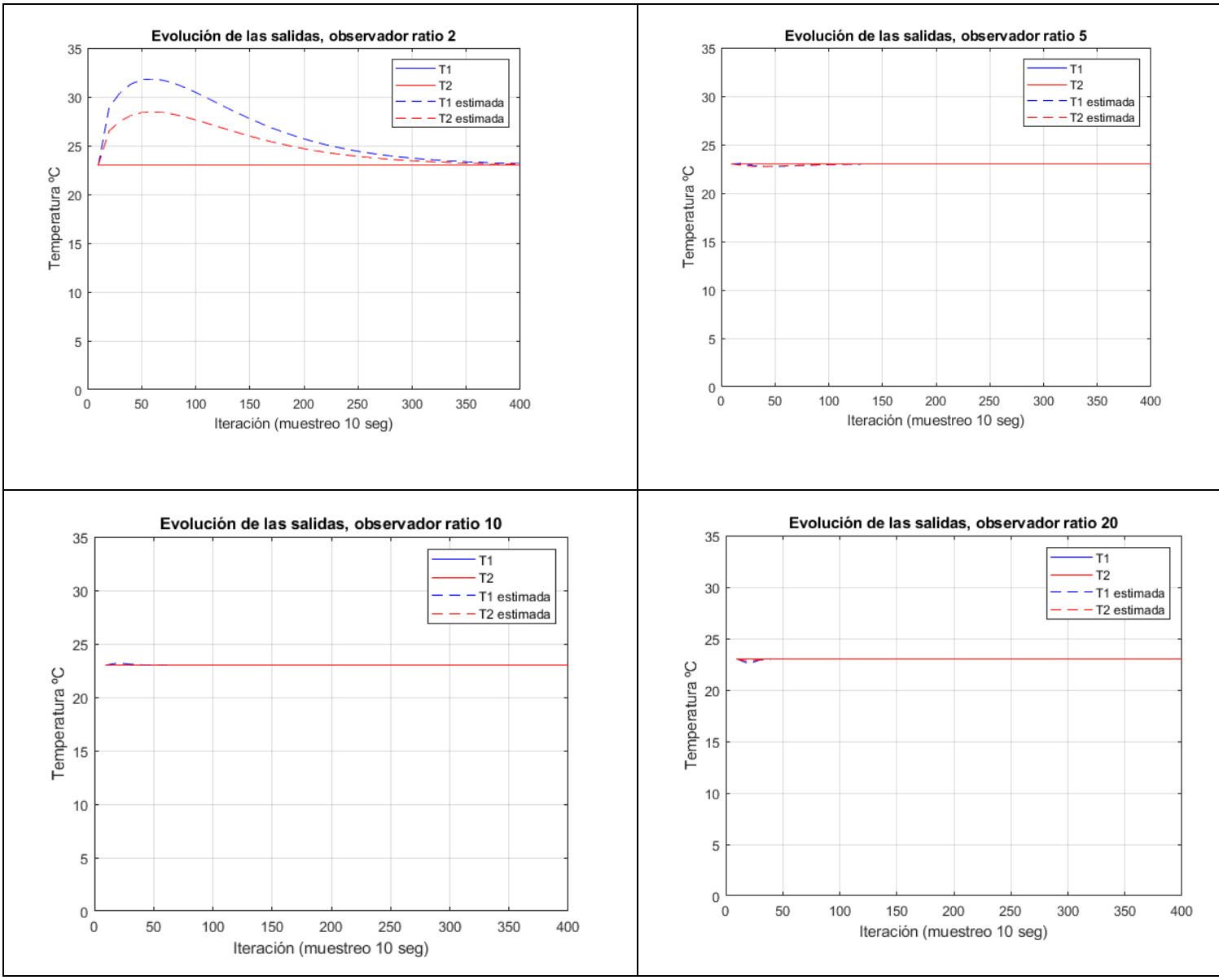

Figura 4-3. Observadores modelo B.

Considero interesante señalar que los estados de este sistema no representan la temperatura, pero se puede utilizar la salida estimada que es equivalente a la temperatura del sistema. En línea continua se representa la temperatura del modelo y en línea discontinua al temperatura estimada por el observador.

Para el modelo obtenido por identificación, resulta sencillo elegir el ratio de 7 para el observador, de forma que no se produce una sobreoscilación significativa de los estados. Aunque cualquier valor cercano a un ratio de 10 es aceptable por tener errores muy pequeños.

En la siguiente figura se observa la evolución de la salida con el observador elegido para el modelo B, con ratio 7, de forma que se produce un pico de temperatura muy pequeño para los valores registrados de la salida.

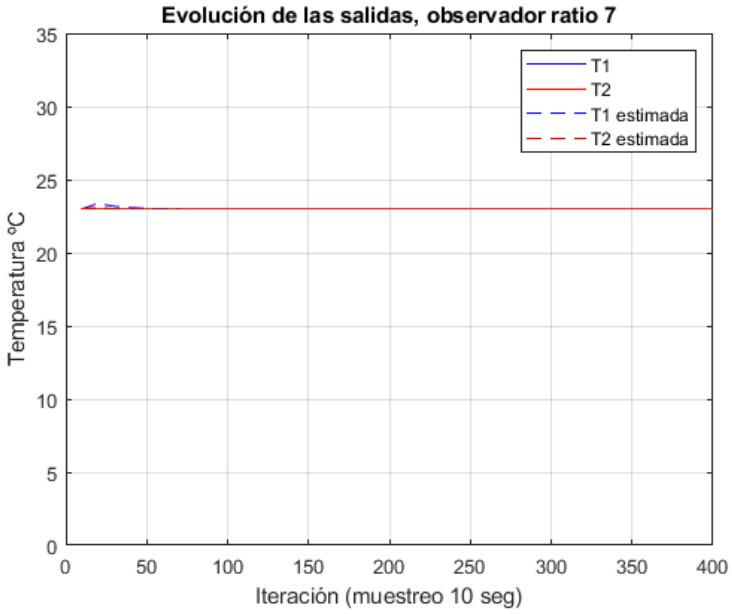

Figura 4-4. Evolución salida observador con ratio 7 modelo B.

La evolución de los estados subyacentes para esa salida estimada del modelo con estados estimados es la siguiente:

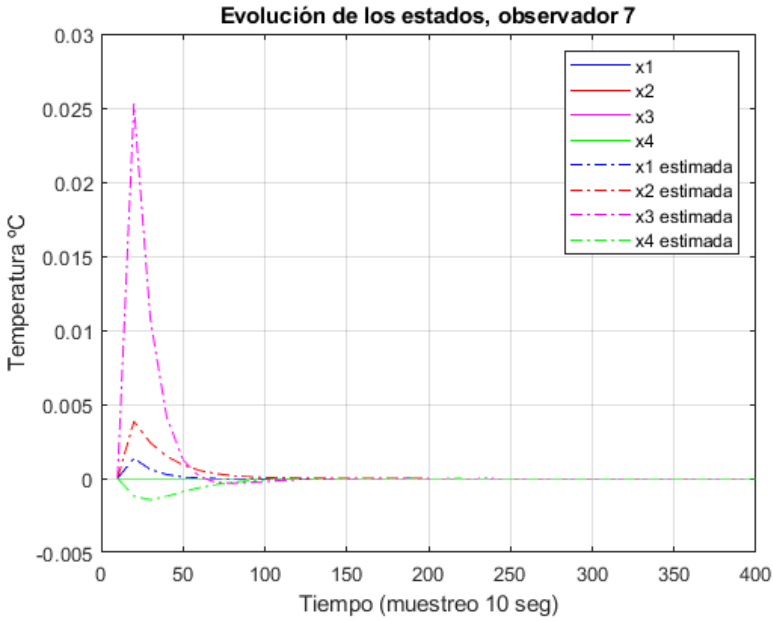

Figura 4-5. Observador modelo B, evolución de los estados

Los estados del modelo A se mueven en rangos de valores de milésimas, en torno a 0.001 o inferiores incluso.

En realidad, lo que funciona de mejor forma, sin que se produzcan picos de temperatura para la salida estimada por el observador, es utilizar para el estado inicial del observador modelo B, un vector próximo al punto de linealización, que es un vector $4x1$ de ceros, en caso de error y asignar un valor de temperatura ambiente al estado inicial del observador, se va a producir un pico de temperatura por la estimación de los estados del sistema que realiza el observador.

5 CONTROL DEL SISTEMA

La base en la que se fundamenta el control, los algoritmos de control aplicados al sistema se van a detallar en este capítulo. En concreto los dos algoritmos de control que se han utilizado, el LQR y el MPC con el fin de lograr el objetivo de control definido. Se entrará en detalle en las ecuaciones y los parámetros que define cada uno de los controladores y se harán simulaciones con los modelos de espacio de estados obtenidos, incluyendo el observador diseñado, para validar los resultados.

5.1 LQR

Las técnicas de control LQR (por sus siglas en inglés *Linear Quadratic Regulator*), es una técnica de control moderna [11].

Partiendo de la ecuación del modelo de espacio de estado en tiempo discreto. (2-8)

El objetivo es calcular la ley de control:

$$u(k) = -Kx(k) \qquad (5\text{-}1)$$

Donde K es la ganancia LQR.

A cada valor $(x(k), u(k))$, se le asocia un coste $L(x,u) > 0$ que mide la distancia hasta el punto de equilibrio (0,0) [11]. Para el control LQR, se usa el índice de desempeño cuadrático.

$$L(x,u) = x^T Q x + u^T R u$$

Donde Q y R son matrices simétricas definidas positivas.

Entonces el índice de desempeño para la trayectoria del control será:

$$J(x_0) = \sum_{k=0}^{\infty} L(x(k), u(k))$$

Este índice de desempeño indica cómo de bueno es la trayectoria, a menor J mejor rendimiento del sistema.

En [11], se demuestra que:

- La ley de control óptima es lineal $u = K_{LQR} x$
- El coste óptimo es $J^*(x) = x^T P x$

P debe satisfacer la ecuación de Riccati para minimizar el índice de desempeño [11].

$$P = Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A$$

Una vez calculada P, que no es trivial, pero si con un ordenador de forma iterativa, se determina el controlador:

$$u = -(R + B^T P B)^{-1} B^T P A x$$

Esta ecuación es fácilmente implementable en Matlab, mediante `dlqr(A,B,Q,R)`, de donde se obtiene la

ganancia K óptima para la ecuación de espacio de estados discretos (2-8).

El control en variables de estado con un observador se define mediante la siguiente ecuación, donde queremos llevar el sistema a una referencia:

$$\begin{matrix} 0 = Ax_r + Bu_r \\ r = Cx_r + Du_r \end{matrix} \longrightarrow \begin{matrix} x_r = N_x r \\ u_r = N_u r \end{matrix}$$

$$u(k) = K(\hat{x}(k) - x_r) + u_r = K\hat{x}(k) + [-K\ I]\begin{bmatrix} x_r \\ u_r \end{bmatrix} = K\hat{x}(k) + [-K\ I]Nr = \\ = K\hat{x}(k) + Mr \qquad (5\text{-}2)$$

Donde (x_r, u_r) es el punto de referencia al que queremos llevar el sistema y la matriz M multiplica al valor de referencia de la salida del sistema deseado. Recordad que $\hat{x}(k)$ es el estado estimado.

Todos estos parámetros del controlador LQR se calculan de la siguiente manera en código de Matlab:

```
q = C'*eye(2)*C;
rho = 0.01;
r = rho*eye(2);

N=[A-eye(4),B;C,D]\[zeros(4,2);eye(2)];

K=-dlqr(A,B,q,r);

M=[-K ,eye(2)]*N;
```

Es de suma importancia las ponderaciones que se le dan a los valores de Q y R para el índice de desempeño, por su sencillez se ha decidido utilizar matrices Q y R diagonales.

- A mayor Q, se pondera en mayor medida la rapidez del control, minimizando el error de seguimiento.
- A mayor R, se pondera en mayor medida el esfuerzo que se realiza en el control.

La u está comprendida en valores entre [0-100], mientras que las variables de estado del modelo A, los límites de operación están en torno a [0-80] por lo que no es necesario realizar una normalización de las variables $u\ y\ x$.

$$Q = C^T * eye(2) * C$$
$$R = \rho * eye(2)$$

Para elegir el valor de ρ se ha probado de forma iterativa distintos valores posibles para la ρ implementando el control LQR con saltos de referencia de $T\_inicial = [25{,}25]$ a la referencia1 $= [45, 40]\ en\ t(k) = 510\ y\ referencia2 = [35{,}30]$ en $t(k) = 1510\ hasta\ t(k) = 2000$, momento en el que termina la simulación.

Esto se ha realizado para los dos modelos de espacio de estados.

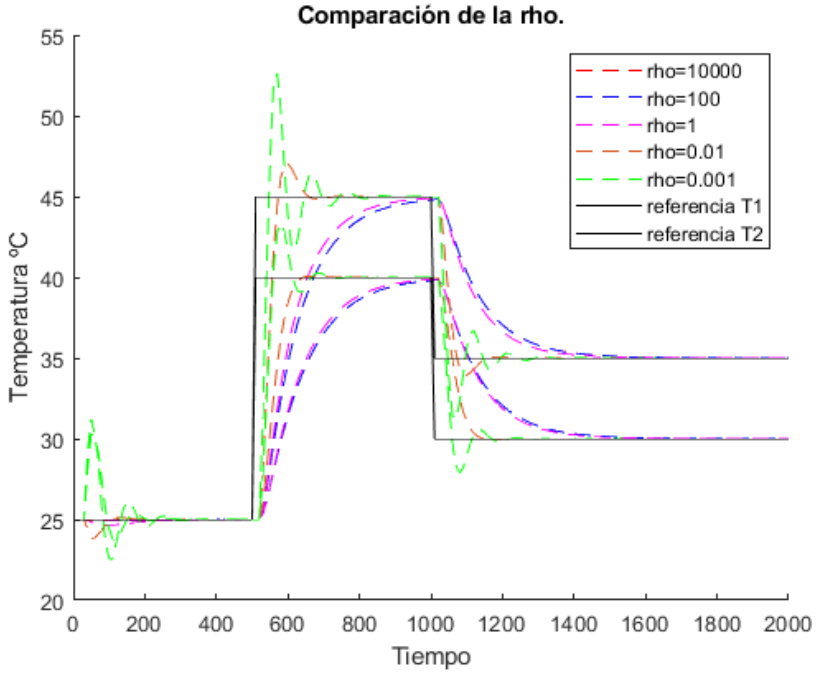

Figura 5-1. Comparativa ρ Modelo A.

Se ha optado por elegir una rho de acuerdo con las características del sistema, de forma que el comportamiento sea parecido al sistema real en bucle abierto, en este caso se ha decidido tomar $\rho = 1$.

Para el modelo B, aquí las variables de estado del sistema identificado no tienen un sentido directamente físico, debido a la técnica de identificación utilizada para obtener el modelo, por lo que se ha optado directamente por utilizar el método de prueba y error para elegir el parámetro ρ.

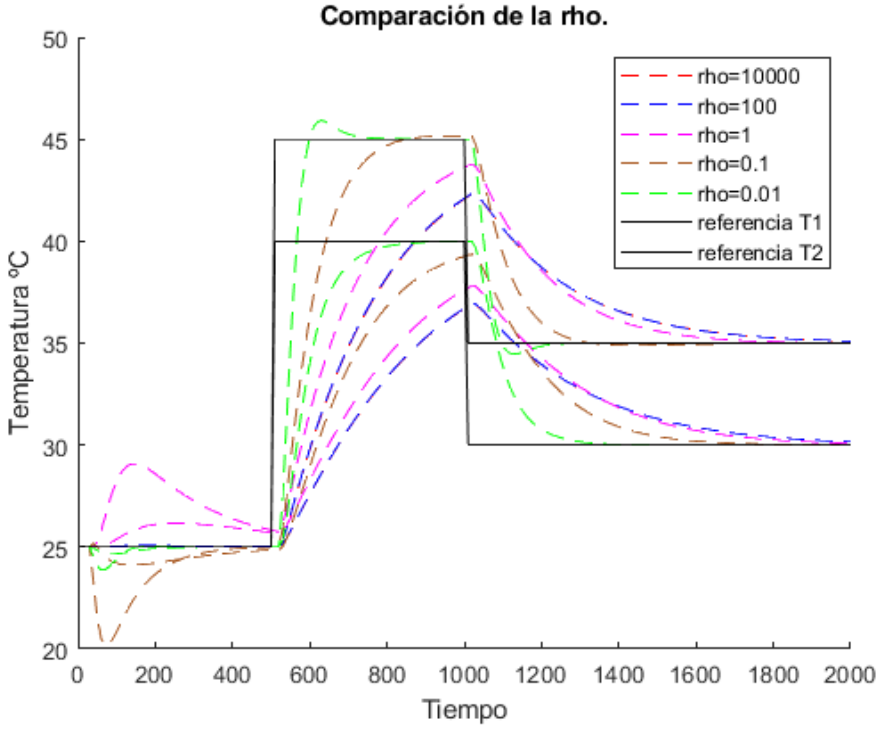

Figura 5-2. Comparación ρ modelo B.

Dado que el sistema se ha obtenido por identificación el sistema real mediante entradas en escalón, se debe pedir al sistema que tenga un comportamiento similar al sistema en bucle abierto de forma inicial, posteriormente, será posible aumentarlo para aumentar la velocidad del control del sistema. En este caso $\rho = 0.1$ parece una elección adecuada.

Se pueden notar la diferencia de comportamiento entre los dos modelos del espacio de estado, ya que la evolución de las temperaturas con control LQR para igual ρ varían según el modelo.

En el siguiente esquema se observa una figura con la estructura del control LQR en la simulación del sistema utilizando el modelo de espacio de estados:

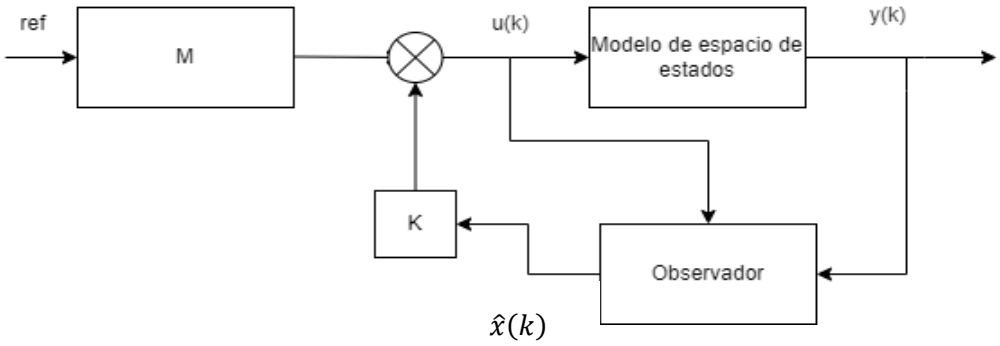

Figura 5-3. Esquema de estructura de control LQR.

Se ha simulado que el sistema parte del punto de funcionamiento durante un periodo de 1000 segundos de forma que se asegura que el sistema y el observador se encuentren en torno al punto de funcionamiento del sistema. Entonces se aplica una referencia de temperatura para la salida de $[45,40]$ desde $t = 1001\ seg$ hasta $t = 2500\ seg$. Posteriormente, se aplica otra referencia de [35,30] desde $t = 2500\ seg$ hasta $t = 4000\ seg$ y finalmente el sistema retorna hasta el punto de funcionamiento.

Recordad que son simulaciones por lo que el tiempo real es de pocos segundos de cálculo del programa al trabajar con los modelos linealizados con su $T_s = 10$ segundos y no con el sistema real, donde si es necesario esperar el tiempo de muestreo real entre cada iteración.

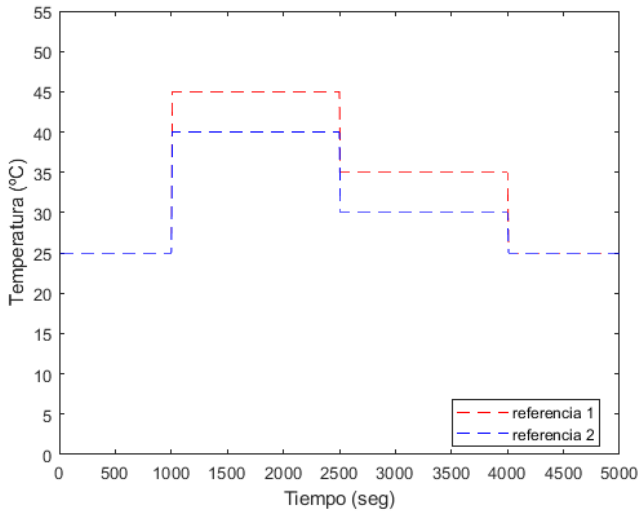

Figura 5-4. Gráfica de la evolución de la referencia utilizada en el control LQR.

5.1.1 Modelo A

Realizando una simulación con el modelo A, se han obtenido las siguientes gráficas donde se muestran la evolución de los distintos parámetros.

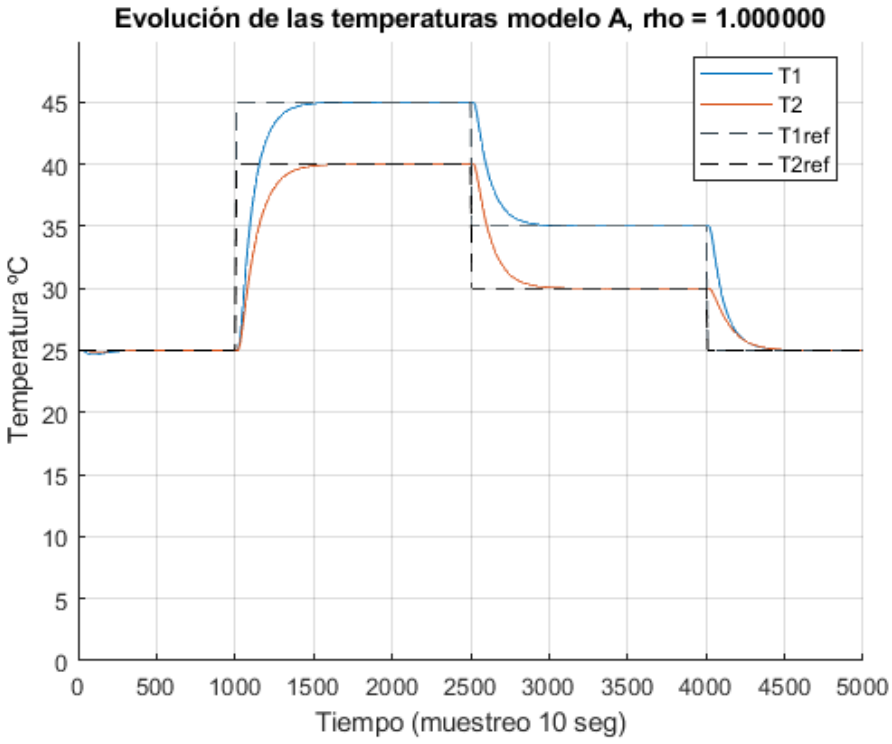

Figura 5-5. Control LQR, evolución de las salidas modelo A.

La línea continua representa la salida del modelo de espacio de estados, T1 y T2. Por el otro lado, en línea discontinua se encuentra la referencia de temperatura.

Se observa que el sistema se ajusta a las referencias de temperatura introducidas, con un transitorio suave, sin sobreoscilaciones.

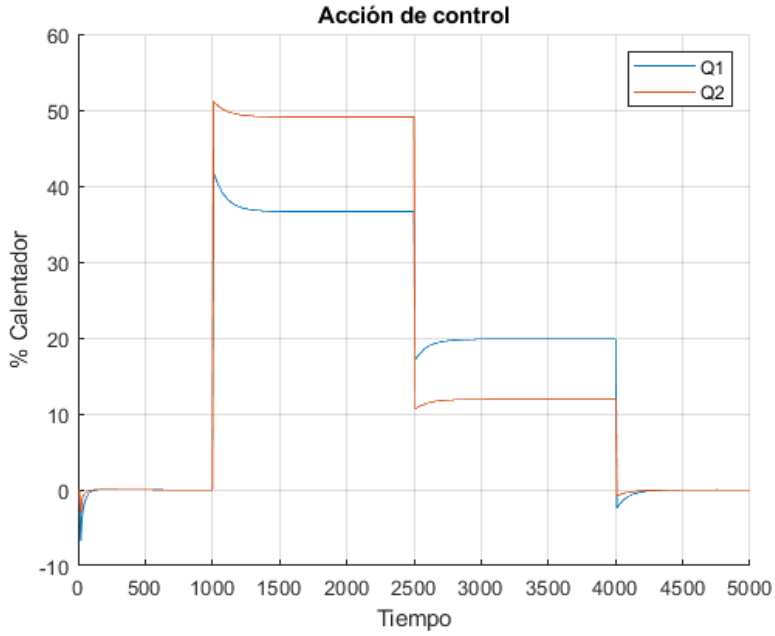

Figura 5-6. Control LQR entradas modelo A.

En esta figura se representan las acciones de control calculadas por el controlador LQR que se aplican al modelo del sistema, siendo la línea azul continua Q1 y la línea naranja Q2.

Inicialmente el observador se encuentra definido en un estado superior respecto al punto de funcionamiento, debido a esto la entrada aplicada es inicialmente negativa.

