Proyecto Fin de Carrera
Ingeniería de Tecnologías Industriales

Controladores en línea basados en datos obtenidos por control iterativo

Autor: F. Javier Cano Marín

Tutor: Teodoro Álamo Cantarero

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

Sevilla, 2019/2020

Proyecto Fin de Carrera
Ingeniería de Tecnologías Industriales

Controladores en línea basados en datos obtenidos por control iterativo

Autor:

F. Javier Cano Marín

Tutor:

Teodoro Álamo Cantarero

Catedrático de Universidad

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

Sevilla, 2019/2020

Proyecto Fin de Carrera: Controladores en línea basados en datos obtenidos por control iterativo

Autor: F. Javier Cano Marín
Tutor: Teodoro Álamo Cantarero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Resumen

En este proyecto se trata un tema ampliamente discutido, la obtención de un controlador sin error en régimen permanente ante la presencia de perturbaciones de media no nula. La idea novedosa que propone este Trabajo de Fin de Grado es aproximarnos a este controlador con un enfoque distinto al de los controladores clásicos. La idea es, a partir de un modelo de planta desconocido generar una base de datos de trayectorias de la cual se obtendrá el controlador final sin error en régimen permanente. Este controlador heredará el comportamiento de las trayectorias almacenadas en la base de datos, manteniendo nulo el error en régimen permanente, con la mejora de que puede ser implementado en línea. Se proponen dos formas de obtener esta base de datos, a través de un controlador diseñado según el método *Truxal-Ragazzini* y un controlador tipo PID. Se realizará una comparativa entre ambos y se discutirá sobre las ventajas e inconvenientes de cada uno de ellos.

Abstract

In this project it is treated a thoroughly discussed topic, the implementation of a controller without steady state error. The new idea that is propossed in this Final Degree Project is the implementation of this controller through a non classic aproximation. The idea is, from an unkown plant model, generate a database of trajectories of which will be implementeted the controller providing a zero steady state error. This controller will inherit the behaviour of the trajectories saved in the database, keeping null the error , with the improvement that this controller could be implemented online. Two different families of controllers are proposed to generate the database of closed-loop trajectories: Truxal-Ragazzini and PID controllers. A comparative between both aproaches is provided.

Índice

Resumen I
Abstract II

1 Introducción **1**
1.1 Definición de sistemas de control 1
1.2 Error en régimen permanente 2
1.3 Control sin error en régimen permanente 3
1.3.1 Controladores PI o PID 3
1.4 Controlador Data Driven 5
1.4.1 Definición del modelo de sistema a controlar 5
1.4.2 Base de datos 5
1.4.3 Controlador Data Driven sin error en régimen permanente 5
1.4.4 Definición del modelo real a controlar 6
1.4.5 Comparativa base de datos PID vs Truxal 6

2 Método de identificación recursiva: Mínimos Cuadrados **8**
2.1 Introducción 8
2.2 Formulación del método de identificación: mínimos cuadrados recursivos 9
2.2.1 Mínimos cuadrados recursivos ponderados 12
2.2.2 Mínimos cuadrados recursivos ponderados, sistema con ruido 12
2.3 Implementación del método de identificación mediante mínimos cuadrados recursivos en MATLAB 13
2.3.1 Filtro del vector paramétrico 20
2.3.2 Aplicación a un sistema con ruido 22

3 Implementación de un controlador discreto según el método de diseño Truxal-Ragazzini **29**
3.1 Introducción 29
3.2 Formulación del controlador Truxal-Ragazzini 30
3.2.1 Condición de realización física 30
3.2.2 Condición de estabilidad interna 30
3.2.3 Condición de error en régimen permanente nulo 31
3.3 Implementación del controlador Truxal-Ragazzini en Matlab 31
3.4 Aplicación del controlador diseñado según el método Truxal-Ragazzini a un sistema conocido. Matlab y Simulink. 33
3.4.1 Respuesta en bucle cerrado del sistema en ausencia de perturbaciones 36
3.4.2 Respuesta en bucle cerrado del controlador diseñado en presencia de ruido de media no nula 39

4 Creación de la Base de Datos **42**
4.1 Base de datos. Definición. 42
4.2 Base de datos. Creación. 42

4.3 Implementación de la base de datos en Matlab. 43

5 Controlador Data Driven 49
5.1 Planteamiento del problema 49
5.2 Definición de la ley de Control. Base de datos 49
5.3 Definición controlador Data Driven 50
5.4 Demostración Data Driven Offset Free 51
5.5 Implementación del controlador Data Driven Offset Free 52
5.6 Implementación del controlador en Matlab 54
5.7 Comparativa Control PI vs Controlador Truxal-Ragazzini 57
5.7.1 Implementación en Matlab. Comparativa de resultados 58

6 Modelado de un Sistema de dos tanques de agua concatenados 67
6.1 Definición del sistema 67
6.2 Modelo no lineal de un tanque de agua 67
6.3 Implementación en Simulink 68

7 Implementación del controlador Data Driven en un sistema real 75
7.1 Identificación del sistema 75
7.2 Controlador Truxal y generación de la Base de datos 78
7.3 Implementación del controlador Data Driven 81
7.4 Comparativa final, Truxal vs PID. Concluciones. 82

Índice de Figuras 85
Índice de Tablas 87
Índice de Códigos 89
Bibliografía 91

1 Introducción

1.1 Definición de sistemas de control

Un sistema de control se define como una agrupación de elementos que actúan de manera conjunta con el objetivo de alcanzar una respuesta deseada a partir de unas condiciones iniciales determinadas. Dentro de un sistema de control se pueden definir los siguientes elementos o variables:

- Planta: Objeto físico o sistema que va a ser controlado.
- Proceso: Conjunto de operaciones que van a ser controladas.
- Controlador: Elemento (o ser humano) que actúa sobre la señal de control de acuerdo a una serie de normas predeterminadas.
- Ley de control: Conjunto de normas predeterminadas usadas por el controlador para ajustar la señal de control.
- Entrada (señal de control): Acción generada fuera de la planta con el objeto de que ésta se comporte de una determinada manera.
- Salida (señal controlada): Respuesta real que se obtiene del sistema de control.
- Referencia: Señal proporcionada al sistema de control que representa la salida deseada del sistema.
- Perturbaciones: Señales no controlables que afectan (generalmente de manera negativa) al sistema de control.

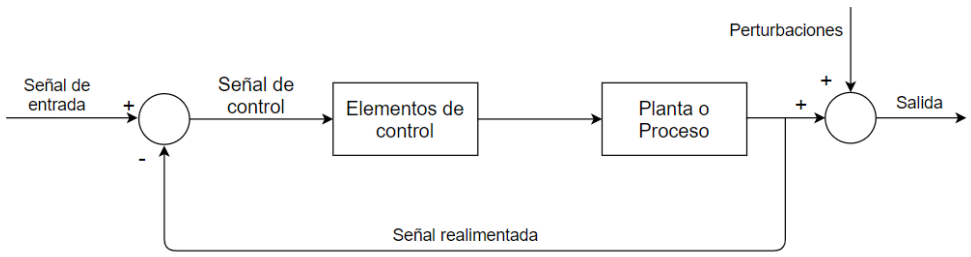

Figura 1.1 Diagrama de bloques sistema de control.

Los sistemas de control se pueden agrupar atendiendo a la dependencia del control con la variable de salida en dos grupos principales:

- Sistemas de control de lazo abierto: Son aquellos en los que no existe dependencia entre la salida del sistema y la acción de control, es decir, la salida y entrada del sistema son independientes.

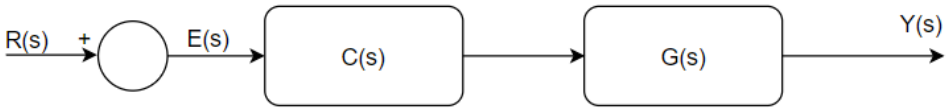

Figura 1.2 Diagrama de bloques sistema de control de lazo abierto.

- Sistemas de control de lazo cerrado: Son aquellos en los que existe de alguna forma una relación entre la señal de salida y entrada del sistema. Generalmente la señal de salida es medida y realimentada para ser comparada con la referencia. A la diferencia entre la referencia y la salida actual del sistema se le denomina error.

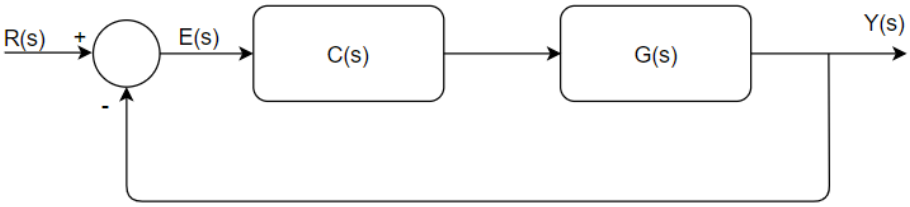

Figura 1.3 Diagrama de bloques sistema de control de lazo cerrado.

El objetivo de este TFG es desarrollar un sistema de control capaz de obtener una respuesta sin error en régimen permanente a pesar de que el sistema presente una serie de perturbaciones aleatorias de media no nula.

1.2 Error en régimen permanente

Para un sistema genérico en bucle cerrado como el que se puede observar en la *Figura 1.3* es posible definir el error en régimen permanente como la diferencia entre el valor de referencia y la salida una vez alcanzado el equilibrio. El objetivo de un controlador sin error en régimen permanente sería conseguir que la media de este error sea cero. Operando podemos definir el error *E(s)* como:

$$E(s) = R(s) - Y(s)$$

$$E(s) = R(s) - G(s)C(s)E(s)$$

$$R(s) = E(s) \cdot [1 + G(s)C(s)]$$

$$E(s) = \frac{R(s)}{1 + G(s)C(s)} \tag{1.1}$$

El error en régimen permanente es el valor que alcanza esta expresión en el infinito, esto se resolvería aplicando el teorema del valor final.

$$erp = \lim_{s \to 1}(s-1)(1 - G_d(s))R(s) \tag{1.2}$$

Siendo $G_d(s)$ la función de transferencia en bucle cerrado del sistema representado por la *Figura 1.3*. La expresión de dicha función de transferencia queda definida en la siguiente ecuación.

$$G_d(z) = \frac{G(z)C(z)}{1+C(z)G(z)} \tag{1.3}$$

1.3 Control sin error en régimen permanente

Ser capaz de alcanzar una referencia sin error en régimen permanente es el objetivo ideal de cualquier controlador en bucle cerrado. En los siguientes subapartados se presentan dos controladores clásicos utilizados para lograr este objetivo.

1.3.1 Controladores PI o PID

Un controlador PID es un modelo clásico de control que actúa en función del error del sistema para alcanzar una salida deseada. La respuesta de este controlador se basa en la suma de tres acciones: proporcional, integral y derivativa.La ecuación que rige el comportamiento de un controlador PID es:

$$U(t) = K_p \cdot e(t) + \frac{1}{\tau_i}\int e(t)dt + \tau_d \frac{de(t)}{dt} \tag{1.4}$$

Como en nuestro caso el objetivo es eliminar el error en régimen permanente nos vamos a centrar en las acciones proporcional e integral.

La acción proporcional es proporcional al error *e(t)* y multiplicará esta variable por una constante K_p. Esta acción intentará minimizar el error del sistema puesto que a medida que *e(t)* sea más grande también lo será la respuesta del controlador. Sin embargo, hay que tener en cuenta que aumentar la acción proporcional también aumenta la inestabilidad del sistema.

La acción integral, como su nombre indica, calcula la integral del error *e(t)* y la multiplica por una constante denominada constante integral τ_i. Esta acción integral proporciona una corrección para compensar las perturbaciones y mantener la variable controlada en el valor de referencia.

Para ejemplificar el efecto de un controlador PID se ha realizado un caso sencillo de control para un sistema de segundo orden subamortiguado. El sistema en cuestión está definido por la función de transferencia.

$$G(s) = \frac{4}{s^2+0.4s+4} \tag{1.5}$$

Mediante la herramienta *'Simulink'* se ha simulado este sistema en tres condiciones distintas. El primer caso se ha simulado solamente la respuesta del sistema ante una entrada tipo escalón unitario, en el segundo caso se ha simulado el sistema ante la misma entrada pero éste estaba controlado mediante un bloque PID realimentado y finalmente el tercer caso presenta las mismas condiciones que en el caso anterior, pero, se ha añadido una perturbación de media no nula a la salida del sistema. En la *Figura 1.4* se puede observar la respuesta del sistema ante los distintos casos expuestos previamente.

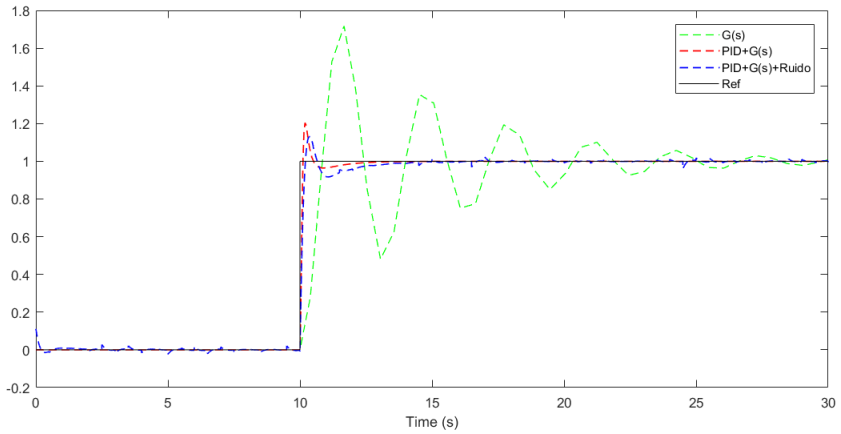

Figura 1.4 Respuesta de un sistema continuo ante una entrada escalón en tres casos distintos.

Cabe destacar que sin presencia de ruido el controlador PID consigue eliminar completamente el error en régimen permanente. Sin embargo, ante la presencia del ruido podemos concluir que el control mediante un bloque PID no es óptimo pero si aceptable, por eso este controlador es usado en modelos sencillos donde las perturbaciones son pequeñas. Finalmente concluir que los valores del bloque PID se han obtenido mediante la función *'Tune'* que presenta el mismo bloque. El bloque PID utilizado presenta la siguiente función de transferencia:

$$PID(s) = P + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}} \tag{1.6}$$

En el caso del control sin ruido los parámetros utilizados para caracterizar el controlador PID han sido:

$$P = 15.6550376623668$$
$$I = 15.0811862620582$$
$$D = 3.38789964706179$$
$$N = 39.9756817046043$$

Para el último caso en el que el sistema si presentaba ruido estos valores han sido:

$$P = 8.12304841022522$$
$$I = 5.55537974625673$$
$$D = 2.1031681358585$$
$$N = 1065.05906546457$$

En todos los casos el tiempo de simulación ha sido de 30 segundos, el tiempo de muestreo ha sido variable y se ha dejado que la opción *'Solver'* sea elegida automáticamente. Para añadir ruido a la salida del sistema se ha utilizado el bloque *'Random Number'* y los parámetros de este bloque han sido: *Mean=0.1*, *Variance=0.0001*, *Seed=0*.

En la *Figura 1.5* se puede observar el diagrama de bloques en *'Simulink'* del control PID del sistema con ruido.

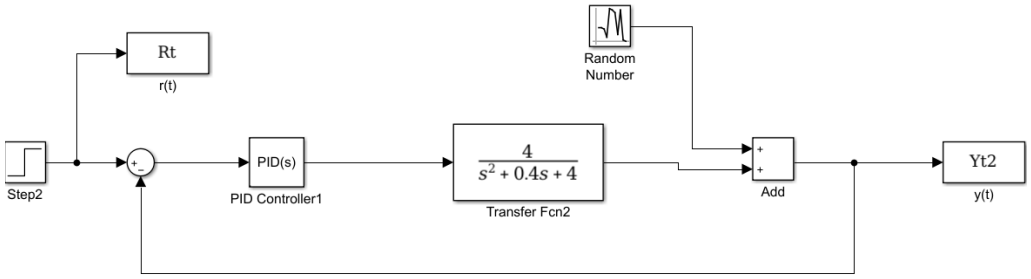

Figura 1.5 Diagrama de bloques control PID.

1.4 Controlador Data Driven

Como se señala más en detalle en el *Capítulo 5*, este modelo de control fue desarrollado por los profesores *J. R. Salvador, D.R. Ramirez, T. Alamo y D. Muñoz de la Peña* pertenecientes al departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla [1][7].

1.4.1 Definición del modelo de sistema a controlar

Como se comentó previamente el objetivo final de este Trabajo de Fin de Grado es desarrollar un controlador capaz de obtener una respuesta sin error en régimen permanente a pesar de que el sistema presente un ruido de media no nula. El sistema a controlar se define como:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + w(t) \\ y(t) &= Cx(t) + Du(t) + v(t) \end{aligned} \tag{1.7}$$

Siendo $x(t) \in \mathbb{R}^{n_x}$ y $u(t) \in \mathbb{R}^{n_u}$ respectivamente los valores del vector de estados y señal de control del sistema para un instante de tiempo t, $y(t) \in \mathbb{R}^{n_y}$ la salida del sistema, $w(t) \in \mathbb{R}^{n_x}$ y $v(t) \in \mathbb{R}^{n_y}$ los valores de las perturbaciones que afectan en cada instante t y finalmente las matrices A, B, C, D que definen al sistema.

1.4.2 Base de datos

La idea es, partiendo de este sistema, generar un histórico de datos a partir del cual se implementará el controlador. Este histórico de datos recoge valores pasados del vector de estados, salidas, señal de control y referencia del sistema definido por la ecuación (*1.7*). Estos valores se obtendrán de simular el sistema utilizando un controlador que nos garantice que el error en régimen permanente sea nulo. El modelo de control seleccionado para definir la base de datos es un controlador discreto diseñado según el método *Truxal-Ragazzini*. Las ventajas de usar este controlador frente a un PID clásico se basan en que para el propio diseño del controlador se ha de definir la función de transferencia en bucle cerrado del sistema. Por tanto el histórico de datos final obtenido entre todas las trayectorias generadas tendrá la característica común de que estas presentan la misma función de transferencia en bucle cerrado. Las características de diseño de un controlador según el método de diseño *Truxal-Ragazzini* se exponen en el *Capítulo 3*. Una vez generado, este histórico de datos se va a utilizar para construir un subconjunto de trayectorias a partir de las cuales se va a implementar el controlador final. Nos referimos a este subconjunto de trayectoria como *Base de Datos*. Las características de esta base de datos y su proceso de creación se han definido en el *Capítulo 4*.

1.4.3 Controlador Data Driven sin error en régimen permanente

La ley de control propuesta para la obtención del controlador será una suma ponderada de las señales de control iniciales almacenadas en la base de datos S:

$$u(t) = \sum_{j \in S} \lambda_j u_j(0) \tag{1.8}$$

La implementación del controlador final se basa entonces en la obtención de los pesos ponderados λ_j para conseguir el error nulo en régimen permanente. Para ellos se impondrán las siguientes restricciones de

igualdad que serán desarrolladas en el *Capítulo 5*.

$$x(t) = \sum_{j \in S} \lambda_j x_j(0) \tag{1.9}$$

$$x(t-1) = \sum_{j \in S} \lambda_j x_j(-1) \tag{1.10}$$

$$u(t-1) = \sum_{j \in S} \lambda_j u_j(-1) \tag{1.11}$$

$$y(t) = \sum_{j \in S} \lambda_j y_j(0) \tag{1.12}$$

$$r = \sum_{j \in S} \lambda_j r_j \tag{1.13}$$

$$1 = \sum_{j \in S} \lambda_j \tag{1.14}$$

1.4.4 Definición del modelo real a controlar

Una vez definido el controlador, aplicaremos el mismo a un sistema no lineal real. Este sistema es el clásico modelo de dos tanques de agua concatenados.

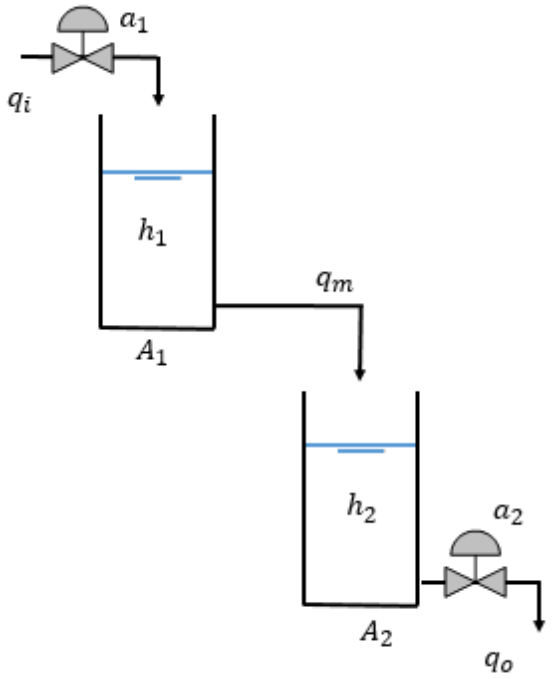

Figura 1.6 Sistema de dos tanques de agua concatenados.

La definición de este modelo se realizará en el *Capítulo 6*. La idea es utilizar un modelo real, pero que el mismo sea desconocido, pues, a priori, no se va a conocer el modelo la planta. Por tanto si desconocemos el modelo, habrá que realizar una caracterización del mismo. Para ello se utilizará el método de identificación recursiva de los mínimos cuadrados ponderados, que será expuesto en el *Capítulo 2*. Una vez identificado el sistema es posible aplicar la algoritmia para el diseño del controlador discreto *Truxal*, generar la base de datos y obtener el controlador final.

1.4.5 Comparativa base de datos PID vs Truxal

La idea novedosa que propone este TFG es la generación de la base de datos para la aplicación del controlador final mediante un controlador discreto diseñado según el método *Truxal-Ragazzini*. Previamente por parte del departamento de Ingeniería de Sistemas de la Universidad de Sevilla, ya se había llevado a cabo el diseño y pruebas de implementación de este controlador. Sin embargo, para generar la base de datos se habían utilizado trayectorias generadas mediante un controlador tipo *PID*. Al utilizar un controlador Truxal, todas las trayectorias almacenadas en la base de datos presentarán la misma función de transferencia en bucle cerrado. De esta forma, el conjunto de trayectorias quedará acotado dando lugar a una menor disparidad entre los estados almacenados en la base de datos y finalmente resultando en una mejor implementación del controlador *Data Driven Offset Free*. En el capítulo final (*7*) se podrá observar dicha comparativa así como la aplicación final al modelo real.

2 Método de identificación recursiva: Mínimos Cuadrados

En este proyecto partimos de un sistema a priori desconocido. Para obtener una aproximación de este modelo es necesario aplicar la teoría de identificación de sistemas. La identificación de sistemas tiene como objetivo crear un modelo matemático que represente con suficiente exactitud las características dinámicas de un determinado proceso o sistema del cual conocemos o no su modelo. Para realizar este proceso de identificación es necesario obtener previamente una representación de las variables de entrada y salida del sistema en condiciones de funcionamiento. Estos modelos matemáticos estarán compuestos por una serie de parámetros a priori desconocidos. En la identificación de sistemas existen numerosos métodos para hallar estos parámetros, uno de los mas usados es el método de los mínimos cuadrados. Este método se basa en la minimización del sumatorio de errores al cuadrado, considerando como error la diferencia entre el valor medido y el valor experimental del sistema.

2.1 Introducción

El objetivo de la identificación por mínimos cuadrados es obtener una aproximación del modelo de un sistema a partir de su comportamiento ante una determinada perturbación. Consideramos un sistema con salida *y(k)* y señal de control *u(k)* ambos medidos en cada instante *k*. El modelo de mínimos cuadrados plantea que la salida de este sistema puede ser aproximada por una expresión del tipo

$$\hat{y}(k) = m(k) \cdot \theta \qquad\qquad k = 0,1,...,n \tag{2.1}$$

Siendo $m(k)$ un vector fila (denominado vector regresor) que contiene valores pasados de las salidas y entradas del sistema y θ un vector columna (denominado vector paramétrico) que contiene los distintos coeficientes que relacionan el valor de la salida aproximada en un instante k con los valores medidos pasados o actuales tanto de las salidas como de la acción de control del sistema. Este vector regresor es a priori desconocido y su cálculo es el objetivo final del método de los mínimos cuadrados. Como ejemplo podemos definir un sistema cuya salida aproximada presente la siguiente expresión:

$$\hat{y}(k) = \theta_1 y_{k-1} + \theta_2 u_{k-1} + \theta_3 u_{k-2} \tag{2.2}$$

Para este ejemplo los vectores paramétrico θ y regresor *m(k)* estarían definidos para un instante cualquiera k como:

$$m(k) = \begin{bmatrix} y_{k-1} & u_{k-1} & u_{k-2} \end{bmatrix}$$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Cómo se ha mencionado antes el vector regresor contiene valores pasados tanto de la salidas como de la señal de control del sistema. Esto implica que para determinados valores de k no sea posible crear dicho vector dado que no existen medidas válidas. Si tomamos como referencia el sistema ejemplo y suponemos que el

primer instante a partir del cual se obtienen valores medidos válidos es *k=0* el vector regresor no podría construirse hasta el instante *k=2* y tendría la siguiente forma:

$$m(2) = \begin{bmatrix} y_1 & u_1 & u_0 \end{bmatrix}$$

Supondremos a partir de ahora que a partir del instante $k = j$ se obtiene el primer valor válido del vector regresor.

El objetivo final como se ha mencionado anteriormente es obtener este vector paramétrico. Definimos $\theta(k)$ a la aproximación de este vector en el instante k. Definiremos también el error *e(k)* como la diferencia entre el valor medido *y(k)* y el valor aproximado $\hat{y}(k)$. Pues bien, el principio de mínimos cuadrados define los parámetros del vector regresor θ de tal forma que se minimice la función de costo:

$$J(\theta) = \frac{1}{2}\sum_{k=j}^{n} e(k)^2 \tag{2.3}$$

Siendo n el número total de valores medidos del sistema.

Para minimizar este funcional vamos a agrupar en una matriz Y el conjunto de valores que representa las distintas salidas medidas del sistema y en otra matriz M el conjunto de vectores regresores del mismo.

$$Y = \begin{bmatrix} y(j) \\ y(j+1) \\ \vdots \\ y(n) \end{bmatrix} \qquad M = \begin{bmatrix} m(j) \\ m(j+1) \\ \vdots \\ m(n) \end{bmatrix}$$

Si definimos ahora el conjunto de valores aproximados de la salida como $\hat{Y} = M\theta$ se puede reescribir la expresión del funcional J como:

$$J(\theta) = \frac{1}{2}(Y-\hat{Y})'(Y-\hat{Y}) = \frac{1}{2}(Y-M\theta)'(Y-M\theta) = \frac{1}{2}(Y'Y - Y'M\theta - \theta'M'Y + \theta'M'M\theta) \tag{2.4}$$

Como la matriz *M'M* es siempre semidefinida o definida positiva la función $J(\theta)$ tendrá mínimo. Además como $Y'M\theta = (\theta'M'Y)'$ entonces:

$$J(\theta) = \frac{1}{2}(Y'Y - 2\theta'M'Y + \theta'M'M\theta) \tag{2.5}$$

Para calcular el mínimo de dicho funcional buscamos el valor de θ que satisfaga la igualdad $\frac{dJ(\theta)}{d\theta} = 0$ de donde obtendremos la siguiente expresión:

$$-2M'Y + 2M'M\theta = 0 \tag{2.6}$$

$$M'M\theta = M'Y \tag{2.7}$$

Resolviendo esta última ecuación obtenemos:

$$\theta = \hat{\theta} = (M'M)^{-1}M'Y \tag{2.8}$$

Dicho valor de θ es el que minimizará el funcional J siempre y cuando la matriz *M'M* sea no singular. Hay que tener en cuenta que para que la matriz *M'M* no presente problemas de singularidad o esté mal condicionada el sistema tendrá que encontrarse bien caracterizado, es decir, tenemos que recoger un amplio número de medidas de la simulación del sistema.