Entonces se puede observar el problema que se encuentra con el control LQR es el hecho de que no opera de forma intrínseca con restricciones a la acción de control aplicada al sistema, por lo que se llega a una entrada negativa, cuando el límite operativo de las entradas es de [0,100] debido al diseño del TCLab, luego esta acción de control inicial no sería posible su aplicación al sistema real.

También se ha obtenido el tiempo de cálculo del control LQR y se representa gráficamente por iteración:

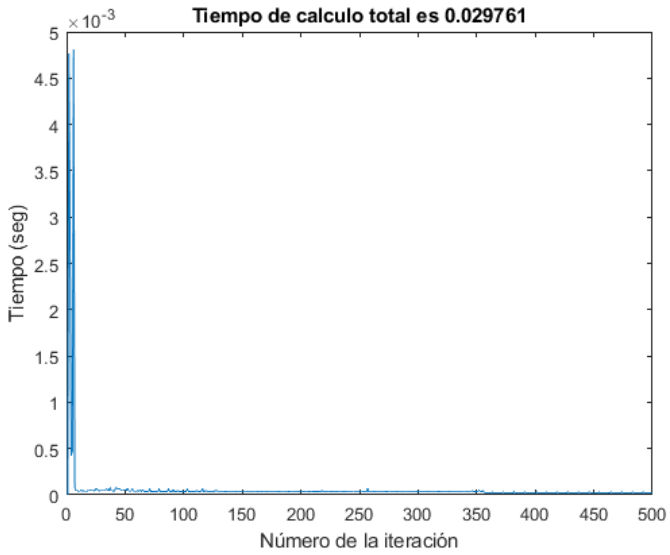

Figura 5-7. Tiempo de cálculo LQR modelo A.

En esta figura se representa el número de la iteración, $T_s = 10\ segundos$ por lo que se realiza una iteración cada 10 segundo de simulación de los sistemas.

Excepto en las iteraciones iniciales, los tiempos de cálculo del controlador LQR están en torno a tiempos de 10^{-5} segundos, el tiempo en las primeras iteraciones es el que ocupa casi todo el porcentaje de tiempo de cálculo, cuyo total es 0.0298 segundos.

5.1.2 Modelo B

Para el modelo B, se ha realizado la misma prueba con las mismas características y cambios de la referencia, pero cambiando los parámetros del observador y la ρ, de acuerdo con los resultados del diseño del modelo y observador. Se ha tomado un punto inicial del observador cercano al punto de funcionamiento del sistema linealizado

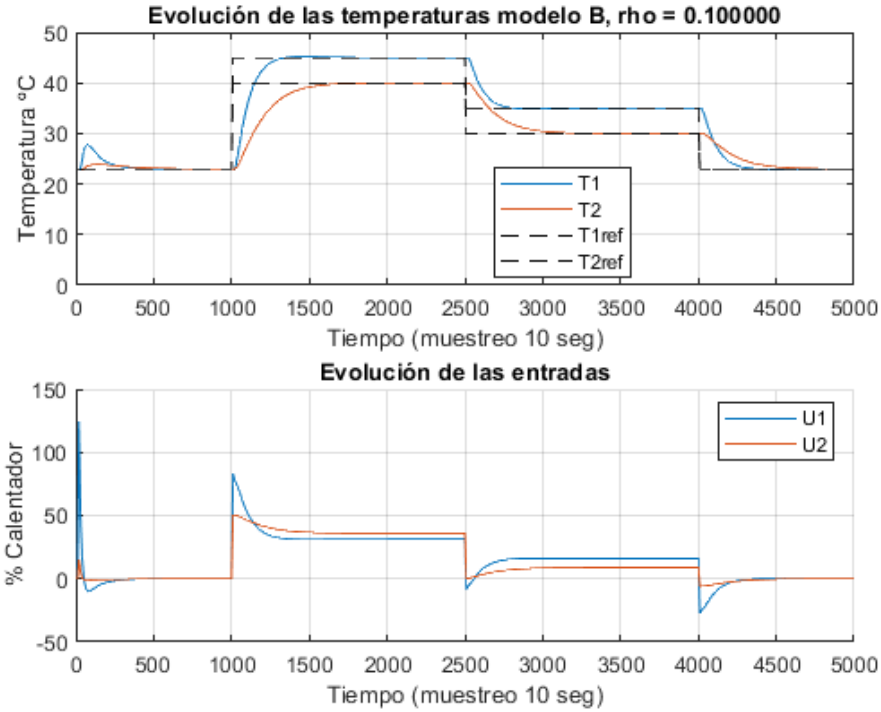

Figura 5-8. Control LQR evolución salidas y entradas modelo B.

El comportamiento es muy similar con el del modelo A, aunque parece que el modelo B es más inestable en el periodo de estabilización del observador hasta converger con el estado del modelo, esto depende principalmente de la ratio del observador y de la forma de definir el estado inicial del observador, con un ajuste más cercano al punto de funcionamiento no existiría ese salto tan grande de la entrada en los instantes iniciales, como se puede observar en la siguiente figura:

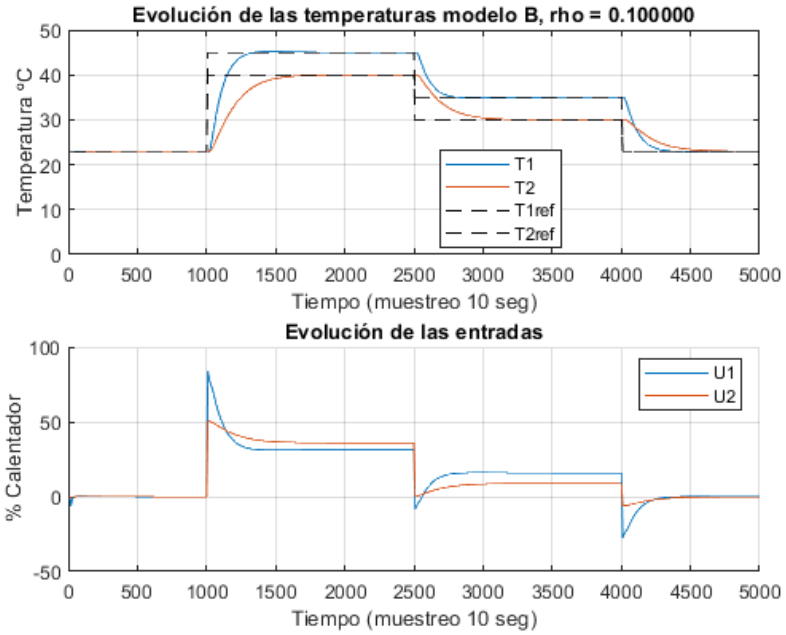

Figura 5-9. LQR evolución salidas sin pico inicial de temperatura estimada.

Simplemente se ha definido un valor inicial del estado del observador más próximo a cero, de esta forma no se produce el pico de temperatura y en la entrada, % calentador, en los instantes iniciales en los que se activa el observador.

El tiempo de cálculo del LQR es el siguiente:

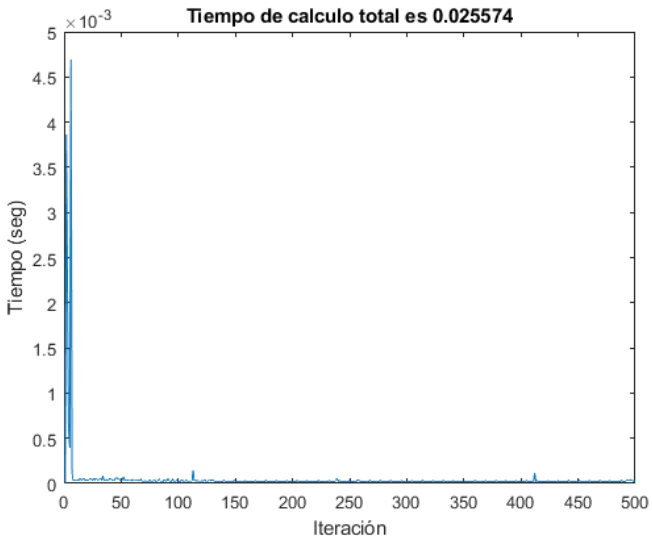

Figura 5-10. Tiempo de cálculo LQR modelo B.

No existen diferencias significativas en el tiempo de cálculo respecto con el modelo A, es simple la razón, los cálculos a realizar son similares modificando los valores de las matrices según el modelo.

5.2 MPC

Model Predictive Control, o MPC, se caracteriza por la siguiente estrategia, tal como describe en [12]:

-Las salidas en un horizonte N, el horizonte de predicción, se predice en cada instante t con el modelo del proceso.

-Las señales de control futuras se calculan optimizando mediante un criterio, una función de costes, de forma que el proceso se mantenga lo más cercano posible a una trayectoria de referencia $w(t+k)$. El criterio suele ser una función de tipo cuadrática del error entre la salida y trayectoria predichas.

-La señal de control $u(t|t)$ es enviada al bucle de control mientras que las siguientes se desechan, porque se conoce el siguiente muestreo de $y(t+1)$ y se calcula $u(t+1|t+1)$. Esto se conoce como estrategia de horizonte deslizante. Donde $u(t|t)$ es la acción de control para el tiempo t estimada en el tiempo t , mientras que $u(t+1|t+1)$ es la acción de control en $t+1$, el siguiente tiempo de muestreo del controlador, estimada en $t+1$.

La principal ventaja es que permite incorporar restricciones de operación y trabaja de forma sencilla para sistemas multivariables. Se puede aplicar a casi cualquier sistema y es muy fácil de sintonizar. Por el otro lado, el inconveniente que tiene el método es su alto coste computacional y que requiere de un modelo preciso del sistema.

La función objetivo, $L(*)$, mide el error de seguimiento de la trayectoria objetivo [13].

$$L(x,u) \geq 0 \; \forall (x,u) \tag{5-3}$$

En concreto, una de las funciones objetivos que se pueden utilizar, es el coste cuadrático.

$$L(x,u) = x^T Q x + u^T R u$$

Se han definido las siguientes restricciones para la salida de nuestro sistema nuestro sistema:

$$LBu \leq u_k \leq UBu \; \epsilon \; Z_{\geq 0} \tag{5-4}$$

Donde UBu en el limite superior para la acción de control del sistema y es igual a 100. LBu es el límite inferior para la acción de control del sistema y es igual a 0.

También se han definido límites en los estados por razones de seguridad y evitar que la temperatura suba hasta niveles demasiados altos.

$$LBx \leq x_k \leq UBx \; \epsilon \; Z_{\geq 0} \tag{5-5}$$

Donde UBx es el límite superior del sistema y se ha definido como $UBx = 100 - X_0 \; y \; LBx = 0 - X_0$ ya que las restricciones se definen en variables incrementales, luego es necesario restar el punto de linealización. Se ha decidido trabajar con grados centígrados porque es más sencillo de visualizar la temperatura que utilizando Kelvin.

El MPC puede realizar una estimación práctica del control óptimo LQR. El problema de optimización del MPC es el siguiente:

$$\min_{u} V_N(x,u) = \sum_{j=0}^{N-1} L\big(x(j),u(j)\big) + V_f\big(x(N)\big)$$

Sujeto a:

$$x(k+1) = Ax(k) + Bu(k) \qquad (5\text{-}6)$$

Y las restricciones del sistema:

$$LBu \leq u_k \leq UBu \; \epsilon \; Z_{\geq 0}$$

$$LBx \leq x_k \leq UBx \; \epsilon \; Z_{\geq 0}$$

Donde N es el horizonte de predicción, la duración de tiempo hacia el futuro en el que la evolución del sistema se predice desde el instante inicial. V_f es el coste terminal, para garantizar la estabilidad del MPC.

Cuyo objetivo es encontrar la secuencia de actuaciones de control que minimizan el valor de V_N

$$u = [u(0), u(1), u(2), \dots, u(N-1)] \qquad (5\text{-}7)$$

En el caso particular de este trabajo, la función a minimizar es la siguiente [14]:

$$\min_{\mathrm{x,u}} \sum_{i=0}^{i=N-1} \left(\left|\left|x_i - x_r\right|\right|_Q^2 + \left|\left|u_i - u_r\right|\right|_R^2 \right) + \left|\left|x_N - x_r\right|\right|_T^2 \qquad (5\text{-}8)$$

Donde $\left|\left|x_i\right|\right|_P = \sqrt{x^T P x}$. T es una función de coste para el estado final del sistema.

La forma en la que se implementa la estrategia es mediante la siguiente estructura de control.

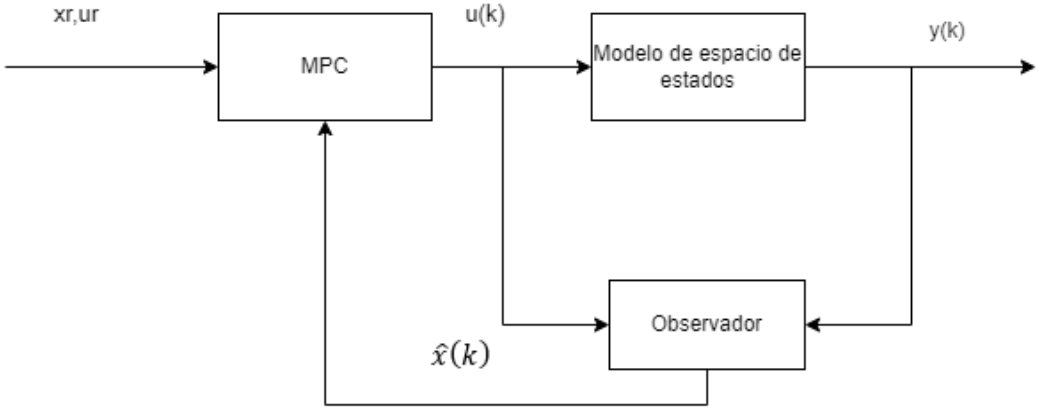

Figura 5-11. Esquema control MPC.

Para la implantación del control MPC se va a utilizar el toolbox SPCIES MPC [14].

Esta Toolbox para Matlab tiene la función de generar solvers para controladores de modelo predictivo (MPC), con la posibilidad de utilizar distintas formulaciones del MPC.

Es capaz de generar código en C o en funciones *mex* para Matlab, de forma que pueda ejecutarse en otras plataformas, como Python o Arduino. Dentro de los ficheros descargados existen tutoriales para aprender a utilizar la herramienta.

Su instalación es muy sencilla, hay que descargar el repositorio de GitHub en una carpeta $SPCIES$, entonces mover el directorio de trabajo de Matlab a esa carpeta creada y ejecutar *spcies('install')* en la barra de comandos.

El código se ha implementado en Matlab en LQR_ambos_sinTCLab.m (8.1).

Al inicio de esta sección, se he explicado una de las ventajas del MPC y es la posibilidad de definir restricciones a los estados y a la acción de control, esto es posible con el toolbox de Matlab SPCIES.

Se define el sistema con las restricciones indicadas en (5-4) y (5-5) en Matlab. Se guardan las restricciones junto a las matrices A y B del modelo en la misma estructura, es un requisito para generar el controlador con SPCIES. Notad que el programa se ha definido de forma que el usuario puede elegir el modelo con el realizar el control, el modelo A o el modelo B.

A continuación, se realiza el diseño del controlador MPC.

En el control MPC se han modificado los valores de los parámetros de forma que el sistema tenga un comportamiento más rápido, ya que se ha visto que el sistema se controla de forma correcta, pero con una respuesta en forma de escalón dado que el valor de R era similar al de Q. Luego se va a introducir una variable a para multiplicar a la Q de la siguiente forma:

$$Q = a * eye(4)$$

Mientras que ρ no cambia de forma.

Un parámetro que no existía como concepto en el control LQR es la variable T. Se calcula de la siguiente forma:

```
[K,T] = dlqr(A,B,Q,R)
 T = diag(sum(T,2))
```

El horizonte de predicción, N, se ha decidido tomar como N = 10.

Todos estos parámetros descritos deben guardarse en una estructura de forma conjunta.

Por último, es necesario elegir las opciones con las que se va a generar el solver, entre las cuales:

- Options.rho: Tamaño del paso del algoritmo ADMM usado para resolver el problema de optimización.
- Options.k_max : Es el número máximo de iteraciones.
- Options.tol : Es la tolerancia de salida del solver.
- Options.in_engineering. Si es 0, indica que estamos trabajando con variables incrementales, x,u,y, si es igual a 1, que se utilizan variables de ingeniería, las X,U,Y.

Se ha empleado la formulación laxMPC [15] para calcular la ley de control.

5.2.1 Modelo A

Se ha optado por elegir un valor de $a = 100$, De forma que la evolución del sistema hacia la referencia sea rápida.

La simulación del sistema se ha llevado a cabo con una prueba con escalones en la referencia de temperatura, resolviendo el sistema de ecuaciones (2-8) para obtener un punto de equilibrio del sistema para una u_{ref} elegida manualmente. La evolución de la referencia puede observarse en la siguiente figura:

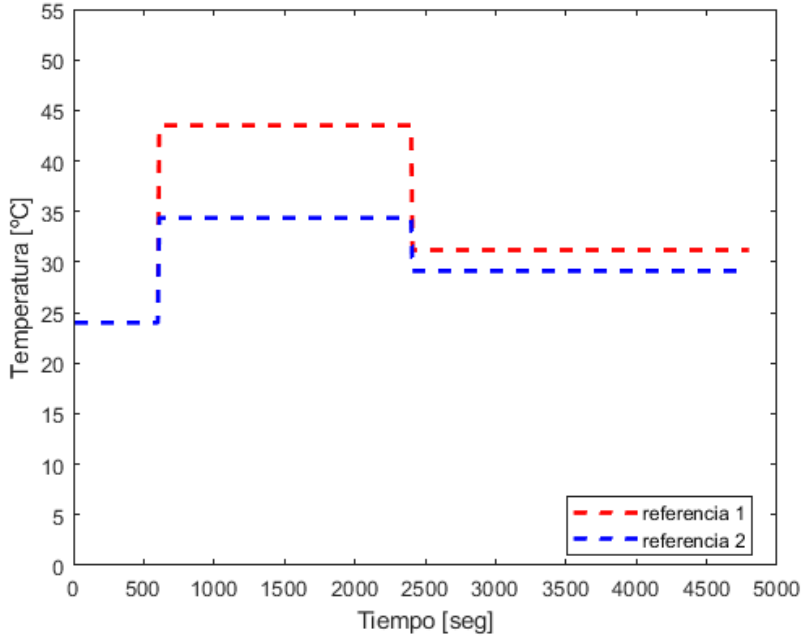

Figura 5-12. Referencia del controlador MPC.

Una vez aplicados las acciones de control al sistema, se obtiene la siguiente evolución de la salida del modelo A.

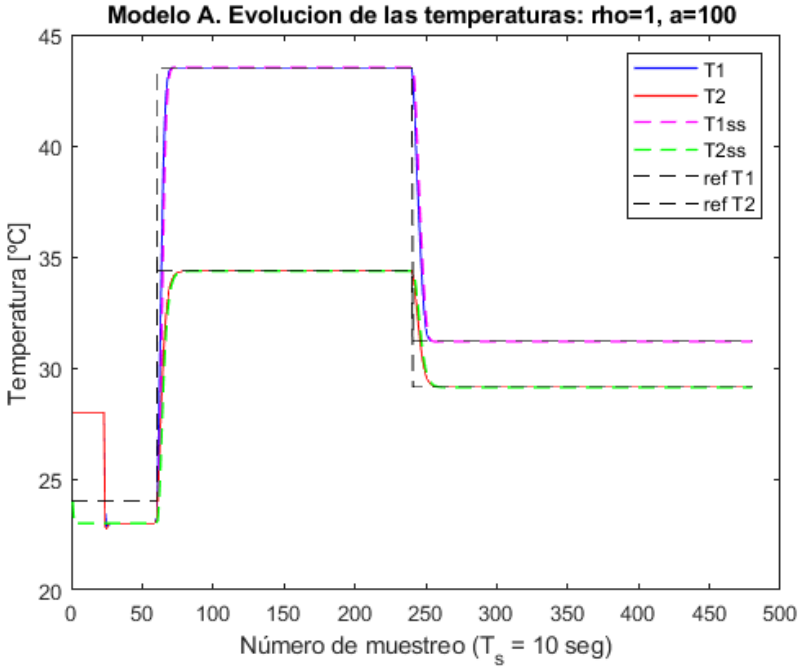

Figura 5-13. Evolución temperaturas MPC para el modelo A.

Se ha optado por modificar el significado del eje X de las gráficas para hacer entender que no se está ejecutando el sistema real en el tiempo, simplemente se calculan las iteraciones del controlador con los modelos obtenidos. Esto cambia cuando estemos utilizando el sistema real, donde el tiempo si transcurre para tener la verdadera evolución de las salidas de nuestro sistema, el TCLab. En la gráfica T1 y T2 son las temperaturas estimadas por el observador, $\hat{x}$, se observa que estas temperaturas parten de un valor estimado, $\hat{x}_0$ igual a 28 $ºC$. Este observador se activa cuando el modelo se ha estabilizado en su punto de funcionamiento y converge rápidamente al estado del modelo A alrededor de la iteración 25. Por otro lado, T1ss y T2ss, representadas en líneas discontinuas, son las salidas del modelo de espacio de estados A. Finalmente, en línea discontinua, de color

negro, se representan las referencia de temperaturas descritas en la Figura 5-12. Referencia del controlador MPC.

En la gráfica se observa que el control tarda muy poco tiempo desde que se produce el escalón en la referencia hasta que el sistema llega a estabilizarse en torno a la referencia. Las acciones de control que se obtienen mediante el algoritmo del MPC son las siguientes:

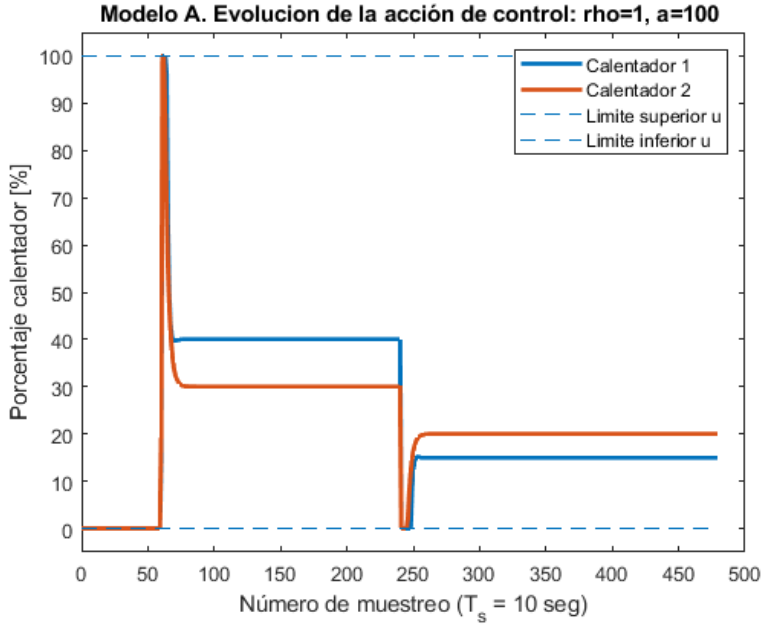

Figura 5-14. Evolución de la actuación de control MPC para el modelo A.

La acción de control cumple en todo momento con las restricciones de control impuestas.

Si se calcula el tiempo de cálculo del algoritmo MPC implementado:

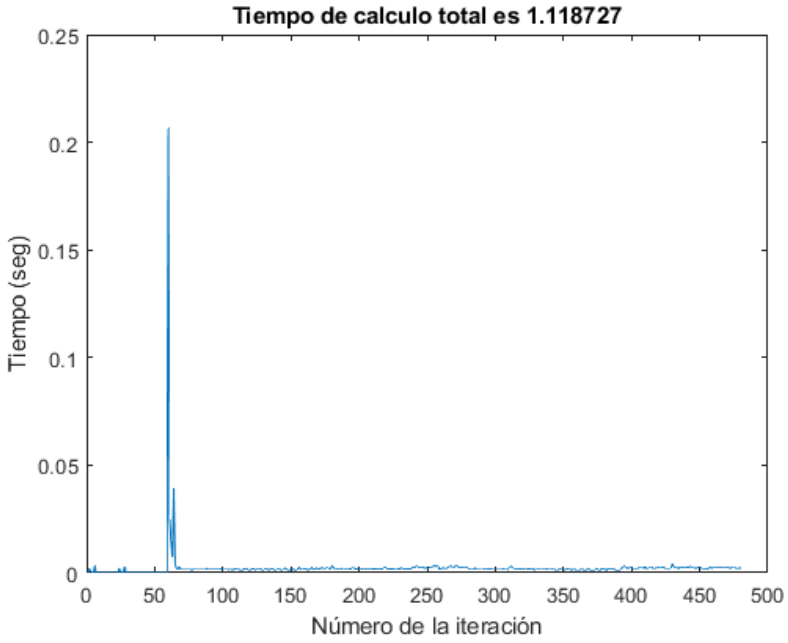

Figura 5-15. Tiempo de cálculo MPC modelo A.

Se observa que el MPC tiene tiempo de cálculo total del orden de un segundo, debido a un pico cuando se conecta el observador. Por cada iteración es del orden de 0.002 segundos, según la media de todos los tiempos de cálculo.