2.2 Formulación del método de identificación: mínimos cuadrados recursivos

El método explicado en el apartado anterior se considera de *'fuera de línea'*, dado que se necesitan todas las medidas hasta el instante final para poder implementarlo. Generalmente en identificación de sistemas los parámetros de los mismos presentan una dinámica que varía con el tiempo por tanto es más óptimo ir identificando los parámetros en cada instante k, este método se denomina de identificación *én línea'*. En el caso de los mínimos cuadrados el objetivo es obtener una aproximación del vector paramétrico θ en el

instante *k+1* a partir de su aproximación en el instante anterior. Si desarrollamos la expresión *(2.8)* para un instante cualquiera *k+1* obtendríamos:

$$\hat{\theta}(k+1) = [M(k+1)'M(k+1)]^{-1}M(k+1)'Y(k+1) \tag{2.9}$$

donde:

$$Y(k) = \begin{bmatrix} y(j) \\ y(j+1) \\ \vdots \\ y(k) \end{bmatrix} \quad Y(k+1) = \begin{bmatrix} Y(k) \\ y(k+1) \end{bmatrix} \quad M(k) = \begin{bmatrix} m(j) \\ m(j+1) \\ \vdots \\ m(k) \end{bmatrix} \quad M(k+1) = \begin{bmatrix} M(k) \\ m(k+1) \end{bmatrix} \tag{2.10}$$

Hemos supuesto al igual que en el caso anterior que el primer instante a partir del cuál se obtiene un valor válido del vector paramétrico es *k=j*. Haciendo uso de las expresiones anteriores podemos desarrollar el siguiente producto

$$M'(k+1)M(k+1) = \begin{bmatrix} M(k) \\ m(k+1) \end{bmatrix}' \begin{bmatrix} M(k) \\ m(k+1) \end{bmatrix} = M(k)'M(k) + m(k+1)'m(k+1) \tag{2.11}$$

Ahora si unimos las expresiones *(2.9)*, *(2.10)* y *(2.11)*

$$\begin{aligned} \hat{\theta}(k+1) &= [M(k)'M(k) + m(k+1)'m(k+1)]^{-1} \begin{bmatrix} M(k) \\ m(k+1) \end{bmatrix}' \begin{bmatrix} Y(k) \\ y(k+1) \end{bmatrix} \\ &= [M'(k)M(k)]^{-1}M(k)'Y(k) - [M'(k)M(k)]^{-1}M(k)'Y(k) \\ &+ [M(k)'M(k) + m(k+1)'m(k+1)]^{-1}[M(k)'Y(k) + m(k+1)'y(k+1)] \end{aligned} \tag{2.12}$$

Agrupando términos obtenemos:

$$\begin{aligned} \hat{\theta}(k+1) &= [M'(k)M(k)]^{-1}M(k)'Y(k) \\ &+ \{[M(k)'M(k) + m(k+1)'m(k+1)]^{-1} - [M'(k)M(k)]^{-1}\}M(k)'Y(k) \\ &+ [M(k)'M(k) + m(k+1)'m(k+1)]^{-1}[m(k+1)'y(k+1)] \end{aligned} \tag{2.13}$$

En la ecuación *(2.13)* se puede observar que el primer término a la derecha de la igualdad se corresponde con $\hat{\theta}(k)$. Centrándonos ahora en el segundo término a la derecha de la igualdad, si sacamos como factor común la primera componente dentro del símbolo llave $(\{)$ obtenemos:

$$[M(k)'M(k) + m(k+1)'m(k+1)]^{-1}\{I - [M(k)'M(k) + m(k+1)'m(k+1)][M'(k)M(k)]^{-1}\}M(k)'Y(k) \tag{2.14}$$

Ahora desarrollando el término dentro de las llaves y simplificando:

$$\begin{aligned} &[M(k)'M(k) + m(k+1)'m(k+1)]^{-1}\{I - I - [m(k+1)'m(k+1)][M'(k)M(k)]^{-1}\}M(k)'Y(k) \\ &= -[M(k)'M(k) + m(k+1)'m(k+1)]^{-1}\{[m(k+1)'m(k+1)][M'(k)M(k)]^{-1}M(k)'Y(k)\} \\ &= -[M(k)'M(k) + m(k+1)'m(k+1)]^{-1}\{[m(k+1)'m(k+1)]\hat{\theta}(k)\} \end{aligned} \tag{2.15}$$

Utilizando la ecuación simplificada *(2.15)* y sustituyéndola en *(2.13)*:

$$\begin{aligned} \hat{\theta}(k+1) &= \hat{\theta}(k) - [M(k)'M(k) + m(k+1)'m(k+1)]^{-1}\{[m(k+1)'m(k+1)]\hat{\theta}(k)\} \\ &+ [M(k)'M(k) + m(k+1)'m(k+1)]^{-1}[m(k+1)'y(k+1)] \end{aligned} \tag{2.16}$$

Podemos sacar factor común en la expresión *(2.16)*. Al hacerlo observamos que el término común se corresponde con la expresión desarrollada *(2.11)*, por lo que podemos simplificarlo.

$$\begin{aligned} \hat{\theta}(k+1) &= \hat{\theta}(k) + [M(k)'M(k) + m(k+1)'m(k+1)]^{-1}m(k+1)'[y(k+1) - m(k+1)\hat{\theta}(k)] \\ &= \hat{\theta}(k) + [M(k+1)'M(k+1)]^{-1}m(k+1)'[y(k+1) - m(k+1)\hat{\theta}(k)] \end{aligned} \tag{2.17}$$

Vamos a definir ahora dos nuevas matrices que nos servirán para implementar el algoritmo recursivo de identificación por mínimos cuadrados. Estas matrices van a ser una matriz de ganancia *K(k)* y una segunda matriz *P(k)*.

$$K(k) = [M(k)'M(k)]^{-1}m(k)' \tag{2.18}$$

$$P(k) = [M(k)'M(k)]^{-1} \tag{2.19}$$

Evaluando ambas expresiones en el instante *k+1* y operando en la matriz *P* para volver a utilizar la expresión *(2.11)* obtendríamos:

$$K(k+1) = [M(k+1)'M(k+1)]^{-1}m(k+1)' \tag{2.20}$$

$$\begin{aligned} P(k+1) &= [M(k+1)'M(k+1)]^{-1} \\ &= [M(k)'M(k) + m(k+1)'m(k+1)]^{-1} \end{aligned} \tag{2.21}$$

Utilizando ahora el lema de inversión de matrices en la expresión *(2.20)* y simplificando:

$$\begin{aligned} &P(k+1) = [M(k+1)'M(k+1)]^{-1} \\ &- [M(k+1)'M(k+1)]^{-1}m(k+1)'\{I + m(k+1)[M(k)'M(k)]^{-1}m(k+1)\}^{-1}m(k+1)[M(k+1)'M(k+1)]^{-1} \\ &= P(k) - P(k)m(k+1)'[I + m(k+1)P(k)m(k+1)']^{-1}m(k+1)P(k) \end{aligned} \tag{2.22}$$

Uniendo las expresiones *(2.19)*, *(2.20)*, *(3.21)* y operando

$$\begin{aligned} K(k+1) &= P(k+1)m(k+1)' = P(k)m(k+1)' \\ &\quad - P(k)m(k+1)'[I + m(k+1)P(k)m(k+1)']^{-1}m(k+1)P(k)m(k+1)' \\ &= P(k)m(k+1)' - P(k)m(k+1)'[I + m(k+1)P(k)m(k+1)']^{-1}[I + m(k+1)P(k)m(k+1)' - I] \\ &= P(k)m(k+1)' - P(k)m(k+1)' + P(k)m(k+1)'[I + m(k+1)P(k)m(k+1)']^{-1} \\ &= P(k)m(k+1)'[I + m(k+1)P(k)m(k+1)']^{-1} \end{aligned} \tag{2.23}$$

Para finalizar solo nos queda reemplazar la expresión *2.23* en la *2.20* y obtenemos:

$$\begin{aligned} P(k+1) &= P(k) - K(k)m(k+1)P(k) \\ &= [I - K(k)m(k+1)]P(k) \end{aligned} \tag{2.24}$$

Con esta última expresión ya hemos definido el conjunto de ecuaciones que nos permiten implementar el método de los mínimos cuadrados recursivos. El primer paso ahora será definir los valores que inicializarán el algoritmo. Tenemos un sistema con un total de *n* medidas, muestreadas para cada tiempo de muestreo *k*. Como se definió con anterioridad el vector regresor solo existe a partir del instante de muestreo *j*, por tanto el primer vector paramétrico que se puede calcular es $\hat{\theta}(j)$. Sin embargo, este método implica que para calcular el valor de $\hat{\theta}$ en un instante cualquiera, necesitamos su valor en el instante anterior. También hay que señalar que a parte del valor de $\hat{\theta}$ necesitamos el valor inicial de la matriz *P* para inicializar el algoritmo. Caben aquí dos posibilidades:

1. Que el usuario defina los valores iniciales de $\hat{\theta}$ y *P*, y a partir de estos, se implemente el algoritmo. Generalmente se escoge un valor elevado para la matriz *P* dado que éste va decreciendo con cada iteración.
2. Suponemos que el primer valor válido de $\hat{\theta}$ se da en el instante *j+1*. Por tanto, podemos definir $\hat{\theta}(j)$ y *P(j)* utilizando las siguientes expresiones:

$$\begin{aligned} P(j) &= [M'(j)M(j)]^{-1} \\ \hat{\theta}(j) &= P(j)M'(j)Y(j) \end{aligned}$$

Una vez definidos los valores iniciales solo queda comenzar el algoritmo que se repetirá tantas veces como medidas del sistema tengamos. Conociendo $\hat{\theta}(j)$ y *P(j)* definimos el vector regresor y la salida para el

siguiente instante de muestreo: *m(j+1)* y *y(j+1)*. Ahora utilizamos las siguientees expresiones para calcular el próximo valor de $\hat{\theta}$.

$$K(j+1) = P(j)m(j+1)'[I + m(j+1)P(j)m(j+1)']^{-1} \tag{2.25}$$

$$\hat{\theta}(j+1) = \hat{\theta}(j) + K(j)[y(j+1) - m(j+1)\hat{\theta}(k)] \tag{2.26}$$

Una vez obtenido el valor de $\hat{\theta}$ calculamos el valor de la matriz P para el mismo instante

$$P(j+1) = [I - K(j)m(j+1)]P(j) \tag{2.27}$$

Y con esto queda definido el algoritmo de identificación por mínimos cuadrados recursivos. Solamente habría que ir repitiendo el desarrollo explicado con anterioridad hasta la última medida del sistema y así obtendríamos el valor óptimo de $\hat{\theta}$.

2.2.1 Mínimos cuadrados recursivos ponderados

En el algoritmo previamente desarrollado se suele incluir un nuevo término denominado factor de olvido λ. El factor de olvido $\lambda \in (0,1]$ es un escalar que pondera el peso que vamos a darle a las distintas estimaciones del sistema. Si lambda es igual a uno entonces todas las estimaciones tendrán el mismo peso, sin embargo, a medida que el valor de lambda se vaya haciendo más próximo a cero los errores recientes (más cercanos al valor actual de k) tendrán mas importancia en la estimación. En este caso el problema de minimización a resolver sería:

$$J(\theta) = \frac{1}{2}\sum_{k=j}^{n} w(k)e(k)^2 = \frac{1}{2}\|E(N)'W(N)W(N)\|^2 \tag{2.28}$$

donde

$$\begin{pmatrix} w(j) & & \\ & \ddots & \\ & & w(n) \end{pmatrix}$$

El vector w recoge los distintos valores del factor de olvido para cada instante k. Generalmente se suele usar un factor de olvido exponencial donde $w(k) = \lambda^{n-k}$, de esta forma los errores pasados presentan menos peso que los actuales. La elección de un correcto factor de olvido es muy importante dado que influye directamente en la identificación del sistema. Si el valor del factor de olvido es próximo a uno éste se va a adaptar más lentamente a las variaciones del sistema pero será mas insensible al ruido. Y en el caso contrario, mientras más próximo sea este valor a cero nuestro sistema se adaptará más rápido pero será más sensible al ruido. En este caso las tres ecuaciones que habría que utilizar para implementar el algoritmo de identificación por mínimos cuadrados serían:

$$K(k+1) = P(k)m(k+1)'[w(k+1) + m(k+1)P(k)m(k+1)']^{-1} \tag{2.29}$$

$$\hat{\theta}(k+1) = \hat{\theta}(k) + K(k)[y(k+1) - m(k+1)\hat{\theta}(k)] \tag{2.30}$$

$$P(k+1) = [I - K(k)m(k+1)]\frac{P(k)}{w(k+1)} \tag{2.31}$$

La metodología para aplicar este algoritmo es la misma que la explicada en el apartado anterior

2.2.2 Mínimos cuadrados recursivos ponderados, sistema con ruido

Se presenta ahora una variación del método de identificación mediante mínimos cuadrados recursivos para sistemas que presenten una variable de ruido cuyos valores, tantos actuales como pasados, afecten a la salida. Estamos hablando de sistemas tipo *ARMAX*.

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + C(z^{-1})v(k) \tag{2.32}$$

Dado que desconocemos al identificar el modelo, cual va a ser la dependencia del mismo con el ruido, se va a presuponer que este es un sistema tipo *ARMAX*. Sin embargo, surge un problema al identificar un modelo tipo *ARMAX*. Dado que el vector de parámetros $C(z^{-1})$ implica que exista una dependencia entre el valor actual de la salida *y(k)* y valores actuales y pasados del ruido *v(k-1)*, *v(k-2)* ... existe un sesgo entre el valor

del vector paramétrico real y el identificado, por tanto, no se asegura la convergencia al valor verdadero si el proceso es tipo *ARMAX*.

Para evitar el problema del sesgo en este tipo de modelos, se incluyen los parámetros del ruido en el vector paramétrico

$$\theta = \begin{bmatrix} a1 & \dots & an & b1 & \dots & bn & c1 & \dots & cn \end{bmatrix}^T \tag{2.33}$$

Como, a priori, en un sistema real los valores de ruido no pueden ser medidos, estos se van a estimar por el error de estimación en cada instante:

$$\begin{bmatrix} m(k) = -y(k-1) & \dots & -y(k-n) & u(k-1) & \dots & u(k-n) & e(k-1) & \dots & e(k-n) \end{bmatrix} \tag{2.34}$$

donde:

$$e(k) = y(k) - m(k)\hat{\theta}(k-1) \tag{2.35}$$

Como efecto negativo de la identificación según este método es que la convergencia de los parámetros c_i es mas lenta, debido a que la señal de ruido es la menos ponderante.

Para implementar este método simplemente habría que seguir el procedimiento expuesto en la sección *2.2 Formulación del método de identificación: mínimos cuadrados recursivos* y utilizar las ecuaciones (*2.29*), (*2.30*) y (*2.31*).

2.3 Implementación del método de identificación mediante mínimos cuadrados recursivos en MATLAB

El código implementado se basa principalmente en un desarrollo iterativo de la formulación presentada en el apartado anterior, *2.2.1 Mínimos cuadrados recursivos ponderados*. En primer lugar, y para comprobar el correcto funcionamiento del algoritmo, se va a definir una función de transferencia. La entrada a esta función de transferencia será una señal de tren de pulsos rectangular. El resultado final deseado sería obtener a la salida del algoritmo la misma función de transferencia que hemos definido nosotros. Las variables de simulación referentes a la función de transferencia, así como el modelo de bloques en *Simulink* se pueden observar a continuación.

Código 2.1 Inicialización del sistema a identificar, mínimos cuadrados recursivos.

```
%PARÁMETROS SIMULACIÓN
puntos=7000; %Número de Samples Times que tendrá la simulación
Pt=7; %Número de periodos que tendrá la simulación
[ts,Period,Width]=DefinePeriod(puntos,tm,Pt);
K=0.3; %Amplitud de la señal Generador de Pulsos
%Numerador y Denominador de la función de transferencia
Num=0.5;
Den=[1 -0.95];
```

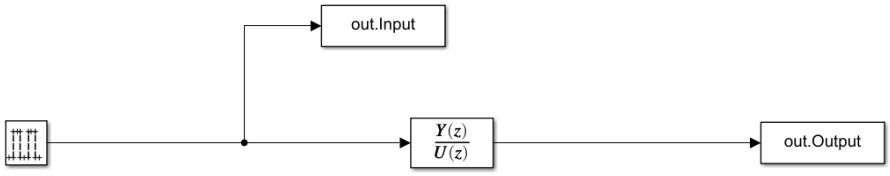

Figura 2.1 Diagrama de bloques del modelo a identificar, mínimos cuadrados.

Una vez caracterizado el sistema a identificar, se pasa a definir el algoritmo desarrollado. En primer lugar, se parte de la idea de que tanto el tiempo de muestreo óptimo como el tamaño y forma del vector regresor son, a priori, desconocidos. Respecto al tiempo de muestro, se va a definir un vector tm_{array} donde se van a encontrar definidos los posibles tiempos de muestreo que va a utilizar el algoritmo. En cuanto al vector regresor se define un modelo inicial del mismo según la siguiente ecuación:

$$m(k) = \begin{bmatrix} y_{k-1} & u_k & u_{k-1} \end{bmatrix} \tag{2.36}$$

También se va a definir una variable *maxIT* que representa el número máximo de iteraciones del algoritmo. La idea es que para cada iteración, el algoritmo aumente el tamaño del vector regresor en uno, añadiendo en primer lugar un valor pasado de y_k y después de u_k hasta el valor máximo definido como *maxIT*. A modo de ejemplo, en la siguiente iteración el vector regresor pasaría a tener la siguiente forma:

$$m(k) = \begin{bmatrix} y_{k-1} & y_{k-2} & u_k & u_{k-1} \end{bmatrix}$$

Y en la siguiente:

$$m(k) = \begin{bmatrix} y_{k-1} & y_{k-2} & u_k & u_{k-1} & u_{k-2} \end{bmatrix}$$

Y así sucesivamente hasta que el número de iteraciones sea igual a *maxIT*. Para finalizar el ejemplo, para un caso concreto en el que se definiera $maxIT = 6$, la forma final del vector paramétrico sería:

$$m(k) = \begin{bmatrix} y_{k-1} & y_{k-2} & y_{k-3} & y_{k-4} & u_k & u_{k-1} & u_{k-2} & u_{k-3} & u_{k-4} \end{bmatrix}$$

Se presenta a continuación el algoritmo desarrollado para implementar el método de identificación mediante mínimos cuadrados recursivos:

Código 2.2 Algoritmo de identificación mediante mínimos cuadrados recursivos.

```
%Variables de Simulación
tm_init=0.1; %Tiempo de muestreo inicial
tm_end=0.1; %Tiempo de muestreo final
Vp=0.1; %Valor de paso para el tiempo de muestreo
tm_array=tm_init:Vp:tm_end; %Vector que contiene los diferentes tiempos de
    muestreo
Size_tm_array=length(tm_array); %Número de tiempos de muestreo
maxIT=5; %Número de iteraciones por cada tiempo de muestreo

%Variables de estructura, resultados finales
[S_Main,S_Aux,S_FinalResults,S_FilterResults]=Define_Structcs(Size_tm_array,
    maxIT);

for z=1:Size_tm_array %Bucle principal, se repite para cada tiempo de muestreo

    %VARIABLES MÍNIMOS CUADRADOS
    k1=2;%fila a partir de la cual se obtienen valores válidos del vector
    %paramétrico
    k2=3; %tamaño inicial del vector regresor
    yk=1; %Número de valores pasados de y contando desde yk-1

    %Tiempo de muestreo y Simulación del sistema de tanques de agua
    tm=tm_array(z);
    Generic_Inicialization

    %Variables medidas del sistema
    time=Sim_Data.tout;
    t=length(time);
    x=Sim_Data.Input.Data;
    y=Sim_Data.Output.Data;
```

```
    for cnt=1:maxIT %Bucle secundario, se repite para cada iteración del
        %sistema variando el tamaño del vector regresor

        %Creación de las matrices "M" y "TETA" que recogen los vectores
        %paramétricos "m" y "teta" para cada tiempo de muestreo
        [M,TETA,I]=Define_LeastSquare_Matrix(k2,t,yk,x,y);

        %Algoritmo recursivo: Mínimos Cuadrados Ponderados
        Lambda=0.9;%Valor inicial del factor de olvido (se mantiene constante
        %para todo el algorítmo
        P0=998*I;%Valor inicial de la matriz P
        teta=LeastSquare_Algorithm(k1,M,TETA,I,t,y,Lambda,P0);

        %Variable estructura que recoge todos los parámetros importantes
        %referentes al algoritmo de identificación
        [S_Aux(cnt)]=Fill_Structs(M,teta,t,k1,k2,yk,time,tm,y);

        %CONDICIÓN BUCLE, aumentamos en uno el tamaño del vector regresor,
        %por cada iteración se añade un valor pasado más, primero de "y",
        %posteriormente de "u". Así hasta cumplir el número de iteraciones
        %máximas marcadas.
        k2=k2+1;
        resto=mod(k2,2);
        if(resto==0)
              yk=yk+1;
              k1=k1+1;
        end
    end

%Entre todas las iteraciones realizadas se escoge la que menor valor de
%error absoluto presente y se almacena en la variable MainStruct
[S_Main(z)]=Optimal(maxIT,S_Aux);
end

%Cálculo de los valores óptimos entre los distintos tiempos de muestreo
[S_FinalResults]=Optimal(Size_tm_array,S_Main);
```

Haciendo un desarrollo más en detalle del algoritmo, en primer lugar, se definen las variables referentes al tiempo de muestreo y el número de iteraciones máximas, las más importantes, tm_{array} y *maxIT* ya fueron explicadas previamente. Posteriormente se van a definir una serie de estructuras que almacenen los resultados que se van generando durante el algoritmo. Para ello se va a hacer uso de la función *Define Structs*

Código 2.3 Function Define Structs.

```
function [Struct1,Struct2,Struct3,Struct4]=Define_Structcs(lw,maxIT)

Size1=zeros(lw,1);
Size2=zeros(maxIT,1);
Size3=zeros(1,1);

Struct1=struct('teta',Size1,'M',Size1,'k1',Size1,'k2',Size1,'yk',Size1,'uk',
    Size1,'time',Size1,'tm',Size1,'Y',Size1,'Yaprox',Size1,'Yerror',Size1,'
    Error',Size1);
Struct2=struct('teta',Size2,'M',Size2,'k1',Size2,'k2',Size2,'yk',Size2,'uk',
    Size2,'time',Size2,'tm',Size2,'Y',Size2,'Yaprox',Size2,'Yerror',Size2,'
    Error',Size2);
```

```
Struct3=struct('teta',Size3,'M',Size3,'k1',Size3,'k2',Size3,'yk',Size3,'uk',
    Size3,'time',Size3,'tm',Size3,'Y',Size3,'Yaprox',Size3,'Yerror',Size3,'
    Error',Size3);
Struct4=struct('teta',Size3,'M',Size3,'Num',Size3,'Den',Size3,'time',Size3,'tm
    ',Size3,'Y',Size3,'Yaprox',Size3,'Yerror',Size3,'Error',Size3);

end
```

Los parámetros que se van a almacenar en estas estructuras son:

- *teta*: Vector que recoge el último valor válido del vector paramétrico tras realizar el algoritmo de identificación.
- *M*: Matriz que recoge los distintos valores de el vectores regresor para cada punto en la simulación.
- *k1*: Variable que representa el primer valor válido a partir del cual se puede implementar el vector regresor.
- *k2*: Tamaño del vector regresor.
- *yk*: Número de valores pasados de *Y* contando desde *yk-1*.
- *uk*: Número de valores pasados de *U* contando desde *yk*.
- *time*: vector que recoge la variable referente al tiempo en la simulación.
- *tm*: valor del tiempo de muestreo utilizado.
- *Y*: Valores de la salida muestreada obtenidos tras la simulación.
- *Yaprox*: Valor de la salida aproximada según la *ecuación (2.1)*.
- *Yerror*: Diferencia punto a punto entre la salida real y la aproximada.
- *Error*: Valor absoluto del error entre la salida real y la aproximada.
- *Num*: Numerador de la función de transferencia final obtenida tras el proceso de identificación.
- *Den*: Denominador de la función de transferencia final obtenida tras el proceso de identificación.

Posteriormente, definidas las estructuras, se entra en el primer bucle del algoritmo de identificación. Este se va a repetir para cada tiempo de muestreo almacenado en el vector tm_{array}. A continuación, se definen las variables referentes al tamaño y forma del vector regresor: *k1*, *k2* e *yk*, (no hace falta definir *uk* porque tal como se ha planteado el modelo se cumple que $k2 = yk + uk$). El siguiente paso es simular el sistema definido según el *código (2.1)* y que viene representado en la *figura (2.1)*. Tras la simulación se almacenan para cada tiempo de muestreo la entrada y salida del sistema.

Llegamos ahora al segundo bucle, donde se realiza el proceso de identificación. Este bucle se repite hasta que el número de iteraciones sea igual a *maxIT*, aumentando el tamaño del vector regresor como se comentó previamente. En primer lugar se van a definir las matrices *TETA* y *M* que corresponden a las ecuaciones *(2.9)*. Para ello se va a hacer uso de la función *Define LeastSquare Matrix*.

Código 2.4 Function Define LeastSquare Matrix.

```
function function [M,TETA,I]=Define_LeastSquare_Matrix(k2,t,yk,x,y)

%Las variables a y b recogen el primer valor válido tanto de la salida como
%entrada en el vector regresor
a=1;%El primer valor válido de y será yk-1
b=0;%El primer valor válido de u será uk

%Variables auxiliares para rellenar el vector paramétrico
aux1=0;%Variable contador para asegurar que se han rellenado todos los valores
%respectivos a "yk"
flag=0;%Cuando toma el valor cero al vector paramétrico se le asignan valores
%"y", cuando es uno, valores "x"
```

```
%Creación de M y TETA
M=zeros(t,k2);
TETA=zeros(k2,t);
I=eye(k2);

for i=1:t
    for j=1:k2
        if(flag==0)
            if(i>a)
                M(i,j)=y(i-a);
            end
            a=a+1;
            aux1=aux1+1;
        end
        if(flag==1)
            if(i>b)
            M(i,j)=x(i-b);
            end
            b=b+1;
        end
        if(aux1==yk)
             flag=1;
        end
    end
       aux1=0;
       flag=0;
       a=1;
       b=0;

end
```

Posteriormente, solo queda inicializar la matriz P y el factor de olvido λ (que en este caso va a tomar un valor contante) para pasar al algoritmo de identificación. El algoritmo de identificación se encuentra implementado en la función *LeastSquare Algorithm*. Este se basa en la aplicación recursiva de las ecuaciones *(2.28)*, *(2.29)* y *(2.30)* tal como se explicó en en apartado *2.2.1 Mínimos cuadrados recursivos ponderados*.

Código 2.5 Function LeastSquare Algorithm.

```
function [teta]=LeastSquare_Algorithm(k1,m,TETA,I,t,y,Lamda,Pi)

for i=k1:t
       Ki=(Pi*m(i,:).')/(Lamda+m(i,:)*Pi*m(i,:).');
       TETA(:,i)=TETA(:,i-1)+Ki*(y(i)-m(i,:)*TETA(:,i-1));
       Pi=(1/Lamda)*((I-Ki*m(i,:))*Pi);
end

teta_aux=rmmissing(TETA')';

teta=teta_aux(:,length(teta_aux));
end
```

Finalmente, una vez obtenido el vector paramétrico (objetivo último del proceso de identificación mediante mínimos cuadrados) solo queda almacenar los resultados obtenidos en las estructuras definidas en el código *(2.3)* y que el algoritmo siga iterando. Para guardar las variables resultado en dichas estructuras se utiliza la función *Fill Structs*.

Código 2.6 Function Fill Structs.

```
function [S]=Fill_Structs(m,teta,t,k1,k2,yk,time,tm,y)

S.Y=y;
S.teta=teta;
S.M=m;
S.k1=k1;
S.k2=k2;
S.yk=yk;
S.uk=k2-yk;
S.time=time;
S.tm=tm;

r1=zeros(t,1);
r2=zeros(t,1);

for i=k1:t
    r1(i)=m(i,:)*teta; %Salida aproximada
end

for i=k1:t
    r2(i)=y(i)-r1(i); %Curva que representa la diferencia entre la
    %salida real y la aproximada
end


Atotal=0;

for i=1:t
    Atotal=Atotal+abs(r2(i)); %Error de aproximación
end


S.Yaprox=r1;
S.Yerror=r2;
S.Error=Atotal;
```

Dentro de la función *Fill Structs* también se calcula la salida aproximada, definida en la ecuación *(3.1)* y el error absoluto, definido como el sumatorio de errores absolutos punto a punto entre la salida real y aproximada. Para finalizar el proceso de identificación, solo queda seleccionar entre todas las iteraciones realizadas la que mejor haya caracterizado el sistema. Se podrían haber utilizado aquí distintas metodologías para la selección del resultado final. Pero, a modo de simplificación, se va a escoger el vector paramétrico cuya salida aproximada presente menor absoluto según la ecuación:

$$Error = \sum_{k=j}^{n} abs(y(k) - \hat{y}(k)) \tag{2.37}$$

Tal como se calcula en la función *Fill Structs*. Este proceso de selección se va a realizar dos veces. En primer lugar por cada tiempo de muestreo definido, al haberse definido distintos tamaños del vector regresor, se va a escoger el par vector paramétrico - vector regresor con menor error absoluto. Posteriormente se realizará esta misma selección comparando el "*mejor*" resultado de los distintos tiempos de muestreo. Se ha definido una función denominada *Optimal* encargada de realizar dicha comparativa entre los errores absolutos y seleccionar el resultado más óptimo según esta metodología.

Código 2.7 Function Optimal.

```
function [S1]=Optimal(maxIT,S2)

S1.Error=S2(1).Error;
Aux=1;

for i=2:maxIT

    if(abs(S2(i).Error)<=S1.Error)
        S1.Error=abs(S2(i).Error);
        Aux=i;
    end
end

S1.teta=S2(Aux).teta;
S1.M=S2(Aux).M;
S1.Y=S2(Aux).Y;
S1.k1=S2(Aux).k1;
S1.k2=S2(Aux).k2;
S1.yk=S2(Aux).yk;
S1.uk=S2(Aux).uk;
S1.time=S2(Aux).time;
S1.tm=S2(Aux).tm;
S1.Yaprox=S2(Aux).Yaprox;
S1.Yerror=S2(Aux).Yerror;
```

Finalizado el proceso de identificación y habiendo elegido el par vector regresor - vector paramétrico con menor error absoluto, se pasa a representar la salida real del sistema frente a la salida muestreada así como el error cometido.

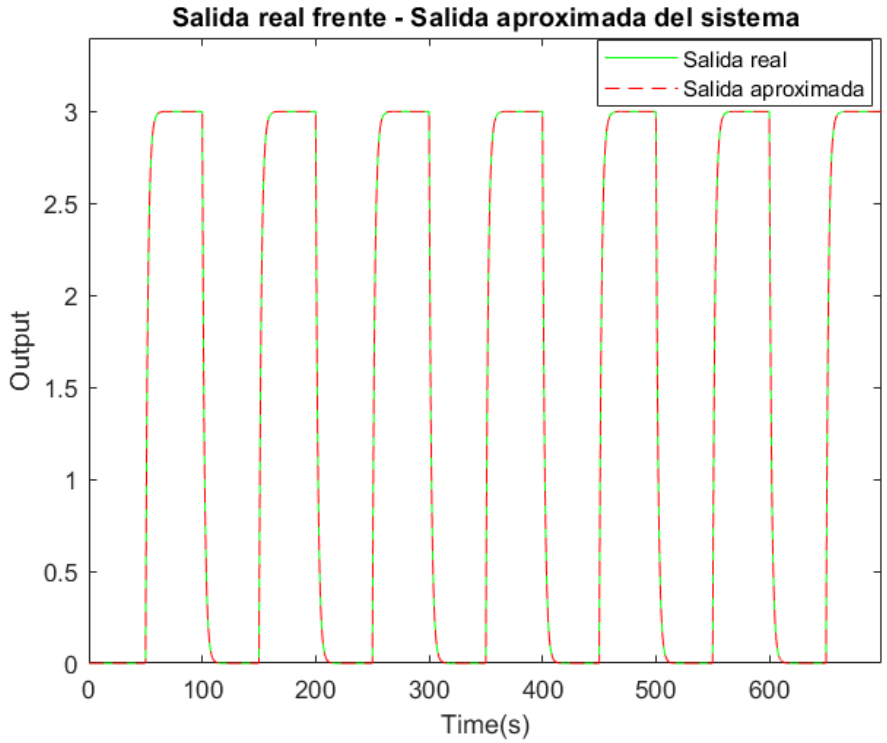

Figura 2.2 Salida real vs Salida aproximada del sistema.

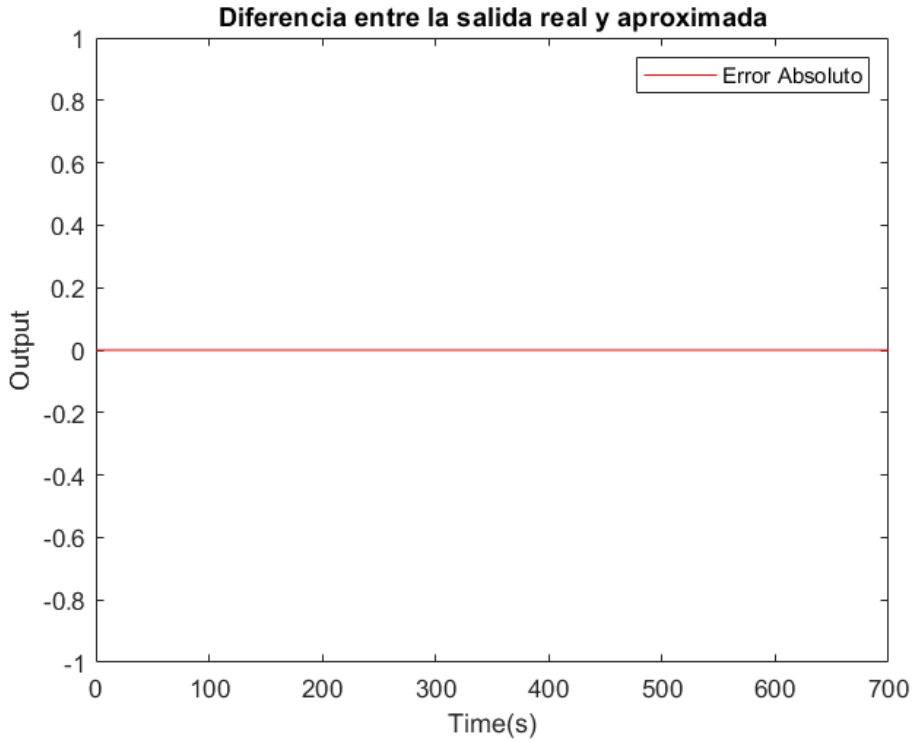

Figura 2.3 Error absoluto proceso de identificación.

Se puede observar de las figuras anteriores, que la salida muestreada es idéntica a la salida real y no se comete error en ningún punto de la simulación.

2.3.1 Filtro del vector paramétrico

El algoritmo de identificación por mínimos cuadrados quedaría implementado con el código *(2.2)* que fue explicado en el punto anterior. Pero además, se ha desarrollado una función tipo filtro que optimice la identificación y evite posteriores errores numéricos. La idea es la siguiente, una vez obtenido el vector paramétrico se va a pasar un filtro al mismo, este filtro va a eliminar los elementos del vector paramétrico que presenten un valor absoluto menor a un parámetro tolerancia "tol_{LS}". Ejemplificando la idea, si tomamos como referencia la ecuación *(2.2)*, si el valor de θ_1 fuera menor que 10^{-8} se podría concluir que el valor de la salida un periodo de muestreo antes del actual (y_{k-1}) no influye en la salida del sistema y por tanto eliminarlo de la ecuación. Para implementar esta idea se ha desarrollado una función denominada *Filter*.

Código 2.8 Function Filter.

```
function [SF] = Filter (S,filter)

S_aux=S;
teta_aux=S.teta;
cnt=0;
for i=1:S.k2
    if abs(S.teta(i)) <= filter
        S_aux.teta(i-cnt,:)=[];
        S_aux.M(:,i-cnt)=[];
        teta_aux(i)=0;
        cnt=cnt+1;
    end
end
```

```
SF.teta=S_aux.teta;
SF.M=S_aux.M;
SF.Y=S.Y;
SF.time=S.time;
SF.tm=S.tm;

t=length(SF.Y);

r1=zeros(t,1);
r2=zeros(t,1);

for i=S.k1:t
    r1(i)=SF.M(i,:)*SF.teta; %Salida aproximada
end

for i=S.k1:t
    r2(i)=SF.Y(i)-r1(i); %Curva que representa la diferencia entre la
    %salida real y la aproximada
end


Atotal=0;

for i=1:t
    Atotal=Atotal+abs(r2(i)); %Error de aproximación
end

SF.Yaprox=r1;
SF.Yerror=r2;
SF.Error=Atotal;

[SF.Num,SF.Den]=FunTransfer(teta_aux,S.yk,S.uk);
```

Dentro de esta función filtro, se vuelve a realizar la acción de almacenamiento en una estructura final con el objetivo de recoger todos los resultados del algoritmo. A su vez, también se ejecuta al finalizar el filtro, la función *FunTransfer*. En dicha función se calcula el numerador y denominador de la función de transferencia obtenida tras el proceso de identificación a partir del vector paramétrico final y la composición del vector regresor.

Código 2.9 Function FunTransfer.

```
function [Num,Den]=FunTransfer(teta,ykopt,ukopt)

Den=zeros(1,(ykopt+1));
Num=zeros(1,ukopt);
Den(1)=1;
for i=2:(ykopt+1)
    Den(i)=-teta(i-1);
end
uk=ykopt+1;
for i=1:ukopt
    Num(i)=teta(uk);
    uk=uk+1;
end
```

Finalmente se van a poner en conjunto todos los resultados obtenidos:

- *Vector paramétrico resultante*:

$$\theta = \begin{bmatrix} 0.95 \\ 0 \\ 0.5 \end{bmatrix}$$

- *Forma del vector regresor resultante*:

$$m(k) = \begin{bmatrix} y_{k-1} & u_k & u_{k-1} \end{bmatrix}$$

- *Vector paramétrico tras aplicar el filtro*:

$$\theta = \begin{bmatrix} 0.95 \\ 0.5 \end{bmatrix}$$

- *Forma del vector paramétrico tras aplicar el filtro*:

$$m(k) = \begin{bmatrix} y_{k-1} & u_{k-1} \end{bmatrix}$$

- *Función de transferencia a identificar*:

$$G(z) = \frac{0.5}{z - 0.95}$$

- *Función de transferencia resultante*:

$$G(z) = \frac{0.5}{z - 0.95}$$

Observando los resultados anteriores, es visible que la función de transferencia obtenida tras el proceso de identificación es idéntica a la que se modelo el sistema. Por tanto, se puede concluir, que el algoritmo de identificación funciona correctamente.

2.3.2 Aplicación a un sistema con ruido

En el caso previo, se utilizo la algoritmia desarrollada durante la sección *2.2.1 Mínimos cuadrados recursivos ponderados* para implementar un script en Matlab capaz de identificar un sistema lineal genérico en el caso de que el mismo no presentara ruido. En este apartado,se sigue la base del código desarrollado en las secciones previas, junto con la algoritmia descrita en la sección *2.2.2 Mínimos cuadrados recursivos ponderados, sistema con ruido* para implementar un método de identificación capaz de identificar sistemas cuya salida se vea afectada por un ruido de media no nula. Partimos de la misma base que el apartado anterior siendo la función a identificar igual a:

$$G(z) = \frac{0.5}{z - 0.95} \tag{2.38}$$

Pero en este caso, el modelo de bloques a simular en *Simulink* se corresponde con la siguiente figura:

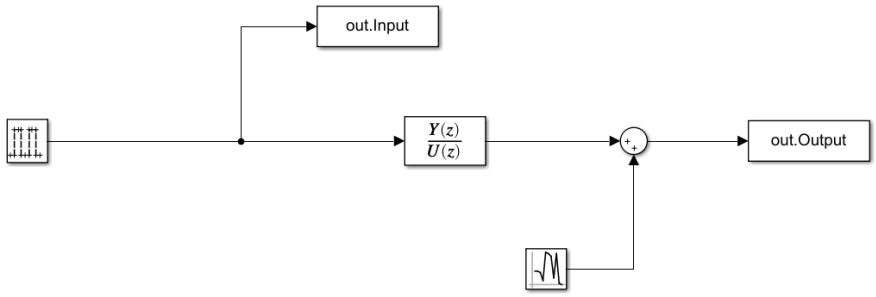

Figura 2.4 Diagrama de bloques del modelo a identificar, sistema con ruido.

Se puede observar que el modelo de bloques es idéntico al que se utilizó en el caso anterior y aparece en la figura (*2.1*), salvo, que la salida de este sistema se ve afectada por una variable ruido. Para definir este ruido se ha utilizado el bloque *Random Number*. Para este caso el valor de esperanza y media del ruido V_e son:

$$\begin{aligned} \Sigma(V_e) &= 0.001 \\ \sigma(V_e) &= 0.0001 \end{aligned} \tag{2.39}$$

En la siguiente sección de código que hace referencia al script *Generic Inicialization Wnoise* se definen los parámetros referentes al sistema a controlar, variables de simulación y el ya mencionado ruido.

Código 2.10 Generic Inicialization Wnoise.

```
%PARÁMETROS SIMULACIÓN
tm=0.1;
puntos=7000; %Número de Samples Times que tendrá la simulación
Pt=7; %Número de periodos que tendrá la simulación
[ts,Period,Width]=DefinePeriod(puntos,tm,Pt);
K=0.3; %Amplitud de la señal Generador de Pulsos
%Numerador y Denominador de la función de transferencia
Num=0.5;
Den=[1 -0.95];

%Ruido
Ve=0.01;
Qve=0.0001;

Sim_Data=sim('Generic_Simulation_WNoise');
```

Volviendo al algoritmo de los mínimos cuadrados, para este ejemplo se va a ser menos detallado en la explicación del mismo, puesto que es una ampliación del código descrito en las secciones (*2.3*) y (*2.3.1*), No se van a tener en cuenta distintos bucles y se va a identificar la función *G(z)* para un único tiempo de muestreo y un tamaño fijo del vector paramétrico. Para este caso concreto y siguiendo la algoritmia de la sección *2.2.2 Mínimos cuadrados recursivos, sistema con ruido* el vector regresor tendrá la siguiente forma:

$$m(k) = \begin{bmatrix} y_{k-1} & y_{k-2} & u_{k-1} & e_{k-1} \end{bmatrix} \tag{2.40}$$

El tiempo de muestreo utilizado será $t_m = 0.1$. Centrándonos en el código, en la siguiente sección adjunta se observa todo lo relativo a la inicialización del sistema a identificar según se vio en el código *Generic Inicialization WNoise*, definición de las variables referentes a la forma del vector regresor (*k1, k2, uk e yk*) y la creación de dos variables estructura (*S Aux* y *S Results*) con el fin de recoger los resultados obtenidos.

Código 2.11 Inincialización algoritmo mínimos cuadrados recursivos, sistema con ruido.

```
maxIT=1; %Número de iteraciones por cada tiempo de muestreo

%Variables de estructura, resultados finales
%Variables de estructura, resultados finales
S_Aux=struct('teta',0,'M',0,'Num',0,'Den',0,'time',0,'tm',0,'Y',0,'Yaprox',0,'
    Yerror',0,'Error',0);
S_Results=struct('teta',0,'M',0,'Num',0,'Den',0,'time',0,'tm',0,'Y',0,'Yaprox
    ',0,'Yerror',0,'Error',0);


%VARIABLES MÍNIMOS CUADRADOS
k1=3;%fila a partir de la cual se obtienen valores válidos del vector
%paramétrico
k2=2; %tamaño inicial del vector regresor
yk=1; %Número de valores pasados de y contando desde yk-1
uk=1; %Número de valores pasados de y contando desde uk-1

%Simulación del sistema de tanques de agua
Generic_Inicialization_WNoise

%Variables medidas del sistema
time=Sim_Data.tout;
t=length(time);
x=Sim_Data.Input.Data;
y=Sim_Data.Output.Data;
```

Inicializada la algoritmia, se pasa a la identificación del sistema. En primer lugar se inicializa el factor de olvido λ y la matriz P. Se definen las matrices M y $TETA$ que recogerán respectivamente los vectores regresor y paramétrico durante toda la simulación. Posteriormente, se pasa a ejecutar el algoritmo de identificación, que se ha definido dentro de la función *Least Square Algorith WNoise*. Calculado el vector paramétrico *teta* se almacenarían los parámetros deseados en la variable estructura *S Aux* y se modificaría el tamaño y forma del vector paramétrico (no ocurre para este caso particular). Finalmente, se utiliza la función *Optimal* que se definió en el código (*2.7*) para elegir entre el vector paramétrico que presente menor valor absoluto de error. Se adjunta la sección de código que implementa todo lo definido en este párrafo.

Código 2.12 Algoritmo de identificación, mínimos cuadrados recursivos, sistema con ruido.

```
for cnt=1:maxIT %Bucle

    %Algoritmo recursivo: Mínimos Cuadrados Ponderados
    Lambda=0.9;%Valor base del factor de olvido
    Pi=998;%Valor inicial de la matriz P

    %Creación de M y TETA
    M=zeros(t,k2);
    TETA=zeros(k2,t);

    %Algoritmo mínimos cuadrados
    [teta,M]=LeastSquare_Algorithm_WNoise(k1,k2,M,TETA,t,y,x,Lambda,Pi,yk,uk);

    %Variable estructura que recoge todos los parámetros importantes
    %referentes al algoritmo de identificación
    S_Aux(cnt)=Fill_Structs_WNoise(M,teta,t,k1,k2,time,tm,y,yk,uk);
```

```
        %CONDICIÓN BUCLE
        k2=k2+1;
        resto=mod(k2,2);
        if(resto==0)
              yk=yk+1;
              k1=k1+1;
        else
               uk=uk+1;
        end
   end

%Entre todas las iteraciones realizadas se escoge la que menor valor de
%error absoluto presente y se almacena en la variable MainStruct
S_Results=Optimal(maxIT,S_Aux);
```

Nos vamos a centrar ahora en la función *LeastSquare Algorith WNoise* que es la que implementa la algoritmia de identificación.

Código 2.13 Function LeastSquare Algotithm WNoise.

```
function [teta,M]=LeastSquare_Algorithm_WNoise(k1,k2,M,TETA,t,y,x,Lambda,Pi,yk,
    uk)

        for i =k1:t
            %Variable ruido
            if (i==k1)
                e=y(i-1);
            else
                e=y(i-1)-M(i-1,:)*TETA(:,i-2);
            end
            %Creación de los vectores "m" y "teta"
            [m,I]=Define_LeastSquare_Vectors_Noise(i,k2,yk,uk,x,y,e);

            %Inicialización Matriz P
            if (i==k1)
                Pi=998*I;
            end
            M(i,:)=m; %M
           % Lambda=Lambda^(t-k1);
            Ki=(Pi*M(i,:).')/(Lambda+M(i,:)*Pi*M(i,:).');
            TETA(:,i)=TETA(:,i-1)+Ki*(y(i)-M(i,:)*TETA(:,i-1));
            Pi=(1/Lambda)*((I-Ki*M(i,:))*Pi);
        end

        teta_aux=rmmissing(TETA')';
        teta=teta_aux(:,length(teta_aux));
end
```

En primer lugar, se calcula el error del sistema según la ecuación (*2.35*). Posteriormente se usa la función *Define LeastSquare Vectors Noise* una variante de la función expuesta en el código (2.4) *Define LeastSquare Matrix* donde se actualizan los valores del vector regresor, en este caso modificados para añadir el valor referente al ruido identificado.

Código 2.14 Function Define LeastSquare Vectors Noise.

```
function function [m,I]=Define_LeastSquare_Vectors_Noise(j,k2,yk,uk,x,y,e)
```

```
%Las variables a y b recogen el primer valor válido tanto de la salida como
%entrada en el vector regresor
a=1;%El primer valor válido de y será yk-1
b=0;%El primer valor válido de u será uk-1

%Variables auxiliares para rellenar el vector paramétrico
aux1=0;%Variable contador para asegurar que se han rellenado todos los valores
%respectivos a "yk"
aux2=0;%Variable contador para asegurar que se han rellenado todos los valores
%respectivos a "uk"
flag=0;%Cuando toma el valor cero al vector paramétrico se le asignan valores
%"y", cuando es uno, valores "x"

%Creación de M y TETA
m=zeros(1,k2);
I=eye(k2);

for i=1:k2
    if(flag==0)
        m(i)=y(j-a);
        a=a+1;
        aux1=aux1+1;
    end
    if(flag==1)
        m(i)=x(j-b);
        b=b+1;
        aux2=aux2+1;
    end
    if(flag==2)
        m(i)=e;
    end
    if(aux1==yk)
        flag=1;
        if(aux2==uk)
            flag=2;
        end
    end
end
```

Definido el vector regresor, solo queda aplicar las ecuaciones (*2.29*), (*2.30*) y (*2.31*) e iterar tantos puntos como tenga la simulación para identificar el valor del vector paramétrico *teta*. En las dos siguientes gráficas se puede observa la salida real del sistema frente a la obtenida tras la identificación del sistema. Así mismo también se puede observar el error absoluto cometido tras el proceso de identificación.

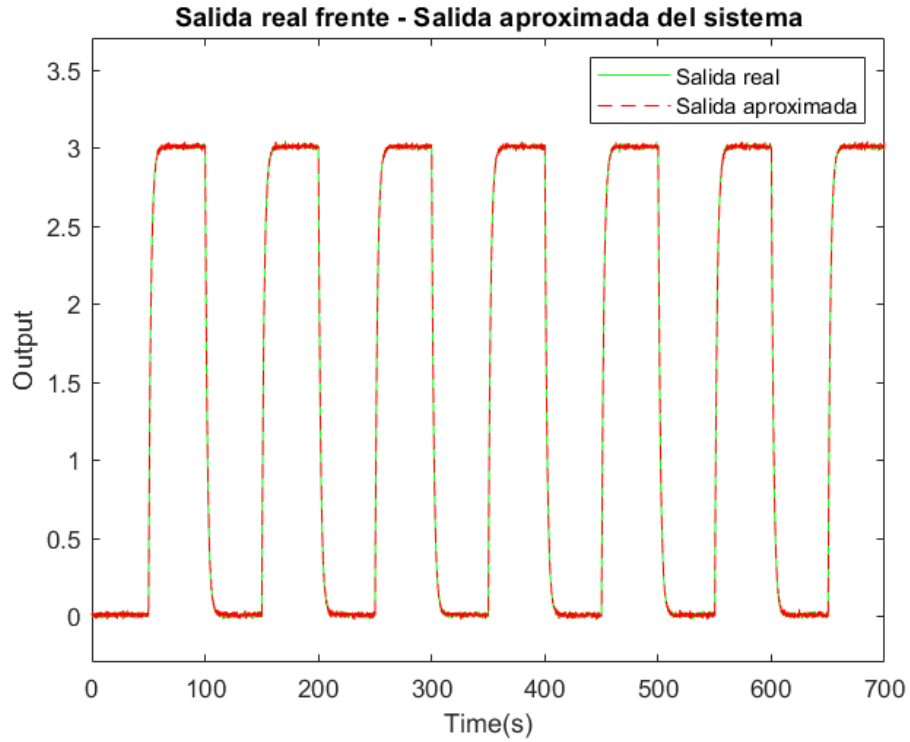

Figura 2.5 Salida real vs Salida aproximada, sistema con ruido.

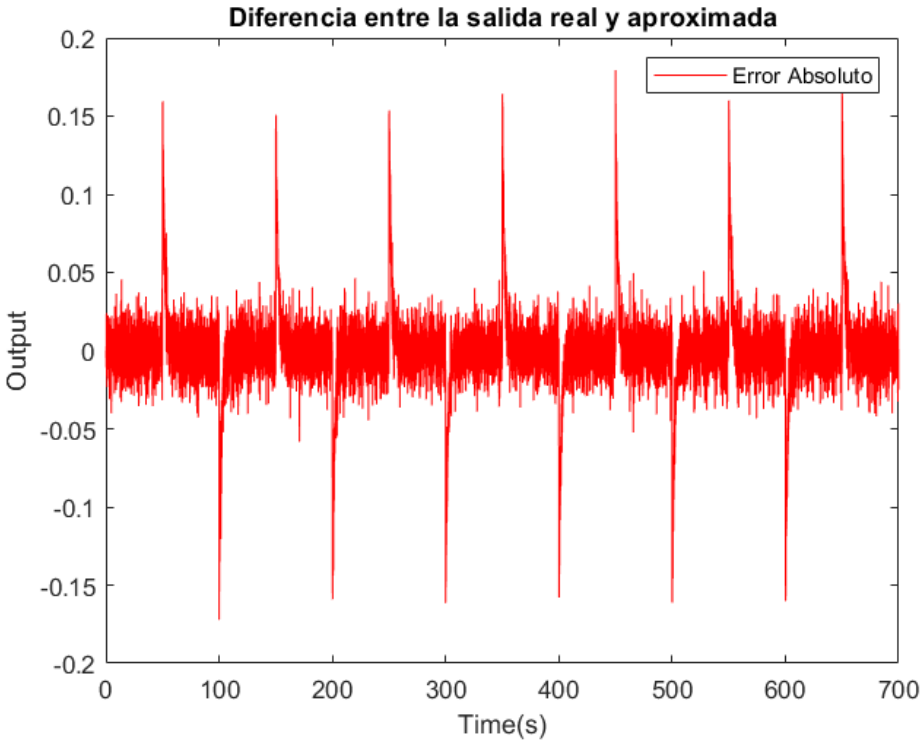

Figura 2.6 Error absoluto proceso de identificación, sistema con ruido.

Finalmente se calcula la función de transferencia del sistema identificado. Se observa que no se corresponde con la misma función de transferencia definida en la ecuación (*2.37*). Esto se debe a que el ruido introducido

imposibilita que la identificación sea exacta. Pero observando la *Figura 2.6* se concluye que el error es aproximadamente igual al ruido del sistema por tanto se convalida el proceso de identificación.

$$G(z) = \frac{-0.01336}{z - 1.01} \tag{2.41}$$

3 Implementación de un controlador discreto según el método de diseño Truxal-Ragazzini

El controlador elegido para implementar la base de datos es un controlador en discreto generado a partir del método de diseño Truxal-Ragazzini.

3.1 Introducción

El método de síntesis directa o de Truxal-Ragazzini es una técnica de diseño de controladores en régimen discreto. Dicho método permite diseñar un controlador *C(z)* de forma que la función en bucle cerrado del sistema $G_d(z)$ presente error en régimen permanente nulo ante una referencia particular, tras un número *N* de periodos de muestro. La idea básica del método de Truxal-Ragazzini consiste en pre-diseñar la función en bucle cerrado del sistema $G_d(z)$ (imponiendo a ésta una serie de condiciones) para posteriormente, obtener la función de transferencia del propio controlador del sistema. En la *Figura 3.1* podemos observar un ejemplo del diagrama de bloques de un controlador Truxal.

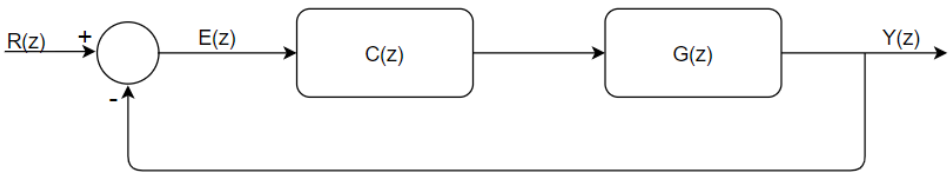

Figura 3.1 Diagrama de bloques control sistema discreto en bucle cerrado.

Tal como se ha diseñado el sistema, la función en bucle cerrado del mismo tendrá la siguiente forma:

$$G_d(z) = \frac{G(z)C(z)}{1+C(z)G(z)} \tag{3.1}$$

Y según la idea del método, como conocemos la función de transferencia en bucle cerrado del sistema $G_d(z)$, la hemos diseñado nosotros, podemos obtener la función del propio controlador despejando de la ecuación anterior.

$$C(z) = \frac{1}{G(z)} \frac{G_d(z)}{1-G_d(z)} \tag{3.2}$$

3.2 Formulación del controlador Truxal-Ragazzini

Un controlador diseñado según éste método deberá cumplir 3 condiciones: tendrá que ser físicamente realizable, debe presentar estabilidad interna y error en régimen permanente nulo.

3.2.1 Condición de realización física

Para que el controlador sea realizable primero deberá ser causal, es decir, definiendo el controlador *C(z)* como:

$$C(z) = \frac{N_c(z)}{D_c(z)} \tag{3.3}$$

Se deberá cumplir que:

$$grado(D_c(z)) \geq grado(N_c(z)) \tag{3.4}$$

Si definimos la función del sistema tanto en bucle abierto como cerrado de la misma forma, lo unimos a la ecuación *(3.3)* y operamos:

$$G(z) = \frac{N(z)}{D(z)}; G_d(z) = \frac{N_d(z)}{D_d(z)} \tag{3.5}$$

$$C(z) = \frac{D(z)}{N(z)} \frac{N_d(z)}{D_d(z) - N_d(z)} \tag{3.6}$$

Observando esta última expresión se puede concluir que para que el sistema sea causal se ha de cumplir que:

$$grado(D(z)) + grado(N_d(z)) \leq grado(N(z)) + grado(Dd(z)) \tag{3.7}$$

3.2.2 Condición de estabilidad interna

No puede haber cancelación de polos inestables entre *C(z)* y *G(z)*. Dado que no se puede asegurar la exactitud de la cancelación, esto daría lugar a problemas de inestabilidad interna del sistema. Lo mismo ocurriría con cancelaciones de ceros inestables o críticos fuera del circulo unidad. Suponiendo que *G(z)* tenga un polo inestable $\alpha \geq 1$ definimos de nuevo la función de transferencia del sistema en bucle cerrado.

$$G(z) = \frac{G_1(z)}{z - \alpha} \tag{3.8}$$

$$G_d(z) = \frac{C(z)\frac{G_1(z)}{z-\alpha}}{1 + C(z)\frac{G_1(z)}{z-\alpha}} \tag{3.9}$$

$$1 - G_d(z) = \frac{1}{1 + C(z)\frac{G_1(z)}{z-\alpha}} = \frac{z - \alpha}{z - \alpha + C(z)G_1(z)} \tag{3.10}$$

Por tanto para que no haya cancelación, los polos inestables de *G(z)* deberán aparecer como ceros de 1-$G_d(z)$.

Para el segundo caso se va a considerar un cero inestable en bucle abierto de valor $\beta \geq 1$ .Definiendo la función de transferencia en bucle abierto como *G(z)*=$\frac{z-\beta}{G_2(z)}$ y volviendo a operar como en el caso anterior obtenemos:

$$G_d(z) = \frac{C(z)\frac{z-\beta}{G_2(z)}}{1 + C(z)\frac{z-\beta}{G_2(z)}} = C(z) \cdot \frac{z - \beta}{G_2 + C(z)(z - \beta)} \tag{3.11}$$

Por lo que finalmente podemos concluir que los ceros inestables de la función de transferencia en bucle abierto deberán aparecer como ceros de la función de transferencia en bucle cerrado.

3.2.3 Condición de error en régimen permanente nulo

Finalmente el controlador debe actuar de manera que el sistema presente error nulo en régimen permanente. Tanto el error en bucle cerrado, como el error en régimen permanente de un sistema se definen respectivamente:

$$E(z) = R(z) - Y(z) = (1 - G_d(z))R(z) \tag{3.12}$$

$$erp = \lim_{z\to 1}(z-1)(1-G_d(z))R(z) \tag{3.13}$$

La entrada *R(z)* se puede definir de forma genérica como $R(z) = \frac{P(z)}{(1-z^{-1})^{q+1}}$. Si particularizamos para el caso de una entrada tipo escalón donde *q=0* y *P(z)=1* se ha de cumplir:

$$erp = \lim_{z\to 1}(z-1)(1-G_d(z))\frac{z}{z-1} = \lim_{z\to 1} z(1-G_d(z)) = 0 \tag{3.14}$$

Por tanto:

$$G_d(1) = 1$$

En este caso se ha particularizado para una entrada tipo escalón pues es la que se va a implementar en el proyecto. Pero el controlador Truxal es extensible a cualquier tipo de entrada: impulso, rampa, parabólica...

3.3 Implementación del controlador Truxal-Ragazzini en Matlab

El código implementado tratará de obtener un controlador en discreto *C(z)* aplicando el método de diseño de controladores Truxal-Ragazzini a partir de los coeficientes del numerador y denominador de una función de transferencia en discreto cualquiera *G(z)* y el tiempo de paso de la misma. Se presenta a continuación el código de la función diseñada.

Código 3.1 Function ControlTruxal.

```
function [Num_Cz,Den_Cz,Num_Gd,Den_Gd,Gd_z,C_z]=ControlTruxal(Num_G,Den_G,
    tm_opt)
```

En esta versión del código no se considera el caso de que la función de transferencia presente polos inestables en bucle abierto.

En primer lugar se calculan las raíces del numerador del sistema en bucle abierto en la búsqueda de posibles ceros inestables. Estas raíces se almacenan en el vector $roots_{NumG}$ y se redondean al quinto valor decimal con el fin de evitar posibles problemas numéricos futuros. Posteriormente se revisa si el valor absoluto de los elementos almacenados en dicho vector es mayor que uno. Dado que en discreto los polos y ceros inestables se encuentran fuera del círculo unidad.

Código 3.2 Cálculo de ceros inestables.

```
%Raices de la función de transferencia en bucle abierto
roots_DenG=round(roots(Den_G),5);
roots_NumG=round(roots(Num_G),5);

%Raices de la función de transferencia en bucle cerrado
Grade_DenG=length(roots_DenG);
Grade_NumG=length(roots_NumG);

%Cálculo del número de ceros inestables
cnt=0;
j=1;
for i=1:Grade_NumG
    if(abs(roots_NumG(i))>1)
        cnt=cnt+1;
        zeros_Gd_Inestables(j)=roots_NumG(i);
```

```
        j=j+1;
    end
end
```

Conociendo el número de ceros inestables, se define el grado del numerador de la función de transferencia en bucle cerrado como $Grado_{NumG_d} = N^o_{CerosInestables} + 1$. Y dado que conocemos el grado del numerador y denominador de la función de transferencia en bucle abierto, se resuelve la ecuación para calcular el grado del denominador de la función en bucle cerrado que evita posibles problemas de causalidad.

$$grado(D(z)) = grado(N_d(z)) - grado(N(z)) + grado(D_d(z)) \tag{3.15}$$

Código 3.3 Cálculo del grado del numerador y denominador de la función de transferencia en bucle cerrado.

```
%Cálculo del Grado de la función de transferencia en bucle cerrado
Grade_NumGd=1+cnt;
Grade_DenGd=Grade_NumGd + Grade_DenG - Grade_NumG;
```

Posteriormente, se definen los vectores que van a recoger los polos y ceros de la función de transferencia en bucle cerrado $poles_{G_d}$ y $zeros_{G_d}$. Los polos en bucle cerrado se van a definir como valores aleatorios favorables dentro del círculo unidad. El vector de ceros, en primer lugar, se va a completar con los valores de ceros inestables obtenidos previamente, evitando así la cancelación de ceros inestables. Aun así, queda pendiente un valor en el vector de ceros por definir. El algoritmo se ha definido dejando libre este valor, para que sea posteriormente utilizado como parámetro en la resolución de la ecuación que anula el error en régimen permanente ante una entrada tipo escalón, $G_d(1) = 1$.