5.2.2 Modelo B

En este apartado se trata la aplicación del controlador MPC al modelo B. En este modelo podemos observar la importancia de la elección del parámetro a, de forma que afecta a la salida y el control de forma muy significativa. La referencia que se ha utilizado es igual a la introducida con el modelo A.

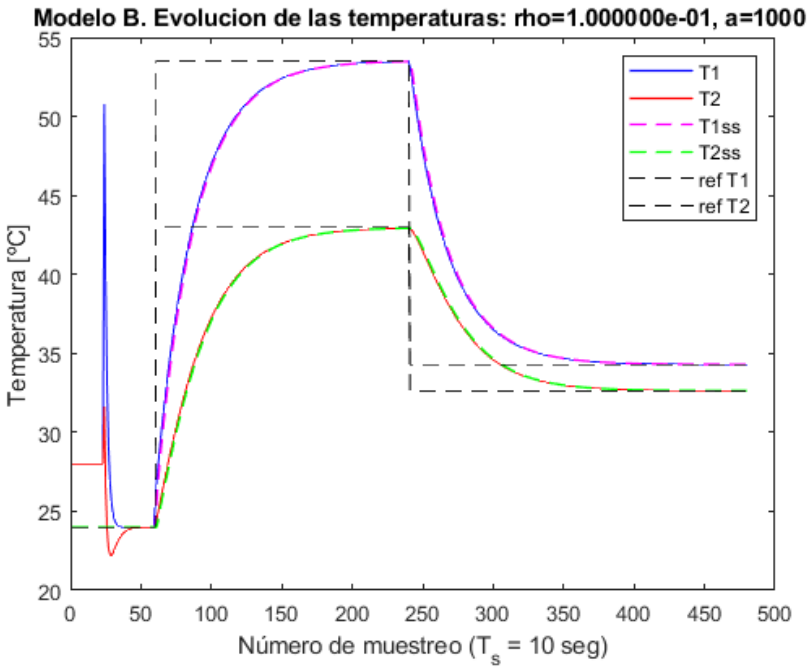

Figura 5-16. Evolución temperaturas MPC para el modelo B si $a = 1000$.

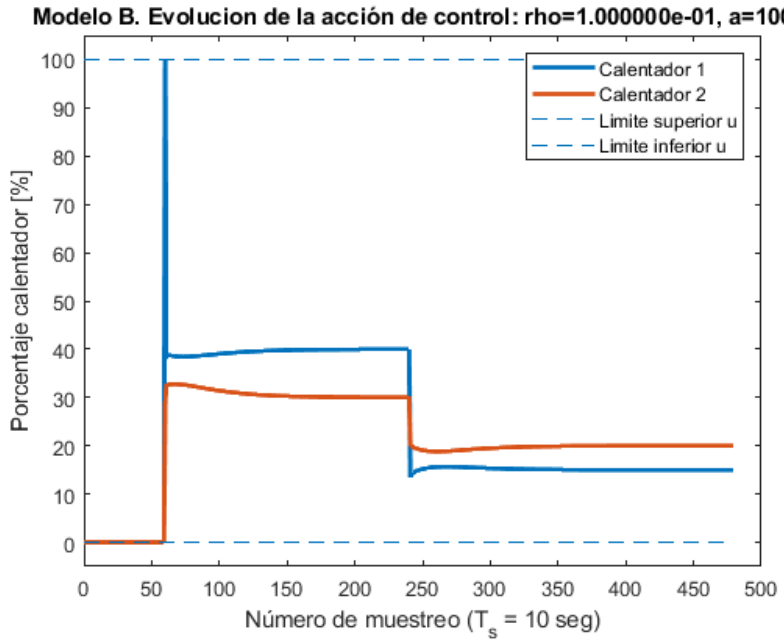

Figura 5-17. Evolución de la actuación de control MPC para el modelo B si $a = 1000$.

Puede observarse que la señal de control es prácticamente una señal de forma cuadrada, esto puede ser debido a que R sea mayor que Q lo que provoca que el sistema aplique para la referencia (x_r, u_r) que es adonde queremos llevar el sistema, la acción de control $u = u_r$, de forma que el sistema evoluciona de una forma lenta hacia el punto donde queremos llegar. Pero si prefiere llegar antes a la referencia, se puede hacer aumentando el valor de $a = 100000$.

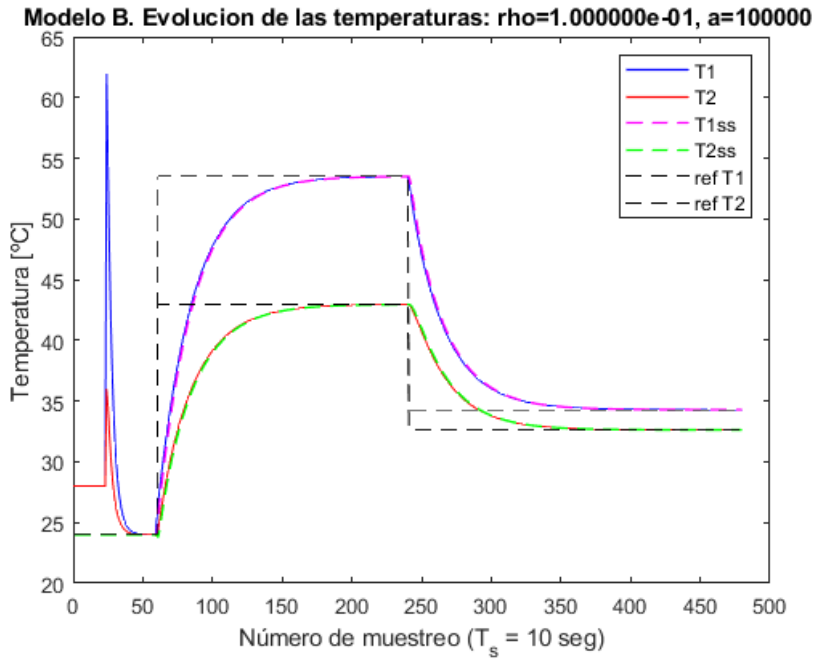

Figura 5-18. Evolución temperaturas MPC para el modelo B si $a = 100000$.

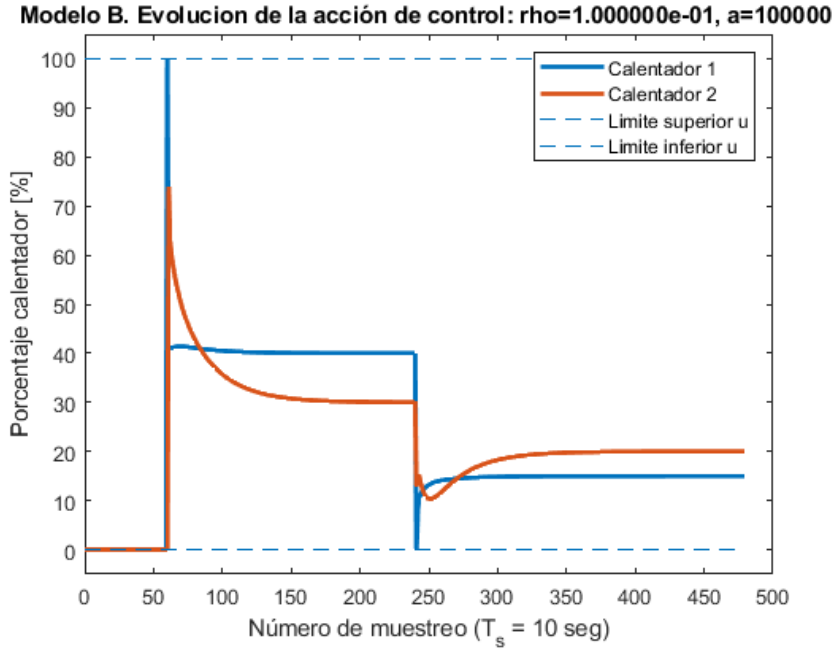

Figura 5-19. Evolución de la actuación de control MPC para el modelo B si $a = 1000$.

Si se representa el tiempo de cálculo del algoritmo MPC en la siguiente figura:

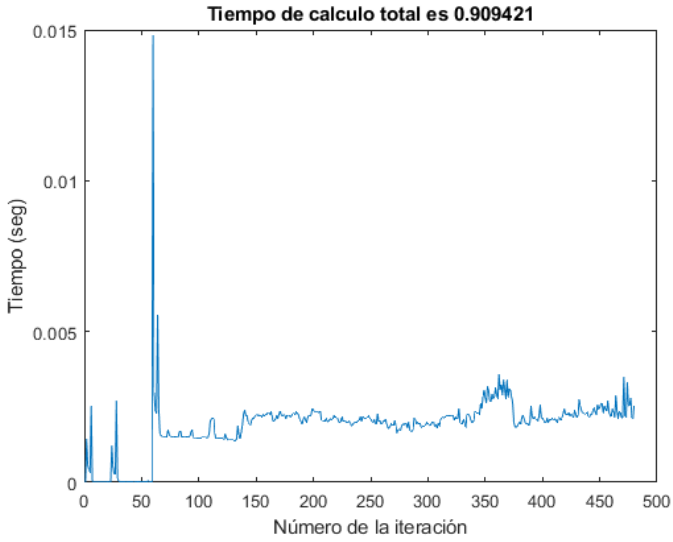

Figura 5-20. Tiempo de cálculo MPC modelo B.

Los órdenes de magnitud del tiempo de cálculo son prácticamente idénticos al del modelo A, con un tiempo por iteración en torno a 0.002 segundos.

Además, para evitar el pico de temperatura en la salida del sistema, se puede cambiar el estado inicial del observador, definiéndolo de forma muy próxima al punto de funcionamiento del modelo B, un vector de ceros. Esto se puede observar en la siguiente figura:

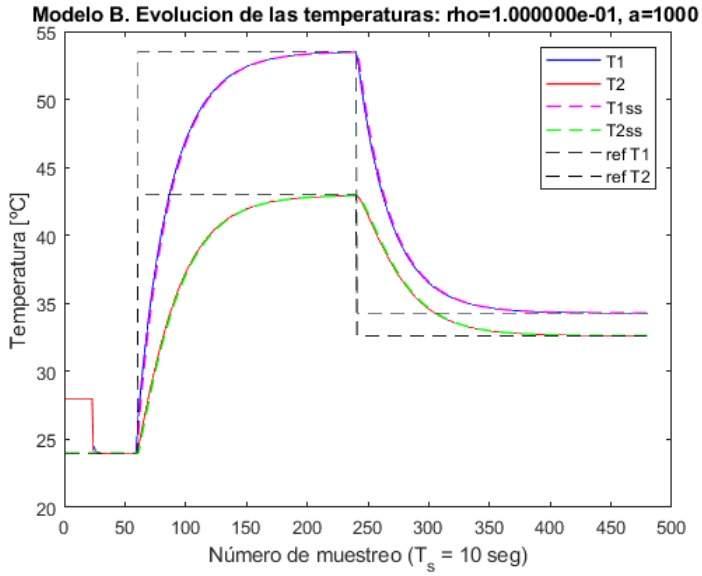

Figura 5-21. Evolución de la temperatura MPC, B, modificando estado inicial observador.

5.2.3 Compilación del controlador en Python

Se ha desarrollado una sencilla aplicación del bucle de control en Python, mediante el uso del toolbox SPECIES que permite generar código C. Para poder ejecutar el código C en Python se ha utilizado el package ctypes, que permite crear tipos de datos compatibles con lenguaje C en Python, y permite realizar llamadas a librerías externas, por ejemplo, en este caso, funciones de C compiladas.

Para obtener la función de C con el controlador, se utiliza el proceso siguiente:

Se hace todo el bucle de control de la misma forma que en Matlab, se ha utilizado en este caso el modelo A, hasta llegar a las opciones del solver, después hay que elegir el directorio donde guardar el solver a crear y

mediante la función `spcies_gen_controller`, utilizando el argumento `'platform'` como `'C'` y se guarda el solver en código C en la ubicación especificada, con el fichero .c y el .h correspondiente.

En Windows 64 bits para poder compilar la función de C, es necesario realizar la instalación del compilador, en mi caso he optado por MinGW-w64, que permite la compilación en sistemas operativos de 64 bits. Es importante una vez instalado añadir el directorio donde se ha realizado la instalación al PATH de Windows, de forma que pueda ejecutar en la consola de comandos de Windows. La instrucción que ejecutar en la siguiente:

```
gcc -o lax_solver.so -shared -fPIC -O2 lax_solver1.c
```

Luego se genera el fichero lax_solver.so en la ubicación que nos encontremos.

Este fichero se va a utilizar como archivo precompilado para utilizar mediante ctypes en Python y poder calcular la acción de control desde Python. Para ello se ha desarrollado la función mpc_function_helper.py, cuya función es definir la función equivalente en Python para el controlador.

Una vez realizado se utiliza el fichero MPC_python.py para realizar el bucle de control, para ello se han cargado las matrices del observador obtenidas anteriormente en Matlab y se ha programado de forma equivalente a Matlab, pero empleando la sintaxis propia de Python, con la ayuda del paquete Numpy.

El resultado en una prueba sencilla, mediante una simulación del sistema con un cambio de referencia de la temperatura de salida del modelo A. El sistema parte del punto de funcionamiento para estabilizar el valor del observador en torno al punto de funcionamiento. En el instante $t = 500$ segundos se introduce un cambio del valor de la consigna de control del sistema, con un aumento de la temperatura. Se puede observar en la siguiente figura la evolución de las variables del sistema:

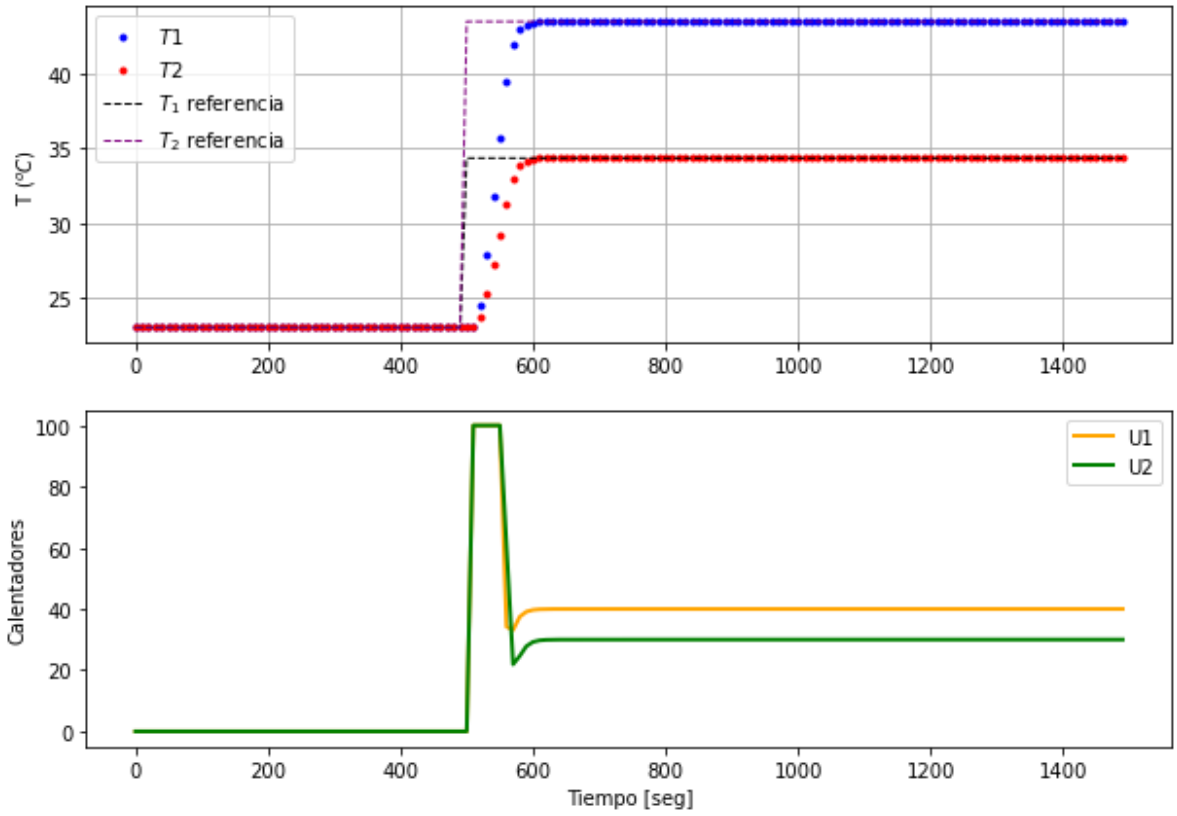

Figura 5-22. Gráfica implementación MPC en Python.

Donde T1 es el valor de la salida de temperatura T1 del modelo A, $T_{1\ \text{referencia}}$ es la temperatura de referencia para la salida del sistema, con línea discontinua, y de la misma forma para las T2. En el gráfico de abajo se puede observar la evolución en el tiempo de simulación de las entradas del sistema, en % de los calentadores.

6 RESULTADOS EXPERIMENTALES

La novedad presente en esta sección del trabajo es que se va a introducir el sistema real en lugar de los modelos de espacio de estados, aplicando los controladores diseñados en la sección 5. Se van a observar las consideraciones prácticas que hay que tener en cuenta a la hora de realizar el control de un sistema real y las diferencias entre la planta TCLab y los modelos linealizados.

Se han tenido diversas situaciones que pueden alterar los resultados y afectar al control.

En este apartado se van a detallar conceptos importantes que hay que tener en cuenta a la hora de programar ambos controladores en Matlab.

Es importante tener en cuenta que estamos trabajando con el sistema real, por lo que es importante que la temporización funcione de forma correcta, para realizar una lectura del sistema real cada 10 segundos. Pero no es simplemente esperar el $T_s = 10$ segundos, ya que entre el muestreo las instrucciones tardan un tiempo en ejecutarse.

Para ello se emplean los comandos de Matlab tic y toc para calcular el tiempo de cálculo en el bucle de control y finalmente, realizar una espera de $10 - t_{calculo} - 0.015$ segundos, el -0.015 se añade porque Matlab tiene un ligero retraso en la señal de espera y así espera un tiempo más exacto.

El bucle de control debe realizar las siguientes funciones:

-Esperar a que el sistema se estabilice en el punto de funcionamiento. Esto depende de la temperatura ambiental del momento en el que se realice la prueba. Tener en cuenta que esto es de vital importancia para asignar los valores iniciales de las variables, así que es de las primeras operaciones que se realizan en los programas diseñados. Para obtener el funcionamiento se mide la temperatura y se espera un tiempo, si la temperatura medida tras el transcurso del tiempo no ha variado de forma significativa, podemos considerar que la temperatura está estabilizada si no han ocurrido fenómenos externos. Esto es importante ya que, si no se tiene en cuenta y se hacen dos pruebas seguidas, al sistema no le ha dado tiempo a enfriarse, lo que altera de forma significativa los resultados de la prueba.

-El observador debe converger con el estado del sistema. Para ello se lee la salida del sistema, la temperatura medida en los sensores al inicio del bucle y después se calcula la evolución de los $\hat{x}$ del observador hasta que el observador converge a una buena estimación del sistema real.

-Una vez estabilizado el observador, se puede calcular la acción de control y aplicarla al TCLab, ya sea LQR o MPC. En cada instante de muestreo empezamos leyendo la salida de temperatura en $ºC$ del sistema y la almacenamos en un vector. Con esta información se calcula la acción de control, con la formulación LQR o MPC. Una vez calculada la acción de control, se actualiza el estado del observador y se almacena el valor de los estados estimados.

La estructura de control es la misma que la de las figuras Figura 5-11. Esquema control MPC. con la diferencia de que en este ocasión se sustituyen los modelos linealizados del sistema que se han obtenido en las secciones **¡Error! No se encuentra el origen de la referencia.**, por el sistema real, el TCLab, de forma que se lee la salida d el sistema en variables incrementales.

Se han realizado dos programas en Matlab, uno para cada tipo de controlador, con la posibilidad de elegir el modelo linealizado a utilizar para los cálculos del observador y el diseño del controlador.

Ya que el modelo B está linealizado respecto a la temperatura ambiente, siendo desconocida en su caso debido

a que se ha obtenido mediante técnicas de identificación. Se ha tomado la decisión de utilizar la temperatura ambiental como punto de funcionamiento del modelo. Para ello, se ha decidido que estabilizar el sistema TCLab en torno al punto de funcionamiento es la primera función que ejecutar en el programa. Para ver si el sistema está enfriándose debido a una prueba anterior y no se encuentra en equilibrio se ha realizado una medición de la temperatura de los sensores, se esperan 15 segundos y se vuelve a realizar una medición. En el caso de que la temperatura hubiera variado por encima de un umbral, se repite el proceso. Por el otro lado, si la temperatura apenas ha cambiado, se guarda esa temperatura como punto de linealización del Modelo B. En cualquier caso, el modelo A está linealizado respecto a una temperatura de 23 $ºC$.

6.1 Control LQR

El control LQR se ha utilizado en los dos modelos, de la misma forma, con la referencia para la salida de la temperatura, medida en el TCLab siguiente:

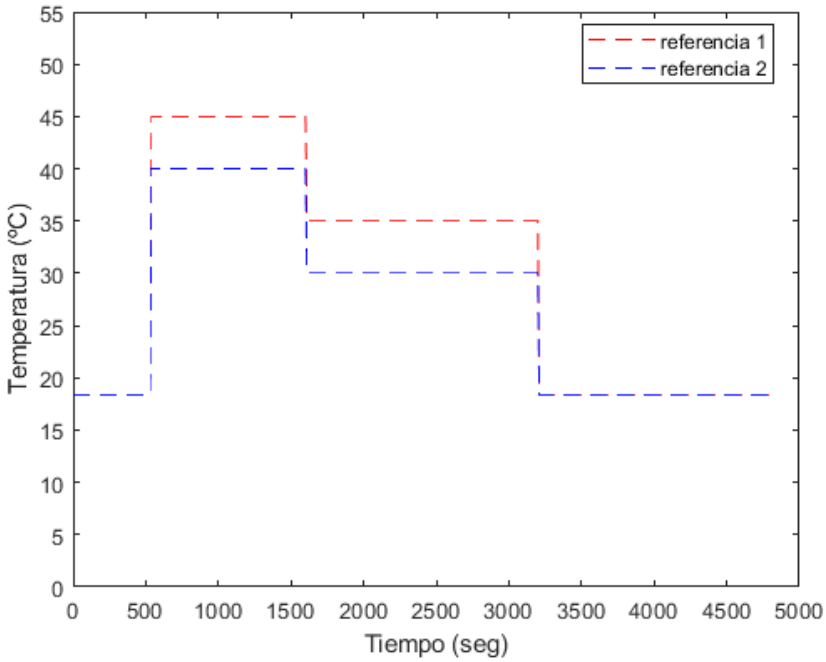

Figura 6-1. LQR referencia de temperatura.

El sistema parte de la temperatura ambiental, que varía según el momento en el que se hace la prueba, y espera a que se estabilice el observador, después se aplica un salto de temperatura hasta los [45,40] $ºC$, a continuación, en $t = 1600\ seg$ se produce un descenso de la temperatura cuyo valor es [35,30] $ºC$ hasta los $t = 3200\ seg$, finalmente se vuelve a la temperatura ambiente.

6.1.1 Modelo A

El modelo A se han obtenido los siguientes resultados para la referencia:

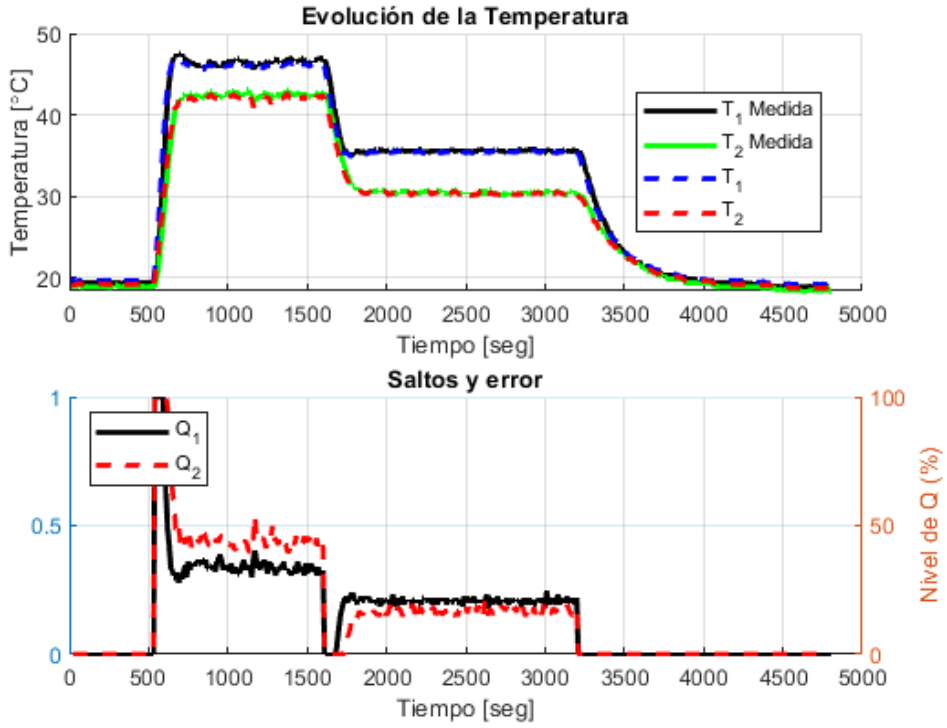

Figura 6-2. Gráfica en tiempo real, LQR, modelo A.