Código 3.4 Cálculo de los ceros y polos de la función de transferencia en bucle cerrado.

```
%Cálculo de los polos y ceros de la función en bucle cerrado
poles_Gd=zeros(1,Grade_DenGd);
zeros_Gd=zeros(1,Grade_NumGd);

%Se colocan los polos en valores favorables dentro del circulo unidad
for i=1:Grade_DenGd
    poles_Gd(i)=-1*((-1)^i)*(0.2+0.2*rand);
end
if Grade_NumGd > 1
    for i=(Grade_NumGd-1)
        zeros_Gd(i)=zeros_Gd_Inestables(i);
    end
end

%Cálculo del ultimo cero de la función de transferencia para que se cumple
%la ecución del error en régimen permanente igual a cero Gd(1)=1
K=1;
B=1;

for i=1:Grade_DenGd
        K=K*(1-poles_Gd(i));
end
for i=1:(Grade_NumGd-1)
     B=B*(1-zeros_Gd(i));
end

zeros_Gd(Grade_NumGd)=1-K/B;
```

Conociendo los polos y ceros de la función de transferencia en bucle cerrado, ésta queda completamente definida. Se calculan posteriormente los valores de los coeficientes del numerador y denominador de la función de transferencia en bucle cerrado para finalmente resolver la *Ecuación 3.6* y obtener la función de transferencia en discreto del controlador Truxal-Ragazzini C_z

Código 3.5 Cálculo de la función de transferencia del controlador Truxal-Ragazzini.

```
%Resultados Finales. Numerador y Denominador de la función de transferencia
%en bucle cerrado, así como del controlador Truxal
Num_Gd=poly(zeros_Gd);
Den_Gd=poly(poles_Gd);

%Ajuste para poder restar numerador y denominador de Gd
Num_Gd_Aux=zeros(1,Grade_DenGd+1);
for i=1:(Grade_NumGd+1)
    Num_Gd_Aux(Grade_DenGd - Grade_NumGd + i)=Num_Gd(i);
end
Num_Cz=conv(Den_G,Num_Gd);
Den_Cz=conv(Num_G,(Den_Gd-Num_Gd_Aux));

Gd_z=tf(Num_Gd,Den_Gd,tm_opt);
C_z=tf(Num_Cz,Den_Cz,tm_opt);
```

3.4 Aplicación del controlador diseñado según el método Truxal-Ragazzini a un sistema conocido. Matlab y Simulink.

En este apartado se muestran los resultados de aplicar el método de diseño de controladores Truxal-Ragazzini a una función de transferencia en discreto. Para ello se va a definir una función de transferencia a la que se va a aplicar el algoritmo definido en el capítulo previo *Capítulo 3.3: Implementación del controlador Truxal-Ragazzini en Matlab*. Para el caso propuesto se han definido el numerador Num_G y denominador Den_G de una función de transferencia en discreto con tiempo de paso tm_{opt}. A partir de estos valores se ha calculado la función de transferencia en discreto de un controlador *C(z)* aplicando la función *ControlTruxal* definida previamente.

Código 3.6 Cálculo de la función de transferencia del controlador Truxal-Ragazzini aplicado a un sistema conocido.

```
%Definición de una Función de Transferencia
Num_G=0.5;
Den_G=[1 -0.95];
tm_opt=0.1;
G_z=tf(Num_G,Den_G,tm_opt);

%Calculo del Controlador Truxal
[Num_Cz,Den_Cz,Num_Gd,Den_Gd,Gd_z,C_z]=ControlTruxal(Num_G,Den_G,tm_opt);
```

Posterior a la ejecución de dicha función, las funciones de transferencia $G(z)$, $Gd(z)$ y $C(z)$ quedan definidas como:

$$G(z) = \frac{0.5}{z-0.95} \tag{3.16}$$

$$Gd(z) = \frac{z-0.1201}{z^2+0.01821z-0.1383} \tag{3.17}$$

$$C(z) = \frac{z^2-1.07z+0.1141}{0.5z^2-0.490*9z-0.009107} \tag{3.18}$$

Una vez obtenida la función de transferencia del controlador *C(z)* se va a simular la respuesta en bucle cerrado del sistema completo. Para la simulación del modelo se utilizará la herramienta *Simulink*. El sistema a simular tendrá como referencia una señal de tren de pulsos rectangular y vendrá definido por el siguiente esquema general:

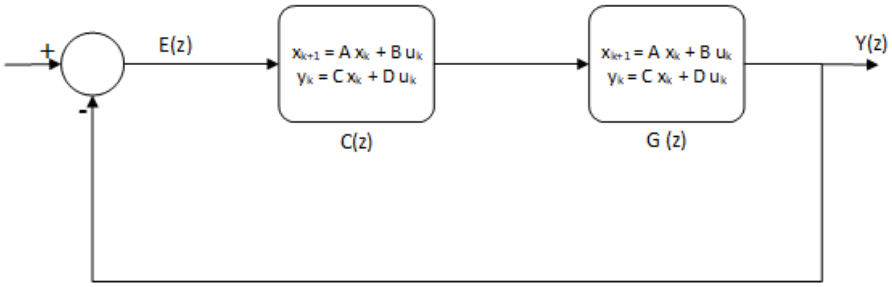

Figura 3.2 Diagrama de Bloques controlador Truxal.

Como puede observarse en la figura previa, se va a utilizar la forma en espacio estados de las funciones de transferencia *C(z)* y *G(z)*. Esto es debido, a que desde el software *Simulink*, el único método de inicializar un modelo y que de esta forma el sistema parta de un punto de estabilidad es haciendo uso de su representación en espacio estados. Para pasar las funciones de transferencia en discreto *C(z)* y *G(z)* a su forma canónica de control se ha utilizado una función propia *StateSpace Transformation*.

Código 3.7 Paso de función de transferencia a espacio estados.

```
%Paso de la función de transferencia en discreto a su forma en espacio
%estados
[A_G,B_G,C_G,D_G]=StateSpace_Transformation(Num_G,Den_G);
[A_Cz,B_Cz,C_Cz,D_Cz]=StateSpace_Transformation(Num_Cz,Den_Cz);
```

Código 3.8 Function StateSpace Transformation.

```
function [A,B,C,D]=StateSpace_Transformation(Num,Den)

%Asumo que el denominador siempre será mayor que el numerador
size1ant=length(Num);
size2=length(Den);
aux=Den(1);
    for i=1:size1ant
        if (Num(i)~=0)
            Num(i)=Num(i)/aux;
        end
```

```
    end
    for i=1:size2
        if Den(i)~=0
            Den(i)=Den(i)/aux;
        end
    end

if size1ant<size2
    Aux=Num;
    Num=zeros(1,size2);
    j=size2;
    for i=0:size1ant-1
         Num(j)=Aux(size1ant-i);
         j=j-1;
    end
end

size1=size2;
size2=size2-1;
A=zeros(size2);
for i=1:size2
    for j=2:size2
        if((i+1)==j)
            A(i,j)=1;
        end
    end0
end
t=0;
for j=1:size2
    A(size2,j)=-Den(size2+1-t);
    t=t+1;
end

B=zeros(size2,1);
B(size2,1)=1;

C=zeros(1,size2);

for i=1:size2
    C(i)=Num(size1-i+1)-Den(size1-i+1)*Num(1);
end

D=Num(1);
```

Los resultados obtenidos son los siguientes:

- Representación de la función de transferencia *G(z)* en su forma canónica de control:

$$x_{k+1} = 0.95x_k + u_k \tag{3.19}$$
$$y_k = 0.5x_k \tag{3.20}$$

- Representación de la función de transferencia *C(z)* en su forma canónica de control:

$$x_{k+1} = \begin{pmatrix} 0 & 1 \\ 0.0182 & 0.9818 \end{pmatrix} x_k + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_k \tag{3.21}$$
$$y_k = \begin{pmatrix} 0.2647 & -0.1767 \end{pmatrix} x_k + 2u_k \tag{3.22}$$

Para este ejemplo se van a definir dos casos de simulación. En el primero se simulará el sistema sin ningún ruido ni perturbación, en el segundo, la entrada y salida del la función de transferencia *G(z)* vendrán afectadas por sendos ruidos de media no nula. En ambos casos, las variables que representan los parámetros más importantes de simulación son los siguientes:

Código 3.9 Variables de simulación.

```
%      Valores que se utilizarán para la simulación del modelo    %
%Valor inferior de la señal de tren de pulsos rectangular
re0=1;
%Valor superior de la señal de tren de pulsos rectangular
re1=2;
%Número de Samples Times que tendrá la simulación
puntos=7000;
%Número de periodos que tendrá la simulación
Pt=7;
%tstrux -> Tiempo de simulación
%Period -> Periodo de la señal de tren de pulsos rectangular
%Width -> Ancho de la señal de tren de pulsos rectangular
[tstrux,Period,Width]=DefinePeriod(puntos,tm_opt,Pt);
```

Código 3.10 Function DefinePeriod.

```
function [t,P,W]=DefinePeriod(puntos,tmopt,K)

t=puntos*tmopt;
P=round(t/(K*tmopt));
W=round(t/(K*2*tmopt));
```

3.4.1 Respuesta en bucle cerrado del sistema en ausencia de perturbaciones

En este ejemplo se ha considerado que el sistema no se encuentra afectado por ningún ruido ni perturbación. La representación en *Simulink* del modelo de bloques definido a simular se puede observar en la siguiente figura:

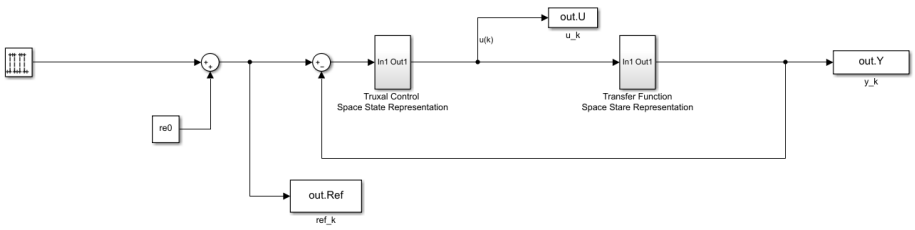

Figura 3.3 Modelo de bloques en Simulink, sistema en lazo cerrado sin perturbaciones. Controlador Truxal-Ragazzini.

Así mismo, la visión detallada de los bloques definidos como *Space State Representation* se encuentra representada en la siguiente figura:

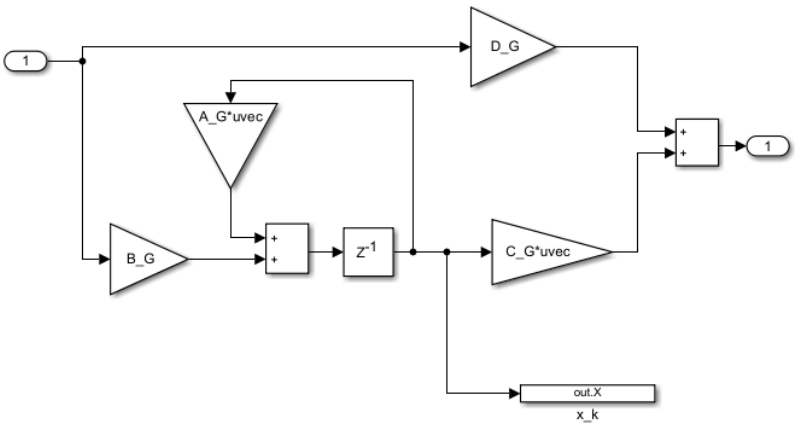

Figura 3.4 Bloque Simulink, Transfer Function Space State Representation.

En cada bloque *State Space Representation* los parámetros *A, B, C y D* se han calculado con la función definida en el *Código 3.8 Function StateSpace Transformation* y aparecen en las ecuaciones *(3.19), (3.20), (3.21) y (3.22)*. Una vez simulado el sistema bajo las premisas definidas en el apartado *Capítulo 3.4 Aplicación del controlador diseñado según el método Truxal-Ragazzini a un sistema conocido. Matlab y Simulink*, estos son los resultados obtenidos:

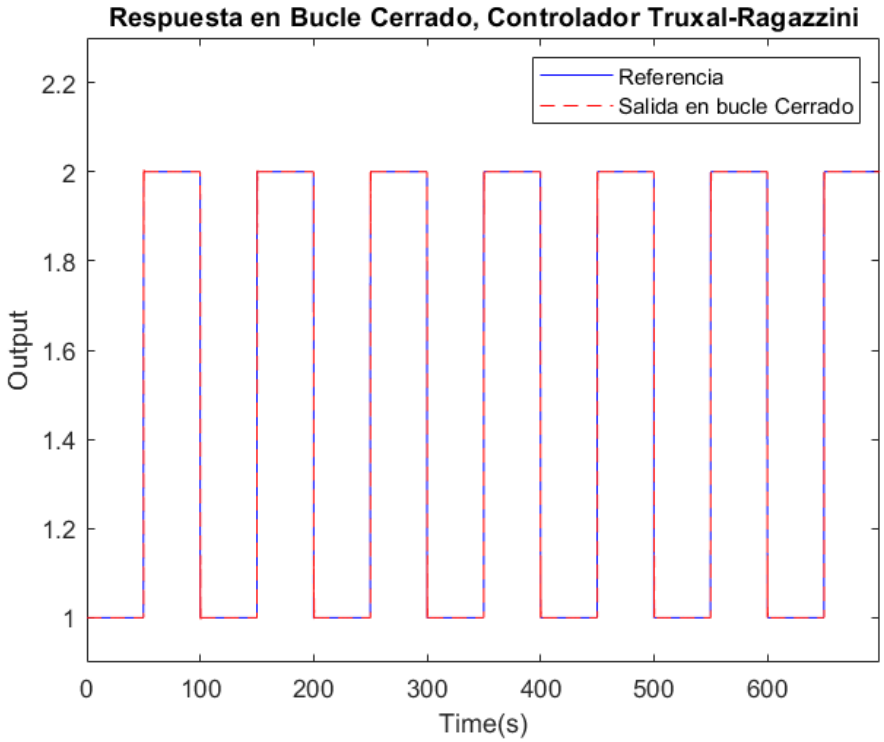

Figura 3.5 Respuesta del sistema en bucle cerrado en ausencia de perturbaciones. Controlador Truxal-Ragazzini.

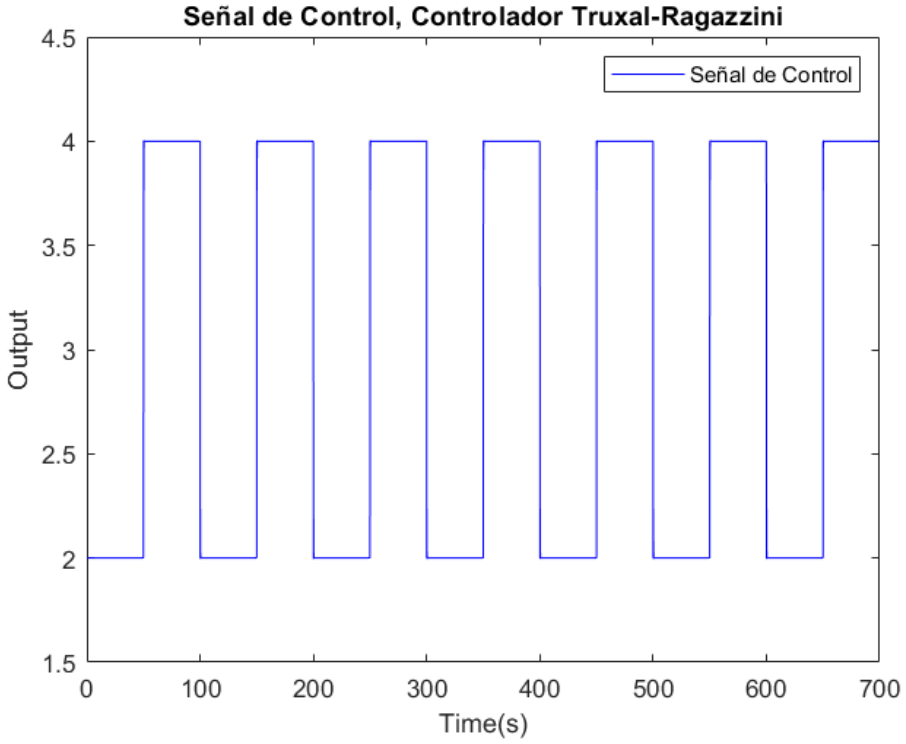

Figura 3.6 Señal de control del sistema en bucle cerrado en ausencia de perturbaciones. Controlador Truxal-Ragazzini.

Como se puede observar en la *Figura 3.5*, la salida muestreada no presenta error en régimen permanente, cumpliéndose así el objetivo final de diseño del controlador.

3.4.2 Respuesta en bucle cerrado del controlador diseñado en presencia de ruido de media no nula

En este segundo caso, se ha añadido un ruido de media no nula que simule una perturbación del sistema. El diagrama de bloques del modelo en *Simulink* es el mismo que el que se puede observar en la *Figura 3.3*, salvo por que el bloque *Transfer Function State Space Representation* ha sido sustituido por *Transfer Function State Space Representation With Noise*.

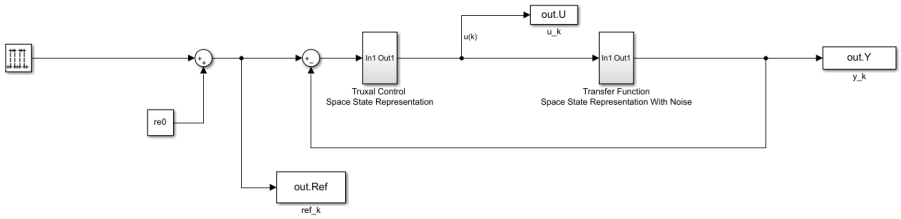

Figura 3.7 Modelo de bloque en Simulink, sistema en lazo cerrado con perturbaciones. Controlador Truxal-Ragazzini.

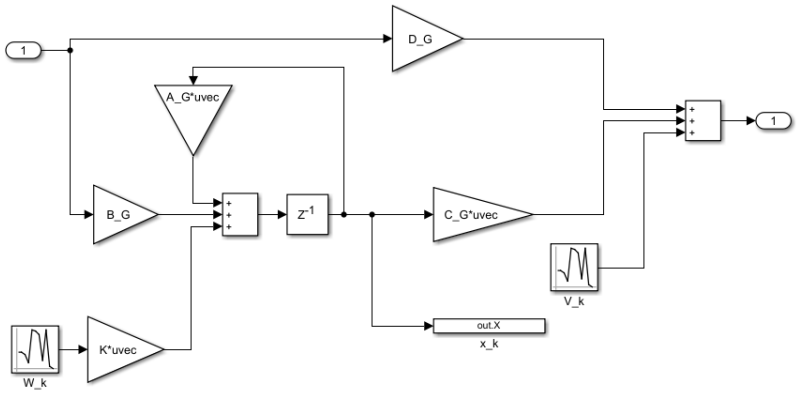

Figura 3.8 Bloque Simulink, Transfer Function Space State Representation With Noise.

Las ecuaciones que rigen este nuevo bloque que representa la forma en espacio estados de la función de transferencia *G(z)* son:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{3.23}$$

$$y_k = Cx_k + Du_k + v_k \tag{3.24}$$

Siendo w_k y v_k las señales que representan el ruido muestreado que afecta al sistema. Los parámetros *A, B, C y D* que aparecen en esta ecuación son los mismo que se han utilizado en el ejemplo anterior y que vienen definidos por las ecuaciones *(3.19) y (3.20)*. Respecto al ruido, éste va a presentar media no nula, estas medias se van a definir como $\mathbb{E}(w_k) = We$ y $\mathbb{E}(v_k) = Ve$. Para simular estos parámetros se ha utilizado el bloque *"Random Number"*.

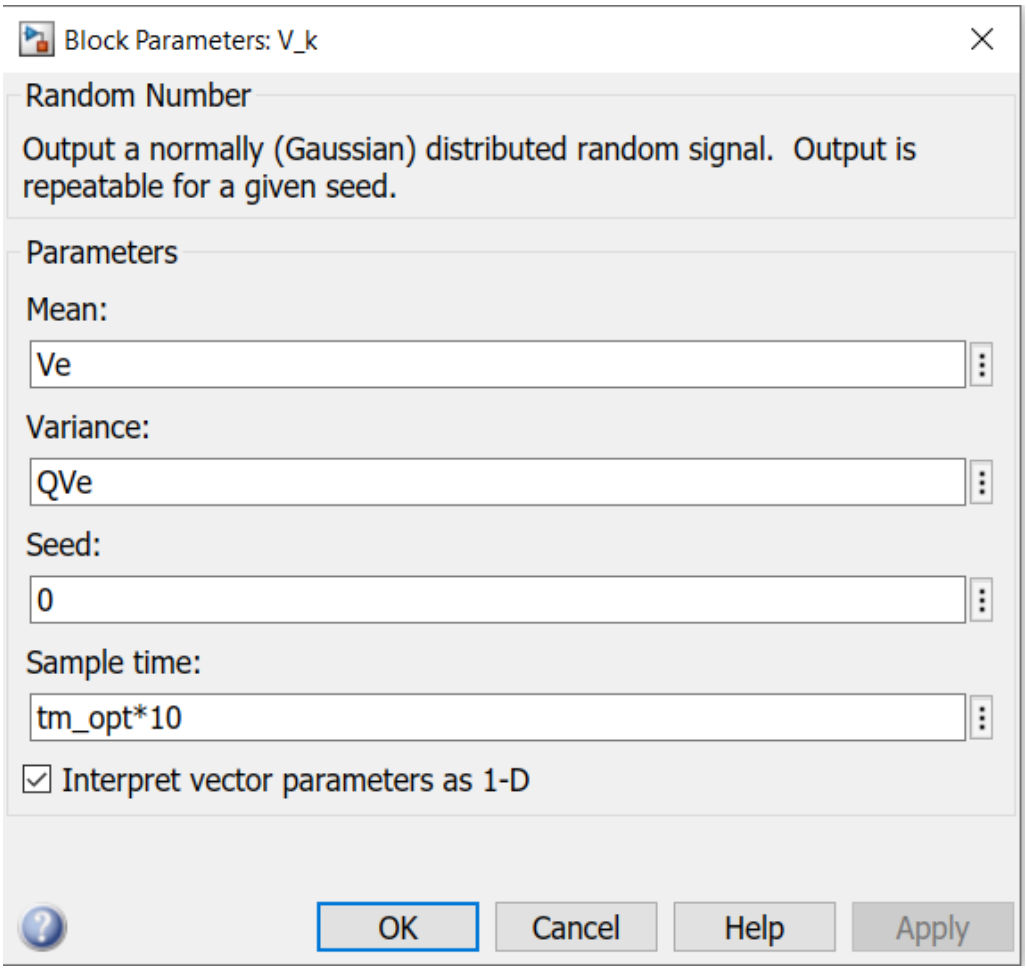

Figura 3.9 Bloque Random Number. Parámetro V(k).

Los valores que representan las medias V_e y W_e y varianzas QV_e y QW_e de sendos ruidos V_k y W_k utilizados como parámetros de los bloques *Random Number* se han definido en la siguiente sección de código.

Código 3.11 Variables referentes a los ruidos V(k) y W(k).

```
%Mínimo valor que puede tomar la media del ruido
tolr1=-0.001;
%Máximo valor que puede tomar la media del ruido
tolr2=0.001;
%Media del ruido V
Ve=tolr1+(tolr2-tolr1)*rand;
%Media del ruido W
We=tolr1+(tolr2-tolr1)*rand;
%Covarianza del ruido V
QVe=0.0001;
%Covarianza del ruido W
QWe=0.0001;
```

Finalmente los resultados de esta simulación se pueden observar en la siguiente figura.

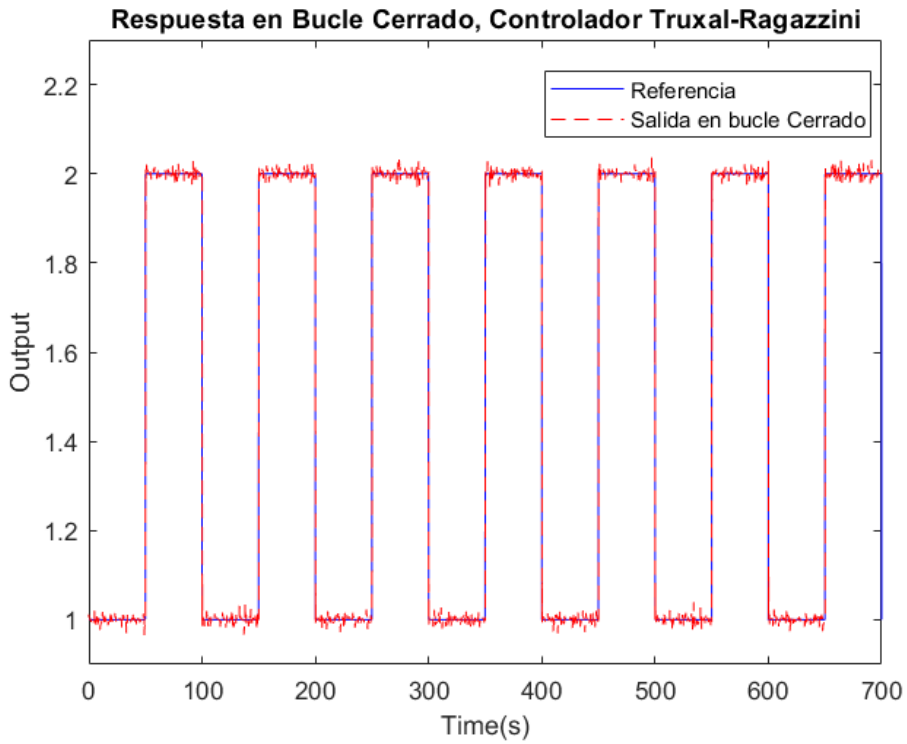

Figura 3.10 Respuesta del sistema en bucle cerrado con perturbaciones. Controlador Truxal-Ragazzini.

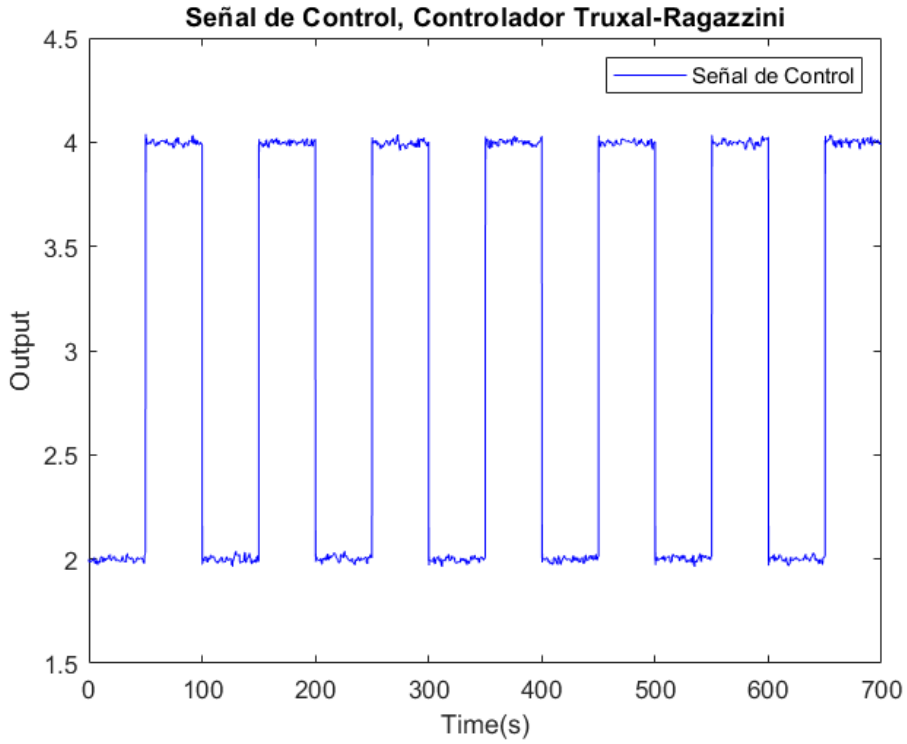

Figura 3.11 Señal de control del sistema en bucle cerrado con perturbaciones. Controlador Truxal-Ragazzini.

4 Creación de la Base de Datos

Como se viene diciendo a lo largo de este proyecto, el objetivo final del mismo es la implementación de un controlador en línea capaz de obtener error en régimen permanente nulo ante la presencia de perturbaciones de media no nula. Uno de los requisitos principales para la correcta implementación de este controlador es la creación de una base de datos. En este capítulo se va a dar forma a dicha base de datos así como explicar su implementación final.

4.1 Base de datos. Definición.

Si se ha definido el sistema a controlar como:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{4.1}$$
$$y_k = Cx_k + Du_k + v_k \tag{4.2}$$

La base de datos recogerá valores pasados sin error en régimen permanente del vector de estados x_i, señal de control u_i, salida y_i y referencia r_i del sistema. El valor *"i"* está definido como *"i=1,...,T"* y representa el numero de trayectorias que estarán contenidas en la base de datos.

4.2 Base de datos. Creación.

En primer lugar, para la creación de la base de datos, se ha de partir de un modelo de la planta a controlar. Para ello en este proyecto se ha aplicado el método de identificación mediante mínimos cuadrados recursivos ponderados (capítulo tal). Este método se explicará en un capítulo posterior y por tanto vamos a partir de la idea de que conocemos el modelo del sistema a controlar definido por las ecuaciones (*5.1*) y (*5.2*).

Conocido el modelo de la planta a controlar se aplica la algoritmia definida en el *Capítulo 2. Implementación de un controlador discreto según el método de diseño Truxal-Ragazzini*, para definir el controlador con el que se generará la base de datos. Como se ha comentado brevemente en el apartado previo, la base de datos no puede presentar error en régimen permanente, dado que el controlador final heredará el mejor o peor comportamiento de las trayectorias almacenadas en la misma, de ahí la elección del método de diseño Truxal Ragazzini. Conocidos planta y controlador, el siguiente paso consiste en la simulación del sistema. Se van a generar un número T de trayectorias, y para cada instante t de cada trayectoria se van a almacenar en la base de datos los valores del vector de estados x_i, señal de control u_i, salida y_i y referencia r_i del sistema. Con esto quedaría implementada la base de datos inicial del sistema.

Una vez definida esta base de datos inicial se va a definir una segunda base de datos a partir de la que se obtendrá el controlador final. Hemos definido como $[x_i,u_i,y_i,r_i]$ al conjunto de trayectorias *"i"* almacenadas en la base de datos para un intervalo de tiempo $t \in [0,p_i]$, siendo p_i el tiempo final de simulación de cada trayectoria i. Pues ahora consideraremos a un conjunto de subtrayectorias $[x_j,u_j,y_j,r_j]$. Cada subtrayectoria j recogerá un conjunto de valores de cada trayectoria i (pudiendo definirse mas de una subtrayectoria j a partir de cada trayectoria i). Esto se debe a que los controladores diseñados según el método Truxal-Ragazzini presentan error en régimen permanente nulo a partir de un valor de N periodos de muestreo. Por tanto, se ha de procurar que la subtrayectorias que formen la base de datos S_j no se correspondan con los primeros periodos de muestreo y solamente recoger aquellos valores que no presenten error en régimen permanente.

Esta nueva base de datos al estar definida a partir de la base de datos original también cumplirá las ecuaciones (*5.1*) y (*5.2*) que ahora se definen como:

$$x_j(k+1) = Ax_j(k) + Bu_k + w_j(k) \tag{4.3}$$

$$y_j(k) = Cx_j(k) + Du_k + v_j(k) \tag{4.4}$$

Cabe comentar que cuando hablamos de la variable t nos referimos al valor actual del tiempo de muestreo de una trayectoria tanto de la base de datos como de la implementación final. Y en cuanto al subíndice k, éste se refiere al desplazamiento relativo de un elemento del conjunto S respecto al tiempo t. Por ejemplo $x_i(t)$ se corresponde con el valor actual del vector de estados de la trayectoria i de la base de datos que en términos de la variable k se definiería como $x_i(0)$.

Este subconjunto de trayectorias deberá de cumplir además otra característica para asegurar la correcta implementación del posterior controlador. La ley de control a aplicar que se definió brevemente en el *Capítulo 1. Introducción*, requiere que para formular la seña de control *u(t)* en un instante t se necesitará el siguiente subconjunto de valores $[x_i(0), x_i(-1), u_i(-1), y_i(0), r_i]$. Por tanto, como se puede observar del subconjunto previo para definir la señal de control en un instante t se necesitan tanto el valor actual como el desplazado un tiempo de muestreo -k de cada trayectoria. Se puede concluir a partir de esto que el subconjunto de trayectorias S_j ha de estar definido como mínimo en el intervalo $t \in [1, p_{ij}]$. Esto se debe a que el valor *x(-1)* solo está definido a partir del instante *t=1*.

4.3 Implementación de la base de datos en Matlab.

En primer lugar, como la teoría sobre identificación de sistemas se explica más adelante, se parte de la idea, como se comentó en la sección previa, que conocemos el modelo de la planta del sistema. Para este ejemplo de implementación se ha definido una función de transferencia cualquiera de tercer orden.

$$G(z) = \frac{z - 0.35}{z^3 - 1.2z^2 + 0.17z + 0.09} \tag{4.5}$$

Que en código, queda implementado con las siguientes líneas.

Código 4.1 Definición de la función de transferencia.

```
%Definición de una Función de Transferencia
poles_G=0.35;
zeros_G=[0.5 -0.2 0.9];
Num_G=poly(poles_G);
Den_G=poly(zeros_G);
tm_opt=0.1;
G_z=tf(Num_G,Den_G,tm_opt);
```

Posteriormente, se aplican las funciones "*ControlTruxal*" y "*StateSpace Transformation*" definidas en el *Capítulo 3* en la secciones de código (*3.1*) y (*3.8*) respectivamente. Con este queda definido el controlador que se va a utilizar para generar la base de datos, así como la representación en espacio estados, tanto de la función de transferencia de la planta como del controlador.

Código 4.2 Cálculo de la ley de control para la generación de la base de datos.

```
%Definición de una Función de Transferencia
poles_G=0.35;
zeros_G=[0.5 -0.2 0.9];
Num_G=poly(poles_G);
Den_G=poly(zeros_G);
tm_opt=0.1;
G_z=tf(Num_G,Den_G,tm_opt);
```

Tras compilar la sección de código previa estos son los resultados obtenidos: Función de transferencia del controlador:

$$C(z) = \frac{z^4 - 1.832z^3 + 0.9287z^2 - 0.01748z + 0.0569}{z^4 - 0.8365z^3 - 1.106z^2 + 1.21z - 0.267} \tag{4.6}$$

Representación en espacio estados de la función de transferencia de la planta:

$$x_{k+1} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -0.09 & -0.17 & 1.2 \end{pmatrix} x_k + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u_k \tag{4.7}$$

$$y_k = \begin{pmatrix} -0.35 & 1 & 0 \end{pmatrix} x_k \tag{4.8}$$

Representación en espacio estados de la función de transferencia del controlador:

$$x_{k+1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0.2670 & -1.2097 & 1.1062 & 0.8365 \end{pmatrix} x_k + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} u_k \tag{4.9}$$

$$y_k = \begin{pmatrix} 0.2101 & -1.2272 & 2.0349 & -0.9957 \end{pmatrix} x_k + u_k \tag{4.10}$$

Una vez definido lo que es el esquema del modelo a simular, el posterior paso será definir los parámetros de simulación. Para la creación de la base da datos se va a definir una variable denominada "*NumTrayects*" que será igual al numero de trayectorias que se van a incluir en la base de datos. Cada trayectoria va a estar definida desde una referencia de valor "*A*" a una referencia de valor "*B*". Para este ejemplo se va a cumplir que las referencia están contenidas en el intervalo *(0,2)* y además $A \leq B$. El modelo va a estar afectado por un ruido de media no nula. Este ruido se va a modelar con la función "*Random Number*" que ya se utilizó en casos previos. Al igual que con las referecias, se van a definir dos límites para la esperanza del ruido. Cada simulación estará afectada por un valor de continua de ruido definido entre estos límites. El valor de la covarianza del ruido si será constante para todas las simulaciones. Por último solo queda definir una variable tipo estructura, en nuestro caso denominada "S_j" que hará la función de base de datos, y donde se almacenaran todas las variables referentes a cada trayectoria simulada.

Código 4.3 Parámetros de simulación. Base da datos.

```
%      Valores que se utilizarán para la simulación del modelo    %
%Tiempo de simulación
ts=120;
%Mínimo valor que puede tomar la media del ruido
tolL_Ref=0;
%Mínimo valor que puede tomar la media del ruido
tolH_Ref=2;
%Mínimo valor que puede tomar la media del ruido
tolL_Noise=-0.001;
%Máximo valor que puede tomar la media del ruido
tolH_Noise=0.001;
%Covarianza del ruido V
QVe=0.0001;
%Covarianza del ruido W
QWe=0.0001;

%Definición variables vase de datos
NumTrajects=1500;
Tstruct=zeros(NumTrajects,1);
Sj=struct('X',Tstruct,'U',Tstruct,'Y',Tstruct,'R0',Tstruct,'R1',Tstruct,'Ref',
    Tstruct);
```

Finalmente, con todo definido, el paso final es la simulación del modelo para su posterior almacenamiento en la base de datos. Como se comentó en la sección *5.1 Base de datos. Definición*, para cada instante de

tiempo t de cada trayectoria i se va a almacenar en la base de datos la siguiente matriz:

$$\begin{bmatrix} x_i(t) \\ x_i(t-1) \\ u_i(t-1) \\ y_i(t) \\ r_i(t) \end{bmatrix} \tag{4.11}$$

Donde:

- $x_i(t)$: Valor del vector de estados de la función de transferencia *G(z)*, de la trayectoria "i" en el instante "t".
- $x_i(t-1)$:Valor del vector de estados de la función de transferencia *G(z)*, de la trayectoria "i" en el instante "*t-1*".
- $u_i(t-1)$:Valor de la señal de control de la función de transferencia *G(z)*, de la trayectoria "i" en el instante "t".
- $y_i(t)$: Valor de la salida muestreda de la función de transferencia *G(z)*, de la trayectoria "i" en el instante "t".
- $r_i(t)$: Valor de referencia de la trayectoria "i" en el instante "t".

Por último solo queda definir la función *Aleatory References* que se utilizará posteriormente para la creación de la base de datos. Para cada trayectoria "i" definida en la base de datos, la función *Aleatory References* genera de forma aleatoria los dos valores de referencia que seguirá el sistema así como las esperanzas del ruido que afecta al sistema.

Código 4.4 Function Aleatory References.

```
function [re0,re1,We,Ve]=Aleatory_References(A1,A2,B1,B2)

re0=A1+(A2/2-A1)*rand;
re1=A2/2+(A2-A2/2)*rand;
We=B1+(B2-B1)*rand;
Ve=B1+(B2-B1)*rand;
```

Código 4.5 Implementación final Base de datos.

```
for i=1:NumTrajects

    %Titulo
    [re0,re1,We,Ve]=Aleatory_References(tolL_Ref,tolH_Ref,tolL_Noise,tolH_Noise
        );
    [Ref]=DataBase_References(re0,re1,ts,tm_opt,2);


    %Cálculos de las características estáticas del sistema
    [X0_G,U0_G]=SteadyState_Characterization(A_G,B_G,C_G,D_G,re0,We,Ve);
    [X0_Cz,~]=SteadyState_Characterization(A_Cz,B_Cz,C_Cz,D_Cz,U0_G,0,0);

    Sim_Data=sim('TruxalModel_WNoise_Simulation_Step');

    [Sj(i)]=DataBase_Generation(Sim_Data.X,Sim_Data.U,Sim_Data.Y,re0,re1,Ref);
end
```

Para almacenar los valores definidos en la ecuación (*5.11*) se ha definido la función *DataBase Generation*.

Código 4.6 Implementación final Base de datos.

```
function [Sj]=DataBase_Generation(Xt,Ut,Yt,re0,re1,Ref)

    Sj.X=Xt';
    Sj.U=Ut;
    Sj.Y=Yt;
    Sj.R0=re0;
    Sj.R1=re1;
    Sj.Ref=Ref;
```

El modelo de bloques de *Simulink* utilizado para las simulaciones del sistema es el mismo que se puede observar en las figuras (*3.7*) y (*3.8*). Para finalizar se adjuntan dos ejemplos de trayectorias (escogidas aleatoriamente) de la base de datos.

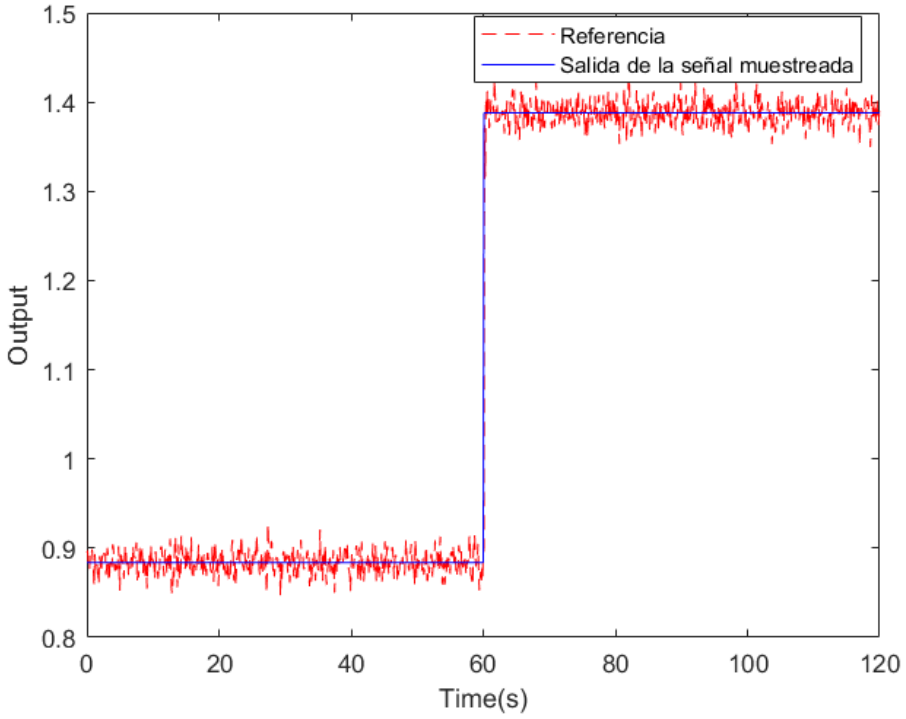

Figura 4.1 Referencia vs Salida muestreada, trayectoria 982.

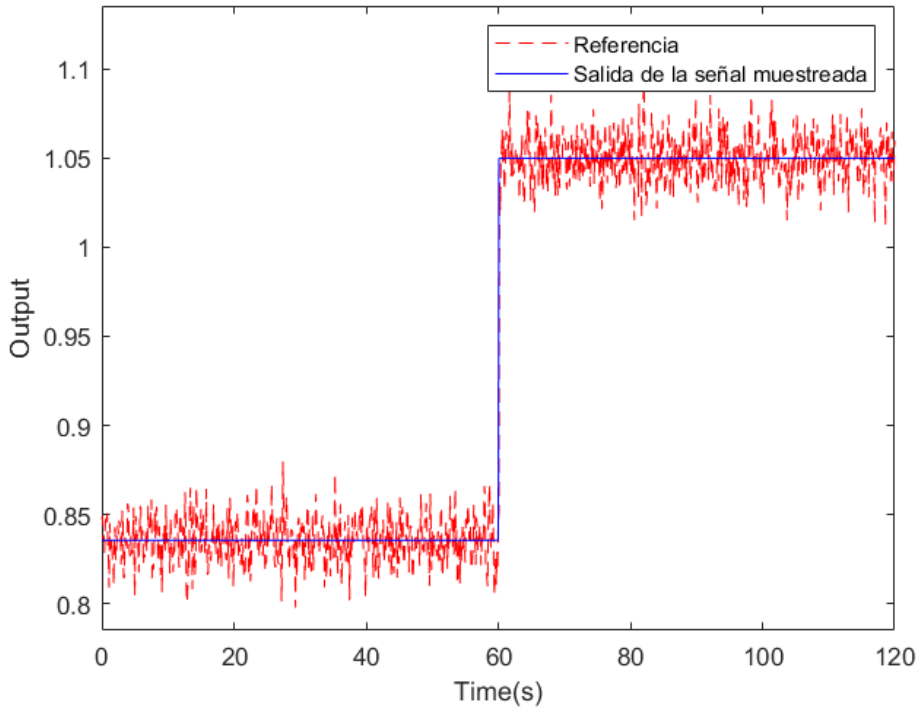

Figura 4.2 Referencia vs Salida muestreada, trayectoria 650.

Finalmente, para terminar esta sección se añade una captura de la forma que tiene la variable estructura "S_j" en Matlab. Por condiciones de espacio solo se muestran las primeras 29 filas que tiene la estructura.

1x1500 struct with 6 fields

Fields	X	U	Y	R0	R1	Ref
1	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.9134	1.6324	1x1201 dou...
2	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.5469	1.9575	1x1201 dou...
3	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.9706	1.9572	1x1201 dou...
4	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.1419	1.4218	1x1201 dou...
5	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.9595	1.6557	1x1201 dou...
6	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.9340	1.6787	1x1201 dou...
7	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.3922	1.6555	1x1201 dou...
8	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.0318	1.2769	1x1201 dou...
9	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.8235	1.6948	1x1201 dou...
10	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.0344	1.4387	1x1201 dou...
11	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.7952	1.1869	1x1201 dou...
12	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.6463	1.7094	1x1201 dou...
13	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.6797	1.6551	1x1201 dou...
14	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.4984	1.9597	1x1201 dou...
15	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.2238	1.7513	1x1201 dou...
16	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.6991	1.8909	1x1201 dou...
17	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.1386	1.1493	1x1201 dou...
18	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.2543	1.8143	1x1201 dou...
19	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.3500	1.1966	1x1201 dou...
20	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.4733	1.3517	1x1201 dou...
21	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.5497	1.9172	1x1201 dou...
22	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.7537	1.3804	1x1201 dou...
23	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.0540	1.5308	1x1201 dou...
24	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.1299	1.5688	1x1201 dou...
25	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.3371	1.1622	1x1201 dou...
26	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.5285	1.1656	1x1201 dou...
27	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.6541	1.6892	1x1201 dou...
28	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.0838	1.2290	1x1201 dou...
29	3x1201 dou...	1201x1 dou...	1201x1 dou...	0.8258	1.5383	1x1201 dou...

Figura 4.3 Estructura S_j que representa a la base de datos.

5 Controlador Data Driven

En este capítulo se desarrolla la algoritmia para la implementación del controlador *Data Driven Offset Free*. Este controlador fue desarrollado por los profesores *J. R. Salvador, D.R. Ramirez, T. Alamo y D. Muñoz de la Peña* pertenecientes al departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla [1][7]. Por tanto las ideas y ecuaciones que se hacen en este capítulo son una completa referencia al trabajo realizado por los profesores citados con anterioridad.

5.1 Planteamiento del problema

La idea, repetida varias veces a lo largo de este trabajo de fin de grado, es desarrollar un controlador en línea capaz de obtener una respuesta sin error en régimen permanente a pesar de que el sistema presente un ruido de media no nula. Hemos definido el sistema a controlar como:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + w(t) \\ y(t) &= Cx(t) + Du(t) + v(t) \end{aligned} \tag{5.1}$$

Siendo $x(t) \in \mathbb{R}_x^n$ y $u(t) \in \mathbb{R}_u^n$ respectivamente los valores del vector de estados y señal de control del sistema para un instante de tiempo t, $y(t) \in \mathbb{R}_y^n$ la salida del sistema, $w(t) \in \mathbb{R}_x^n$ y $v(t) \in \mathbb{R}_y^n$ los valores de las perturbaciones que afectan al sistema en cada instante t y finalmente las matrices A, B, C, D que definen al sistema.

5.2 Definición de la ley de Control. Base de datos

El objetivo pues será alcanzar una referencia de valor "r" sin error en régimen permanente. Se ha de comentar que tal como se ha definido el ruido en el apartado anterior, alcanzar un controlador sin error en régimen permanente es matemáticamente imposible. El resultado final esperado es obtener un controlador capaz de forzar que la salida "y" esté limitada entre una serie de valores cuya media sea igual a "r". El primer paso para la obtención de este controlador "*offset free*" es la generación de una base datos. La definición e implementación de la base de datos se ha realizado en el capítulo previo *4. Creación de la base de datos*. La necesidad de esta base de datos viene dada por la propia definición de la ley de control. La ley de control que se va a aplicar en cada instante "t" se define como:

$$u(t) = \sum_{j \in S} \lambda_j u_j(0) \tag{5.2}$$

Es decir, la señal de control es una suma ponderada de las señales de control de cada trayectoria "j" almacenadas en la base datos. De ahí la necesidad de este histórico de datos. En la próximas secciones se definiran las condiciones que ha de cumplir "λ_j" para conseguir el control sin error en régimen permanente.

5.3 Definición controlador Data Driven

En este apartado se van a ir imponiendo condiciones a "λ_j" con el fin de conseguir el tan deseado control sin error en régimen permanente. En primer lugar, queremos que nuestra variable "λ" este contenida en el intervalo (0,1), por tanto:

$$1 = \sum_{j \in S} \lambda_j \tag{5.3}$$

Así mismo, ha de existir una dependencia entre el vector de estados de las trayectorias almacenadas en la base de datos y el vector de estados final. Para ello se impone la siguiente condición:

$$x(t) = \sum_{j \in S} \lambda_j x_j(0) \tag{5.4}$$

Unificando las ecuaciones (*5.1*), (*5.2*) y (*5.4*) obtenemos:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + w(t) \\ &= A \sum_{j \in S} \lambda_j x_j(0) + B \sum_{j \in S} \lambda_j u_j(0) + w(t) \\ &= \sum_{j \in S} \lambda_j (Ax_j(0) + Bu_j(0)) + w(t) \\ &= \sum_{j \in S} \lambda_j A x_j(1) + \sum_{j \in S} \lambda_j (w(t) - w_j(0)) \end{aligned} \tag{5.5}$$

Esto implica que:

$$x(t+1) \approx \sum_{j \in S} \lambda_j A x_j(1) \tag{5.6}$$

Donde el error de la ecuación anterior viene definido por el término:

$$\sum_{j \in S} \lambda_j (w(t) - w_j(0)) \tag{5.7}$$

Tal como ha sido definido el término error por la ecuación (*5.7*) no se puede presuponer que su esperanza sea nula. Por tanto hay que imponer más condiciones a la variable "λ_j" con este objetivo. En primer lugar, queremos que el control presente memoria. Al hablar del término "*memoria*", nos referimos a que exista una relación entre las variables actuales y pasadas del sistema. Esto se consigue imponiendo una dependencia de la variable de control "λ_j" tanto con el vector de estados, como con la señal de control un instante previo al instante de control, es decir:

$$x(t-1) = \sum_{j \in S} \lambda_j x_j(-1) \tag{5.8}$$

$$u(t-1) = \sum_{j \in S} \lambda_j u_j(-1) \tag{5.9}$$

Si nos referimos al último párrafo de la sección *4.2 Base de datos. Creación*. Se comprueba que los valores $x_j(-1)$ y $u_j(-1)$ están bien definidos dados que las trayectorias de la base de datos se definieron para el intervalo $t \in [1, p_{ij}]$. Finalmente, se va imponer una relación tanto de la salida como de la referencia finales de control con aquellas almacenadas en la base de datos.

$$y(t) = \sum_{j \in S} \lambda_j y_j(0) \tag{5.10}$$

$$r = \sum_{j \in S} \lambda_j r_j \tag{5.11}$$

Con esto quedarían definidas las restricciones que ha de cumplir λ_j para conseguir el control sin error en régimen permanente. En el siguiente apartado se demuestra que aplicando dichas restricciones se consigue el control "*offset free*".

5.4 Demostración Data Driven Offset Free

En este apartado se trata de demostrar de como aplicando la ley de control definida en la ecuación (*5.2*) sujeta a las restricciones impuestas en las ecuaciones (*5.3*), (*5.4*), (*5.8*), (*5.9*), (*5.10*) y (*5.11*) es posible obtener una ley de control sin error en régimen permanente. Partiendo de la ecuación (*5.1*) pero considerando el instante de tiempo *t* en vez de *t-1* y aplicando las ecuaciones (*5.4*), (*5.8*) y (*5.9*) obtenemos:

$$\begin{aligned} x(t) &= Ax(t-1) + Bu(t-1) + w(t-1) \\ w(t-1) &= x(t) - Ax(t-1) - Bu(t-1) \\ &= \sum_{j\in S} \lambda_j x_j(0) - A\sum_{j\in S} \lambda_j x_j(-1) - B\sum_{j\in S} \lambda_j u_j(-1) \\ &= \sum_{j\in S} \lambda_j (x_j(0) - Ax_j(-1) - Bu_j(-1)) \\ &= \sum_{j\in S} \lambda_j w_j(-1) \end{aligned} \tag{5.12}$$

Volviendo a aplicar las restricciones impuestas por las ecuaciones (*5.4*), (*5.8*) y (*5.9*) a la ecuación (*5.1*), pero en este caso si considerando el instante de tiempo *t+1*:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + w(t) \\ &= \sum_{j\in S} \lambda_j A x_j(1) + \sum_{j\in S} \lambda_j (w(t) - w_j(0)) \\ &= \sum_{j\in S} \lambda_j A x_j(1) + w(t) - \sum_{j\in S} \lambda_j w_j(0) \end{aligned} \tag{5.13}$$

Si definimos ahora el error de predicción $e_x(t)$ como:

$$e_x(t) = x(t+1) - \sum_{j\in S} \lambda_j x_j(1) \tag{5.14}$$

Y lo sustituimos en la ecuación (*5.13*)

$$e_x(t) = w(t) - \sum_{j\in S} \lambda_j w_j(0) \tag{5.15}$$

Aplicando esta última igualdad a la ecuación (*5.12*) antes calculada obtendríamos:

$$-w(t-1) + \sum_{j\in S} \lambda_j w_j(-1) = 0 \tag{5.16}$$

Finalmente operamos junto a la ecuación (*5.14*)

$$\begin{aligned} e_x(t) &= w(t) - w(t-1) - \sum_{j\in S} \lambda_j (w_j(0) - w_j(-1)) \\ &= \triangle w(t) - \triangle w(t-1) - \sum_{j\in S} \lambda_j (\triangle w_j(0) - \triangle w_j(-1)) \end{aligned} \tag{5.17}$$

Donde, $\triangle w(\cdot) = w(\cdot) - w^e$ y $\triangle w_j(\cdot) = w_j(\cdot) - w_j^e$. Dado que por construcción las variables $\triangle w_j(\cdot)$ y $\triangle v_j(\cdot)$ tienen media igual a cero, queda demostrado que el error de estimación es un término de error con media igual a cero. Si reconstruimos la igualdad (*5.14*):

$$x(t+1) = \sum_{j\in S} \lambda_j x_j(1) + e_x(t) \tag{5.18}$$

Queda demostrado la obtención del controlador sin error en régimen permanente, pues el cálculo del siguiente estado es función del estado anterior más un término error de media igual a cero.

5.5 Implementación del controlador Data Driven Offset Free

Es de fácil asunción que el error del sistema es la diferencia entre la referencia a alcanzar y la salida.

$$e(t) = r - y(t) \tag{5.19}$$

Esto es extensible a cada subtrayectoria "j" de la base de datos "S". Por tanto:

$$e_j(t) = r_j - y_j(t) \tag{5.20}$$

La idea es reducir la varianza del error de muestreo con cada iteración realizada. Por tanto el término a minimizar se podría definir como:

$$\sum_{j \in S} \lambda_j^2 E \| e_j(0) \|^2 \tag{5.21}$$

El término $E\| e_j(0) \|^2$ es, a priori, difícil de caracterizar. Pero si asumimos que tiene un límite superior "σ" tal que:

$$E\| e_j(0) \|^2 \leq \sigma, \forall j \in S \tag{5.22}$$

El problema de optimización a resolver se reduce a:

$$\min_{\lambda_j} \sum_{j \in S} \sigma^2 \lambda_j^2 \tag{5.23}$$

Ahora si consideramos que todos los valores "σ_j" son iguales a un determinado valor "σ" y aplicamos todas las restricciones que se definieron en la sección *5.3 Demostración controlador Data Driven*, en específico, las ecuaciones (*5.3*), (*5.4*), (*5.8*), (*5.9*), (*5.10*) y (*5.11*) el problema de optimización es:

$$\begin{aligned}
\lambda_j^* = arg \min_{\lambda_j} & \sum_{j \in S} \lambda_j^2 \\
s.t. x(t) &= \sum_{j \in S} \lambda_j x_j(0) \\
x(t-1) &= \sum_{j \in S} \lambda_j x_j(-1) \\
u(t-1) &= \sum_{j \in S} \lambda_j u_j(-1) \\
y(t) &= \sum_{j \in S} \lambda_j y_j(0) \\
r &= \sum_{j \in S} \lambda_j r_j \\
1 &= \sum_{j \in S} \lambda_j
\end{aligned} \tag{5.24}$$

Una vez obtenido la variable "λ_j", la ley de control a aplicar viene definida por la ecuación (*5.2*).

Este problema puede ser presentado como un problema de optimización cuadrática sujeto a una serie de restricciones de igualdad.

$$\begin{aligned}
\lambda^*(t) = arg \min_{\lambda} & \lambda^T \lambda \\
s.t. H\lambda &= b,
\end{aligned} \tag{5.25}$$

Donde "λ" es un vector que recoge los valores de cada variable "λ_j". Y "H" y "b" son respectivamente una matriz y un vector de dimensiones $H \in \Re^{n_h x n_T}$ y $b \in \Re^{n_h}$. La variables "n_h" que representa el número de filas tanto de la matriz "H" como del vector "b" se corresponde también con el número de restricciones de igualdad. Esta variable se define como $n_h = 2 \cdot n_x + n_u + 2 \cdot n_y + 1$, donde respectivamente "n_x" es el tamaño del vector de estados, "n_u" es el tamaño de la señal de control y "n_y" es el tamaño de la salida del sistema. Finalmente señalar que la variable "n_T" representa el número de trayectorias totales de la base de datos. En

una vista más a detalle de las matrices "H" y "b", estás se definen como:

$$H = \begin{bmatrix} x_j(0) & x_{j+1}(0) & ... & x_T(0) \\ x_j(-1) & x_{j+1}(-1) & ... & x_T(0) \\ u_j(-1) & u_{j+1}(-1) & ... & u_T(0) \\ y_j(0) & y_{j+1}(0) & ... & y_T(0) \\ r_j(0) & r_{j+1}(0) & ... & r_T(0) \\ 1 & 1 & ... & 1 \end{bmatrix} \tag{5.26}$$

$$b = \begin{bmatrix} x(t) \\ x(t-1) \\ u(t-1) \\ y(t) \\ r(t) \\ 1 \end{bmatrix} \tag{5.27}$$

Con el objetivo de mejorar el problema de optimización, se asume que en la base de datos el número de trayectorias almacenadas "T" nos asegura de sobra la correcta caracterización de la matriz "H". Pero, para resolver el problema de optimización, debido a las restricciones impuestas para calcular λ_j, queremos que no exista un exceso de discrepancia entre las variables de control reales y las almacenadas en la base de datos. Por ello, se va a utilizar un criterio de selección, definido a posteriori, con el que se va a reducir el número de candidatos de la base de datos a un valor "n_c", quedándonos así, con el subconjunto de candidatos más óptimos. El criterio de selección utilizado es una función "*distancia*" que compara las trayectorias almacenadas en la base de datos, con el valor actual de las variables de control. Además, como previamente se impuso que el control tuviera un tipo de memoria a partir de las ecuaciones (*5.8*) y (*5.9*), también se van a tener en cuenta los valores pasados del vector de estados y la señal de control. Se van a definir pues dos vectores:

$$z(t) = \begin{bmatrix} x(t) \\ x(t-1) \\ u(t-1) \\ y(t) \\ r(t) \end{bmatrix} \tag{5.28}$$

$$z_j = \begin{bmatrix} x_j(0) \\ x_j(-1) \\ u_j(-1) \\ y_j(0) \\ r_j(0) \end{bmatrix} \tag{5.29}$$

El número final de "n_c" candidatos vendrá dado por las trayectorias que ponderen un menor valor en la función distancia definida como:

$$d(z(t),z_j) = |x(t) - x_j(0)|_2 + |x(t-1) - x_j(-1)|_2 + |u(t-1) - u_j(-1)|_2 + |y(t) - y_j(0)|_2 + |r(t) - r_j(0)|_2 \tag{5.30}$$

Dicha función distancia se ha de implementar cada tiempo de muestreo para cada trayectoria $j \in S$. Tras aplicar dicha función distancia, el rango de la matriz "H" queda reducida a $H \in \Re^{n_h x n_c}$.