Esta gráfica ha sido importante para comprobar los resultados de la prueba en tiempo real, de forma que si se ha cometido algún error en el código no se ha tenido que esperar todo el tiempo de simulación a generar las gráficas.

T_1 y T_2 son las temperaturas estimadas por el observador, $\hat{x}$. T_1 medida y T_2 medida son las temperaturas que se van midiendo en los sensores del TCLab a medida que se aplican las acciones de control al sistema.

A continuación, se van a mostrar las salidas del sistema TCLab en comparación con la referencia de temperatura.

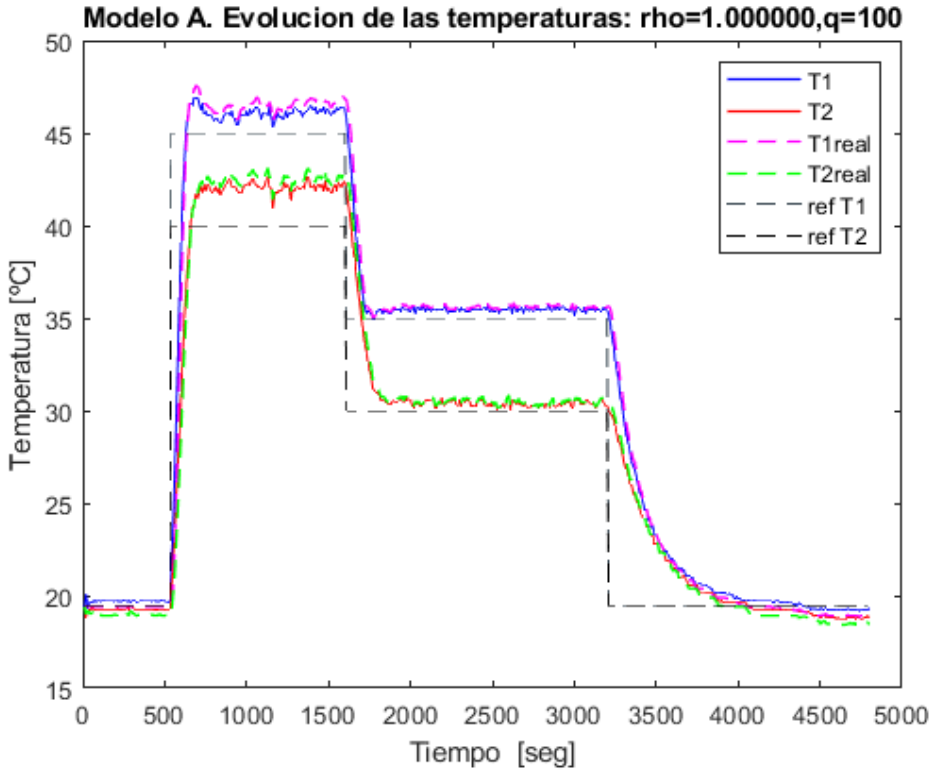

Figura 6-3. Control LQR salidas del TCLab, modelo A.

Se puede observar el error cometido en la linealización del modelo A, ya que cuanto más nos alejamos del punto de funcionamiento del sistema, más error se comete respecto a la referencia de temperatura.

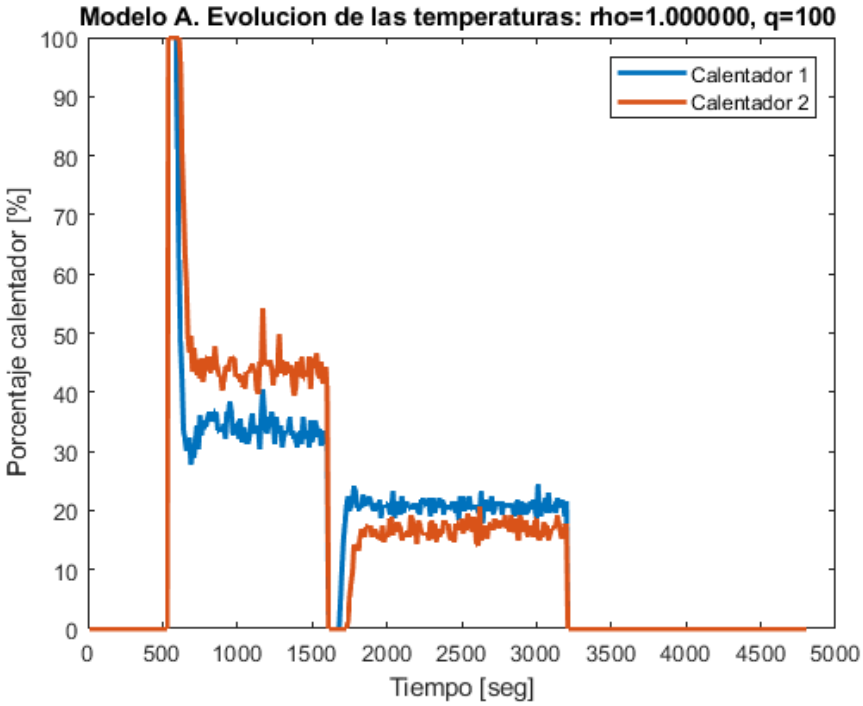

Figura 6-4. Control LQR acciones de control con TCLab, modelo A.

Esta figura es muy interesante, dado que se observa que, con el sistema real, la señal no es cuadrada, a diferencia de la Figura 5-6. Control LQR entradas modelo A., donde la acción de control es prácticamente de forma escalonada, esto es debido a la interferencia de los factores externos, como la presencia de una persona cerca del equipo, corrientes de aire, variaciones de la temperatura ambiental, que afectan a la temperatura del TCLab, provocando que, evidentemente, la temperatura real tenga pequeñas variaciones, que el controlado intenta corregir.

Además, el LQR no tiene capacidad de imponer restricciones a la acción de control, se ha realizado una restricción de forma manual con los valores límites de las entradas [0-100], pero el controlador LQR intenta aplicar acciones de control fuera de estos límites en los picos superiores de la gráfica de 100 o inferiores de 0.

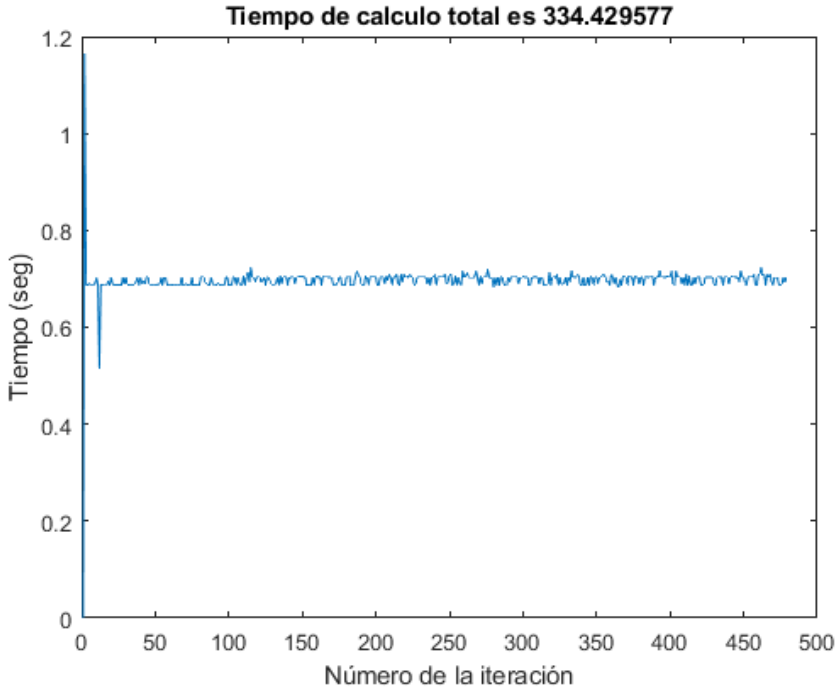

Figura 6-5. LQR con TCLab, tiempo de cálculo, modelo A.

Este tiempo de cálculo es tan elevado debido a que en cada iteración se lee las entradas del sistema por el puerto y además representar gráficamente en directo la evolución de las temperaturas y las entradas para seguir la evolución del sistema mientras se hacen pruebas. En la Figura 5-7 y Figura 5-10 se observa que el tiempo de cálculo del controlador LQR es del orden de 10^{-5} segundos por iteración del controlador LQR.

6.1.2 Modelo B.

La referencia introducida es similar a la del modelo A, teniendo en cuenta que la temperatura ambiental tenía un valor distinto.

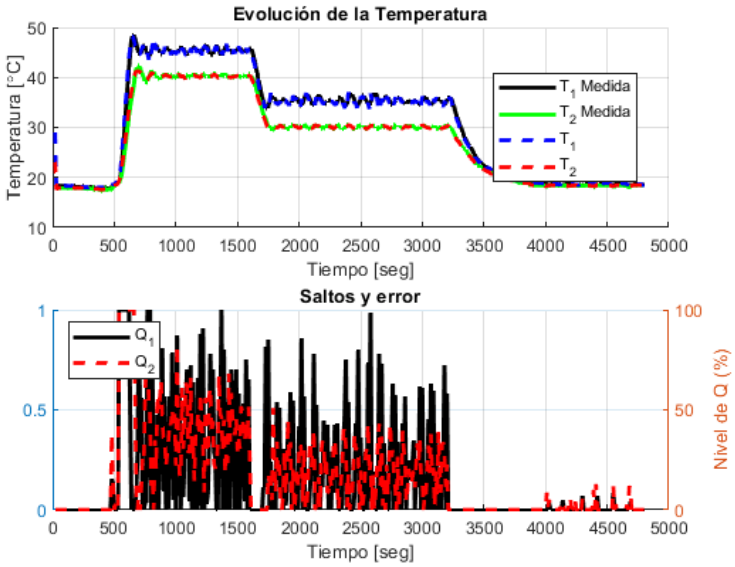

Figura 6-6. Gráfica en tiempo real, LQR, modelo B.

Se muestran las temperaturas de salida del sistema, en comparación con la referencia indicada al controlador LQR.

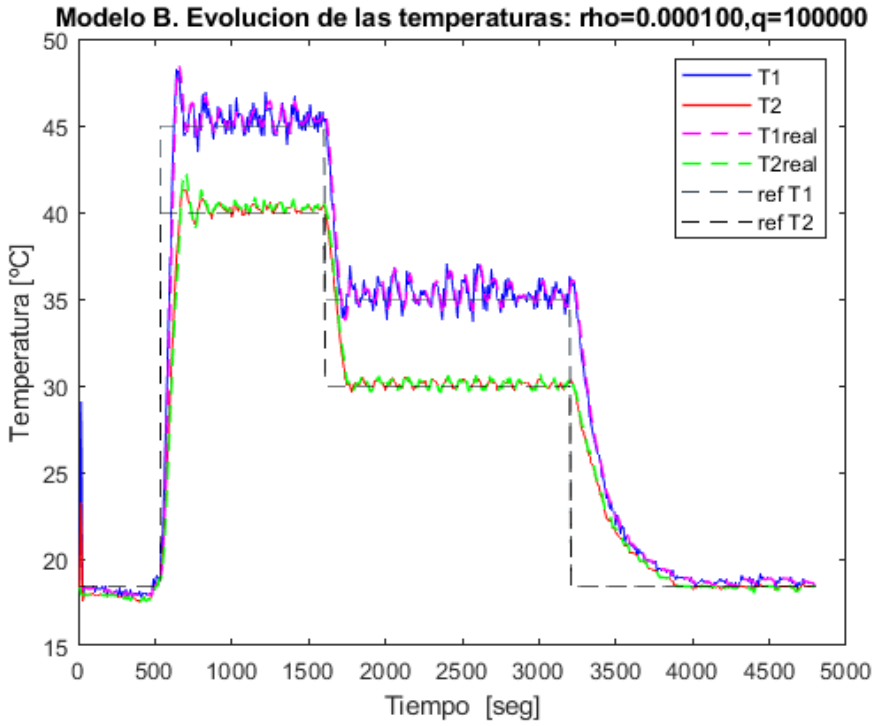

Figura 6-7. Control LQR salidas del TCLab, modelo B.

El control realizado es mucho más enérgico, como se observa en la gráfica siguiente sobre las acciones de control aplicadas a los calentadores del TCLab.

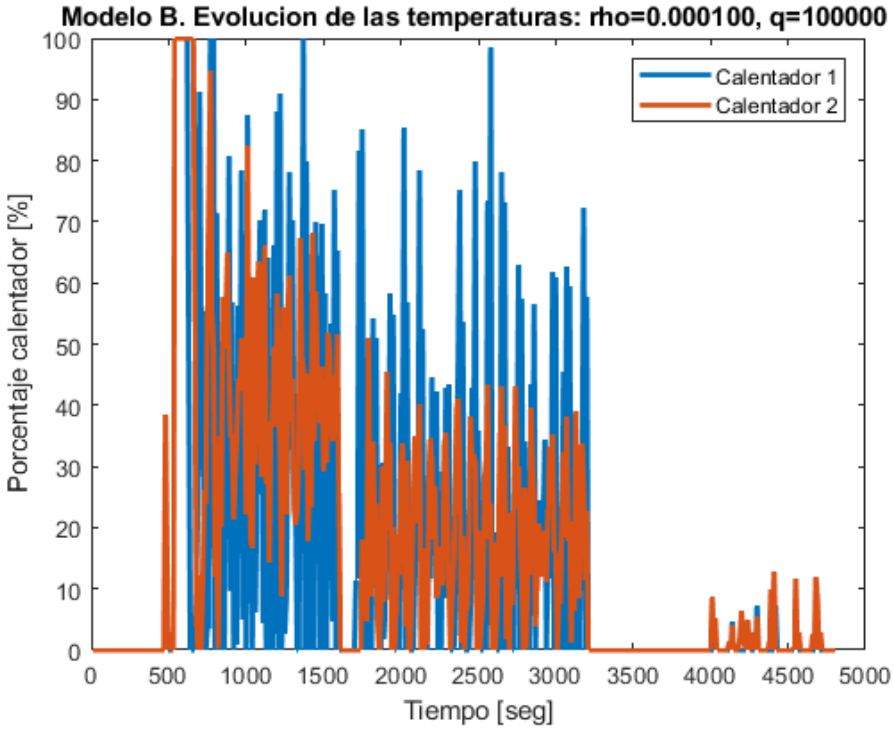

Figura 6-8. Control LQR acciones de control con TCLab, modelo B.

Esta situación tan distinta de las acciones de control puede ser debido a la elección del parámetro Q realizada, que tiene un valor distinto al del modelo A.

6.2 Control MPC

La referencia de temperatura introducida al controlador MPC ha cambiado un poco, se ha definido de forma distinta para los dos modelos, el modelo A, termina en el punto de linealización, 23ºC. Por el otro lado, el modelo B termina en su punto de linealización, la temperatura ambiente inicial. Se muestran a continuación:

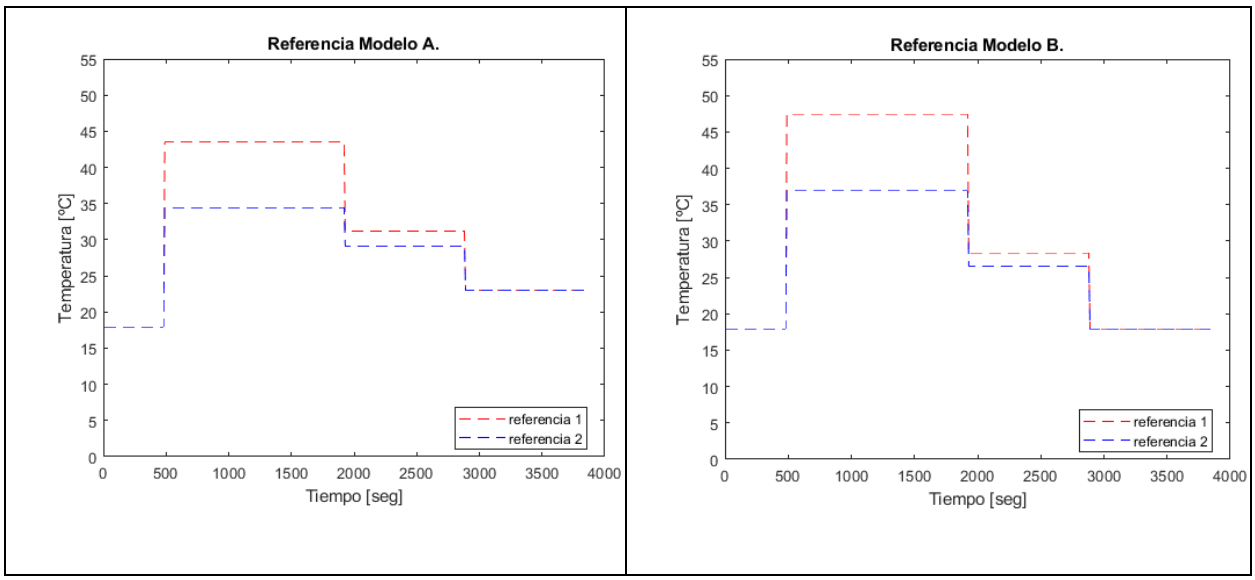

Figura 6-9. Referencias MPC, modelo A y B.

La prueba es de 64 minutos en total para el controlador MPC.

6.2.1 Modelo A

Las variables son iguales a las definidas en el LQR.

Si representamos las salidas del sistema junto con la referencia de temperatura introducida:

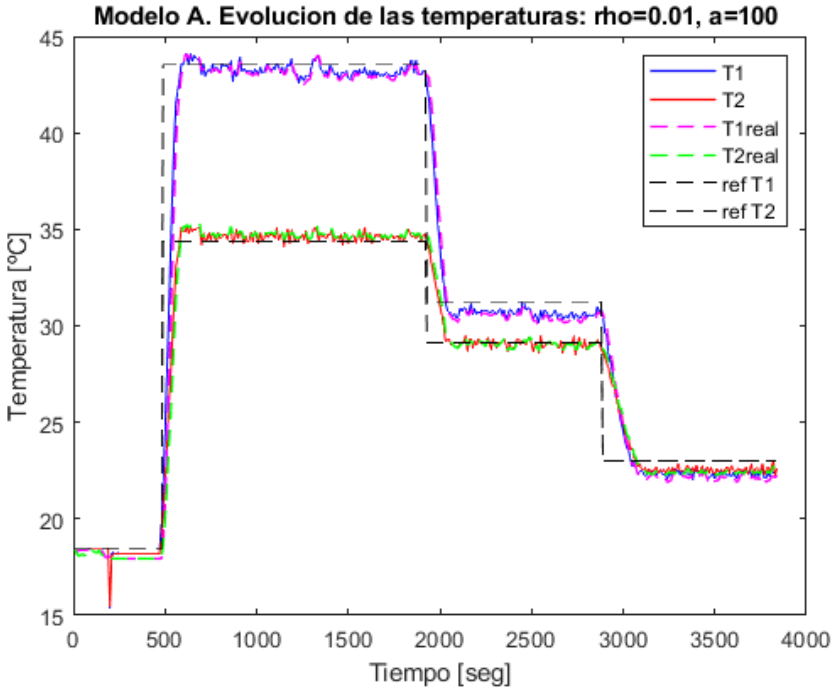

Figura 6-10. Control MPC salidas del TCLab, modelo A.

A continuación, las entradas aplicadas al TCLab.

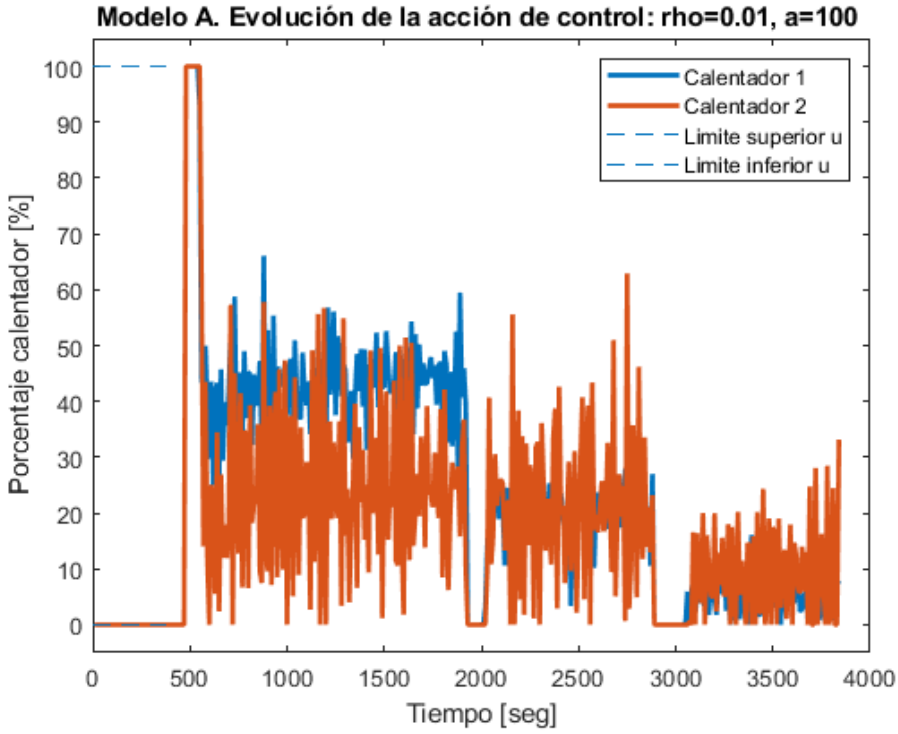

Figura 6-11. Control MPC, acciones de control con TCLab, modelo A

Se puede observar que este controlador tiene una importante ventaja y es la posibilidad de implementar las restricciones a la señal de control del sistema de forma autónoma.

El tiempo de cálculo es el siguiente:

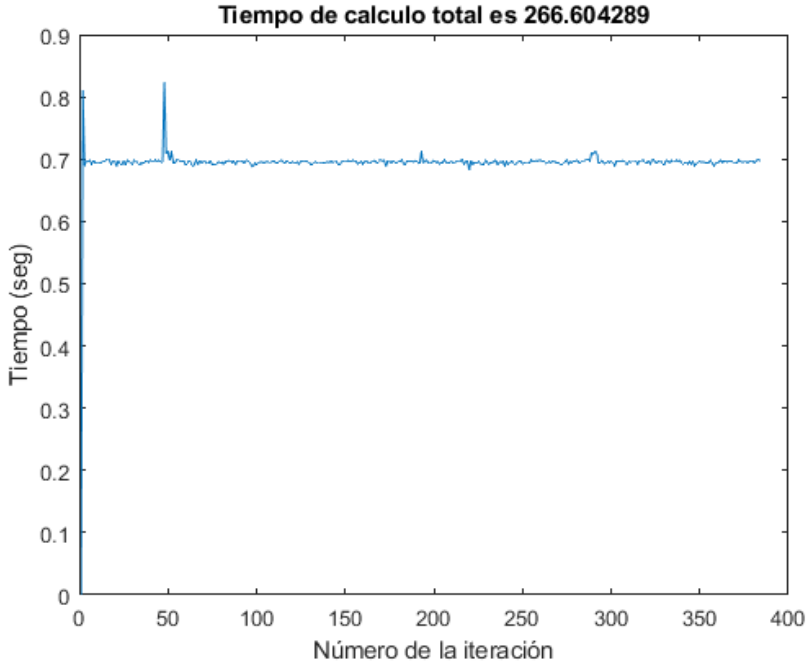

Figura 6-12. MPC con TCLab, tiempo de cálculo, modelo A.

En la Figura 5-15 y Figura 5-20, se ha representado el tiempo de cálculo del algoritmo MPC, que está en torno a 0.02 segundo por iteración.

6.2.2 Modelo B.

Se ha simulado durante el mismo tiempo que Si se compara la salida con la referencia deseada:

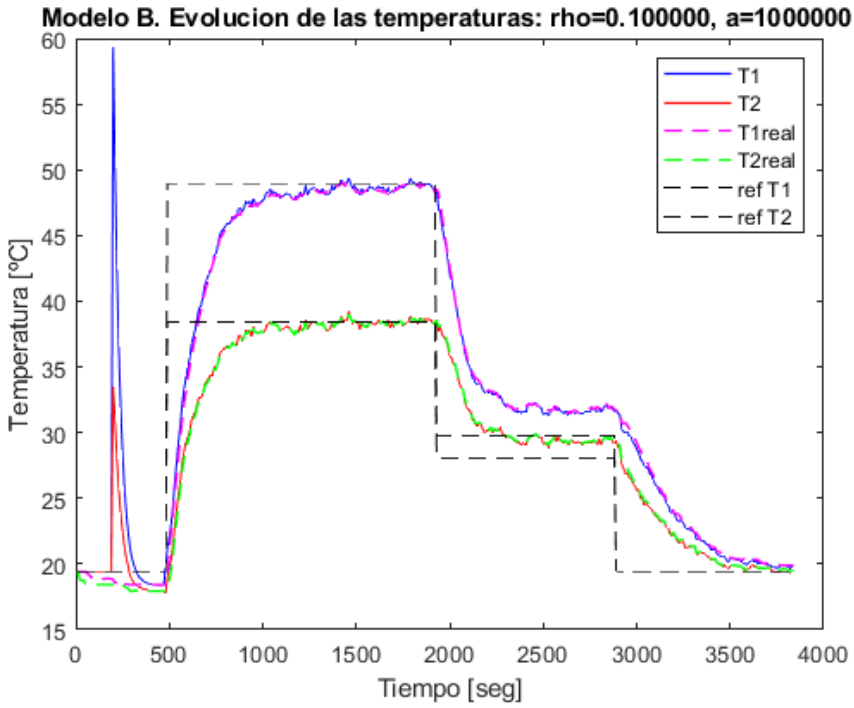

Figura 6-13. Control MPC salidas del TCLab, modelo B.

Se observa un pico en la salida estimada del observador porque se está ajustando a partir de ese punto. Por algún motivo, parece que este modelo no responde bien ante la dinámica de enfriamiento de la planta, por lo que comete un error significativo.

A continuación, las entradas aplicadas a los calentadores producidas mediante el controlador MPC:

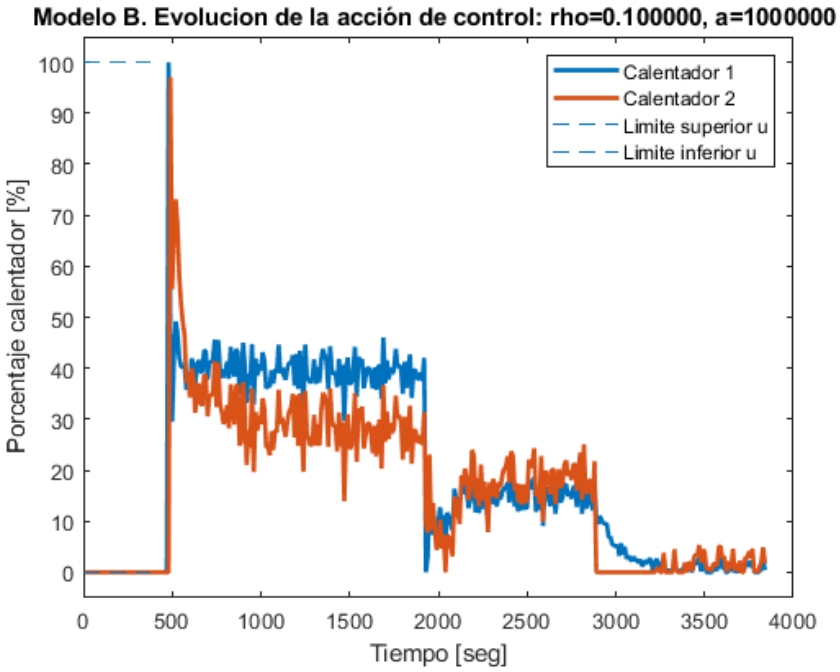

Figura 6-14. Control MPC acciones de control con TCLab, modelo B.

El tiempo de cálculo del bucle de control total durante los 64 minutos de simulación es el siguiente:

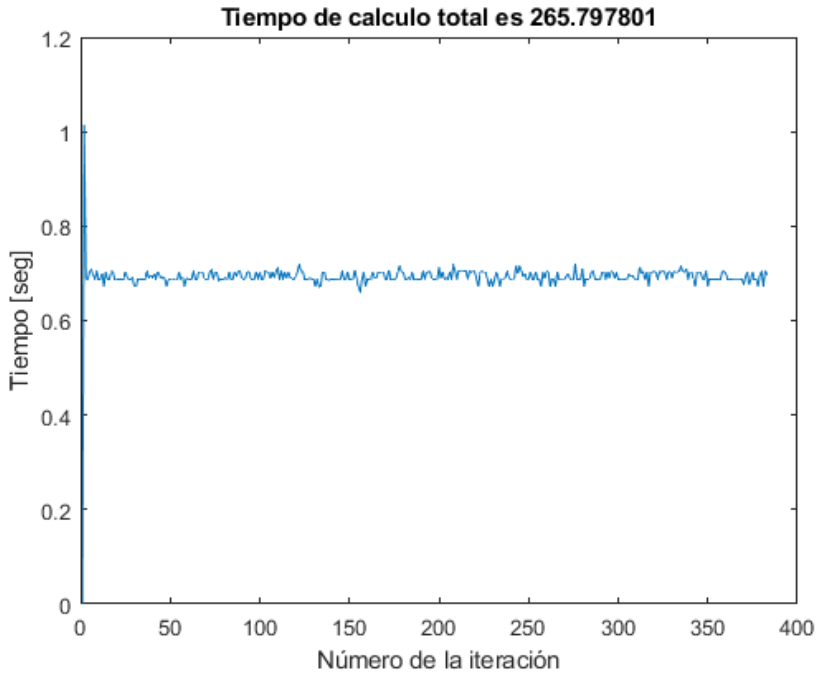

Figura 6-15. MPC con TCLab, tiempo de cálculo, modelo B.

Los tiempos de cálculo son en la práctica, idénticos a los del Modelo A

Se han recogido la evolución del control MPC con ambos modelos. En la siguiente sección se van a comparar los controladores MPC y LQR.

6.3 Comparación controladores.

Considero interesante comparar los resultados que producen ambos controladores sobre el sistema real, utilizando el modelo A, como forma de examinar la diferencia entre las metodologías. Para poder comparar los controladores se han ajustado las referencias para que tengan la misma forma, realizando una prueba con el TCLab con una duración de una hora.

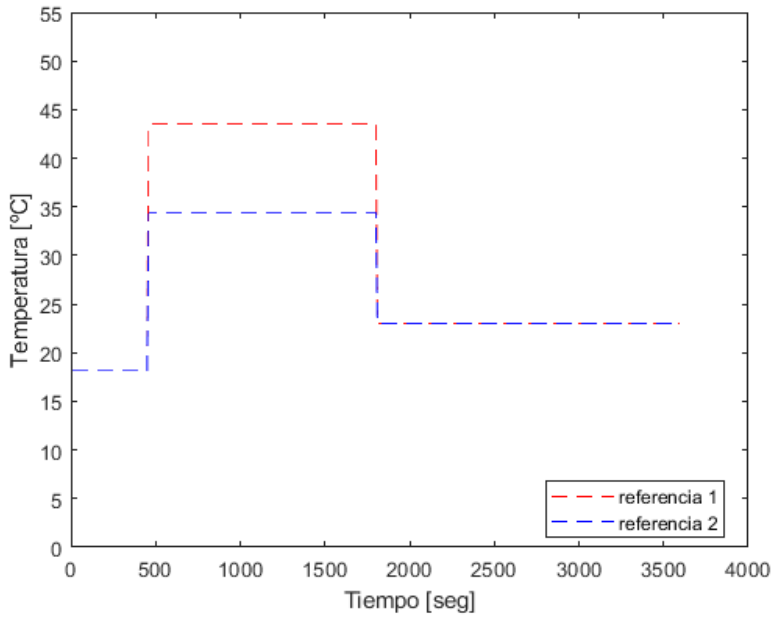

Figura 6-16. Referencia comparación controladores.

Donde el sistema parte de la temperatura ambiente y a los $t = 450$ segundos, hay un salto de la referencia, para tener una temperatura de salida de [43,51, 34,36] $ºC$. Posteriormente en el instante de tiempo $t = 1800$ segundos se vuelve al punto de linealización del modelo con las temperaturas igual a [23,23] $ºC$, hasta la finalización de la prueba cuando $t = 3600$ segundos.

Una vez realizado el bucle de control se han obtenido los siguientes resultados:

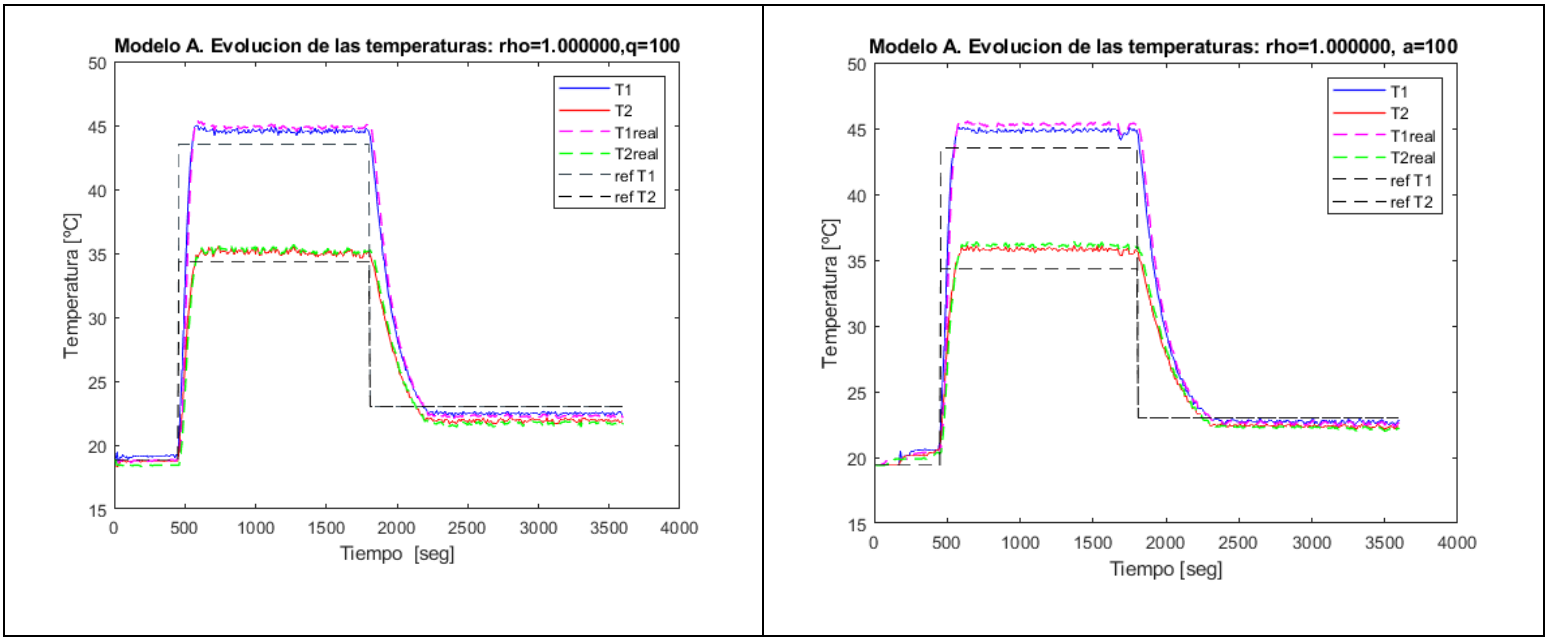

Figura 6-17. Salidas de los controladores.

A la izquierda se ha colocado el LQR, mientras que a la derecha el MPC. El error en régimen permanente es menor para el LQR. Ambos modelos parecen cometer un ligero error en torno al punto de linealización, debido a las perturbaciones existentes que no se han podido eliminar.

A continuación, se muestran las acciones de control aplicadas en cada caso:

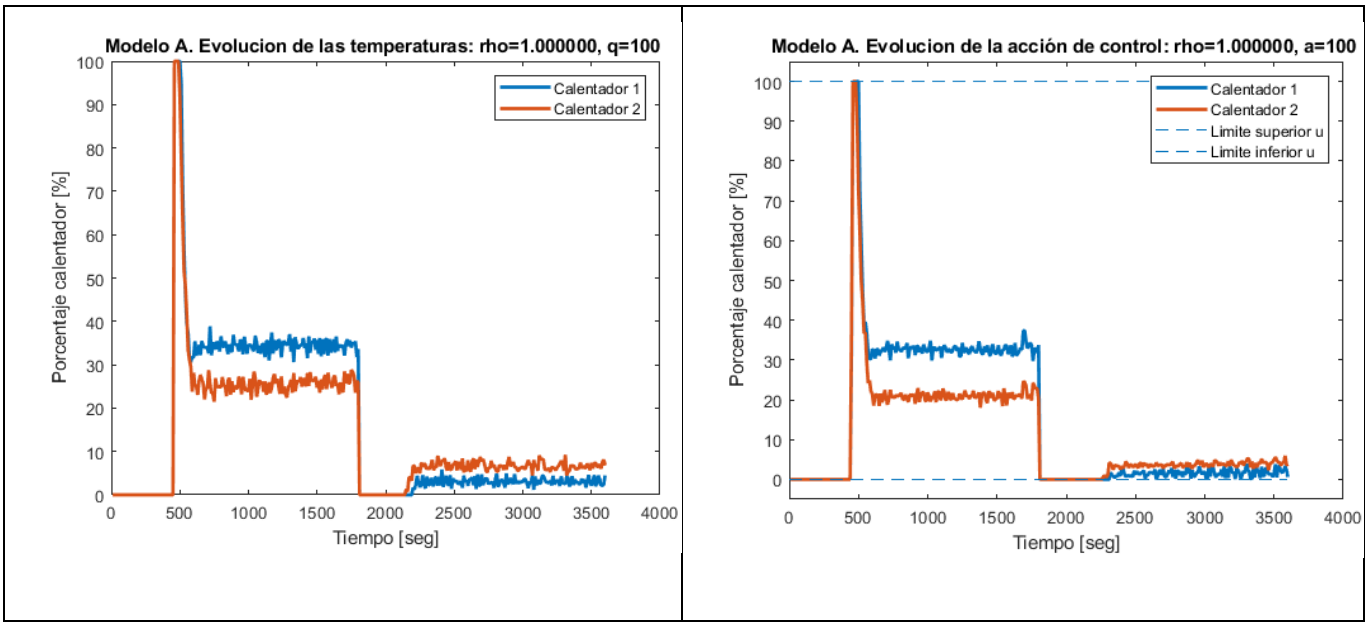

Figura 6-18. Evolución de la acción de control.

La gráficas muestran que la acción de control es muy similar en ambos controladores, pero hay que tener en cuenta que la acción de control obtenida por el LQR se ha limitado de forma manual, ya que el TCLab no acepta valores fuera de los límites establecidos, por esta razón no se representa la acción de control real LQR.

En cuanto al tiempo de cálculo para cada algoritmo de control, se va a mostrar una gráfica con los tiempos de

cada iteración y un histograma para observar los valores más comunes de tiempo de cálculo de ambos controladores:

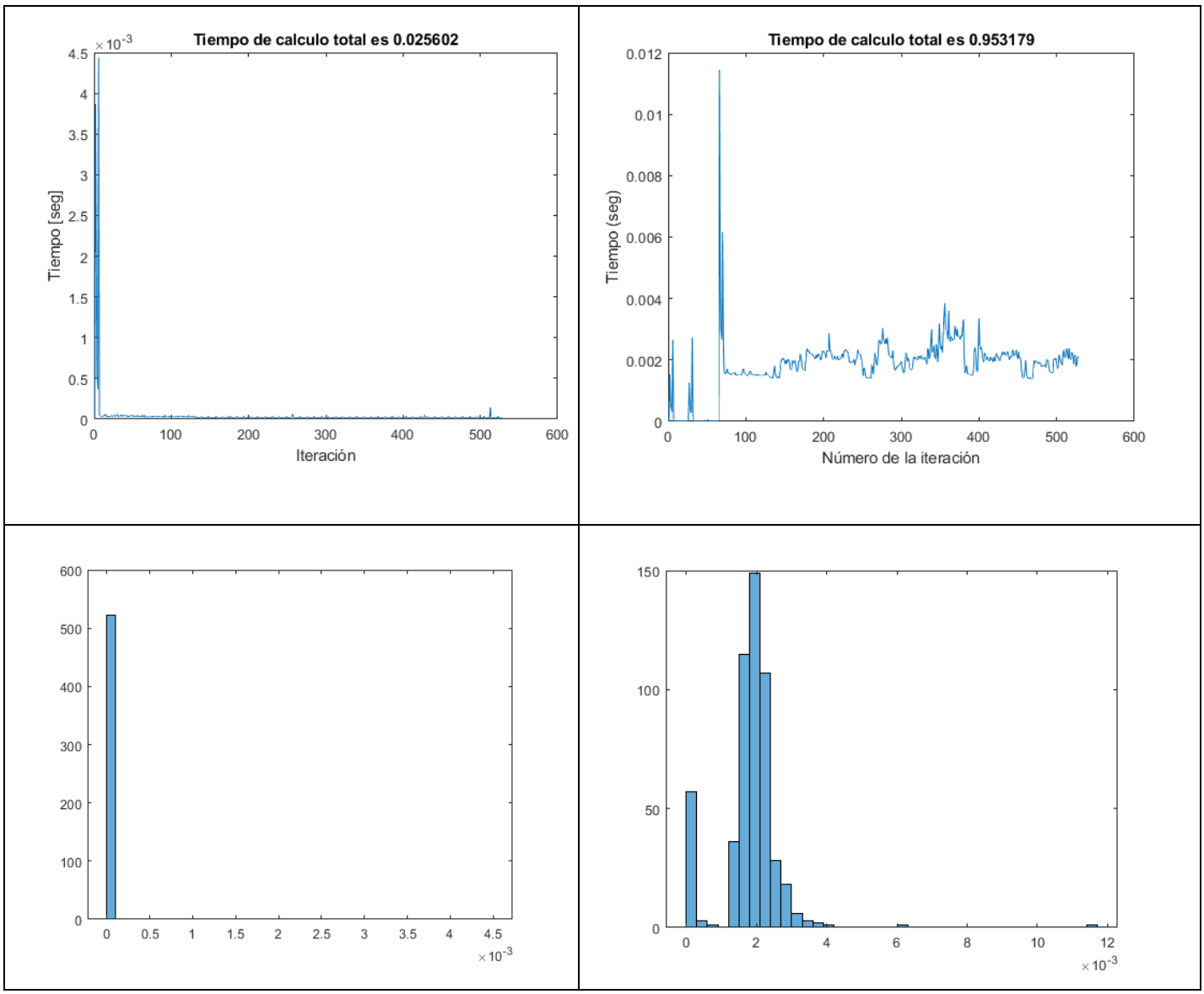

Figura 6-19. Tiempo de cálculo de los controladores LQR (izquierda) y MPC (derecha).

Los tiempos de cálculo son similares para ambos controladores, la mayor parte del tiempo de cada iteración se usa en la comunicación entre TCLab y controlador por el puerto serie, ya que, si comparamos con la Figura 6-5 y Figura 6-11, los tiempos de cálculo incluyendo el TCLab son muy superiores a los de la figura anterior.

Se observa que es superior el tiempo de cálculo del controlador MPC respecto al LQR. La media de cada iteración de tiempo para el MPC es de 0.02 segundos y la del LQR en torno a $4 * 10^{-5}$, de forma que el controlador MPC tarda casi 500 veces más en resolver los cálculos del problema de optimización que el controlador LQR.

Si se calcula el error cometido respecto a la referencia en términos de MSE:

CONTROLADOR	MSE
LQR	27,97
MPC	28,38

Tabla 6-1. Error MSE del seguimiento de la referencia.

En esta prueba en específico el controlador LQR tiene menor error en el seguimiento de la trayectoria, pero en otras pruebas realizadas con el controlador MPC, por ejemplo, con un valor de la variable $\rho = 0.01$, la salida observada en el TCLab se muestra en la siguiente figura:

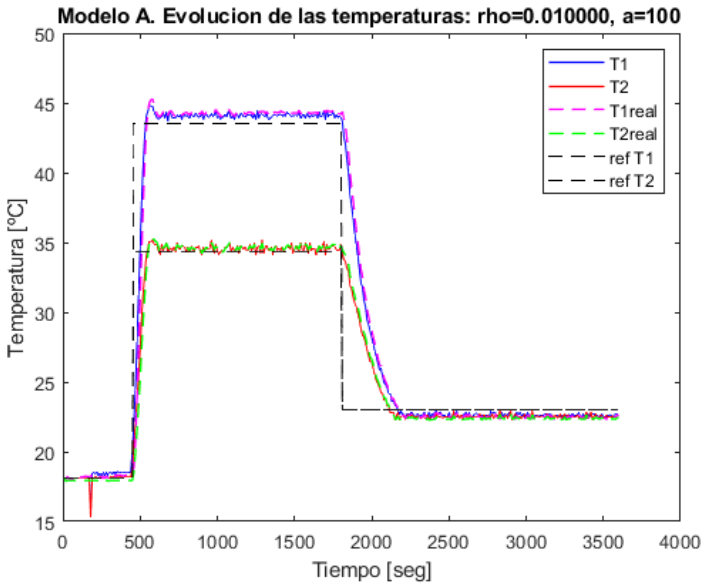

Figura 6-20. MPC prueba 2.

Si se calcula el MSE para esta prueba el error MSE es de 23,67. Es muy posible que modificar los parámetros de los controladores hayan afectado a la capacidad de seguir la trayectoria del modelo, incluso los distintos parámetros con los que se ha diseñado el observador. Finalmente es muy difícil evitar todas las posibles perturbaciones que pueden afectar a la medida de temperatura y hayan afectado a la calidad de las pruebas con el modelo real, como una temperatura ambiente distinta, corrientes de viento, presencia de personas cerca, etc.

7 CONCLUSIONES

Las conclusiones a las que hemos llegado a lo largo del desarrollo del trabajo se detallan en este capítulo, relacionándolo con los objetivos expuestos inicialmente.

Se ha comprobado que el TCLab es una herramienta útil y sencilla de utilizar, ideal para ser utilizadas en prácticas de control, para aplicar los conocimientos adquiridos sobre control a la práctica.

Se han logrado obtener modelos de espacio de estado con un comportamiento prácticamente idéntico al sistema real, aunque cuando más nos alejemos del punto de funcionamiento desde el que se han linealizado, aumenta el error respecto al propio TCLab.

Es posible controlar la temperatura del sistema real, el TCLab, mediante LQR y MPC, de forma que se ajusta con un error en régimen permanente muy bajo. De forma que se valida la posibilidad de utilizar estos controladores para aplicaciones de control de temperatura, dado que son problemas de carácter multivariable funcionan de una forma adecuada, de especial importancia actualmente debido a la necesidad de disminuir el consumo de energía de acuerdo con los tratados internaciones, como el Acuerdo de Glasgow en relación con las emisiones de $C0_2$ máximas permitidas para la industria y los países.

El control MPC parece ser más efectivo debido a la posibilidad de incluir las restricciones del sistema en el controlador, de forma que respeta los límites del sistema para calcular la acción de control. Además, permite incluir restricciones de seguridad a la temperatura del sistema en el caso del modelo A de forma sencilla, también se podría hacer en el caso del modelo B, aunque de forma más compleja.

Las ideas para expandir el alcance del trabajo se recogen a continuación:

- Implementar el controlador en el propio Arduino, de forma que sea posible la resolución del problema de control con la capacidad de cálculo del Arduino y sin tener que conectar el sistema por el puerto serie a otro ordenador donde se realizan las operaciones hasta el momento.
- Comparar los resultados obtenidos con más metodologías de controlador. Por ejemplo: PID, Control por IMC, etc.
- Desarrollar en más detalle pruebas que permitan la comparación de ambos algoritmos en condiciones externas lo más similares posibles de forma que sea posible decidir sobre que algoritmo obtiene mejores resultados en pruebas controladas.
- Incluir nuevas funcionalidades al TCLab modificando el fichero de sketch de Arduino, por ejemplo, para añadir el controlador en línea, subiéndolo a la propia memoria del Arduino.

REFERENCIAS

[1] H. G. Malkapure y M. Chidambaram, «Comparison of Two Methods of Incorporating an Integral Action in Linear Quadratic Regulator,» *IFAC Proceedings Volumes,* vol. 47, nº 1, pp. 55-61, 2014.

[2] J. Richalet, A. Rault, J. L. Testud y J. Papon, «Model predictive heuristic control,» *Automatica (Journal of IFAC),* vol. 14, nº 5, pp. 413-428, 1978.

[3] C. R. Cutler y B. L. Ramaker, «Dynamic Matrix Control. A computer control algorithm,» *Joint Automatic Control Conference,* nº 17, p. 72, 1980.

[4] S. J. Qin y T. A. Badgwell, «An Overview of Nonlinear Model Predictive Control Applications.,» *Nonlinear Model Predictive Control,* vol. 26, pp. 369-392, 2000.

[5] S. D. Cairano, «An Industry Perspective on MPC in Large Volumes Applications: Potential Benefits and Open Challenges,» *IFAC Proceedings Volumes,* vol. 45, nº 17, pp. 52-59, 2012.

[6] apmonitor, «apmonitor,» 2021. [En línea]. Available: http://apmonitor.com/pdc/index.php/Main/ArduinoTemperatureControl. [Último acceso: Noviembre 2021].

[7] Apmonitor.com, «Apmonitor.com,» Septiembre 2019. [En línea]. Available: http://apmonitor.com/pdc/index.php/Main/ArduinoModeling2. [Último acceso: Noviembre 2021].

[8] Apmonitor.com, «Apmonitor.com,» Noviembre 2019. [En línea]. Available: http://apmonitor.com/pdc/index.php/Main/ArduinoEstimation2. [Último acceso: Noviembre 2021].