Con todo definido el próximo paso sería la implementación del control. Partiendo de la existencia de una base de datos definida bajo los criterios expuestos en el capítulo *4. Creación de la Base de Datos*. Para la implementación del controlador "*Data Driven Offset Free"* habría que seguir los siguiente pasos, que se repetirán para cada tiempo de muestreo "tm" que se realice la simulación:

1. Se construye el vector *z(t)*.
2. Para candidato $j \in S$ se aplica la función distancia definida en la ecuación (*5.23*)
3. Tras aplicar la función distancia, Se construye el subconjunto $\hat{S}$ con los "n_c" mejores candidatos.
4. Se resuelve el problema de optimización definido por la ecuación (*5.18*). La solución a este problema de optimización viene dada por:

$$\lambda^*(t) = H^T(HH^T)^{-1}b \tag{5.31}$$

5. Una vez conocido el valor de las variables $\lambda_j^*(t)$ contenidas en el vector $\lambda^*(t)$, solo quedaría aplicar la ley de control $u(t) = \sum_{j \in S} \lambda_j u_j(0)$ y repetir el proceso.

5.6 Implementación del controlador en Matlab

Partimos del código implementado en la sección *4.3 Implementación de la base de datos en Matlab* donde se definió la algoritmia y condiciones para crear la base de datos. Una vez definida la base de datos, el primer paso es inicializar las condiciones del nuevo control. Haciendo uso de la función *Aleatory References* que se definió en el código (*4.4*) se crean de forma aleatoria las condiciones de continua del ruido, así como los set points que serán impuestos a nuestro sistema. Así mismo, se calcula el estado estacionario del sistema con el valor de referencia de partida, para que el modelo inicialice de forma estable. Finalmente, a partir de las matrices A, B y C se calcula el valor de la variable "n_h", que define el numero de restricciones de igualdad que tendrá el problema de optimización. Solo quedaría definir el número de candidatos que forman el subconjunto de trayectorias finales "n_c", que en este caso se han definido en la variable "*NumTrajects Selected*". Con esto quedan definidas las condiciones iniciales de partida para implementar el control.

Código 5.1 Inicialización problema de optimización final.

```
%Variables
Points=length(Sj(1).X(1,:));
NumTrajects_Selected=150;
X_Size=length(Sj(1).X(:,1));
H_Size=2*X_Size+4;

%Valores inciales
[re0,re1,We,Ve]=Aleatory_References(tolL_Ref,tolH_Ref,tolL_Noise,tolH_Noise);
[Ref]=DataBase_References(re0,re1,ts,tm_opt,2);
[X0,U0]=SteadyState_Characterization(A_G,B_G,C_G,D_G,re0,We,Ve);

%Inicializar
[Xx,Yx,Ux]=Iniciar(Points,X0,U0,X_Size,A_G,B_G,C_G);
```

Código 5.2 Function Iniciar.

```
function [Xx,Yx,Ux]=Iniciar(Points,X0,U0,X_Size,A,B,C,We,Ve,QWe,QVe)

Ux=zeros(1,Points);
Ux(1)=U0;
Xx=zeros(X_Size,Points);
Xx(:,1)=X0;
Xx(:,2)=A*X0+B*U0 +We +QWe -2*QWe*rand;
Yx=zeros(1,Points);
Yx(1)=C*X0 +Ve +QVe -2*QVe*rand;
Yx(2)=C*Xx(:,2) +Ve +QVe -2*QVe*rand;
```

Con el objetivo de recoger el valor de todas las variables cada tiempo de muestreo, se ha definido una estructura de nombre "*Solutions Struct*". En dicha estructura se van a recoger para cada tiempo de muestreo, los valores de las matrices "H", "b" y "λ" así como el valor del vector de estados "x", señal de control "u" y salida del sistema "y".

Código 5.3 Estructura de resultados finales.

```
H_Structs_Size=zeros(H_Size,NumTrajects_Selected);
b_Structs_Size=zeros(H_Size,1);
Lambda_Structs_Size=zeros(H_Size,1);
```

```
Solutions_Struct=struct('H',H_Structs_Size,'b',b_Structs_Size,'Lambda',
    Lambda_Structs_Size,'X',X0,'Y',re0,'U',U0,'Ref',re0);

Solutions_Struct(2).X=Xx(:,2);
Solutions_Struct(2).Y=Yx(2);
```

Para la implementación de la algoritmia para resolver el problema de optimización simplemente se han seguido los pasos estipulados al final de la sección *5.5 Implementación del controlador Data Driven Offset Free*. En la siguiente sección se desarrolla el código creado, que posteriormente será desglosado y explicado más en detalle.

Código 5.4 Implementación controlador Data Driven Offset Free.

```
%Funcion distancia
Distance_Flags=Distance_Calculation(Xx(:,cnt-1),Xx(:,cnt),Ux(cnt-1),Yx(cnt),Ref
    (cnt),Sj,NumTrajects,NumTrajects_Selected,cnt);

%Construimos H
H=Matrix_H_Generation(NumTrajects,cnt,Sj,Distance_Flags,H_Size);

%Construimos b
b=Matrix_b_Generation(Xx,Yx,Ux,Ref,cnt,H_Size,X_Size);

%Solución
H_Ajusted=(H*H') + 0.00001*eye(H_Size);
Beta=(H_Ajusted)\b;
Lambda=H'*Beta;

% %Nueva señal de control
Ux(cnt)=Summultli(Sj,Lambda,cnt,Distance_Flags);

%Valor en espacio estados siguiente tiempo de muestreo
Xx(:,cnt+1)=A_G*Xx(:,cnt) + B_G*Ux(:,cnt) + Ve - QWe +2*QWe*rand;
Yx(:,cnt+1)=C_G*Xx(:,cnt+1) + We - QVe +2*QVe*rand;


[Solutions_Struct(cnt)]=RellenaStruct_v2(H,b,Lambda,Xx,Yx,Ux,Ref,cnt);
```

En primer lugar, se ha creado la función "*Distance Calculation*". En esa función se implementa la ecuación (*5.23*). El resulado de aplicar dicha función es un vector denominado "*Distance Flags*", dicho vector es un vector tipo "*bandera*" (solo contiene los valores cero y uno). Si el vector *Distance Flags* tiene el valor "*1*" en su columna "*n*" significa que la trayectoria "*n*" del conjunto S se encuentra dentro del subconjucto $\hat{S}$ de trayectorias finales seleccionadas.

A partir del vector *Distance Flags* se reduce el conjunto de trayectorias finales a un valor predefinido *NumTrayects Selected* y se crean las matrices "*H*" y "*b*".

Código 5.5 Function Matrix H Generation.

```
function H=Matrix_H_Generation(NumTrajects,cnt,S,Kdist,H_Size)

H=zeros(H_Size,1);


for k=1:NumTrajects
   if(Kdist(k)==1)
          A=[S(k).X(:,cnt);S(k).X(:,cnt-1);S(k).U(cnt-1);S(k).Y(cnt);S(k).Ref(
              cnt);1];
```

```
            H=[H A];
    end
end

H(:,1)=[];
```

Código 5.6 Function Matrix b Generation.

```
function B=Matrix_b_Generation(Xx,Yx,Ux,Ref,cnt,H_Size,X_Size)

B=zeros(H_Size,1);

for i=1:X_Size
    B(i)=Xx(i,cnt);
end

for i=1:X_Size
    B(X_Size+i)=Xx(i,cnt-1);
end

Size=2*X_Size+1;

B(Size)=Ux(cnt-1);
B(Size+1)=Yx(cnt);
B(Size+2)=Ref(cnt);

B(Size+3)=1;
```

Una vez creadas las matrices "H" y "b", solo quedaría resolver la ecuación (*5.24*) y aplicar la ley de control. Se ha de comentar que se ha modificado el término $(HH^T)^{-1}$ de la ecuación original, dado que al invertir la matriz HH^T se producían problemas de inestabilidad. Para solucionarlo, se suma al término HH^T la matriz identidad multiplicada por un término muy pequeño, como se puede observar en la siguiente ecuación

$$\hat{H} = (H * H') + 0.00001 * I; \tag{5.32}$$

Tras este ajuste, ya es posible resolver el problema de optimización y obtener el vector $\lambda^*(t)$. Conocido $\lambda^*(t)$ ya es posible resolver la ecuación (*5.2*) y obtener el valor de la señal de control *u(t)*.

Código 5.7 Function Summulti.

```
function U=Summultli(S,LAMBDA,cnt,Kdist)

Include_Structs=length(Kdist);
U=0;
i=1;

for j=1:Include_Structs
    if (Kdist(j)==1)
       U=LAMBDA(i)*S(j).U(cnt)+U;
       i=i+1;
    end
end
```

Quedando completamente implementado la resolución del problema de optimización. Ya solo quedaría, calcular los nuevos valores del vector de estado y salida según las ecuación (*5.1*) y volver a repetir el proceso, tantas veces como puntos se desee que tenga el sistema final. Si partimos de la base de datos que se desarrollo

en el apartado *4.3 Implementación de la base de datos en Matlab* y se aplica el modelo de control desarrollado en la sección previa. Se puede observar en la *figura 5.1* como se obtiene el control final sin error en régimen permanente.

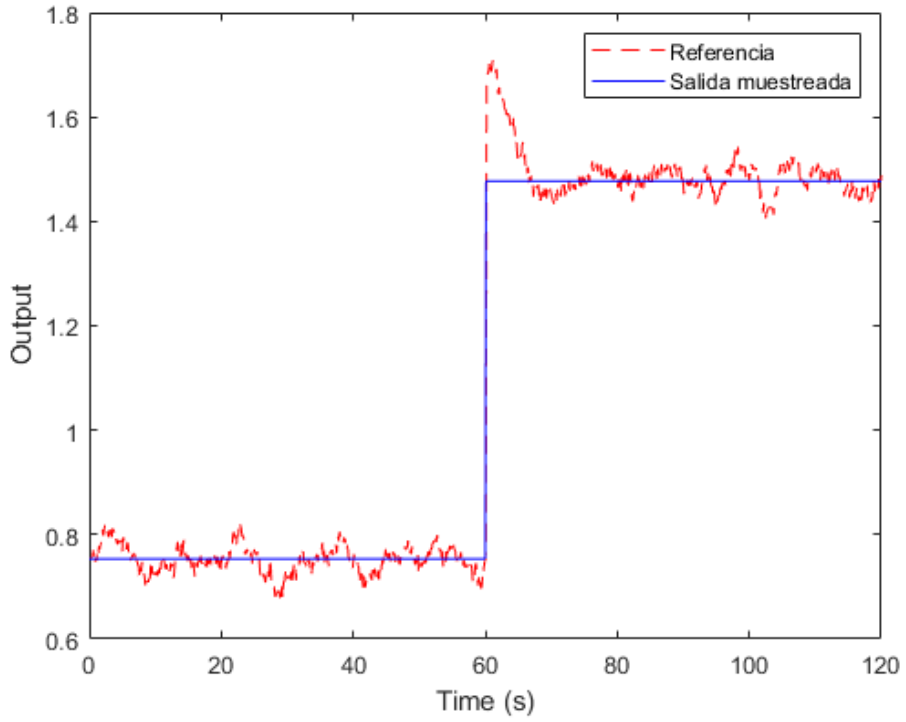

Figura 5.1 Salida del sistema. Controlador Data Driven Offset Free.

5.7 Comparativa Control PI vs Controlador Truxal-Ragazzini

Al implementar la base de datos, uno de los requisitos fundamentales es que las trayectorias almacenadas no presenten error en régimen permanente, dado que el controlador heredará el mejor o peor comportamiento de estas trayectorias. Para generar esta base de datos se podría haber implementado cualquier controlador que presentará un elemento integral que nos permitiera poder anular así el error en régimen permanente. Pero, sin embargo, entre todas las opciones se ha escogido un controlador diseñado según el método Truxal-Ragazzini. ¿Pero, a qué se debe esto?. En los siguientes apartados se tratará de justificar la elección del controlador Truxal-Ragazzini y se comparará su rendimiento frente a un controlador PI.

En la sección previa se ha expuesto el desarrollo y ejemplo de la aplicación del controlador *Data Driven* a una función de transferencia de tercer orden, en específico está se definió en la sección *4.3 Implementación de la base de datos en Matlab* y tenía la siguiente forma:

$$G(z) = \frac{z - 0.35}{z^3 - 1.2z^2 + 0.17z + 0.09} \tag{5.33}$$

Éste, sin embargo, es un ejemplo bastante teórico que solo nos sirve para demostrar la correcta resolución del problema de optimización. Como se comentó en el *Capítulo 1 Introducción* durante el *Capítulo 7 Implementación del controlador Data Driven en un sistema real*, se irá desarrollando la aplicación a un modelo real no lineal . Al identificar un modelo no lineal entorno a diferentes puntos de operación, el vector paramétrico puede modificar su forma dando lugar a diferentes funciones de transferencia que representen el mismo modelo de la planta. Es aquí donde resalta la importancia de utilizar un controlador diseñado según el método Truxal-Ragazzini frente a un controlador tipo PI. Ambos controladores nos permitirían obtener un conjunto de trayectorias sin error en régimen permanente, sin embargo, como bien se explicó en la sección *3.2 Formulación del controlador Truxal-Ragazzini*, para el diseño del mismo el primer paso es diseñar la función de transferencia en bucle cerrado del sistema $G_d(z)$. De este modo, a pesar de que para cada proceso

de identificación se genere un modelo distinto de la planta *G(z)*, si para diseñar el controlador *C(z)* se impone a todos los modelos de la planta la misma función de transferencia en bucle cerrado, el resultado será un conjunto de trayectorias más centradas en torno a los mismos puntos de operación. Esta implicación ve su resultado final al resolver el problema de optimización, dando lugar a mejores resultados en el diseño del controlador final. Si hubiéramos utilizado un controlador tipo PI para la generación de la base de datos, a pesar de haber conseguido que las trayectorias no presentaran error en régimen permanente, no se hubiese logrado centralizar el conjunto de trayectorias, resultando en inestabilidades en el sistema.

5.7.1 Implementación en Matlab. Comparativa de resultados

Para dar forma a los comentarios antes expuestos, se va a implementar en Matlab dos base de datos distintas, una generada a partir de un controlador Truxal y otra generada a partir de un controlador PI. A partir de estas bases de datos se implementará el controlador *Data Driven* y se compararán los resultados obtenidos.

Supongamos que partimos de la función de transferencia definida en la ecuación (*5.27*). Para hacer la comparativa más real, de acuerdo a lo especificado en la sección anterior, vamos a suponer que el modelo de la planta representado por la ecuación (*5.27*) varía con cada trayectoria generada. Para ello se ha definido una función denominada "*Plants Params Variation*". Esta función recibe como parámetros los polos y ceros de la planta así como una variable en tanto por ciento denominada "*modification*". La función devolverá dos nuevos vectores de polos y ceros donde los parámetros de los mismos han sido modificados un valor aleatorio máximo de *modification*%. De esta forma se simula la variación de la planta al identificar un modelo no lineal.

Código 5.8 Function Plants Params Variation.

```
function [c,p]=Plant_Params_Variation(c_init,p_init,Variation)

size_c=length(c_init);
size_p=length(p_init);

c=zeros(1,size_c);
p=zeros(1,size_p);

for i=1:size_c
    c(i)=c_init(i) + c_init(i)* ( -Variation/100 + 2*Variation*rand()/100);
end

for i=1:size_p
    p(i)=p_init(i) + p_init(i)* ( -Variation/100 + 2*Variation*rand()/100);
end
```

Se adjuntan ahora dos secciones de código. No se va a entrar en mucho detalle en ellas porque simplemente son una ampliación del código expuesto en las secciones *4.3 Implementación de la base de datos en Matlab* y *5.6 Implementación del controlador en Matlab* extendidas al caso actual. Simplemente señalar una serie de detalles, en primer lugar, ambas bases de datos (PI y Truxal) se generan contando con el mismo número de trayectorias. Si denominamos S_j^{PI} a la trayectoria *j* de la base de datos generada a partir del controlador PI y S_j^{Truxal} a la trayectoria *j* de la base de datos generada a partir del controlador Truxal se va a cumplir siempre que en ambas tanto la componente del ruido en continua como las referencias inicial y final son iguales. De este modo se puede dar lugar a una comparación más precisa de ambas bases de datos.

Código 5.9 Generación de la Base de datos, PI vs Truxal.

```
%Definición de una Función de Transferencia
zeros_G_init=0.35;
poles_G_init=[0.5 -0.2 0.9];
Num_G0=poly(zeros_G_init);
Den_G0=poly(poles_G_init);
```

```
tm_opt=0.1;
G0_z=tf(Num_G0,Den_G0,tm_opt);
modification=7.5;

%Paso de la función de transferencia en discreto a su forma en espacio
%estados
[A_G,B_G,C_G,D_G]=StateSpace_Transformation(Num_G0,Den_G0);

%CLOSED LOOP TRANSFER FUNCTION
pd=[0.8 -0.6 0.2];
Den_Gd=poly(pd);
cd=1-polyval(Den_Gd,1);
Num_Gd=poly(cd);
Gd_z=tf(Num_Gd,Den_Gd,0.1);

%Modelos de Control PI
P_min=0.05; %Coeficiente proporcional mínimo
P_max=0.15; %Coeficiente proporcional máximo
I_min=0.1; %Coeficiente integral mínimo
I_max=0.2; %Coeficiente integral máximo

%      Valores que se utilizarán para la simulación del modelo    %
%Tiempo de simulación
ts=120;
%Mínimo valor que puede tomar la media del ruido
tolL_Ref=0;
%Mínimo valor que puede tomar la media del ruido
tolH_Ref=2;
%Mínimo valor que puede tomar la media del ruido
tolL_Noise=-0.001;
%Máximo valor que puede tomar la media del ruido
tolH_Noise=0.001;
%Covarianza del ruido V
QVe=0.0001;
%Covarianza del ruido W
QWe=0.0001;

%Definición variables vase de datos
NumTrajects=1500;
Tstruct=zeros(NumTrajects,1);
Sj_Truxal=struct('X',Tstruct,'U',Tstruct,'Y',Tstruct,'R0',Tstruct,'R1',Tstruct
    ,'Ref',Tstruct);
Sj_PID=struct('X',Tstruct,'U',Tstruct,'Y',Tstruct,'R0',Tstruct,'R1',Tstruct,'
    Ref',Tstruct);

for i=1:NumTrajects

    %Variación de parámetros de la función de transferencia
    [zeros_G,poles_G]=Plant_Params_Variation(zeros_G_init,poles_G_init,
        modification);
    Num_G=poly(zeros_G);
    Den_G=poly(poles_G);
    G_z=tf(Num_G,Den_G,tm_opt);

     %Referencias
    [re0,re1,We,Ve]=Aleatory_References_v3(tolL_Ref,tolH_Ref,tolL_Noise,
        tolH_Noise);
```

```
    [Ref]=DataBase_References(re0,re1,ts,tm_opt,2);

    %Controlador Truxal
    [C_z,Num_Cz,Den_Cz]=ControlTruxal_Variation(Num_G,Den_G,Num_Gd,Den_Gd,
        tm_opt);

    %PI
    P=P_min +(P_max-P_min)*rand;
    I=I_min +(I_max-I_min)*rand;

    %Paso de la función de transferencia en discreto a su forma en espacio
    %estados
    [A_Cz,B_Cz,C_Cz,D_Cz]=StateSpace_Transformation(Num_Cz,Den_Cz);

    %Cálculos de las características estáticas del sistema
    [X0_G,U0_G]=SteadyState_Characterization(A_G,B_G,C_G,D_G,re0,We,Ve);
    [X0_Cz,~]=SteadyState_Characterization(A_Cz,B_Cz,C_Cz,D_Cz,U0_G,0,0);
    Integrator_Init=U0_G;

    Sim_Data=sim('TruxalModel_WNoise_Simulation_Step');

    [Sj_Truxal(i)]=DataBase_Generation(Sim_Data.X,Sim_Data.U,Sim_Data.Y,re0,re1
        ,Ref);

    Sim_Data=sim('PID_Model_WNoise_Simulation_Step');

    [Sj_PI(i)]=DataBase_Generation(Sim_Data.X,Sim_Data.U,Sim_Data.Y,re0,re1,Ref
        );
end
```

Código 5.10 Controlador Data Driven Offset Free, PID vs Truxal.

```
%SOLVER
DataBase_Definition_Truxal_PID_Modifiable

%Variables
Points=length(Sj_Truxal(1).X(1,:));
NumTrajects_Selected=150;
X_Size=length(Sj_Truxal(1).X(:,1));
H_Size=2*X_Size+4;


%Valores inciales
[re0,re1,We,Ve]=Aleatory_References_v3(tolL_Ref,tolH_Ref,tolL_Noise,tolH_Noise)
    ;
[Ref]=DataBase_References(re0,re1,ts,tm_opt,2);
[X0,U0]=SteadyState_Characterization(A_G,B_G,C_G,D_G,re0,We,Ve);

%Inicializar
[Xx_T,Yx_T,Ux_T]=Iniciar_Noise(Points,X0,U0,X_Size,A_G,B_G,C_G,We,Ve,QWe,QVe);
Xx_PI=Xx_T;
Yx_PI=Yx_T;
Ux_PI=Ux_T;

%Estructura de resultados Finales
H_Structs_Size=zeros(H_Size,NumTrajects_Selected);
```

```
b_Structs_Size=zeros(H_Size,1);
Lambda_Structs_Size=zeros(H_Size,1);
Solutions_Struct_T=struct('H',H_Structs_Size,'b',b_Structs_Size,'Lambda',
    Lambda_Structs_Size,'X',X0,'Y',re0,'U',U0,'Ref',re0);
Solutions_Struct_PI=struct('H',H_Structs_Size,'b',b_Structs_Size,'Lambda',
    Lambda_Structs_Size,'X',X0,'Y',re0,'U',U0,'Ref',re0);

Solutions_Struct_T(2).X=Xx_T(:,2);
Solutions_Struct_T(2).Y=Yx_T(2);

Solutions_Struct_PI(2).X=Xx_PI(:,2);
Solutions_Struct_PI(2).Y=Yx_PI(2);

for cnt=2:(Points-1)

%Funcion distancia
Distance_Flags_T=Distance_Calculation(Xx_T(:,cnt-1),Xx_T(:,cnt),Ux_T(cnt-1),
    Yx_T(cnt),Ref(cnt),Sj_Truxal,NumTrajects,NumTrajects_Selected,cnt);
Distance_Flags_PI=Distance_Calculation(Xx_PI(:,cnt-1),Xx_PI(:,cnt),Ux_PI(cnt-1)
    ,Yx_PI(cnt),Ref(cnt),Sj_PI,NumTrajects,NumTrajects_Selected,cnt);

%Construimos H
H_T=Matrix_H_Generation(NumTrajects,cnt,Sj_Truxal,Distance_Flags_T,H_Size);
H_PI=Matrix_H_Generation(NumTrajects,cnt,Sj_PI,Distance_Flags_PI,H_Size);

%Construimos b
b_T=Matrix_b_Generation(Xx_T,Yx_T,Ux_T,Ref,cnt,H_Size,X_Size);
b_PI=Matrix_b_Generation(Xx_PI,Yx_PI,Ux_PI,Ref,cnt,H_Size,X_Size);

%Solución para Truxal
H_Ajusted_T=(H_T*H_T') + 0.00001*eye(H_Size);
Beta_T=(H_Ajusted_T)\b_T;
Lambda_T=H_T'*Beta_T;

% %Nueva señal de control
Ux_T(cnt)=Summultli(Sj_Truxal,Lambda_T,cnt,Distance_Flags_T);


%Valor en espacio estados siguiente tiempo de muestreo
Xx_T(:,cnt+1)=A_G*Xx_T(:,cnt) + B_G*Ux_T(:,cnt) +We +QWe -2*QWe*rand;
Yx_T(:,cnt+1)=C_G*Xx_T(:,cnt+1) +Ve +QVe -2*QVe*rand;

[Solutions_Struct_T(cnt)]=RellenaStruct_v2(H_T,b_T,Lambda_T,Xx_T,Yx_T,Ux_T,Ref,
    cnt);

%Solución para PI
H_Ajusted_PI=(H_PI*H_PI') + 0.00001*eye(H_Size);
Beta_PI=(H_Ajusted_PI)\b_PI;
Lambda_PI=H_PI'*Beta_PI;

% %Nueva señal de control
Ux_PI(cnt)=Summultli(Sj_PI,Lambda_PI,cnt,Distance_Flags_PI);

%Valor en espacio estados siguiente tiempo de muestreo
Xx_PI(:,cnt+1)=A_G*Xx_PI(:,cnt) + B_G*Ux_PI(:,cnt) +We +QWe -2*QWe*rand;
Yx_PI(:,cnt+1)=C_G*Xx_PI(:,cnt+1) +Ve +QVe -2*QVe*rand;
```

```
[Solutions_Struct_PI(cnt)]=RellenaStruct_v2(H_PI,b_PI,Lambda_PI,Xx_PI,Yx_PI,
    Ux_PI,Ref,cnt);
end
```

Se ha realizado la comparativa para cinco casos distintos. Se ha partido de una variación de los parámetros de la planta de un 1 % y se ha aumentado este valor un 0.5 % hasta alcanzar el 3 % como máximo. Para todos los controladores generados según el método de diseño *Truxal-Ragazzini* se les ha impuesto la misma función de transferencia en bucle cerrado. Que para este ejemplo en específico se corresponde con la siguiente ecuación:

$$G_d(z) = \frac{z - 0.744}{z^3 - 0.4z^2 - 0.44z + 0.096} \tag{5.34}$$

El modelo de controlador PI se corresponde con la siguiente ecuación:

$$PI(z) = P + I \cdot T_s \frac{1}{z-1} \tag{5.35}$$

Los parámetros "P" e "I" del control PI se han elegido para que también varíen de forma aleatoria con cada simulación. Los límites de dichos parámetros se exponen en las siguientes igualdades:

$$\begin{aligned} P_{min} &= 0.05 \\ P_{max} &= 0.15 \\ I_{min} &= 0.1 \\ I_{max} &= 0.2 \end{aligned}$$

En las dos siguientes figuras se pueden observar los modelos de bloques utilizados para simular ambos modelos de control PI y Truxal:

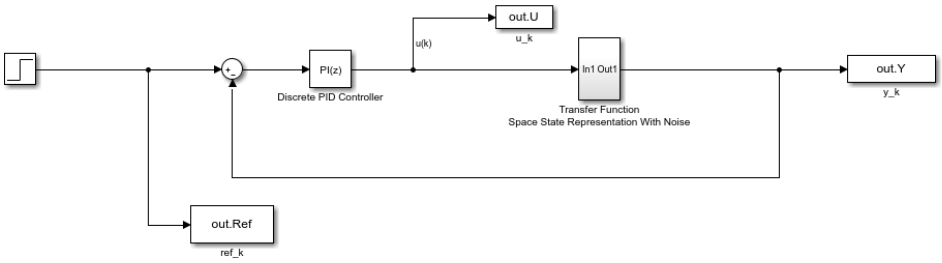

Figura 5.2 Diagrama de bloques Simulink. Control PI.

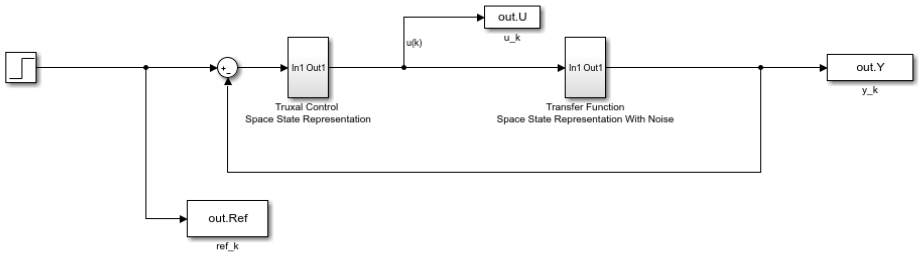

Figura 5.3 Diagrama de bloques Simulink. Control Truxal.

Los bloques *Space State Representation With Noise* son los mismos que se corresponden con la *Figura 3.8*. Tras la generación de ambas bases de datos implementadas en el código (*5.9*), se aplica el código (*5.10*) que implementa el controlador *Data Driven*. Para la comparativa, se ha reducido según la función distancia el número de candidatos de ambas bases de datos a 150. Ambos controladores *Data Driven* se han forzado a seguir las mismas referencias inicial y final para asemejar la comparativa. Se adjunta una tabla en la que se muestran los errores absolutos obtenidos para cada modelo de base de datos en función del parámetro *modification*:

Tabla 5.1 Error de identificación del modelo para las distintas formas del vector regresor.

Modification	Error Base de Datos PI	Error Base de Datos Truxal
1 %	166.1227	60.3345
1.5 %	155.5957	55.6808
2 %	284.4336	108.9912
2.5 %	265.9085	162.1106
3 %	147.75	175.1203

Se puede concluir que para variaciones pequeñas de la función de transferencia, si utilizamos el controlador Truxal, al acotarse los valores de estados almacenados en la base de datos, los resultados obtenidos resultan evidentemente mejores. Sin embargo, al aumentar la variación de los polos y ceros de la función de referencia respecto a la original se impone el controlador tipo PI. Estos resultados se ampliaran con la comparativa que se realice en el *Capítulo 7*.

Finalmente, se adjuntan una serie de gráficas que hacen referencia a los resultados finales de la comparativa del controlador *Data Driven Offset Free* implementado mediante la base de datos generada por un controlador PI y un controlador Truxal. Todas estas gráficas hacen referencia al caso previo de $modification = 2.5\,\%$. En primer lugar, a modo de ejemplo, es posible observar dos de las trayectorias almacenadas en S_j^{PI} yS_j^{Truxal} respectivamente. Las trayectorias mostradas se han elegido de forma aleatoria.

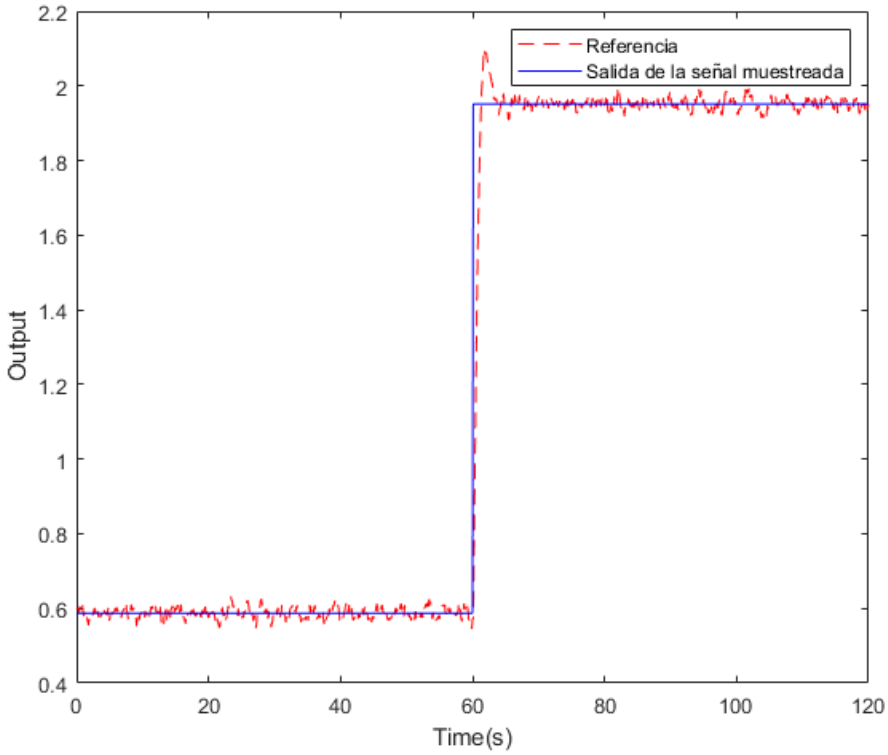

Figura 5.4 Salida vs Referencia. Base de datos PI.

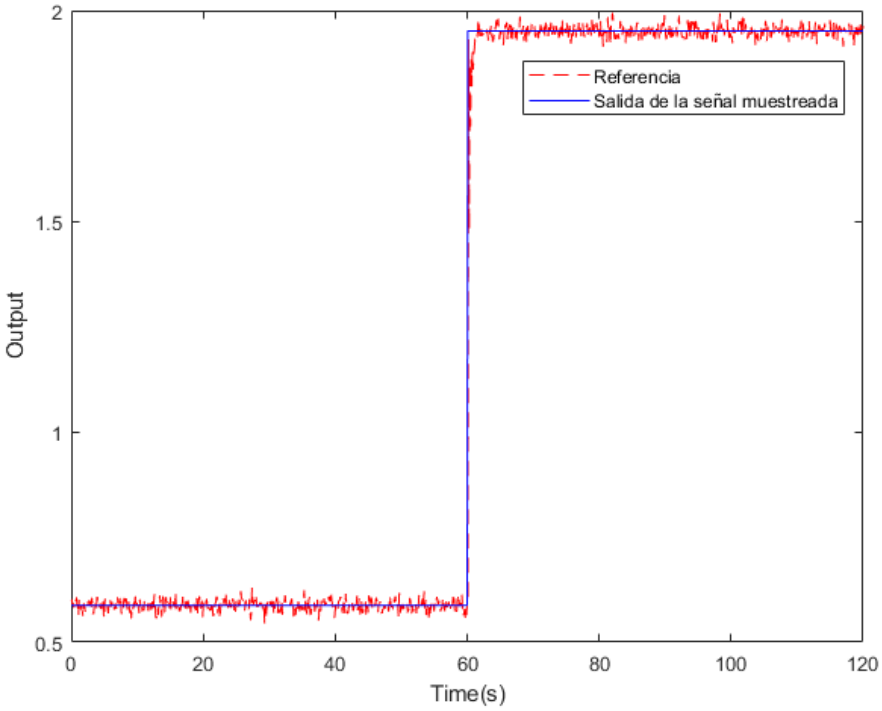

Figura 5.5 Salida vs Referencia. Base de datos Truxal.

Se adjuntan las dos gráficas de resultados finales tras la generación de los controladores *Data Driven* a partir de las respectivas bases de datos.

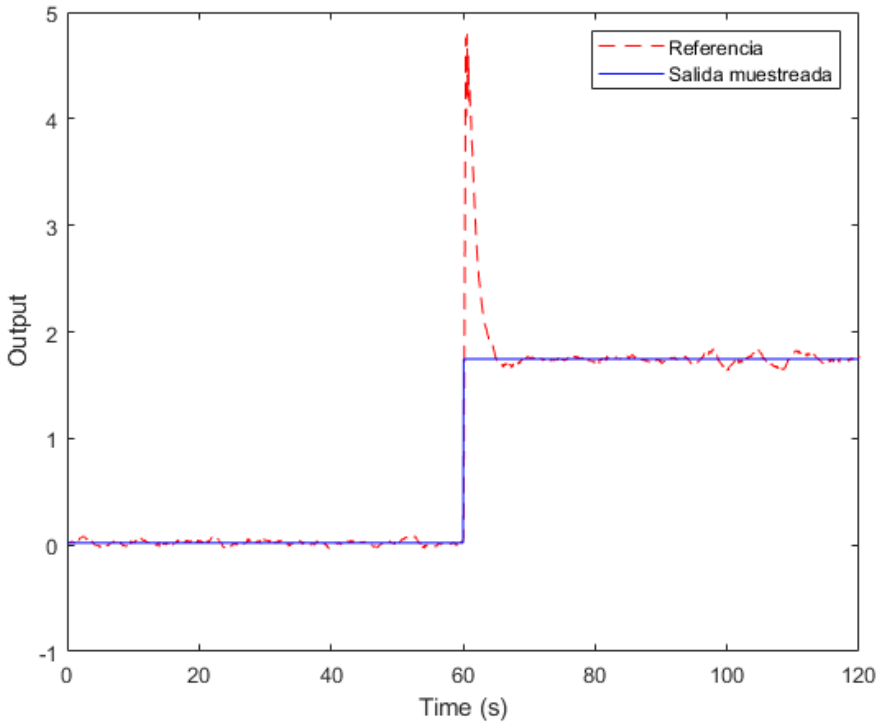

Figura 5.6 Controlador Data Driven. Base de datos PI.

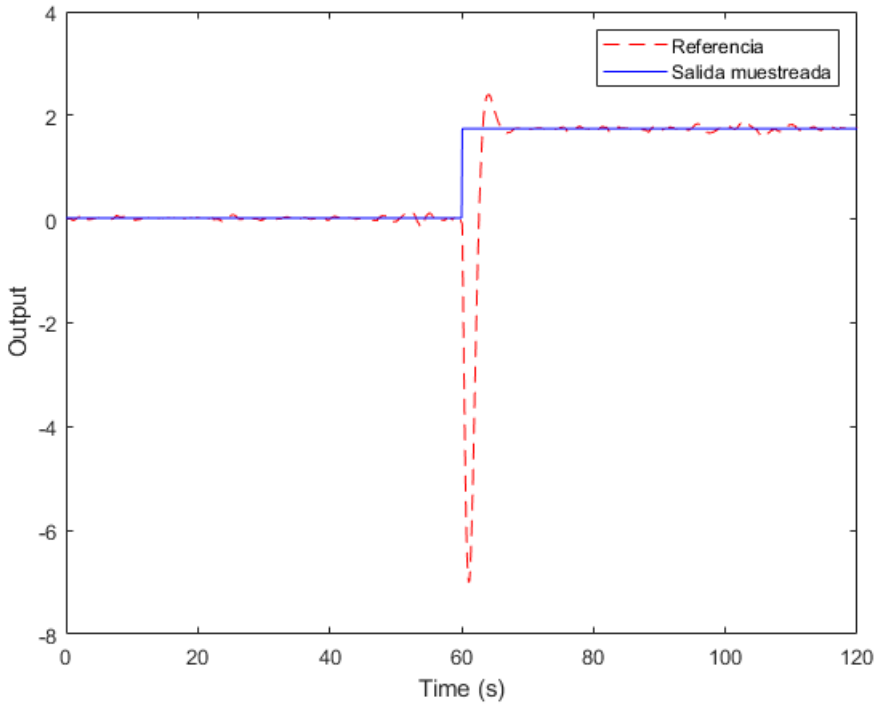

Figura 5.7 Controlador Data Driven. Base de datos Truxal.

6 Modelado de un Sistema de dos tanques de agua concatenados

Con el objetivo de dar veracidad al control, se va a probar el mismo en un una simulación, que no deja de ser un modelo, pero que sirva para representar a un sistema real. Se ha escogido un ejemplo de modelo no lineal bien conocido, dos tanques de agua concatenados. A partir de este modelo, se generaran las simulaciones a identificar y que posteriormente serán usadas para implementar el control en línea del sistema. La elección de este modelo, que puede parecer simplista y muy estudiado, se debe a que la importancia no recae en este. Con la elección de este modelo se quiere centrar los resultados en la generación de la base de datos y la implementación del controlador final y no el modelado del mismo. Posteriormente se explican el desarrollo de las ecuaciones matemáticas del modelo y la implementación en la herramienta de diseño*Simulink*.

6.1 Definición del sistema

Nuestro sistema consta de dos tanques de áreas A_1 y A_2 respectivamente.El agua entra al primer tanque mediante una tubería de área transversal a_1. La salida del tanque 1 está directamente conectada con la entrada del tanque 2 por otra tubería de área transversal a_2. El tanque 2 presenta una segunda entrada de agua de caudal continuo de valor q_x. Finalmente el segundo tanque desemboca en otra tubería de área transversal a_3. Consideraremos que todas las tuberías están reguladas por unas válvulas de presión cuyos coeficientes de descarga supondremos iguales entre sí y a un valor C_q.

6.2 Modelo no lineal de un tanque de agua

Suponemos un tanque de volumen V que recibe una entrada y salida continua de caudal q_i y q_s respectivamente. Por tanto la variación de volumen del tanque será:

$$\frac{dV}{dt} = q_i - q_s \tag{6.1}$$

Cómo el caudal por una tubería se define como el producto del área de paso por la velocidad del fluido.

$$q_n = a \cdot v \tag{6.2}$$

Y según la ecuación de Torricelli la velocidad del fluido se rige según la ecuación:

$$v = C_q \cdot \sqrt{2 \cdot g \cdot h} \tag{6.3}$$

Siendo g la aceleración de la gravedad, h la altura de la columna de agua del tanque y C_q el coeficiente de descarga de la válvula de salida del tanque. Uniendo las ecuaciones (*6.1*) (*6.2*) (*6.3*) tenemos que:

$$\begin{aligned} q_s &= a \cdot C_q \cdot \sqrt{2 \cdot g \cdot h} \\ \frac{dV}{dt} = A \cdot \frac{dh}{dt} &= q_i - a \cdot C_q \cdot \sqrt{2 \cdot g \cdot h} \end{aligned} \tag{6.4}$$

En cuanto al caudal de entrada q_i , suponiendo que aguas arribas la válvula entrega una presión constante Δp y el agua tiene un peso específico de valor γ:

$$q_i = C_q \cdot A_0 \cdot \sqrt{\frac{2 \cdot g}{\gamma} \cdot \Delta p} \tag{6.5}$$

A_0 será la sección de paso real de la tubería de entrada al tanque 1, y estará regulada por la mayor o menor apertura de la válvula, siendo x la apertura instantánea de la válvula y x_{max} la apertura máxima de la misma.

$$\begin{aligned} A_0 &= a_i \cdot \frac{x}{x_{max}} \\ 0 \leq \frac{x}{x_{max}} &\leq 1 \end{aligned} \tag{6.6}$$

Agrupando los términos constantes que aparecen en las ecuaciones de los caudales de entrada y salida en sendos coeficientes C_1 y C_2:

$$\begin{aligned} C_1 &= C_q \cdot a_i \cdot \sqrt{\frac{2 \cdot g}{\gamma} \cdot \Delta p} \\ C_2 &= a \cdot C_q \cdot \sqrt{2 \cdot g} \end{aligned} \tag{6.7}$$

Finalmente uniendo las ecuaciones (*6.4*) (*6.5*) (*6.6*) obtenemos el modelo no lineal del sistema.

$$\frac{dh}{dt} = \frac{1}{A} \cdot \left(C_1 \cdot \frac{x}{x_{max}} - C_2 \cdot \sqrt{h} \right) \tag{6.8}$$

6.3 Implementación en Simulink

Una vez definido el sistema y las ecuaciones que regirán su comportamiento, pasaremos a implementar dicho modelo en Simulink. El primer tanque de área A_1, altura inicial h_{01}, altura máxima h_{01max}, resistencia hidráulica R_{h1}, área transversal de la tubería de entrada a_1 y área transversal de la tubería de salida a_2 quedará representado por este modelo:

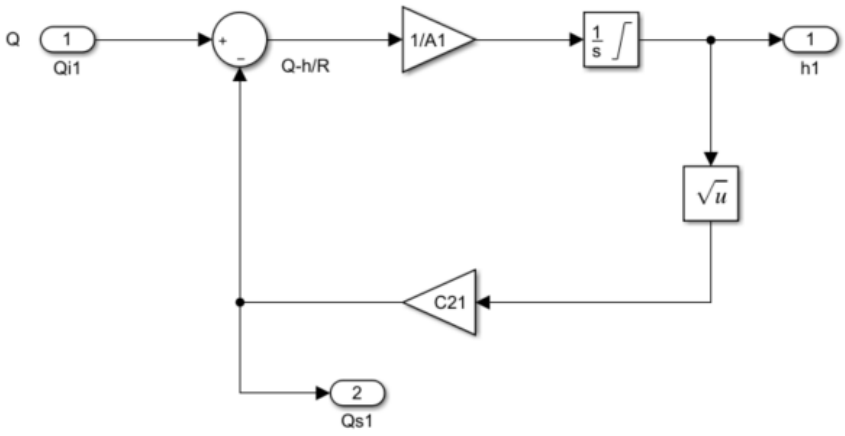

Figura 6.1 Modelo no lineal del primer tanque de agua. Bloque Simulink.

El integrador tendrá impuesto un límite superior igual a la altura máxima del tanque h_{01}, un límite inferior igual a cero y un valor inicial h_{01}. La entrada a este primer subsistema es el caudal Q_{i1} y su salida Q_{s1} será el caudal de entrada del segundo subsistema que representa al tanque 2. Este tanque tendrá un área transversal A_2, altura inicial h_{02}, altura máxima h_{02max}, resistencia hidráulica R_{h2}, área transversal de la tubería de entrada a_2 (que es la misma que la tubería de salida del primer tanque) y área transversal de la tubería de salida a_3. El modelo de este segundo tanque será igual al primero (incluyendo las mismas limitaciones al integrador con sus correspondientes valores).

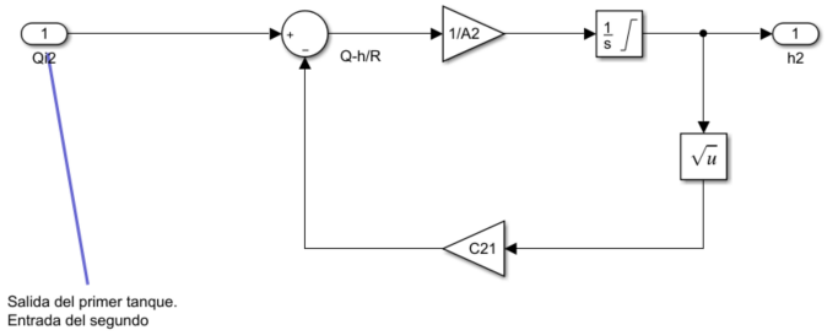

Figura 6.2 Modelo no lineal del segundo tanque de agua. Bloque Simulink.

Finalmente solo queda definir los parámetros de simulación del sistema. Nuestro sistema tendrá un tiempo de simulación t_S y el método de integración o "*Solver*" utilizado será ode4 (Runge-Kutta) con un tiempo de

muestreo fijo de valor t_m. Respecto al modelado y control del sistema, se va a realizar un control realimentado mediante un PID. Se van a controlar dos set points que representan la altura inicial y final del segundo tanque "h_2".Para representar estas referencia se va utilizar el bloque *PulseGenerator*. Para el control se va a utilizar una función tipo PID, cuyos parámetros se corresponden con la siguiente ecuación.

$$PID(s) = P(1 + I\frac{1}{s}) \tag{6.9}$$

Donde

$$P = 0.5$$
$$I = 2$$

Los valores del controlador PI se han elegido mediante la opción "*Tune*" que presenta el propio bloque del controlador en Simulink. La salida del PID estará limitada entre cero y uno para simular la apertura de la válvula de entrada ($0 \leq \frac{x}{x_{max}} \leq 1$). Finalmente la salida de este PID se multiplicará por el coeficiente C_1 de la ecuación (4.1). El modelo final del sistema (agrupando los dos tanques en sendos subsistemas) se puede observar en la siguiente figura

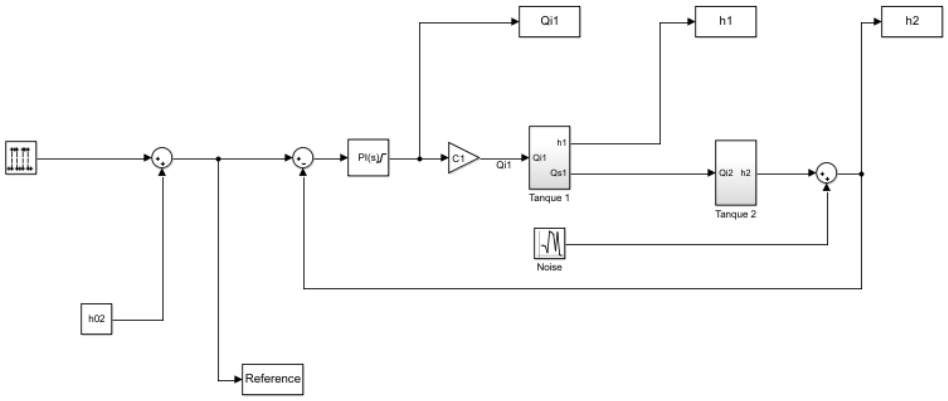

Figura 6.3 Modelo no lineal sistema de dos tanques de agua concatenados. Simulink.

Para simular el ruido del sistema se ha añadido una perturbación a la salida h_2. Esta perturbación representa un ruido de media no nula. Se ha utilizado la función "*Random Number*", ya comentada en otras ocasiones, para simular dicho ruido. Tanto los valores de ruido como referencias del sistema se han escogido de forma aleatoria haciendo uso de la función definida en la sección de código *4.4 Aleatory References*. Se han forzado que tanto las referencias como el valor medio del ruido se encuentre dentro de unos límites establecidos que se muestran a continuación:

$$h2_{min} = 0.5$$
$$h2_{max} = 1.5$$
$$Ve_{min} = 0.01$$
$$Ve_{max} = 0.03$$

Finalmente comentar, para que el sistema inicialice plano, se han de cumplir las siguientes condiciones iniciales:

$$h_{01} = h_{02}$$
$$Q_{i0} = \frac{\sqrt{h_{01}} \cdot C_2}{P \cdot C_1}$$

Se cumple también que Q_{i0} es el valor inicial del integrador de PID(s) necesario para que el sistema arranque plano.

En la siguiente sección de código adjunta se muestra la inicialización de todos los parámetros necesarios para la simulación. A su vez se ha representado en tres gráficas distintas el valor del caudal de entrada Q_{i1}, las alturas del primer y segundo tanque h_1 y h_2 y la referencia del sistema que varía entre un valor inicial h_{02} y final $h_{02}+K$ para un tiempo de simulación $t_s = 840s$ y un tiempo de muestreo $t_m = 0.1s$. Las referencia utilizadas se escogen de forma aleatoria utilizando la función ya definida *Aleatory References* en el código (*4.4*). Pero, para el caso mostrado en las figuras adjuntas, el valor utilizado ha sido:

$$h_{02} = 0.9567$$
$$h_{02+K} = 1.3162$$

Código 6.1 Inicialización variables/parámetros de simulación sistema de dos tanques concatenados.

```
clc
clear
%En este función se declaran todas las variables respectivas a la
%inicialización de la simulación que representa la variación de la altura
%de dos tanques de agua concatenados.

%PARÁMETROS TUBERÍA DE ENTRADA
a1=0.75; %Área conducto de entrada [m^2]
Cv=0.9; %Constante de la válvula
g=9.81; %gravedad [m^2/s]
Lambda=9807; %Peso espesífico del agua [N/m^3]
Ap=7000; %Variación de presión
C1=Cv*a1*sqrt(2*g*Ap/Lambda); %Coeficiente 1

%PARÁMETROS PRIMER TANQUE
A1=5; %Área transversal del tanque [m^2]
h01max=4; %Altura maxima del tanque [m]
a2=0.5; %Área transversal tubería de salida [m^2]
C21=a2*Cv*sqrt(2*g); %Coeficiente 2


%PARÁMETROS SEGUNDO TANQUE
A2=2; %Área transversal del tanque [m^2]
h02max=4; %Altura maxima del tanque [m]
a3=a2; %Área transversal tubería de salida [m^2]
C22=a3*Cv*sqrt(2*g); %Coeficiente 2

%PARÁMETROS SIMULACIÓN Y CONTROL
ts=1640; %Tiempo de simulación [s]
tm=0.1; %Tiempo de muestreo [s]
N_Period=7; %Número de Periodos
Period=round(ts/(N_Period*tm)); %Periodo de la señal Generador de Pulsos
Width=round(ts/(N_Period*2*tm)); %Ancho de Pulso
P=0.5; %Coeficiente proporcional
```

```
I=2; %Coeficiente integral


%Inicialización
h2_min=0.5;
h2_max=1.5;
Noise_min=0.01;
Noise_max=0.03;
[h02,h2,Ve]=Aleatory_References_v2(h2_min,h2_max,Noise_min,Noise_max);
h01=h02; %Altura inicial del tanque [m]
Qi0=sqrt(h01)*C21/(P*C1); %Caudal inicial de entrada [m^3/s]
K=h2-h02; %Ancho de la señal Generador de Pulsos

%Ruido
QVe=0.001;

sim('Concatenate_Tanks_WNoise_Simulation_NoLineal')
end
```

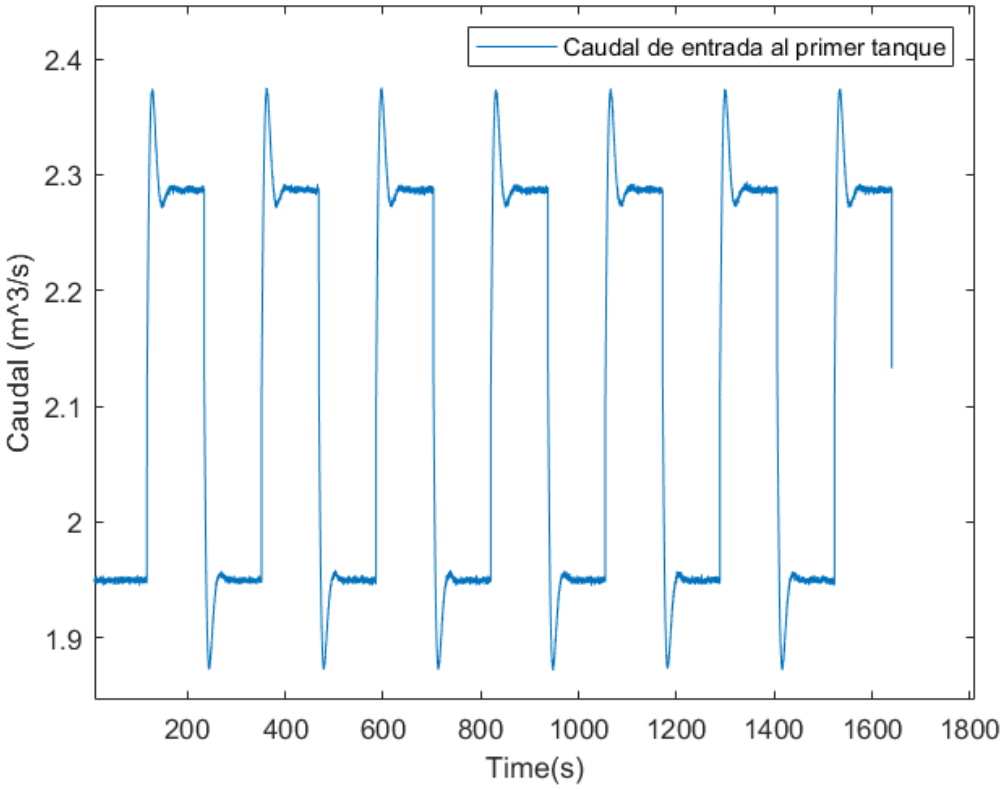

Figura 6.4 Caudal de entrada del primer tanque. Simulación sistema dos tanque concatenados.

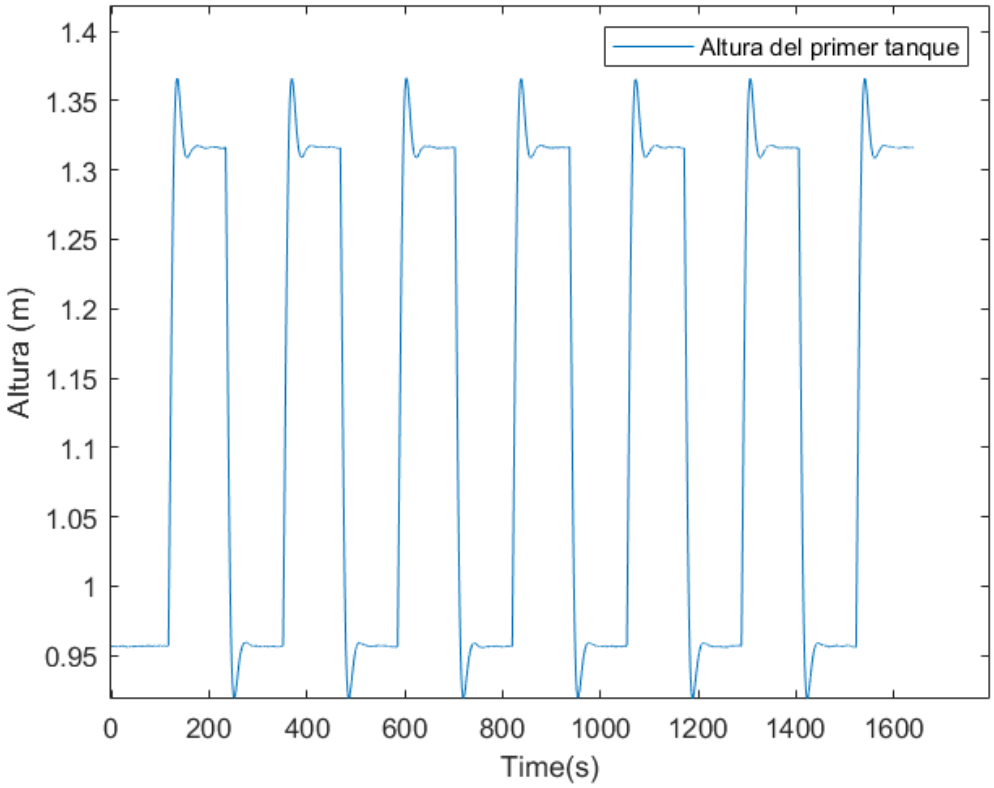

Figura 6.5 Altura del primer tanque. Simulación sistema dos tanque concatenados.

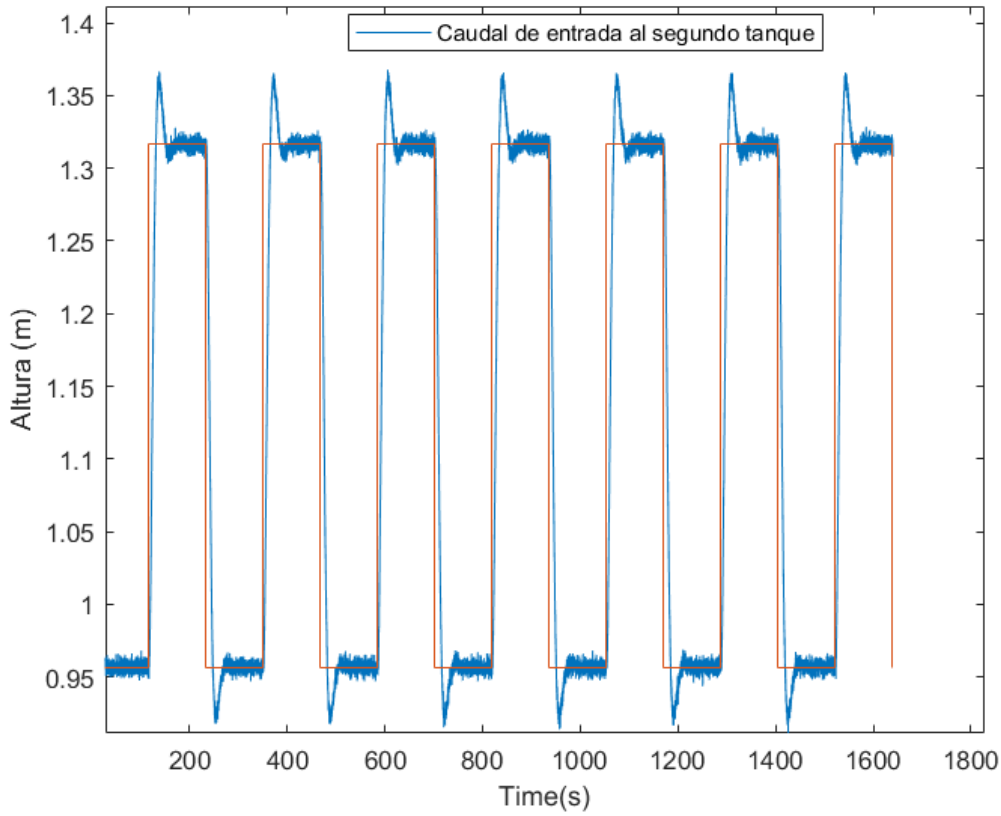

Figura 6.6 Altura del segundo tanque. Simulación sistema dos tanque concatenados.

7 Implementación del controlador Data Driven en un sistema real

En este apartado se lleva a cabo el desarrollo final de todo lo explicado en los capítulos previos. Partiendo del modelo de dos tanques de agua concatenados desarrollado en el *Capítulo 6* se llevará a cabo la identificación utilizando el método de los mínimos cuadrados recursivos ponderados. Una vez identificada la función de transferencia del sistema, se calculará un controlador según el método de diseño *Truxal-Ragazzini*. Este proceso será iterativo y servirá para generar la base de datos con la que se implementará el controlador final.

7.1 Identificación del sistema

Partiendo del modelo de bloques de Simulink que puede ser observado en la *Figura 6.3*. y la inicialización del mismo llevada a cabo de el *Código 6.1* se va a llevar a cabo la identificación del mismo. Para el proceso de identificación nos basaremos en la algoritmia desarrollada durante la sección *2.2.2 Mínimos cuadrados recursivos ponderados, sistema con ruido* y, en concreto, haremos uso de una adaptación para este caso particular de las secciones de código *Código 2.12 Algoritmo de identificación, mínimos cuadrados recursivos, sistema con ruido*, *Código 2.13 Function LeastSquare Algortithm WNoise* y *Código 2.14 Function Define LeastSquare Vectors Noise*.

En primer lugar, se expone la sección de código específica utilizada para el proceso de identificación.