[9] N. S. Nise, Control System Engineering, Wiley, 2011.

[10] K. Ogata, Ingeniería de control moderna, Pearson Educación, 2010.

[11] S. Skoogestad y I. Postlethwaite, Multivariable feedback Control: Analysis and Design (2nd ed.), John Wiley and Sons, 2005.

[12] E. F. Camacho y C. B. Alba, Model Predictive Control, Springer, 2007.

[13] J. B. Rawlings, M. David Q y M. M. Diehl, Model Predictive Control: Theory, Computation and Desing, Nob Hill Publishing, 2017.

[14] P. Krupa, D. Limón y T. Alamo, «Spcies: Suite of Predictive Controllers for Industrial Embedded Systems,» Dec 2020. [En línea]. Available: https://github.com/GepocUS/Spcies. [Último acceso: 2021].

[15] P. Krupa, D. Limón y T. Alamo, «Implementation of model predictive control in programmable logic controllers,» *IEE Transactions on Control Systems Technology,* vol. 29, nº 3, pp. 1117-1130, 2021.

8 ANEXO A

8.1 Código Matlab.

Se adjuntan todos los códigos desarrollados en Matlab con su nombre y una pequeña descripción de su función.

- **Generar_respuesta_escalon.m**

Se generan las entradas en forma de escalón y se obtiene las temperaturas tras aplicar los escalones al TCLab.

```
%Generar datos con entradas de forma en escalon

clear all; close all; clc

n = 60*30+1;  % Numero de segundos (20 min)
tiempo = zeros(n,1);
tclab
% Porcentaje de calentadores (0-100%)
Q1 = zeros(n,1);
Q2 = zeros(n,1);
Q1s = zeros(n,1);
Q2s = zeros(n,1);


for k=1:length(Q1)/40
    n_salto=1800/45;

    if logical(mod(k,2))
        amp_salto = randi([0 40],1);
        Q1(k*n_salto:k*n_salto+n_salto)= amp_salto;
        amp_salto2 = randi([0 40],1);
        Q2(k*n_salto:k*n_salto+n_salto)= amp_salto2;
    elseif not(logical(mod(k,2)))

        Q1(k*n_salto:k*n_salto+n_salto)= 0;

        Q2(k*n_salto:k*n_salto+n_salto)= 0;
    end


end

% for k=1:length(Q2)/45
%     n_salto=1800/40;
%
%     if logical(mod(k,2))     %Si es impar
%
%         amp_salto2 = randi([0 40],1);
%         Q2(k*n_salto:k*n_salto+n_salto)= amp_salto2;
%     elseif not(logical(mod(k,2)))
%
%         Q2(k*n_salto:k*n_salto+n_salto)= 0;
```

```
%     end
%
%
% end


Q=[Q1(1:1801), Q2(1:1801)];
% T. inicial. Comprobar con una medida inicial de T
T0_C1 = T1C();
T0 = T0_C1;
T0_C2 = T2C();
% Guardar los valores de temperatura
T1 = ones(n,1)*T0_C1;
T2 = ones(n,1)*T0_C2;


tiempo_inicial = clock;
tiempo_anterior = tiempo_inicial;
pausa_max = 1.0;

%Graficas animadas
figure(1)
subplot(2,1,1)
hold on, grid on
anexp1 = animatedline('LineStyle','-', 'Color', 'k', 'LineWidth', 2);
anexp2 = animatedline('LineStyle','-', 'Color', 'g', 'LineWidth', 2);

xlabel('Time (sec)')
ylabel('Temperature \circC')
legend('T_1 Measured', 'T_2 Measured', ...
    'Location', 'northwest')
title('Temperature Simulation')

subplot(2,1,2)
hold on, grid on

xlabel('Time (sec)')

yyaxis right
title('Steps')
anQ1 = animatedline('LineStyle','-', 'Color', 'k', 'LineWidth', 2);
anQ2 = animatedline('LineStyle','--', 'Color', 'r', 'LineWidth', 2);
ylabel('Power Level Q (%)')
xlabel('time (sec)')
legend('Q_1', 'Q_2', 'Location', 'northwest')

for k= 2:n

    %Punto inicial
    x0 = [T1(k-1), T2(k-1)];

    Q1s(k) = Q(k,1);
    Q2s(k) = Q(k,2);
    h1(Q1(k));
    h2(Q2(k));

    %Pausar el tiempo
    pausa_tiempo = pausa_max - etime(clock, tiempo_anterior);
    if pausa_tiempo >= 0
        pause(pausa_tiempo -0.01)
```

```
    else
        pause(0.01)
    end
    %Guardar el tiempo
    t = clock;
    dt = etime(t,tiempo_anterior);

    if k>=2
        tiempo(k) = tiempo(k-1) + dt;
    end
    tiempo_anterior = t;


    %Lectura de las temperaturas
    T1(k) = T1C();
    T2(k) = T2C();


    %Plot
    addpoints(anexp1,tiempo(k),T1(k))
    addpoints(anexp2,tiempo(k),T2(k))
    addpoints(anQ1,tiempo(k),Q1s(k))
    addpoints(anQ2,tiempo(k),Q2s(k))
    drawnow

end

h1(0);
h2(0);
disp('Calentadores apagados')

% turn off heater but keep LED on if T > 50
if (T1C() || T2C()) > 50
    led(1)
    disp(['Warning, heater temperature 1 =', num2str(T1C())])
    disp(['Warning, heater temperature 2 =', num2str(T2C())])
else
    led(0)
end
%Referir la T respecto al punto de equilibrio T0, para la identificacion
T1(:) = T1(:)-T0_C1;
T2(:) = T2(:)-T0_C2;

datos = [tiempo,Q1s,Q2s,T1,T2];
csvwrite('datos_escalon_30min.txt',datos);
clear a
```

- **Lectura_de_datos.m**

Se leen los datos que se generan con los escalones, con el fin de usar estos datos para identificar el modelo de espacio de estados B.

```
%Lectura de los datos del fichero .txt para la identificación de las
%matrices

%Se han generado escalones de forma aleatoria.
```

```
% 66 % (20 min) de los datos para estimar modelo.
%Usar el 33% final de los datos para validar el modelo.
%Las temperaturas medidas se refieren al punto de equilibrio con Tambiente.
%Tambiente era 27, pero rehacer generar_respuesta_escalon

close all; clear all; clc
%
% tclab;

datos = readmatrix('datos_escalon_30min.txt');
% datos = readtable('data2.csv');
% datos = table2array(datos);
tiempo = datos(:,1);
Q1 = datos(:,2);
Q2 = datos(:,3);
T1 = datos(:,4);
T2 = datos(:,5);

y = [T1, T2];
u = [Q1, Q2];

%Filtro
s = y;
alfa = 0.85;

for k=2:length(y)
    y(k,:) = alfa*y(k-1,:)+(1-alfa)*s(k,:);
end

%Comparar la señal medida y la filtrada
figure()
plot(s,'r','DisplayName', 'Señal sin filtrar')
hold on
plot(y,'b','DisplayName', 'Señal filtrada')
% legend('Señal sin filtrar','Señal filtrada')
xlabel('Tiempo [seg]')
ylabel('Temperatura incremental')
legend('Señal sin filtrar','','Señal filtrada')

data = iddata(y,u,1)
plot(data)

yest = y(1:1200,:);
y_val = y(1201:1800,:);

uest = u(1:1200,:);
u_val = u(1201:1800,:);

% %Para evitar fallos del puerto serie, cerrar el puerto.
% fclose( serial(a.Port) ); %Create a serial object with the port Arduino is
connected to it and close it
% clear a; %Remove the variable



%Guardar las matrices obtenidas del system identification toolbox
A = ss1.A
B = ss1.B
C = ss1.C

[A,B,C,D,K,x0] = idssdata(ss1)
```

```
x0_est1=x0;

save('Matrices', 'A','B','C','x0_est1')
```

- **Lqr_sinTCLab.m**

Diseño del controlador LQR con el modelo linealizado del TCLab.

```
%Controlador sin observador para el modelo identificado.

clear, clc

%Tiempo de muestreo es 10 seg.
Tm = 10;

%Matrices contiene la estimacion de Matlab de A,B,C y el x0. Linealizado
%respecto a que valor? A la t_ambiente.


D = zeros(2,2);
flag = 0;
mensaje = ['Elegir el espacio de estado a utilizar. \n 1 = Modelo de
primeros principios. ' ...
        '\n 2 = Modelo identificado con la respuesta ante escalones. \n'];

while flag == 0
    fprintf(mensaje)

    opcion = input('Introducir valor: ')

    if opcion == 1
        load Matrices_python
        flag = 1;
        sysc_1 = ss(A_python,B_python,C_python,D);
        ss_dis = c2d(sysc_1,Tm);
        T0 = 24;
        Tpy = 23;


    elseif opcion == 2
        load Matrices
        flag = 1;
        %El tiempo de muestreo esta en 1 segundo, pero queremos cambiarlo a
10
        %segundos para estar de acuerdo con el de python.
        sysd_1 = ss(A,B,C,D,1);
        %d2d resamplea al Tm especificado.
        ss_dis = d2d(sysd_1,Tm);
%         T0 = T1C();
        T0 = 24; %Para cuando no este tclab conectado, asigno esta Tambi
        Tpy = T0; %Para evitar tener que hacer cosas complicadas, asigno
como x0 esa.
    else
        disp('Valor no reconocido, vuelve a intentarlo.')
    end
end
```

```
%Asignar las matrices.
A = ss_dis.A;
B = ss_dis.B;
C = ss_dis.C;
D = ss_dis.D;

n=size(A,1);
m=size(B,2);


%¿Es controlable el sistema?

Controlab=ctrb(A,B)

%Es controlable si el rango de la matriz es igual al del vector de estados.
if rank(Controlab)==4
    disp("controlable")
else
    disp("error, no es controlable")
    return
end

%Si, es controlable.

%El observador tiene distinto valor según el modelo.
if opcion==1
   x0 = T0*ones(4,1);
   xe0 = 28*ones(4,1); %Valor inicial del observador.
   xincr_e = (xe0-Tpy*ones(4,1)).*ones(4,1);
elseif opcion==2
    xe0 = ones(4,1);
    x0 = ones(4,1);
    xincr_e = 0.5*ones(4,1);
end


%Parametros q y r.
%Elegir un q >> r para obtener una respuesta más rápida.


if opcion==1

    rho = 1;
%     a = 100;

elseif opcion==2
    rho = 0.1;
%       a = 100000;  %Este es el valor elegido
%     a = 1000;
end

q = C'*eye(2)*C;
r = rho*eye(2);

N=[A-eye(4),B;C,D]\[zeros(4,2);eye(2)];
K=-dlqr(A,B,q,r);
```

```
M=[-K ,eye(2)]*N;

Abc=(A+B*K);
Bbc=(B*M);

sysbc=ss(Abc,Bbc,C,D,Tm);
%Polos del sistema en b.c
disp('Polos del sistema controlado')
eig(sysbc)

%%% DISEÑO DEL OBSERVADOR %%%


if opcion==1
    ratio = 3;
elseif opcion ==2
    ratio=7;
end
polos=eig(A+B*K);  %O Abc
Polos_Obs=polos.^(ratio);
L=place(A',C',Polos_Obs)'; %Polos del observador
disp('Polos de la dinámica del observador')
eig(A-L*C)

Aobs=[A-L*C];
Bobs=[B L];
Cobs=[C; zeros(2,4)]; %Queremos h1 y h2
Dobs=zeros(4);  %Mismo tamaño que Bobs
sysobs=ss(Aobs,Bobs,Cobs,Dobs,Tm);

%Ya tengo el observador.

tmax = 500; %El tiempo total es tmax*tmuestreo = tmax*10
tiempo2 = 1:tmax;
tiempo2 = tiempo2*10;

% T. inicial. Comprobar con una medida inicial de T
T0 = 23;


%Salida.
y0 = [T0;T0];
y(1,1) = T0;
y(2,1) = T0;

u0 = zeros(2,1);

xincr = zeros(4,tmax);
uincr = zeros(2,tmax);
yincr = zeros(2,tmax);

%Los x del observador.
xincr_e = xe0;

%Referencia.
ref = ones(2,tmax);

ref(1,1:100) = T0;
```

```
ref(2,1:100) = T0;

ref1=45;
ref2=40;

ref(1,101:250)=ref(1,101:250)*(ref1);
ref(2,101:250)=ref(2,101:250)*(ref2);

ref(1,251:400)=ref(1,251:400)*35;
ref(2,251:400)=ref(2,251:400)*30;


ref(1,401:500)=ref(1,401:500)*T0;
ref(2,401:500)=ref(2,401:500)*T0;


ref_incr(1,1) = 0;
ref_incr(2,1) = 0;

uincr = zeros(2,1);

tiempo_inicial = clock;
tiempo_anterior = tiempo_inicial;
pausa_max = 10;
tiempo = zeros(tmax,1);
T1 = ones(tmax,1)*T0;
T2 = ones(tmax,1)*T0;

for k=2:tmax

    tic

    % Cálculo de la acción de control
    ref_incr(:,k)=ref(:,k)-[T0; T0];
    uincr(:,k)=K*xincr_e(:,k-1)+M*ref_incr(:,k);
    u(:,k)=uincr(:,k)+u0;

    xincr(:,k)=A*xincr(:,k-1)+B*uincr(:,k-1);
    y(:,k)=C*xincr(:,k-1)+y0;  %Usar variables incrementales.

    yincr(:,k)=y(:,k)-y0;

    xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];

    %Sumar a las variables incrementales el valor inicial .

    x_u(:,k)=x0+xincr(:,k);
    x_est(:,k)=x0+xincr_e(:,k);

    tiempo(k) = toc;
end


%Plot de los estados estimados y los del modelo.
figure()
plot(tiempo2,x_u(1,:),'b')
hold on
plot(tiempo2,x_u(2,:),'r')
```

```
plot(tiempo2,x_u(3,:),'m')
plot(tiempo2,x_u(4,:),'g')
plot(tiempo2,x_est(1,:),'b-.')
plot(tiempo2,x_est(2,:),'r-.')
plot(tiempo2,x_est(3,:),'m-.')
plot(tiempo2,x_est(4,:),'g-.')

grid on
title("Evolución de los estados")
xlabel("Tiempo (muestreo 10 seg)")
ylabel("Temperatura °C")
legend('x1','x2','x3','x4','x1 estimada','x2 estimada','x3 estimada', 'x4
estimada')

%Plot de los valores de temperatura.
figure()
subplot(2,1,1)
plot(tiempo2,y(1,:))
hold on
plot(tiempo2,y(2,:))

%Plot de la referencia de temperatura.
plot(tiempo2,ref(1,:),'k--')
plot(tiempo2,ref(2,:),'k--')
str=sprintf("Evolución de las temperaturas modelo B, rho = %f",rho);
title(str)
xlabel("Tiempo (muestreo 10 seg)")
ylabel("Temperatura °C")
ylim([0 50])
grid on
legend('T1','T2','T1ref','T2ref','Location','best')


%Plot de las alturas.
subplot(2,1,2)
hold on
plot(tiempo2,u(1,:))
plot(tiempo2,u(2,:))
grid on
title('Evolución de las entradas')
legend('U1','U2','Location','best')
xlabel("Tiempo (muestreo 10 seg)")
ylabel("% Calentador")

%Tiempo de calculo
figure
plot(tiempo)
total = sum(tiempo)
str = sprintf('Tiempo de calculo total es %f',total)
title(str)
ylabel('Tiempo [seg]')
xlabel('Iteración')
```

- **Lqr_ambos_modelos_TCLab.m**

Cálculo del controlador LQR para los modelos con el TCLab conectado.

```
%Controlador para ambos modelos con el TCLab.

%Diseño del parámetro rho.
clear, close all

tclab

%Tiempo de muestreo es 10 seg.
Tm = 10;
duracion = 80*60; %tiempo en segundos
tmax = duracion/10; %El tiempo total es tmax*tmuestreo = tmax*10
tiempo2=1:tmax;
tiempo2=10*tiempo2;
%Divisible entre 8 o poner round a referencia.


%Matrices contiene la estimacion de Matlab de A,B,C y el x0. ¿Linealizado
%respecto a que valor? A la t_ambiente de generar_respuesta_escalon.m
%En el primer caso es 27 °C
D = zeros(2,2);
flag = 0;
mensaje = ['Elegir el espacio de estado a utilizar. \n 1 = Modelo de
primeros principios. ' ...
        '\n 2 = Modelo identificado con la respuesta ante escalones. \n'];

while flag == 0
    fprintf(mensaje)

    opcion = input('Introducir valor: ')

    if opcion == 1
        load Matrices_python
        flag = 1;
        sysc_1 = ss(A_python,B_python,C_python,D);
        ss_dis = c2d(sysc_1,Tm);
        Tpy = 23; %La T a la que se ha calculado es espacio de estados de
python
%         T0 = T1C();

    elseif opcion == 2
        load Matrices
        flag = 1;
        %El tiempo de muestreo esta en 1 segundo, pero queremos cambiarlo a
10
        %segundos para estar de acuerdo con el de python.
        sysd_1 = ss(A,B,C,D,1);
        %d2d resamplea al Tm especificado.
        ss_dis = d2d(sysd_1,Tm);
        %T0 = T1C(); % Asigno esta Tambiente
        %Tpy = T0;
    else
        disp('Valor no reconocido, vuelve a intentarlo')
    end
end

% T0 = T1C();
% T0 = 20; %Para cuando no este tclab conectado
Tambiente1 = T1C();
Tambiente2 = T2C();
flag2 = 0;
```

```
i=0;

%Para que se enfríe el tclab y se estabilize la temperatura.
while flag2 == 0

   Tambiente1 = T1C();
   Tambiente2 = T2C();

   pause(15)

   if abs(T1C()-Tambiente1)< 0.4 && abs(T2C()-Tambiente1)< 0.4
       flag2 = 1;
       T0 = T1C(); %Ya tengo la temp. ambiente.
       if opcion == 2 %Para evitar tener que hacer cosas complicadas, asigno
como Tpy a la T0
           Tpy = T0;
       end
   end
 i=i+15;
 display(i)

end


%Asignar las matrices.
A = ss_dis.A;
B = ss_dis.B;
C = ss_dis.C;
D = ss_dis.D;

n=size(A,1);
m=size(B,2);

switch opcion
    case 1
    x0 = T0*ones(4,1);
    xe0 = 25*ones(4,1); %Valor inicial del observador.
    xincr_e = (xe0-Tpy*ones(4,1)).*ones(4,1);
    case 2
    x0 = zeros(4,1);
    xe0 = -0.1*ones(4,1);
    xincr_e = -0.1*ones(4,1);
end

%¿Es controlable el sistema?
Controlab=ctrb(A,B);

%Es controlable si el rango de la matriz es igual al del vector de estados.
if rank(Controlab)==4
    disp("controlable")
else
    disp("error, no es controlable")
    return
end

%Si, es controlable.

%Diseño del lqr. Parámetros Q y R.
```

```
%Elegir un q >> r para obtener una respuesta más rápida.
q = C'*eye(2)*C;
%Valor de rho para R = rho*eye(2)
% rho=1;

switch opcion
    case 1
        rho = 1;
        a_q = 100;
        q = a_q*eye(4);
    case 2
        rho = 0.0001;
        a_q = 100000;
        q = C'*eye(2)*C;
end


r = rho*eye(2);
N=[A-eye(4),B;C,D]\[zeros(4,2);eye(2)];
K=-dlqr(A,B,q,r);
M=[-K ,eye(2)]*N;

Abc=(A+B*K);
Bbc=(B*M);
%Se crea el ss de bucle cerrado.
sysbc=ss(Abc,Bbc,C,D,Tm);
%Polos del sistema en b.c
disp('Polos del sistema controlado')
eig(sysbc)


%%% DISEÑO DEL OBSERVADOR %%%
%Poner el ratio elegido según el modelo
if opcion==1
    ratio = 3;
elseif opcion ==2
    ratio = 7;
end

polos=eig(A+B*K);  %O Abc
Polos_Obs=polos.^(ratio);
L=place(A',C',Polos_Obs)'; %Polos del observador
disp('Polos de la dinámica del observador')
eig(A-L*C)

Aobs=[A-L*C];
Bobs=[B L];
Cobs=[C; zeros(2,4)]; %Queremos h1 y h2
Dobs=zeros(4);  %Mismo tamaño que Bobs
sysobs=ss(Aobs,Bobs,Cobs,Dobs,Tm);

%Ya se ha obtenido el observador.

%Salida.
y0 = [T0;T0];
y(1,1) = T0;
y(2,1) = T0;
u0 = zeros(2,1);

%Los x del observador.
```

```
x= x0;
x_est = xe0;

%Definición de matrices y vectores a utilizar en el bucle de control.
uincr = zeros(2,1);
xincr = zeros(4,tmax);
% xincr_e = ones(4,tmax).*(xe0-Tpy);
yincr = zeros(2,tmax);
yssmat = zeros(2,tmax);
Tssmat1 = ones(1,tmax)*T0; %Almacena la salida del espacio de estado.
Tssmat2 = ones(1,tmax)*T0;
T1 = ones(1,tmax)*T0; %Para almacenar la variable medida mediante sensor.
T2 = ones(1,tmax)*T0;

%Referencia.
ref = ones(2,tmax);



ref(1,1:round(tmax/9))=T0;
ref(2,1:round(tmax/9))=T0;

ref1=45;
ref2=40;

ref(1,round(tmax/9)+1:round(tmax/3))=...
    ref(1,round(tmax/9)+1:round(tmax/3))*(ref1);
ref(2,round(tmax/9)+1:round(tmax/3))=...
    ref(2,round(tmax/9)+1:round(tmax/3))*(ref2);

ref(1,round(tmax/3)+1:round(2*tmax/3))=ref(1,round(tmax/3)+1:round(2*tmax/3)
)*35;
ref(2,round(tmax/3)+1:round(2*tmax/3))=ref(2,round(tmax/3)+1:round(2*tmax/3)
)*30;



% ref(1,round(2*tmax/3)+1:end)=ref(1,round(2*tmax/3)+1:end)*T0;
% ref(2,round(2*tmax/3)+1:end)=ref(2,round(2*tmax/3)+1:end)*T0;
ref(1,round(2*tmax/3)+1:end)=ref(1,round(2*tmax/3)+1:end)*Tpy;
ref(2,round(2*tmax/3)+1:end)=ref(2,round(2*tmax/3)+1:end)*Tpy;

% Plot de la referencia.
% figure()
% plot(tiempo2,ref(1,:),'r--')
% hold on
% plot(tiempo2,ref(2,:),'b--')
% legend('referencia 1','referencia 2','Location','best')
% xlabel('Tiempo [seg]')
% ylabel('Temperatura [°C]')
% ylim([0 55])

ref_incr(1,1)=0;
ref_incr(2,1)=0;
u = zeros(2,tmax);



tiempo_inicial = clock;
tiempo_anterior = tiempo_inicial;
pausa_max = Tm;
tiempo = zeros(tmax,1);
```

```
tiempo_rest = zeros(tmax,1);
T1 = ones(tmax,1)*T0;
T2 = ones(tmax,1)*T0;


%Graficas a tiempo real
figure()
subplot(2,1,1)
hold on, grid on
anexp1 = animatedline('LineStyle','-', 'Color', 'k', 'LineWidth', 2);
anexp2 = animatedline('LineStyle','-', 'Color', 'g', 'LineWidth', 2);
anpred1 = animatedline('LineStyle','--','Color', 'b','LineWidth', 2);
anpred2 = animatedline('LineStyle','--','Color', 'r', 'LineWidth', 2);
% anpredmat1 = animatedline('LineStyle','-.','Color', '#FFC1C1','LineWidth',
2);
% anpredmat2 = animatedline('LineStyle','-.','Color', '#FF7F24
','LineWidth', 2);
xlabel('Tiempo [seg]')
ylabel('Temperatura [\circC]')
legend('T_1 Medida', 'T_2 Medida', ...
    'T_1', 'T_2','Location', 'best')
title('Evolución de la Temperatura')
%Plot abajo de la entrada
subplot(2,1,2)
hold on, grid on
xlabel('Tiempo [seg]')
yyaxis right
title('Saltos y error')
anQ1 = animatedline('LineStyle','-', 'Color', 'k', 'LineWidth', 2);
anQ2 = animatedline('LineStyle','--', 'Color', 'r', 'LineWidth', 2);
ylabel('Nivel de Q (%)')
xlabel('Tiempo [seg]')
legend( 'Q_1', 'Q_2', 'Location', 'northwest')


for k=2:tmax

    tic

     %Lectura de temperaturas, la salida del sistema.
    T1(k) = T1C();
    T2(k) = T2C();

    %Leer la salida del sistema, en variable incremental
    yincr(:,k) = [T1(k); T2(k)]-[Tpy; Tpy];

%     if opcion == 2
%         yincr(:,k) = [T1(k); T2(k)];
%     end

%     xincr(:,k)=A*xincr(:,k-1)+B*uincr(:,k-1);
%     y(:,k)=C*xincr(:,k-1)+y0;  %Usar variables incrementales.
%
%     yincr(:,k)=y(:,k)-y0;

    % Esperamos hasta que el observador del sistema converga.
    if k < tmax/10


        %Se mantiene en el punto de funcionamiento.
```

```
        uincr(:,k)=uincr(:,k-1);
        u(:,k)=uincr(:,k)+u0;

        %Aplicación al sistema de la acción de control.
        h1(u(1,k));
        h2(u(2,k));

        %Actualizar observador.
        xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];
        x_est(:,k)=x0+xincr_e(:,k);
        yssmat(:,k) = C*xincr_e(:,k);
        Tssmat1(k) = yssmat(1,k)+Tpy;
        Tssmat2(k) = yssmat(2,k)+Tpy;


    %Una vez el observador converge, se calcula la acción de
    %control LQR a aplicar al sistema
    elseif k>= tmax/10

        % Cálculo de la acción de control
        ref_incr(:,k)=ref(:,k)-[Tpy; Tpy];
        uincr(:,k)=K*xincr_e(:,k-1)+M*ref_incr(:,k);


        %Limitar la señal del control según el sistema.

        if uincr(1,k)<0
            uincr(1,k)=0;
        elseif uincr(1,k)>100
            uincr(1,k)=100;
        end

        if uincr(2,k)<0
            uincr(2,k)=0;
        elseif uincr(2,k)>100
            uincr(2,k)=100;
        end

        u(:,k)=uincr(:,k)+u0;

        %Aplicación al sistema de la acción de control.
        h1(u(1,k));
        h2(u(2,k));
        %Actualizar el observador de estado.
        xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];
        x_est(:,k) = x0+xincr_e(:,k);
        yssmat(:,k) = C*xincr_e(:,k);
        Tssmat1(k) = yssmat(1,k)+Tpy;
        Tssmat2(k) = yssmat(2,k)+Tpy;

    end
    %Almaceno los distintos resultados de rho
    tiempo(k) = toc;

     %Dibujar gráficas
    addpoints(anexp1,tiempo2(k),T1(k))
    addpoints(anexp2,tiempo2(k),T2(k))
    addpoints(anpred1,tiempo2(k),Tssmat1(k))
    addpoints(anpred2,tiempo2(k),Tssmat2(k))
```

```
    addpoints(anQ1,tiempo2(k),u(1,k))
    addpoints(anQ2,tiempo2(k),u(2,k))
    drawnow

    %Espero el tiempo de muestreo
    pause(pausa_max-tiempo(k)-0.015);
    tiempo_rest(k) = pausa_max-tiempo(k)-0.015;
end

%Apagar los calentadores
h1(0);
h2(0);
disp('Calentadores apagados')


if opcion==1
    modelo='A';
elseif opcion==2
    modelo='B';
end


figure()
h1 = plot(tiempo2,Tssmat1,'b','DisplayName','T1','LineWidth',1);
hold on
h2 = plot(tiempo2,Tssmat2,'r','DisplayName','T2','LineWidth',1);

h3 = plot(tiempo2,T1,'m--','DisplayName','T1real','LineWidth',1);
h4 = plot(tiempo2,T2,'g--','DisplayName','T2real','LineWidth',1);

h5 = plot(tiempo2,ref(1,:),'--','Color','#36454F','DisplayName','ref T1');
h6 = plot(tiempo2,ref(2,:),'k--','DisplayName','ref T2');
str = sprintf('Modelo %s. Evolucion de las temperaturas: rho=%f,q=%d',
modelo,rho,a_q);
title(str)
xlabel("Tiempo  [seg]")
ylabel("Temperatura [°C]")
legend()

figure()
plot(tiempo2,u(1,:),'DisplayName','Calentador 1','LineWidth',2)
hold on
plot(tiempo2,u(2,:),'DisplayName','Calentador 2','LineWidth',2)
% line([0,tmax],[umax,umax],'DisplayName','Limite superior u','LineStyle','-
-');
% line([0,tmax],[umin,umin],'DisplayName','Limite inferior u','LineStyle','-
-');
% ylim([umin-5, umax+5]);
legend()
title('Entradas')
xlabel("Tiempo [seg]")
ylabel("Porcentaje calentador [%]")
str = sprintf('Modelo %s. Evolucion de las temperaturas: rho=%f, q=%d ',
modelo,rho,a_q);
title(str)

figure
plot(tiempo)
total = sum(tiempo);
str = sprintf('Tiempo de cálculo total es %f',total);
```

```
title(str)
ylabel('Tiempo (seg)')
xlabel('Número de la iteración')

% fclose( serial(a.Port) ); %Create a serial object with the port Arduino is
connected to it and close it
clear a; %Remove the variable
```

- **Predictivo_spcies_sintclab.m**

Cálculo del control predictivo con el toolbox SPECIES utilizando el modelo en espacio de estados.

```
%El predictivo del tclab pero sin conectar el equipo.

%Para verificar el funcionamiento correcto.

clear, clc,
close all

%Tiempo de muestreo es 10 seg.
Tm = 10;
duracion = 80*60; %tiempo en segundos
tmax = duracion/10; %Para las iteraciones y los vectores dividir entre Ts
tiempo2=1:tmax;
tiempo2=10*tiempo2;
%Divisible entre 8


%Matrices contiene la estimacion de Matlab de A,B,C y el x0. ¿Linealizado
%respecto a que valor? A la t_ambiente de generar_respuesta_escalon.m
%En el primer caso es 27 °C
D = zeros(2,2);
flag = 0;
mensaje = ['Elegir el espacio de estado a utilizar. \n 1 = Modelo de
primeros principios. ' ...
        '\n 2 = Modelo identificado con la respuesta ante escalones. \n'];

while flag == 0
    fprintf(mensaje)

    opcion = input('Introducir valor: ')

    if opcion == 1
        load Matrices_python
        flag = 1;
        sysc_1 = ss(A_python,B_python,C_python,D);
        ss_dis = c2d(sysc_1,Tm);
        T0 = 24;
        Tpy = 23;


    elseif opcion == 2
        load Matrices
        flag = 1;
```

```
        %El tiempo de muestreo esta en 1 segundo, pero queremos cambiarlo a
10
        %segundos para estar de acuerdo con el de python.
        sysd_1 = ss(A,B,C,D,1);
        %d2d resamplea al Tm especificado.
        ss_dis = d2d(sysd_1,Tm);
%         T0 = T1C();
        T0 = 24; %Para cuando no este tclab conectado, asigno esta Tambi
        Tpy = T0; %Para evitar tener que hacer cosas complicadas, asigno
como x0 esa.
    else
        disp('Valor no reconocido, vuelve a intentarlo.')
    end
end




%Asignar las matrices.
A = ss_dis.A;
B = ss_dis.B;
C = ss_dis.C;
D = ss_dis.D;

%Comprobar si es controlable el sistema.
Controlab = ctrb(A,B);
if rank(Controlab)==4
    display("Sistmea controlable")
else
    display("Error, sistema no controlable")
    return
end

%Restricciones.
Tmax = 80;
Tmin = 0;  %El mínimo es la temperatura ambiente.

umax = 100;
umin = 0;


n=size(A,1);
m=size(B,2);
D=zeros(2,2);

%Valores iniciales.


%El observador tiene distinto valor según el modelo.
if opcion==1
   x0 = T0*ones(4,1);
   xe0 = 28*ones(4,1); %Valor inicial del observador.
   xincr_e = (xe0-Tpy*ones(4,1)).*ones(n,tmax);
elseif opcion==2
    xe0 = ones(4,1);
    x0 = ones(4,1);
    xincr_e = 0.5*ones(n,tmax);
end
```

```
u0 = [0;0];

%Las restricciones del sistema. ¿En var incrementales?
LBx=Tmin*ones(4,1)-x0;
UBx=Tmax*ones(4,1)-x0;

LBu=umin*ones(2,1)-u0;
UBu=umax*ones(2,1)-u0;

sys = struct('A', A, 'B', B, 'LBx', LBx, 'UBx', UBx, 'LBu', LBu, 'UBu',
UBu);


%Diseñar el MPC.
%Estos son los valores que he probado en la realimentacion de estados para
%R y Q. R y Q deben ser semidefinidas positivas. No tienen por qué ser
%siempre triangulares.
%Si las salidas son cosas fisicas suele ser triangular (una valvula, un
%piston, ...)

%La rho se ha ajustado de forma experimental, para tener un comportamiento
%similar al bucle abierto, dado que se ha obtenido a partir de
%identificación de escalón.
%Esta forma no es válida para Spcies, al ser Q semidefinida positiva pero
%la formulación utilizada, FISTA, requiere que Q sea definida positiva.
% Q = C'*eye(2)*C;
% rho = 10;
% R = rho*eye(2);

if opcion==1

    rho = 1;
    a = 100;

elseif opcion==2
    rho = 0.1;
     a = 100000;  %Este es el valor elegido
%     a = 1000;
end

Q = a*eye(4);
R = rho*eye(2);

[K, T] = dlqr(A, B, Q, R); % Asi se obtiene la T en el MPC.
K = -K;
T = diag(sum(T,2));
N = 10; %Horizonte de predicción.

parametros = struct('Q', Q, 'R', R, 'T', T, 'N', N);


%Elegir las opciones del solver.
%El rho es importante elegirlo de forma adecuada para el ADMM, probar
valores.
options.rho = 1; % Value of the penalty parameter of the ADMM algorithm
options.k_max = 5000; % Maximum number of iterations of the solver
options.tol = 1e-3; % Exit tolerance of the solver.
```

```
options.in_engineering=0;  %Si es cero es variable incremental (por
defecto).

% solver_options.rho = 15; % Value of the penalty parameter of the ADMM
algorithm
% solver_options.k_max = 5000; % Maximum number of iterations of the solver
% solver_options.tol = 1e-3; % Exit tolerance of the solver
%
% %Todo esto es si uso el compilador gcc, para ficheros MEX.
% %Se guarda en el directorio por defecto en SPECIES/generated_solver
% options.save_name = 'lax_solver2';
% options.directory =
'C:\Users\mario\Dropbox\TFM\TCLab\Programas_tfm\Matlab\solvers';
%
% spcies_gen_controller('sys', sys, 'param', parametros, 'solver_options',
solver_options,...
%     'options', options, 'platform', 'C', 'type', 'laxMPC');
%
% %Limpiar el directorio de anteriores solvers. Evitar fallos.
% spcies_clear;

%%% DISEÑO DEL OBSERVADOR %%%
%El observador se ha diseñado con un ratio elevado para aumentar la
%velocidad de los polos.

if opcion==1
    ratio = 3;
elseif opcion ==2
    ratio=7;
end
polos=eig(A+B*K);
Polos_Obs=polos.^(ratio);
L=place(A',C',Polos_Obs)'; %Polos del observador
disp('Polos de la dinámica del observador')
eig(A-L*C)

Aobs=[A-L*C];
Bobs=[B L];
Cobs=[C; zeros(2,4)]; %Queremos h1 y h2
Dobs=zeros(4);  %Mismo tamaño que Bobs
sysobs=ss(Aobs,Bobs,Cobs,Dobs,Tm);

%Posición inicial.
x = x0;
x_est = xe0; %Se define la posición inical del observador.


%Vector inicial de referencia.
ref = ones(2,tmax);
ref(:,1:tmax/8) = T0.*ones(2,tmax/8);

%El objetivo de control
% Tcontrol(1,1) = 40;
% Tcontrol(2,1) = 30;

y0=x0(1:2,1);
y = y0;
uref_incr = ones(2,tmax);
xr = zeros(4,1);
xref = ones(4,tmax);
```

```
%Se calcula la entrada de referencia, no la salida directamente.
uref(1,1)=40-0; %El u0 es cero.
uref(2,1)=30-0;
ur_incr = uref;
%Añadir al vector de referencia de la entrada.
uref_incr(:,tmax/8:tmax/2) = uref.*uref_incr(:,tmax/8:tmax/2);

%Definición de matrices y vectores a utilizar en el bucle de control.
uincr = zeros(2,1);
xincr = zeros(n,tmax);
% xincr = ones(n,tmax).*(x0-Tpy);


yincr = zeros(2,tmax);
yssmat = zeros(2,tmax);
Tssmat1 = ones(1,tmax)*28; %Almacena la salida del espacio de estado.
Tssmat2 = ones(1,tmax)*28;
T1 = ones(1,tmax)*T0; %Para almacenar la variable medida mediante sensor.
T2 = ones(1,tmax)*T0;

%Se calcula el punto de equilibrio para esa entrada de referencia,
%resolviendo la ecuación del espacio de estados para obtener un punto de
%equilibrio.
xref1 = (sys.A - eye(n))\(-sys.B*ur_incr);
yref = C*xref1;

%Asignar a la matriz para usar en el cálculo de la acción de control en un
%bucle.
xref(:,tmax/8+1:tmax/2) = xref(:,tmax/8+1:tmax/2).*xref1;
ref(:,tmax/8+1:tmax/2)=ref(:,tmax/8+1:tmax/2).*(yref+Tpy);

%Cambio de consigna 2: enfriamiento. Se escoge una menor u de referencia.
uref = [15; 20];
ur_incr = uref; %El u0 es 0.
uref_incr(:,tmax/2+1:end) = uref.*uref_incr(:,tmax/2+1:end);

xr = (sys.A - eye(n))\(-sys.B*ur_incr);
yref = C*xr;

xref(:,tmax/2+1:end) = xref(:,tmax/2+1:end).*xr;
ref(:,tmax/2+1:end) = ref(:,tmax/2+1:end).*(yref+Tpy);

%Plot de la referencia.

figure()
plot(tiempo2,ref(1,:),'r--','Linewidth',2)
hold on
plot(tiempo2,ref(2,:),'b--','Linewidth',2)
legend('referencia 1','referencia 2','Location','best')
xlabel('Tiempo [seg]')
ylabel('Temperatura [°C]')
ylim([0 55])

%Temporización
pausa_max = Tm;  %Esto es el tiempo de muestreo.
tiempo = zeros(tmax,1);
tiempo_inicial = clock;
tiempo_anterior = tiempo_inicial;
```

```
tiempo_rest = zeros(tmax,1);


%Estabilizar en el punto de funcionamiento.
for k=2:tmax

    tic

    % la salida del sistema.
    xincr(:,k)=A*xincr(:,k-1)+B*uincr(:,k-1);
    yincr(:,k) = C*xincr(:,k);

    T1(k) = yincr(1,k)+Tpy;
    T2(k) = yincr(2,k)+Tpy;


    %Se deja inicialmente la misma u. tmax/8, hasta estabilizar el punto de
    %funcionamiento.

    if k < (tmax/20)
        uincr(:,k)=uincr(:,k-1);
        u(:,k)=uincr(:,k)+u0;


        %Revisar esto, no me convence la forma de asignar.
        %La temperatura del observador inicial es 25 eso debería
        %observarse en la evolución de la graficas
        Tssmat1(k) = Tssmat1(k-1);
        Tssmat2(k) = Tssmat2(k-1);


    %Una vez estabilizado, se activa el observador.
    %Se deja un tiempo para que se estabilice.
    elseif k >= (tmax/20) && k < tmax/8

        %Se mantiene en el punto de funcionamiento.
        uincr(:,k)=uincr(:,k-1);
        u(:,k)=uincr(:,k)+u0;

        %Aplicación al sistema de la acción de control.
%         h1(u(1,k));
%         h2(u(2,k));


%         xincr(:,k)=A*xincr(:,k-1)+B*uincr(:,k-1);
%         y(:,k)=C*xincr(:,k-1)+y0;  %Usar variables incrementales.
%
%         yincr(:,k)=y(:,k)-y0;

        %Actualizar el observador.
        xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];
        x_est(:,k) = x0+xincr_e(:,k);
        yssmat(:,k) = C*xincr_e(:,k);
        Tssmat1(k) = yssmat(1,k)+Tpy;
        Tssmat2(k) = yssmat(2,k)+Tpy;

    %Llamada a spcies. Devuelve la acción de control para la referencia.
    %Cálculo de la accion de control.
    elseif k>=tmax/8
```

```
        [uincr(:,k),iter,flag1] = spcies_laxMPC_FISTA_solver(xincr_e(:,k-
1),xref(:,k),uref_incr(:,k),...
                                'sys',sys,'param',parametros,'options',options);

%     [uincr(:,k),iter(k),flag1] = spcies_laxMPC_ADMM_solver(xincr(:,k-
1),xref_incr,ur_incr,...
%
'sys',sys,'param',parametros,'options',options);

        u(:,k)=uincr(:,k)+u0;

        %Aplicación al sistema de la acción de control.
%         h1(u(1,k));
%         h2(u(2,k));
%
        %Actualizar el observador de estado.
        xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];
        x_est(:,k) = x0+xincr_e(:,k);
        yssmat(:,k) = C*xincr_e(:,k);
        Tssmat1(k) = yssmat(1,k)+Tpy;
        Tssmat2(k) = yssmat(2,k)+Tpy;

    end

%     xincr(:,k) = A*xincr(:,k-1)+B*uincr(:,k);

%     x_est(:,k) = x0+xincr_e(:,k);
%     yssmat(:,k) = C*xincr_e(:,k);
%     Tssmat1(k) = yssmat(1,k)+T0;
%     Tssmat2(k) = yssmat(2,k)+T0;
%     x(:,k)=xincr_e(:,k)+x0;
%   u(:,k)=uincr(:,k)+u0;

    tiempo(k) = toc;

end

if opcion==1
    modelo='A';
elseif opcion==2
    modelo='B';
end

figure()
h1 = plot(Tssmat1,'b','DisplayName','T1','LineWidth',1);
hold on
h2 = plot(Tssmat2,'r','DisplayName','T2','LineWidth',1);

h3 = plot(T1,'m--','DisplayName','T1ss','LineWidth',1);
h4 = plot(T2,'g--','DisplayName','T2ss','LineWidth',1);

h5 = plot(ref(1,:),'--','Color','#36454F','DisplayName','ref T1');
h6 = plot(ref(2,:),'k--','DisplayName','ref T2');
% plot(ref(3,:),'m--')
% plot(ref(4,:),'g--')

str = sprintf('Modelo %s. Evolucion de las temperaturas: rho=%d, a=%d ',
modelo,rho,a)
```

```
title(str)

xlabel("Número de muestreo (T_s = 10 seg)")
ylabel("Temperatura [°C]")
% legend(h1,'T1',h2, 'T2',h3,'T1 medida',h4,'T2 medida',h5,'referencia
T1',h6,'referencia2')
legend()

figure()
plot(u(1,:),'DisplayName','Calentador 1','LineWidth',2)
hold on
plot(u(2,:),'DisplayName','Calentador 2','LineWidth',2)
line([0,tmax],[umax,umax],'DisplayName','Limite superior u','LineStyle','--
');
line([0,tmax],[umin,umin],'DisplayName','Limite inferior u','LineStyle','--
');
ylim([umin-5, umax+5]);
legend()
title('Entradas')
xlabel("Número de muestreo (T_s = 10 seg)")
ylabel("Porcentaje calentador [%]")
str = sprintf('Modelo %s. Evolucion de la acción de control: rho=%d, a=%d ',
modelo,rho,a);
title(str)


figure
plot(tiempo)
total = sum(tiempo);
str = sprintf('Tiempo de calculo total es %f',total);
title(str)
ylabel('Tiempo (seg)')
xlabel('Número de la iteración')
```

- **Predictivo_spcies.m**

Cálculo del control predictivo mediante SPECIES con el TCLab.

```
clear, clc, close all

tclab

%Tiempo de muestreo es 10 seg.
Tm = 10;
duracion = 64*60; %tiempo en segundos
%Divisible entre 8 o poner round a referencia.

D = zeros(2,2);
flag = 0;
mensaje = ['Elegir el espacio de estado a utilizar. \n 1 = Modelo de
primeros principios. ' ...
        '\n 2 = Modelo identificado con la respuesta ante escalones. \n'];

while flag == 0
    fprintf(mensaje)
```

```
    opcion = input('Introducir valor: ')

    if opcion == 1
        load Matrices_python
        flag = 1;
        sysc_1 = ss(A_python,B_python,C_python,D);
        ss_dis = c2d(sysc_1,Tm);
        Tpy = 23; %La T a la que se ha calculado es espacio de estados de
python
%         T0 = T1C();

    elseif opcion == 2
        load Matrices
        flag = 1;
        %El tiempo de muestreo está en 1 segundo, pero queremos cambiarlo a
10
        %segundos para estar de acuerdo con el de python.
        sysd_1 = ss(A,B,C,D,1);
        %d2d resamplea al Tm especificado.
        ss_dis = d2d(sysd_1,Tm);
        %T0 = T1C(); % Asigno esta Tambiente
        %Tpy = T0;
    else
        disp('Valor no reconocido, vuelve a intentarlo')
    end
end

% T0 = T1C();
% T0 = 20; %Para cuando no este tclab conectado
Tambiente1 = T1C();
Tambiente2 = T2C();
flag2 = 0;
i=0;

%Para que se enfríe el tclab.
while flag2 == 0

   Tambiente1 = T1C();
   Tambiente2 = T2C();

   pause(15)

   if abs(T1C()-Tambiente1)< 0.5 && abs(T2C()-Tambiente1)< 0.5
       flag2 = 1;
       T0 = T1C(); %Ya tengo la temp. ambiente.
       if opcion == 2 %Para evitar tener que hacer cosas complicadas, asigno
como Tpy a la T0
           Tpy = T0;
       end
   end
 i=i+15;
 display(i)

end

%Asignar las matrices.
A = ss_dis.A;
B = ss_dis.B;
C = ss_dis.C;
D = ss_dis.D;
```

```
%Restricciones.
Tmax = 80;
Tmin = 0;  %El mínimo es la temperatura ambiente.

umax = 100;
umin = 0;

tmax = duracion/10;
tiempo2=1:tmax;
tiempo2=10*tiempo2
n=size(A,1);
m=size(B,2);
D=zeros(2,2);

%Valores iniciales.
switch opcion
    case 1
    x0 = T0*ones(4,1);
    xe0 = 25*ones(4,1); %Valor inicial del observador.
    xincr_e = (xe0-Tpy*ones(4,1)).*ones(n,tmax);
    case 2
    x0 = ones(4,1);
    xe0 = ones(4,1);
    xincr_e = 0.5*ones(n,tmax);
end
u0 = [0;0];

%Las restricciones del sistema. ¿En var incrementales?
LBx=Tmin*ones(4,1)-x0;
UBx=Tmax*ones(4,1)-x0;

LBu=umin*ones(2,1)-u0;
UBu=umax*ones(2,1)-u0;

sys = struct('A', A, 'B', B, 'LBx', LBx, 'UBx', UBx, 'LBu', LBu, 'UBu',
UBu);


%Diseñar el MPC.
%Estos son los valores que he probado en la realimentacion de estados para
%R y Q. R y Q deben ser semidefinidas positivas. No tienen por qué ser
%siempre triangulares.
%Si las salidas son cosas fisicas suele ser triangular (una valvula, un
%piston, ...)

%La rho se ha ajustado de forma experimental, para tener un comportamiento
%similar al bucle abierto, dado que se ha obtenido a partir de
%identificación de escalón.
%Esta forma no es válida para Spcies, al ser Q semidefinida positiva pero
%la formulación utilizada, FISTA, requiere que Q sea definida positiva.
% Q = C'*eye(2)*C;
% rho = 10;
% R = rho*eye(2);

switch opcion
    case 1
        rho = 0.01;
        a_q = 100;
    case 2
```

```
        rho = 0.1;
        a_q = 1000000;
        %a_q = 100;

end

Q = a_q*eye(4);
R = rho*eye(2);

[K, T] = dlqr(A, B, Q, R); % Asi se obtiene la T en el MPC.
K = -K;
T = diag(sum(T,2));
N = 10; %Horizonte de predicción.

parametros = struct('Q', Q, 'R', R, 'T', T, 'N', N);

%Elegir las opciones del solver.
%El rho es importante elegirlo de forma adecuada para el ADMM, probar
valores.
% options.rho = 1; % Value of the penalty parameter of the ADMM algorithm
options.k_max = 5000; % Maximum number of iterations of the solver
options.tol = 1e-3; % Exit tolerance of the solver.
options.in_engineering=0;  %Si es cero es variable incremental (por
defecto).


%%Todo esto es si uso el compilador gcc, para ficheros MEX.
% %Se guarda en el directorio por defecto en SPECIES/generated_solver
% options.save_name = 'lax_solver';
% options.directory = '';
%
% %Limpiar el directorio de anteriores solvers. Evitar fallos.
% spcies_clear;

%%% DISEÑO DEL OBSERVADOR %%%
%El observador se ha diseñado con un ratio específico para cada modelo.
%De forma que el error cometido sea un mínimo.

if opcion==1
    ratio = 3;
elseif opcion ==2
    ratio = 7;
end
polos=eig(A+B*K);
Polos_Obs=polos.^(ratio);
L=place(A',C',Polos_Obs)'; %Polos del observador
disp('Polos de la dinámica del observador')
eig(A-L*C)

Aobs=[A-L*C];
Bobs=[B L];
Cobs=[C; zeros(2,4)]; %Queremos h1 y h2
Dobs=zeros(4);  %Mismo tamaño que Bobs
sysobs=ss(Aobs,Bobs,Cobs,Dobs,Tm);

%Posición inicial.
x = x0;
x_est = xe0; %Se define la posición inical del observador.
```

```
%Vector inicial de referencia.
ref = ones(2,tmax);
ref(:,1:tmax/8) = T0.*ones(2,tmax/8);

%El objetivo de control
% Tcontrol(1,1) = 40;
% Tcontrol(2,1) = 30;

y0=x0(1:2,1);
y = y0;
uref_incr = ones(2,tmax);
% xr = zeros(4,1);
xref = ones(4,tmax);

%Se calcula la entrada de referencia, no la salida directamente.
uref(1,1)=40-0; %El u0 es cero.
uref(2,1)=30-0;
ur_incr = uref;
%Añadir al vector de referencia de la entrada.
uref_incr(:,tmax/8:tmax/2) = uref.*uref_incr(:,tmax/8:tmax/2);

%Definición de matrices y vectores a utilizar en el bucle de control.
uincr = zeros(2,1);
xincr = zeros(n,tmax);


yincr = zeros(2,tmax);
yssmat = zeros(2,tmax);
Tssmat1 = ones(1,tmax)*T0; %Almacena la salida del espacio de estado.
Tssmat2 = ones(1,tmax)*T0;
T1 = ones(1,tmax)*T0; %Para almacenar la variable medida mediante sensor.
T2 = ones(1,tmax)*T0;

%Se calcula el punto de equilibrio para esa entrada de referencia,
%resolviendo la ecuación del espacio de estados para obtener un punto de
%equilibrio.
xref1 = (sys.A - eye(n))\(-sys.B*ur_incr);
yref = C*xref1;

%Asignar a la matriz para usar en el cálculo de la acción de control en un
%bucle.
xref(:,tmax/8+1:tmax/2) = xref(:,tmax/8+1:tmax/2).*xref1;
ref(:,tmax/8+1:tmax/2)=ref(:,tmax/8+1:tmax/2).*(yref+Tpy);

%Cambio de consigna 2: enfriamiento. Se escoge una menor u de referencia.
uref = [15; 20];
ur_incr = uref; %El u0 es 0.
uref_incr(:,tmax/2+1:3/4*tmax) = uref.*uref_incr(:,tmax/2+1:3/4*tmax);

xr = (sys.A - eye(n))\(-sys.B*ur_incr);
yref = C*xr;

xref(:,tmax/2+1:3/4*tmax) = xref(:,tmax/2+1:3/4*tmax).*xr;
ref(:,tmax/2+1:3/4*tmax) = ref(:,tmax/2+1:3/4*tmax).*(yref+Tpy);

%Al final volvemos al punto de funcionamiento.
uref = [0; 0];
ur_incr = uref;
uref_incr(:,tmax*(3/4)+1:end) = uref.*uref_incr(:,tmax*(3/4)+1:end);
```

```
xr = (sys.A - eye(n))\(-sys.B*ur_incr);
yref = C*xr;
xref(:,tmax*(3/4)+1:end) = xref(:,3/4*tmax+1:end).*xr;
ref(:,tmax*(3/4)+1:end) = ref(:,3/4*tmax+1:end).*(yref+Tpy);

% Plot de la referencia.
% figure()
% plot(tiempo2,ref(1,:),'r--')
% hold on
% plot(tiempo2,ref(2,:),'b--')
% legend('referencia 1','referencia 2','Location','best')
% xlabel('Tiempo [seg]')
% ylabel('Temperatura [°C]')
% ylim([0 55])

%Temporización
pausa_max = Tm;  %Esto es el tiempo de muestreo.
tiempo = zeros(tmax,1);
tiempo_inicial = clock;
tiempo_anterior = tiempo_inicial;
tiempo_rest = zeros(tmax,1);


%Graficas a tiempo real
figure()
subplot(2,1,1)
hold on, grid on
anexp1 = animatedline('LineStyle','-', 'Color', 'k', 'LineWidth', 2);
anexp2 = animatedline('LineStyle','-', 'Color', 'g', 'LineWidth', 2);
anpred1 = animatedline('LineStyle','--','Color', 'b','LineWidth', 2);
anpred2 = animatedline('LineStyle','--','Color', 'r', 'LineWidth', 2);
% anpredmat1 = animatedline('LineStyle','-.','Color', '#FFC1C1','LineWidth',
2);
% anpredmat2 = animatedline('LineStyle','-.','Color', '#FF7F24
','LineWidth', 2);
xlabel('Tiempo [seg]')
ylabel('Temperatura [\circC]')
legend('T_1 Medida', 'T_2 Medida', ...
    'T_1', 'T_2','Location', 'best')
title('Evolución de la Temperatura')
%Plot abajo de la entrada
subplot(2,1,2)
hold on, grid on
xlabel('Tiempo [seg]')
yyaxis right
title('Saltos y error')
anQ1 = animatedline('LineStyle','-', 'Color', 'k', 'LineWidth', 2);
anQ2 = animatedline('LineStyle','--', 'Color', 'r', 'LineWidth', 2);
ylabel('Nivel de Q (%)')
xlabel('Tiempo [seg]')
legend( 'Q_1', 'Q_2', 'Location', 'northwest')

%Bucle de control
for k=2:tmax

    tic

    %Lectura de temperaturas, la salida del sistema.
    T1(k) = T1C();
    T2(k) = T2C();
```

```
    %Leer la salida del sistema, en variable incremental
    yincr(:,k) = [T1(k); T2(k)]-[Tpy; Tpy];

    %Se deja inicialmente la misma u. durante tmax/20, hasta estabilizar el
punto de
    %funcionamiento.

    if k < (tmax/20)
        uincr(:,k)=uincr(:,k-1);
        u(:,k)=uincr(:,k)+u0;

        %Aplicación al sistema de la acción de control.
        h1(u(1,k));
        h2(u(2,k));


        %Revisar esto, no me convence la forma de asignar.
        %La temperatura del observador inicial es 25 eso debería
        %observarse en la evolución de la graficas
       Tssmat1(k) = Tssmat1(k-1);
       Tssmat2(k) = Tssmat2(k-1);



    %Una vez estabilizado, se activa el observador.
    %Se deja un tiempo para que se estabilice.
    elseif k >= (tmax/20) && k < tmax/8

        %Se mantiene en el punto de funcionamiento.
        uincr(:,k)=uincr(:,k-1);
        u(:,k)=uincr(:,k)+u0;

        %Aplicación al sistema de la acción de control.
        h1(u(1,k));
        h2(u(2,k));


        %Actualizar el observador.
        xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];
        x_est(:,k) = x0+xincr_e(:,k);
        yssmat(:,k) = C*xincr_e(:,k);
        Tssmat1(k) = yssmat(1,k)+Tpy;
        Tssmat2(k) = yssmat(2,k)+Tpy;

    %Llamada a spcies. Devuelve la acción de control para la referencia.
    %Cálculo de la accion de control.
    elseif k>=tmax/8

        [uincr(:,k),iter,flag1] = spcies_laxMPC_FISTA_solver(xincr_e(:,k-
1),xref(:,k),uref_incr(:,k),...
                            'sys',sys,'param',parametros,'options',options);

%     [uincr(:,k),iter(k),flag1] = spcies_laxMPC_ADMM_solver(xincr(:,k-
1),xref_incr,ur_incr,...
%
'sys',sys,'param',parametros,'options',options);

        u(:,k)=uincr(:,k)+u0;
```

```
        %Aplicación al sistema de la acción de control.
        h1(u(1,k));
        h2(u(2,k));

        %Actualizar el observador de estado.
        xincr_e(:,k)=Aobs*xincr_e(:,k-1)+Bobs*[uincr(:,k); yincr(:,k)];
        x_est(:,k) = x0+xincr_e(:,k);
        yssmat(:,k) = C*xincr_e(:,k);
        Tssmat1(k) = yssmat(1,k)+Tpy;
        Tssmat2(k) = yssmat(2,k)+Tpy;

    end




    tiempo(k) = toc;

       %Dibujar gráficas
    addpoints(anexp1,tiempo2(k),T1(k))
    addpoints(anexp2,tiempo2(k),T2(k))
    addpoints(anpred1,tiempo2(k),Tssmat1(k))
    addpoints(anpred2,tiempo2(k),Tssmat2(k))
    addpoints(anQ1,tiempo2(k),u(1,k))
    addpoints(anQ2,tiempo2(k),u(2,k))
    drawnow

    %Esperamos el tiempo de muestreo, teniendo en cuenta el tiempo de
    %calculo.

    pause(pausa_max-tiempo(k)-0.015);
    %El pause espera más tiempo del deseado por ese le metemos -0.015
    tiempo_rest(k) = pausa_max-tiempo(k)-0.015;

end

%Apagar el calentador
h1(0);
h2(0);
disp('Calentadores apagados')

if opcion==1
    modelo='A';
elseif opcion==2
    modelo='B';
end

figure()
h1 = plot(tiempo2,Tssmat1,'b','DisplayName','T1','LineWidth',1);
hold on
h2 = plot(tiempo2,Tssmat2,'r','DisplayName','T2','LineWidth',1);

h3 = plot(tiempo2,T1,'m--','DisplayName','T1real','LineWidth',1);
h4 = plot(tiempo2,T2,'g--','DisplayName','T2real','LineWidth',1);

h5 = plot(tiempo2,ref(1,:),'k--','DisplayName','ref T1');
h6 = plot(tiempo2,ref(2,:),'k--','DisplayName','ref T2');
```

```
str = sprintf('Modelo %s. Evolucion de las temperaturas: rho=%f, a=%d ',
modelo,rho,a_q)
title(str)
xlabel("Tiempo [seg]")
ylabel("Temperatura [°C]")
% legend(h1,'T1',h2, 'T2',h3,'T1 medida',h4,'T2 medida',h5,'referencia
T1',h6,'referencia2')
legend()

%Gráfica de la acción de control.
figure()
plot(tiempo2,u(1,:),'DisplayName','Calentador 1','LineWidth',2)
hold on
plot(tiempo2,u(2,:),'DisplayName','Calentador 2','LineWidth',2)
line([0,tmax*10],[umax,umax],'DisplayName','Limite superior
u','LineStyle','--');
line([0,tmax*10],[umin,umin],'DisplayName','Limite inferior
u','LineStyle','--');
ylim([umin-5, umax+5]);
legend()
title('Entradas')
xlabel("Tiempo [seg]")
ylabel("Porcentaje calentador [%]")
str = sprintf('Modelo %s. Evolucion de la acción de control: rho=%f, a=%d ',
modelo,rho,a_q)
title(str)

%Gráfica del tiempo de cálculo.
figure
plot(tiempo)
total = sum(tiempo);
str = sprintf('Tiempo de calculo total es %f',total);
title(str)
ylabel('Tiempo [seg]')
xlabel('Número de la iteración')

% fclose( serial(a.Port) ); %Create a serial object with the port Arduino is
connected to it and close it
clear a; %Remove the variable

figure()
histogram(tiempo)
ylabel('Veces')
xlabel('Tiempo [seg]')
title('Histograma del tiempo de cálculo')
```

8.2 Código Python.

Se adjuntan todos los códigos desarrollados en Python con su nombre y una pequeña descripción.

- **Calculo_parametros.py**

Aquí se realiza el cálculo de los parámetros de las ecuaciones de balance, se obtiene las matrices del modelo de

espacio de estados A y se valida el modelo A con el sistema real.

```
#Cálculo de parámetros alfa1, alfa2, tau, u1 y u2 de la ecuación

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from 100arám import GEKKO
import tclab
import time
from scipy import signal

# Importar datos
try:
    # Intentar leer primero el archivo local
    filename = 'data.csv'
    data = pd.read_csv(filename)
except:
    # saltos de los calentadores
    Q1d = np.zeros(601)
    Q1d[10:200] = 80
    Q1d[200:280] = 20
    Q1d[280:400] = 70
    Q1d[400:] = 50

    Q2d = np.zeros(601)
    Q2d[120:320] = 100
    Q2d[320:520] = 10
    Q2d[520:] = 80

    try:
        # Connect to Arduino
        a = tclab.TCLab()
        fid = open(filename,'w')
        fid.write('Time,Q1,Q2,T1,T2\n')
        fid.close()

        # prueba de escalones (10 min)
        for i in range(601):
            # valores de los calentadores
            a.Q1(Q1d[i])
            a.Q2(Q2d[i])
            print('Time: ' + str(i) + \
                  ' Q1: ' + str(Q1d[i]) + \
                  ' Q2: ' + str(Q2d[i]) + \
                  ' T1: ' + str(a.T1)   + \
                  ' T2: ' + str(a.T2))
            # wait 1 second
            time.sleep(1)
            fid = open(filename,'a')
            fid.write(str(i)+','+str(Q1d[i])+','+str(Q2d[i])+',' \
                      +str(a.T1)+','+str(a.T2)+'\n')
        # close connection to Arduino
        a.close()
        fid.close()
    except:
```

```
          filename                                                    =
'https://apmonitor.com/pdc/uploads/Main/tclab_data2.txt'
     # Intentar usar el tclab y si no está conectado leer el fichero
del link
     data = pd.read_csv(filename)

 # Ajustar parámetros al balance de energía
 m = GEKKO() # Create GEKKO Model

 # Parameters to Estimate
 U = m.FV(value=10,lb=1,ub=20)
 Us = m.FV(value=20,lb=5,ub=40)
 alpha1 = m.FV(value=0.01,lb=0.001,ub=0.03)  # W / % heater
 alpha2 = m.FV(value=0.005,lb=0.001,ub=0.02) # W / % heater
 tau = m.FV(value=10.0,lb=5.0,ub=60.0)

 # Measured inputs
 Q1 = m.Param()
 Q2 = m.Param()

 Ta =23.0+273.15                    # K
 mass = 4.0/1000.0                  # kg
 Cp = 0.5*1000.0                    # J/kg-K
 A = 10.0/100.0**2                  # Area no entre calentadores m^2
 As = 2.0/100.0**2                  # Area entre calentadores m^2
 eps = 0.9                          # Emissivity
 sigma = 5.67e-8                    # Stefan-Boltzmann

 #SV = variable de estado.
 TH1 = m.SV()
 TH2 = m.SV()
 #CV es una variable controlada, variables del modelo incluidas
 #en el objetivo de control.
 TC1 = m.CV()
 TC2 = m.CV()

 # Heater Temperatures in Kelvin
 T1 = m.Intermediate(TH1+273.15)
 T2 = m.Intermediate(TH2+273.15)

 # Transferencia de calor entre los dos heaters
 Q_C12 = m.Intermediate(Us*As*(T2-T1)) # Convectivo
 Q_R12 = m.Intermediate(eps*sigma*As*(T2**4-T1**4)) # 101arámetro

 # Balances de energía
 m.Equation(TH1.dt() == (1.0/(mass*Cp))*(U*A*(Ta-T1) \
                 + eps * sigma * A * (Ta**4 – T1**4) \
                 + Q_C12 + Q_R12 \
                 + alpha1*Q1))

 m.Equation(TH2.dt() == (1.0/(mass*Cp))*(U*A*(Ta-T2) \
                 + eps * sigma * A * (Ta**4 – T2**4) \
                 - Q_C12 – Q_R12 \
                 + alpha2*Q2))

 # Conducción a los sensores de temperatura
```

```
  m.Equation(tau*TC1.dt() == TH1-TC1)
  m.Equation(tau*TC2.dt() == TH2-TC2)

  # Options
  # STATUS=1 permite al solver ajustar 102arámetros
  U.STATUS = 1
  Us.STATUS = 1
  alpha1.STATUS = 1
  alpha2.STATUS = 1
  tau.STATUS = 1

  Q1.value=data['Q1'].values
  Q2.value=data['Q2'].values
  TH1.value=data['T1'].values[0]
  TH2.value=data['T2'].values[0]
  TC1.value=data['T1'].values
  TC1.FSTATUS = 1    # minimize fstatus * (meas-pred)^2
  TC2.value=data['T2'].values
  TC2.FSTATUS = 1    # minimize fstatus * (meas-pred)^2

  m.time = data['Time'].values
  m.options.IMODE   = 5 # MHE
  m.options.EV_TYPE = 2 # Objective type
  m.options.NODES   = 2 # Collocation nodes
  m.options.SOLVER  = 3 # IPOPT

  m.solve(102arám=False) # Resolver el modelo físico del sistema.

  # Valores de los 102arámetros
  print('Parámetros estimados')
  print('U     : ' + str(U.value[0]))
  print('Us     : ' + str(Us.value[0]))
  print('alpha1: ' + str(alpha1.value[0]))
  print('alpha2: ' + str(alpha2.value[-1]))
  print('tau: ' + str(tau.value[0]))

  print('Constantes')
  print('Ta: ' + str(Ta))
  print('m: ' + str(mass))
  print('Cp: ' + str(Cp))
  print('A: ' + str(A))
  print('As: ' + str(As))
  print('eps: ' + str(eps))
  print('sigma: ' + str(sigma))

  sae = 0.0
  for I in range(len(data)):
      sae += np.abs(data['T1'][i]-TC1.value[i])
      sae += np.abs(data['T2'][i]-TC2.value[i])
  print(' Balance de Enería error absoluto cometido : ' + str(sae))


  # Crear la gráfica con el balance de energía con los parámetros
obtenidos.
  Plt.figure(figsize=(10,7))
  ax=plt.subplot(2,1,1)
```

```
ax.grid()
plt.plot(data['Time'],data['T1'],'r.',label=r'$T_1$ medida')
plt.plot(m.time,TC1.value,color='black',linestyle='—',\
         linewidth=2,label=r'$T_1$ balance de energia')
plt.plot(data['Time'],data['T2'],'b.',label=r'$T_2$ medida')
plt.plot(m.time,TC2.value,color='orange',linestyle='—',\
         linewidth=2,label=r'$T_2$ balance de energia')
plt.ylabel(r'T ($^oC$)')
plt.legend(loc=2)
ax=plt.subplot(2,1,2)
ax.grid()
plt.plot(data['Time'],data['Q1'],'r-',\
         linewidth=3,label=r'$Q_1$')
plt.plot(data['Time'],data['Q2'],'b:',\
         linewidth=3,label=r'$Q_2$')
plt.ylabel('Calentadores')
plt.xlabel('Tiempo (seg)')
plt.legend(loc='best')
plt.savefig('tclab_parametros.png')
plt.show()
```

- **MPC_python.py**

Este fichero se encarga de implementar el control MPC en Python.

```
#Predictivo en python

from mpc_funcion_helper import calculo_mpc_c
import pathlib
import numpy as np
import tclab
from scipy.linalg import solve
from scipy.signal import cont2discrete
import scipy.io
from ctypes import *
from ctypes import c_double, c_int, CDLL
import matplotlib.pyplot as plt

try:
    datos = np.load('matrices_python.npz')

    #Para ver el nombre con el que se han guardado
    sorted(datos)
    Am = datos['Apy']
    Bm = datos['Bpy']
    Cm = datos['Cpy']
    Dm = np.zeros((2,2))

except:
```

```
    print('Error, no se encuentra el fichero de \
              las matrices en el directorio')

# a = tclab.TCLab()
tmax = 150
Tm = 10
Tpy = 23  #Temperatura utilizada para generar el modelo.
# Tamb = a.T1
Tamb = 27

### DISEÑO DEL OBSERVADOR
obs1=scipy.io.loadmat('observador1.mat')
Aobs = obs1['Aobs']
Bobs = obs1['Bobs']
Cobs = obs1['Cobs']

x = np.zeros((4,1))
xest = np.zeros((4,1))

ref = np.ones((2,tmax))
u0 = np.zeros((2,1))
x0 = np.ones((4,1))*(Tamb-Tpy)
xe0 = np.ones((4,tmax))*28
xincr = np.zeros((4,tmax))
xincre = np.ones((4,tmax))*(28-Tpy)
uincr = np.zeros((2,tmax))
yincr = np.zeros((2,tmax))
#Se pone el objetivo de u.

uref1 = 40
uref2 = 30;
#uref = np.array([[uref1],[uref2]])
uref = np.ones((2,tmax))
xref = np.ones((4,tmax))
yref = np.ones((2,tmax))

xref[:,0:50]=xref[:,0:50]*np.zeros((4,50))

uref[:,0:50] = np.zeros((2,50))
uref[:,51:-1] = np.array([[uref1],[uref2]])
Tss1 = np.ones(tmax)*Tpy
Tss2 = np.ones(tmax)*Tpy

#Pasar a discreto el espacio de estados
sysdisc = cont2discrete((Am,Bm,Cm,Dm), 10)
Am = sysdisc[0]
Bm = sysdisc[1]
Cm = sysdisc[2]

#Utilizar solve para obtener el xss para esa uref
MatrizA = Am-np.eye(4)
MatrizB = np.matmul(-Bm,np.array([uref1,uref2]))
xr = solve(MatrizA, MatrizB)

pru1 = xr
pru2 = np.array(pru1)
```

```
  #Para multiplicar vector por matriz elemento a elemento
  pru3 = np.ones((4,tmax-50))*pru2.reshape(-1,1)
  #pru2 = pru1*np.ones((4,tmax-50))

  xref[:,50:tmax] = xref[:,50:tmax]*pru3
  yref1 = Cm @ xr

  yref[0:50]=[[0],[0]]
  # yref[50:tmax]= np.ones((2,tmax-50))*yref1.reshape(-1,1)

  #Variables para guardar los resultados
  hX = np.zeros((4,tmax))
  hU = np.zeros((2,tmax))
  hiter = np.zeros(tmax)
  tiempos = np.zeros(tmax)
  x = x0
  # iteraccion = np.zeros(tmax)
  flag = np.zeros(tmax)

  for i in range(tmax-1):


      xincr[:,i+1] = Am @ xincr[:,i]+Bm @ uincr[:,i]
      yincr[:,i+1] = Cm @ xincr[:,i+1]

      Tss1[i+1] = yincr[0,i+1]+Tpy
      Tss2[i+1] = yincr[1,i+1]+Tpy

      #Esperar a que el observador se estabilice en torno al punto de
funcionamiento.
      if i>40:
          res = calculo_mpc_c(xincre[:,i], xref[:,i], uref[:,i])
          hU[:,i+1] = res['u']
          # hU[0,i+1] = res['u'][0]
          # hU[1,i+1] = res['u'][1]
          hiter[i+1] = res['iteracion'].value
          flag[i+1] = res['h'].value
          uincr[:,i+1] = np.array(hU[:,i+1])

      elif i<=40:
          uincr[:,i+1]=uincr[:,i]

      #Calculo del observador
      xincre[:,i+1]     =     Aobs     @     xincre[:,i]+Bobs     @
np.concatenate([uincr[:,i+1].T, yincr[:,i+1].T])

      tiempos[i+1] = tiempos[i]+10
      #time.sleep(9)


  # Crear graficas
  plt.figure(figsize=(10,7))
  ax=plt.subplot(2,1,1)
  ax.grid()
  plt.plot(tiempos,Tss1,'r.', label=r'$T1$',color='blue')
  plt.plot(tiempos,Tss2,'r.', label=r'$T2$')
```

```
  plt.plot(tiempos,(xref[1]+Tpy)*np.ones(tmax),color='black',
linestyle='--',\
           linewidth=1,label=r'$T_1$ referencia')

  plt.plot(tiempos,(xref[2]+Tpy)*np.ones(tmax),color='purple',
linestyle='--',\
           linewidth=1,label=r'$T_2$ referencia')
  plt.ylabel(r'T ($^oC$)')
  plt.legend(loc='best')

  #Plot de la acción de control.
  ax=plt.subplot(2,1,2)
  plt.plot(tiempos,uincr[0,:],color='orange', linestyle='-',\
           linewidth=2,label='U1')

  plt.plot(tiempos,uincr[1,:],color='green', linestyle='-',\
           linewidth=2,label='U2')
  plt.legend(loc='best')
  plt.ylabel('Calentadores')
  plt.xlabel('Tiempo [seg]')
```

- **Mpc_funcion_helper.py**

Código necesario para poder emplear la función del Controlador creada en C en lenguaje Python.

```
  """ La llamada a la funcion del solver en python"""

  from ctypes import c_double, c_int, CDLL, byref
  import sys

  # El archivo se compila con

  #gcc -shared -o laxMPC_win32.so -fPIC laxMPC.c
  #  Debe  estar  el  archivo  compilado  en  la  misma  carpeta  que
mpc_funcion_helper.
  lib_path = 'solvers/lax_solver1_win32.so'

  try:
      solver_function_lib = CDLL(lib_path)
      #Esto define el equivalente para python donde se almacenen todas
las funciones
      #y variables del fichero C basic_function.c
  except:
      print('OS %s not recognized' % (sys.platform))


  python_laxMPC_ADMM1 = solver_function_lib.laxMPC_ADMM
  python_laxMPC_ADMM1.restype = None
```

```
  def calculo_mpc_c(xin, xrin, urin):
      """Llamada a la funcion C para calcular la accion de control
      requiere de entradas: el estado, el estado de referencia y la u
de referencia
      Devuelve: la accion de control, el número de iteraciones y una
flag de salida."""
      n = len(xin)
      m = len(urin)
      x_c = (c_double * n)(*xin)
      xr_c = (c_double * n)(*xrin)
      ur_c = (c_double * m)(*urin)

      u = (c_double * m)()
      # iteracion_mpc = len(x_c)-3
      # flaga = len(x_c)+4

      iteracion_mpc = c_int()
      flaga =c_int()

      python_laxMPC_ADMM1(x_c,xr_c,ur_c,                              u,
byref(iteracion_mpc),byref(flaga))

      res = {'u':u, 'iteracion': iteracion_mpc, 'h': flaga}
      return res
```