Código 7.1 Algoritmo mínimos cuadrados recursivos, aplicación modelo de dos tanques de agua concatenados.

```
%Variables de estructura, resultados finales
S_Results=struct('teta',0,'M',0,'Num',0,'Den',0,'time',0,'tm',0,'Y',0,'Yaprox
    ',0,...
    'Yaprox2',0,'Yerror',0,'Yerror2',0,'Error',0,'Error2',0);


%VARIABLES MÍNIMOS CUADRADOS
k1=4;%fila a partir de la cual se obtienen valores válidos del vector paramé
    trico
k2=7; %tamaño inicial del vector regresor
yk=3; %Número de valores pasados de y contando desde yk-1
uk=3; %Número de valores pasados de u contando desde uk-1

%CONDICIÓN ITERACIÓN BUCLE
flag=0;
tolerance=30;


while flag==0
```

```
    %Tiempo de muestreo y Simulación del sistema de tanques de agua
    Concatenate_Tanks_WNoise_Inicialization_NoLineal_v3


    %Variables medidas del sistema
    x=Qi1_sim;
    y=h2_sim;

    %Algoritmo recursivo: Mínimos Cuadrados Ponderados
    Lambda=0.9;%Valor base del factor de olvido
    Pi=998;%Valor inicial de la matriz P

    %Creación de M y TETA
    M=zeros(t,k2);
    TETA=zeros(k2,t);

    %Algoritmo mínimos cuadrados
    [teta,M,TETA_END]=LeastSquare_Algorithm_WNoise(k1,k2,M,TETA,t,y,x,Lambda,Pi
        ,yk,uk);

    %Variable estructura que recoge todos los parámetros importantes
    %referentes al algoritmo de identificación
    S_Results=Fill_Structs_WNoise(M,teta,t,k1,k2,time,tm,y,yk,uk);

     if S_Results.Error2 <= tolerance
        flag = 1;
     end
end
```

No se va a entrar en mucho detalle en la misma, pues ésta es una adaptación de las secciones de código comentadas en el párrafo anterior, pero, si cabe destacar una serie de detalles que se han tenido en cuenta en la implementación. Los procesos de identificación (tantos como trayectorias finales tendrá la base de datos) se han realizado con un tamaño y forma fijas del vector regresor, en concreto:

$$m(k) = \begin{bmatrix} y_{k-1} & y_{k-2} & u_{k-1} & e_{k-1} \end{bmatrix} \tag{7.1}$$

Este vector se ha elegido teniendo en cuenta dos varemos. En primer lugar se ha impuesto un tamaño máximo del mismo, con la intención de evitar una posible sobreparametrización del sistema. El tamaño máximo impuesto al vector regresor fue el siguiente

$$m(k) = \begin{bmatrix} y_{k-1} & y_{k-2} & y_{k-3} & u_{k-1} & u_{k-2} & u_{k-3} & e_{k-1} \end{bmatrix} \tag{7.2}$$

Luego, dentro de las distintas posibilidades de formas recogidas para ese tamaño máximo, se ha escogido la que presente menor error absoluto (mismo parámetro que se utilizó en el *Capítulo 2*). En concreto se utiliza la ecuación (*2.37*). En la siguiente tabla se adjuntan los distintos vectores regresores utilizados y los errores absolutos obtenidos al realizar 3 simulaciones distintas por cada vector regresor.

Tabla 7.1 Error de identificación del modelo para las distintas formas del vector regresor.

	Error Simulación 1	Error Simulación 2	Error Simulación 3
$[y_{k-1}\, y_{k-2}\, u_{k-1}\, e_{k-1}]$	76.8295	21.9285	8.2613
$[y_{k-1}\, y_{k-2}\, u_{k-1}\, u_{k-2}\, e_{k-1}]$	23.2427	14.0003	96.0053
$[y_{k-1}\, y_{k-2}\, y_{k-3}\, u_{k-1}\, u_{k-2}\, e_{k-1}]$	96.1417	38.6225	98.9955
$[y_{k-1}\, y_{k-2}\, y_{k-3}\, u_{k-1}\, u_{k-2}\, u_{k-3}\, e_{k-1}]$	82.591	34.6462	41.413

Como se puede extraer de la información que se recoge en la propia tabla, los mejores resultados se corresponden con la forma del vector regresor asociado a la ecuación (*7.1*). Finalmente se adjuntan dos

capturas referentes, en primer lugar, a la representación de la salida real del modelo frente a la salida estimada tras el proceso de identificación y, en segundo lugar, al error de estimación obtenido para un caso aleatorio. Las gráficas corresponden al vector regresor escogido para la simulación número cuatro.

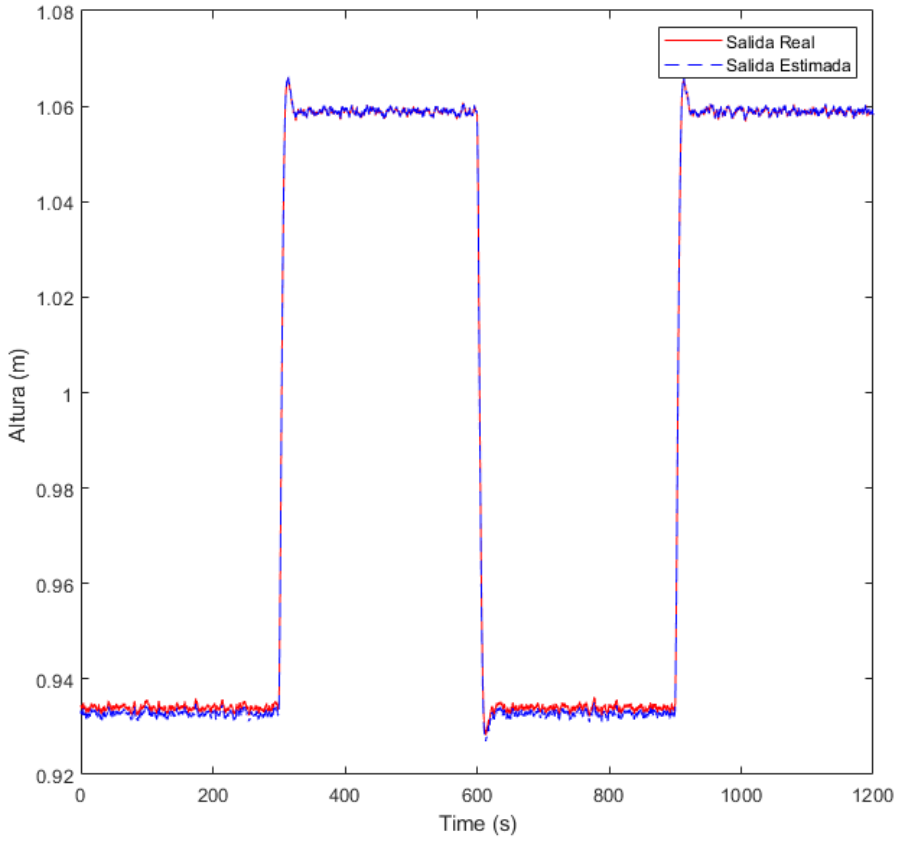

Figura 7.1 Salida real vs Salida estimada del sistema.

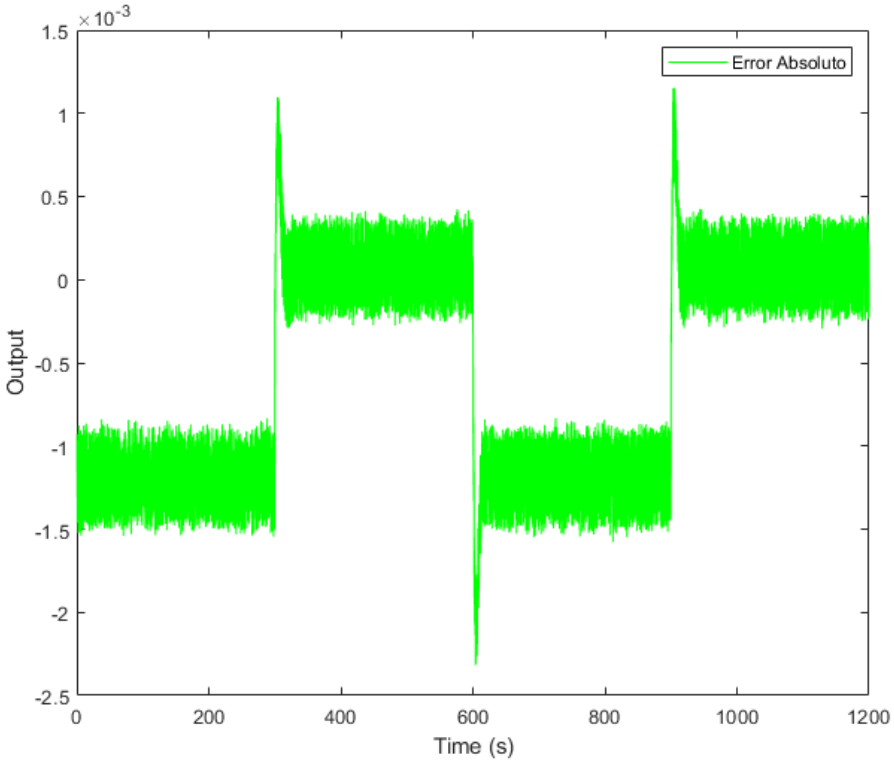

Figura 7.2 Error absoluto del proceso de identificación.

Se añade, a modo de ejemplificación, la función de transferencia final obtenida para la cuarta simulación:

$$G(z) = \frac{0.000688z}{z^2 - 0.8692z - 0.1094} \tag{7.3}$$

7.2 Controlador Truxal y generación de la Base de datos

Una vez se ha identificado el sistema, el siguiente paso es el diseño del controlador Truxal y la generación de la base de datos. Se adjunta la sección de código correspondiente.

Código 7.2 Generación de la base de datos y controlador Truxal, sistema de dos tanques concatenados.

```
NumTrajects=1500;
Data_Base=struct('X',[0;0],'U',0,'Y',0,'R0',0,'R1',0,'Ref',0);

%CLOSED LOOP TRANSFER FUNCTION
pd=[0.5 -0.4];
Den_Gd=poly(pd);
cd=1-polyval(Den_Gd,1);
Num_Gd=poly(cd);
Gd_z=tf(Num_Gd,Den_Gd,0.1);


%Generación de la Base de Datos
for iCounter=1:NumTrajects
```

```
    %Proceso de identificaciónq
    LeastSquare_Concatenate_Tanks_NoLineal_WNoise_v2

    %TRUXAL
    [C_z,Num_Cz,Den_Cz]=ControlTruxal_Variation( S_Results.Num, ...
        S_Results.Den,Num_Gd,Den_Gd,tm);

    %Referencias
    [re0,re1,We,Ve]=Aleatory_References_v3(h2_min,h2_max,Noise_min,Noise_max);
    [Ref]=DataBase_References(re0,re1,ts,tm,2);

    %Paso de la función de transferencia en discreto a su forma en espacio
    %estados
    [A_Cz,B_Cz,C_Cz,D_Cz]=StateSpace_Transformation(Num_Cz,Den_Cz);
    [A_G,B_G,C_G,D_G]=StateSpace_Transformation(S_Results.Num,S_Results.Den);

    %Cálculos de las características estáticas del sistema
    [X0_G,U0_G]=SteadyState_Characterization(A_G,B_G,C_G,D_G,re0,We,Ve);
    [X0_Cz,~]=SteadyState_Characterization(A_Cz,B_Cz,C_Cz,D_Cz,U0_G,0,0);

    Sim_Data=sim('TruxalModel_WNoise_Simulation_Step');

    [Data_Base(iCounter)]=DataBase_Generation(Sim_Data.X,Sim_Data.U,Sim_Data.Y,
        re0,re1,Ref);
end

end
```

Cabe destacar una serie de parámetros de diseño que se han utilizado. En primer lugar, el número de trayectorias almacenadas (que es igual al número de iteraciones que realiza el algoritmo) es de 1500. En segundo lugar, se ha escogido la misma función en bucle cerrado para todas las simulaciones dadas. Esta puede observarse en la siguiente ecuación:

$$G_d(z) = \frac{z-0.3}{z^2-0.1z-0.2} \tag{7.4}$$

La sección de código adjunta es prácticamente idéntica al *Código 4.5* desarrollado en la sección *4.3 Implementación de la base de datos en Matlab*. Las funciones utilizadas se encuentran también ya definidas en las secciones de código *3.1*, *3.8* y *4.4*. Finalmente se adjuntan dos ejemplos de trayectorias escogidas aleatoriamente de la base de datos.

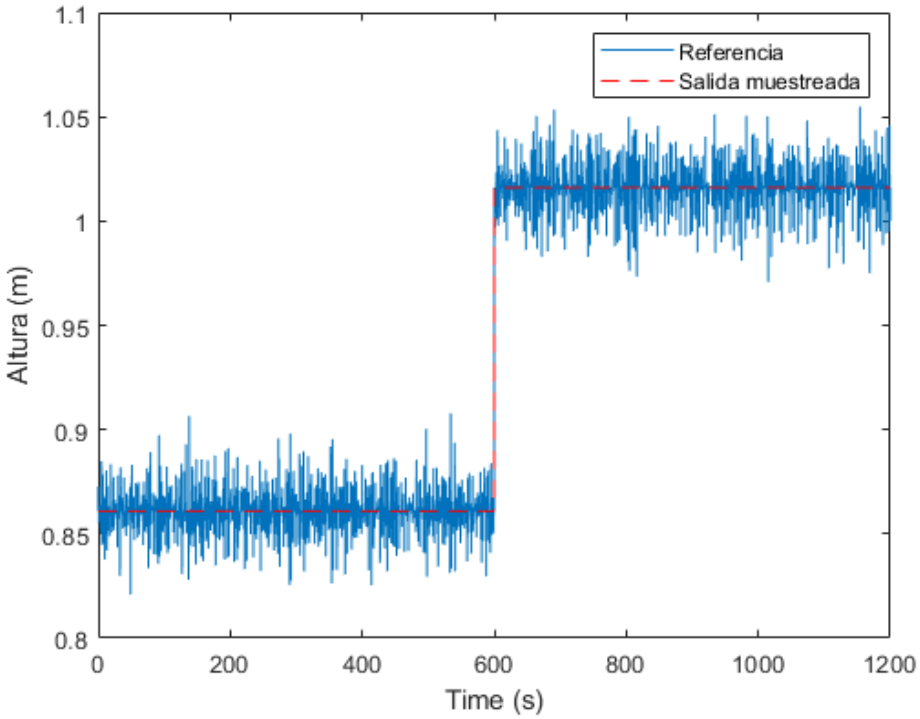

Figura 7.3 Referencia vs Salida muestreada, trayectoria 1191.

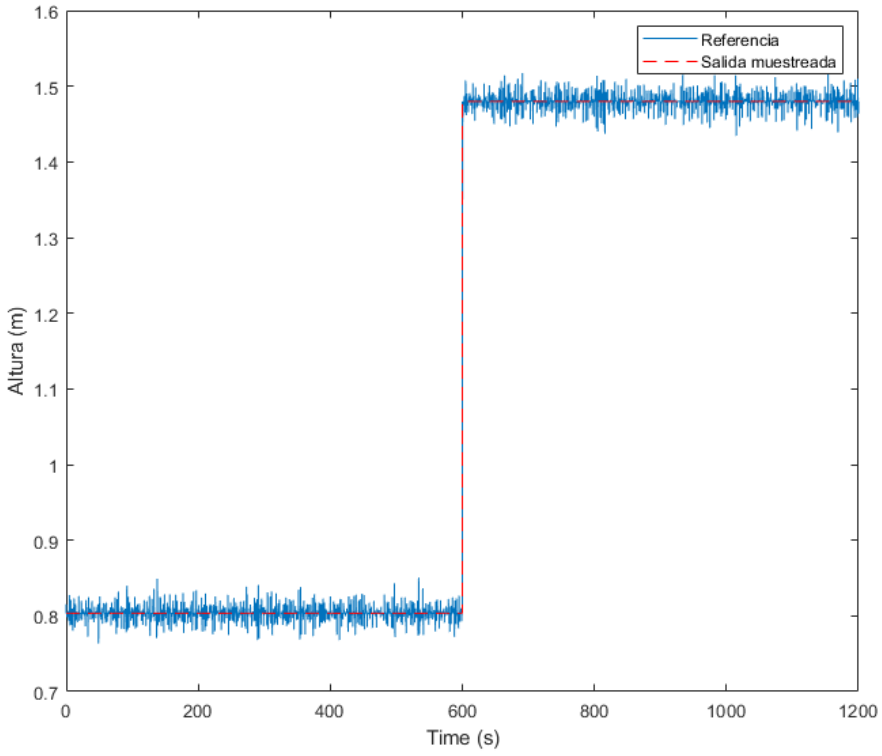

Figura 7.4 Referencia vs Salida muestreada, trayectoria 749.

7.3 Implementación del controlador Data Driven

Con nuestra base de datos generada, solo queda la implementación del controlador final. El código desarrollado, al igual que el correspondiente al *Código 7.1*, es una adaptación del código desarrollado en su propio capítulo para el caso de los dos tanques de agua. Se adjunta la sección de código correspondiente.

Código 7.3 Controlador Data Drive, sistema de dos tanques concatenados.

```
%Variables
Points=Puntos + 1;
NumTrajects_Selected=150;
X_Size=2;
H_Size=2*X_Size+4;

%Valores inciales
[h02,h2,Ve,We]=Aleatory_References_v3(h2_min,h2_max,Noise_min,Noise_max);
[Ref]=DataBase_References_v2(h02,h2,time,3,1);

[X0,U0]=SteadyState_Characterization_v3(h02,C22,C21,C1,Ve,We,A1,A2);
h01=X0(1,1);
Acu_Int_1=0;
Acu_Int_2=0;

%Inicializar
[Xx,Yx,Ux]=Iniciar_v2(Points,X0,U0,X_Size,h02);

%Estructura de resultados Finales
H_Structs_Size=zeros(H_Size,NumTrajects_Selected);
b_Structs_Size=zeros(H_Size,1);
Lambda_Structs_Size=zeros(H_Size,1);
Solutions_Struct=struct('H',H_Structs_Size,'b',b_Structs_Size,'Lambda',
    Lambda_Structs_Size,'X',X0,'Y',h02,'U',U0,'Ref',h02);

Solutions_Struct(2).X=Xx(:,2);
Solutions_Struct(2).Y=Yx(2);

for cnt=2:(Points-1)

%Funcion distancia
Distance_Flags=Distance_Calculation(Xx(:,cnt-1),Xx(:,cnt),Ux(cnt-1),Yx(cnt),Ref
    (cnt),Data_Base,NumTrajects,NumTrajects_Selected,cnt);

%Construimos H
H=Matrix_H_Generation(NumTrajects,cnt,Data_Base,Distance_Flags,H_Size);

%Construimos b
b=Matrix_b_Generation(Xx,Yx,Ux,Ref,cnt,H_Size,X_Size);

%Solución
H_Ajusted=(H*H') + 0.00001*eye(H_Size);
Beta=(H_Ajusted)\b;
Lambda=H'*Beta;

% %Nueva señal de control
Ux(cnt)=Summultli(Data_Base,Lambda,cnt,Distance_Flags);
```

```
U=Ux(cnt);
h1_in=Xx(1,cnt);
h2_in=Xx(2,cnt);
Int_1=Acu_Int_1;
Int_2=Acu_Int_2;

[h1,h2,Acu_Int_1,Acu_Int_2]=ConcatenateTanks_WNoise(U,C21,C22,C1,A1,A2,Int_1,
    Int_2,...
                h01,h02,h1_in,h2_in,Ve,We,QVe,QWe);

%Valor en espacio estados siguiente tiempo de muestreo
Xx(:,cnt+1)=[h1 ; h2];
Yx(:,cnt+1)=h2;


[Solutions_Struct(cnt)]=RellenaStruct_v2(H,b,Lambda,Xx,Yx,Ux,Ref,cnt);

end
```

Para la implementación final se han escogido 150 trayectorias del total de 1500 que recogía la base de datos. Se adjunta una gráfica del resultado final.

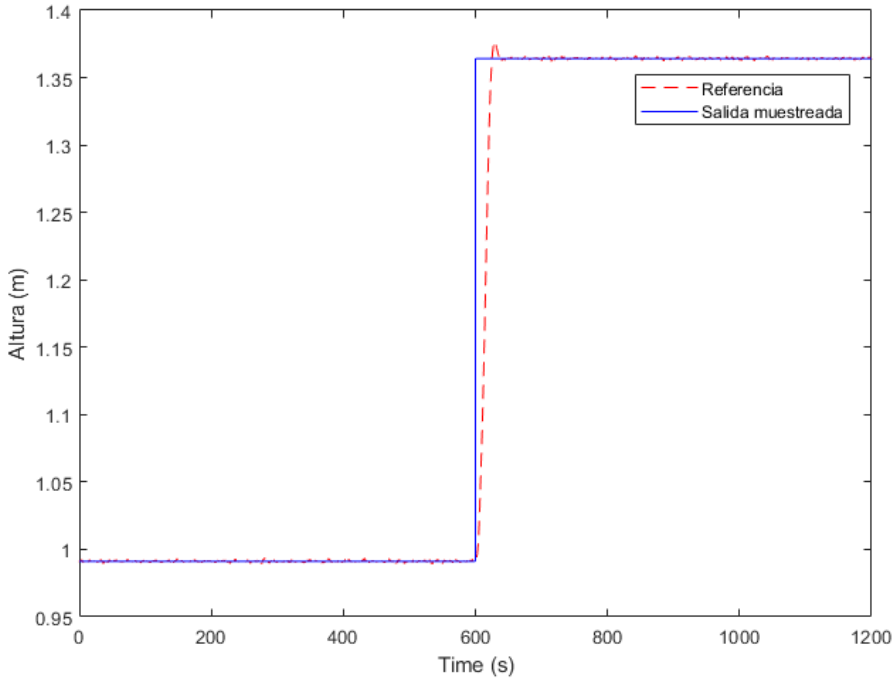

Figura 7.5 Referencia vs Salida muestreada, Controlador Data Driven.

7.4 Comparativa final, Truxal vs PID. Concluciones.

Finalmente se va a realizar una comparativa entre los resultados obtenidos en el funcionamiento del controlador *Data Driven* ante las distintas formas de implementar la base de datos. El parámetros que utilizaremos de baremo será el error absoluto entre la referencia y la salida final controlada. La primera forma de implementar la base de datos se ha desarrollado durante todo este capítulo y se basa en la implementación de un controlador Truxal. En la segunda partiremos del modelo de dos tanques de agua concatenados definido en en la *Figura 6.3* y generaremos la base de datos a partir de un controlador tipo PI. Definiremos dos límites (superior

e inferior) para ambas constantes integrales y se iterará tantas veces como trayectorias tenga la base de datos final. Ambas bases de datos se diseñaran con los mismos valores de referencia y ruido para hacer la comparación lo más equitativa posible. Al controlador final también se le impondrá que siga la misma referencia para ambos casos de simulación. Los límites impuestos a las constantes integral y proporcional del controlador PI son los siguientes:

$$P_{min} = 0.1$$
$$P_{max} = 2.5$$
$$I_{min} = 1.5$$
$$I_{max} = 10$$

Ambas bases de datos contarán con el mismo número de trayectorias, 1500. Además, respecto al controlador Truxal. La función de transferencia en bucle cerrado es idéntica a la que ecuación (*7.4*). Para ambas simulaciones los límites impuestos a las referencias y al ruido son los siguientes:

$$h2_{min} = 0.5$$
$$h2_{max} = 1.5$$
$$Ve_{min} = 0.01$$
$$Ve_{max} = 0.03$$

Se va adjuntar una tabla como la que se desarrollo en la sección *5.7 Comparativa Control PI vs Controlador Truxal-Ragazzini* en la que se van a adjuntar los errores obtenidos para tres casos de simulación distintos.

Tabla 7.2 Comparativa error de control, Base de datos Truxal vs PID.

	Error Simulación 1	Error Simulación 2	Error Simulación 3
Base de datos PID	25.6984	48.4562	41.4585
Base de datos Truxal	12.8564	9.6584	30.2687

De la tabla previa se puede extraer dos conclusiones finales. La primera observable es que los resultados obtenidos son claramente mejores en el caso de la base de datos generada mediante el controlador Truxal. Para este TFG solo se han añadido los tres casos previos, pero el proceso de comparación se ha realizado numerosas veces a lo largo del proceso de implementación, por lo que se puede concluir que los resultados finales no están sesgados. En segundo lugar si comparamos estos resultados con los de la *Tabla 5.1* se puede concluir que el peor o mejor rendimiento del controlador Truxal frente al PI recae principalmente en el proceso de identificación. Mientras mejor se realice el mismo y menos desviación tengan los parámetros identificados, al tener a su vez la misma función de transferencia en bucle cerrado esto da lugar a una agrupación de las trayectorias finales entre los mismo puntos de operación, evitando una desviación entre las variables almacenadas en la base de datos. Por tanto, concluir que para un proceso de identificación bien implementado, se consiguen mejores resultados a través de la generación de la base de datos mediante un controlador Truxal frente a uno PID.

Índice de Figuras

1.1 Diagrama de bloques sistema de control 1
1.2 Diagrama de bloques sistema de control de lazo abierto 2
1.3 Diagrama de bloques sistema de control de lazo cerrado 2
1.4 Respuesta de un sistema continuo ante una entrada escalón en tres casos distintos 4
1.5 Diagrama de bloques control PID 5
1.6 Sistema de dos tanques de agua concatenados 6

2.1 Diagrama de bloques del modelo a identificar, mínimos cuadrados 13
2.2 Salida real vs Salida aproximada del sistema 19
2.3 Error absoluto proceso de identificación 20
2.4 Diagrama de bloques del modelo a identificar, sistema con ruido 23
2.5 Salida real vs Salida aproximada, sistema con ruido 27
2.6 Error absoluto proceso de identificación, sistema con ruido 27

3.1 Diagrama de bloques control sistema discreto en bucle cerrado 29
3.2 Diagrama de Bloques controlador Truxal 34
3.3 Modelo de bloques en Simulink, sistema en lazo cerrado sin perturbaciones. Controlador Truxal-Ragazzini 36
3.4 Bloque Simulink, Transfer Function Space State Representation 37
3.5 Respuesta del sistema en bucle cerrado en ausencia de perturbaciones. Controlador Truxal-Ragazzini 38
3.6 Señal de control del sistema en bucle cerrado en ausencia de perturbaciones. Controlador Truxal-Ragazzini 38
3.7 Modelo de bloque en Simulink, sistema en lazo cerrado con perturbaciones. Controlador Truxal-Ragazzini 39
3.8 Bloque Simulink, Transfer Function Space State Representation With Noise 39
3.9 Bloque Random Number. Parámetro V(k) 40
3.10 Respuesta del sistema en bucle cerrado con perturbaciones. Controlador Truxal-Ragazzini 41
3.11 Señal de control del sistema en bucle cerrado con perturbaciones. Controlador Truxal-Ragazzini 41

4.1 Referencia vs Salida muestreada, trayectoria 982 46
4.2 Referencia vs Salida muestreada, trayectoria 650 47
4.3 Estructura S_j que representa a la base de datos 48

5.1 Salida del sistema. Controlador Data Driven Offset Free 57
5.2 Diagrama de bloques Simulink. Control PI 62
5.3 Diagrama de bloques Simulink. Control Truxal 63
5.4 Salida vs Referencia. Base de datos PI 64
5.5 Salida vs Referencia. Base de datos Truxal 64
5.6 Controlador Data Driven. Base de datos PI 65

5.7 Controlador Data Driven. Base de datos Truxal 66

6.1 Modelo no lineal del primer tanque de agua. Bloque Simulink 69
6.2 Modelo no lineal del segundo tanque de agua. Bloque Simulink 69
6.3 Modelo no lineal sistema de dos tanques de agua concatenados. Simulink 70
6.4 Caudal de entrada del primer tanque. Simulación sistema dos tanque concatenados 72
6.5 Altura del primer tanque. Simulación sistema dos tanque concatenados 73
6.6 Altura del segundo tanque. Simulación sistema dos tanque concatenados 74

7.1 Salida real vs Salida estimada del sistema 77
7.2 Error absoluto del proceso de identificación 78
7.3 Referencia vs Salida muestreada, trayectoria 1191 80
7.4 Referencia vs Salida muestreada, trayectoria 749 80
7.5 Referencia vs Salida muestreada, Controlador Data Driven 82

Índice de Tablas

5.1 Error de identificación del modelo para las distintas formas del vector regresor 63

7.1 Error de identificación del modelo para las distintas formas del vector regresor 76
7.2 Comparativa error de control, Base de datos Truxal vs PID 83

Índice de Códigos

2.1 Inicialización del sistema a identificar, mínimos cuadrados recursivos 13
2.2 Algoritmo de identificación mediante mínimos cuadrados recursivos 14
2.3 Function Define Structs 15
2.4 Function Define LeastSquare Matrix 16
2.5 Function LeastSquare Algorithm 17
2.6 Function Fill Structs 17
2.7 Function Optimal 18
2.8 Function Filter 20
2.9 Function FunTransfer 21
2.10 Generic Inicialization Wnoise 23
2.11 Inincialización algoritmo mínimos cuadrados recursivos, sistema con ruido 24
2.12 Algoritmo de identificación, mínimos cuadrados recursivos, sistema con ruido 24
2.13 Function LeastSquare Algotithm WNoise 25
2.14 Function Define LeastSquare Vectors Noise 25

3.1 Function ControlTruxal 31
3.2 Cálculo de ceros inestables 31
3.3 Cálculo del grado del numerador y denominador de la función de transferencia en bucle cerrado 32
3.4 Cálculo de los ceros y polos de la función de transferencia en bucle cerrado 32
3.5 Cálculo de la función de transferencia del controlador Truxal-Ragazzini 33
3.6 Cálculo de la función de transferencia del controlador Truxal-Ragazzini aplicado a un sistema conocido 33
3.7 Paso de función de transferencia a espacio estados 34
3.8 Function StateSpace Transformation 34
3.9 Variables de simulación 36
3.10 Function DefinePeriod 36
3.11 Variables referentes a los ruidos V(k) y W(k) 40

4.1 Definición de la función de transferencia 43
4.2 Cálculo de la ley de control para la generación de la base de datos 43
4.3 Parámetros de simulación. Base da datos 44
4.4 Function Aleatory References 45
4.5 Implementación final Base de datos 45
4.6 Implementación final Base de datos 46

5.1 Inicialización problema de optimización final 54
5.2 Function Iniciar 54
5.3 Estructura de resultados finales 54
5.4 Implementación controlador Data Driven Offset Free 55
5.5 Function Matrix H Generation 55

5.6 Function Matrix b Generation 56
5.7 Function Summulti 56
5.8 Function Plants Params Variation 58
5.9 Generación de la Base de datos, PI vs Truxal 58
5.10 Controlador Data Driven Offset Free, PID vs Truxal 60

6.1 Inicialización variables/parámetros de simulación sistema de dos tanques concatenados 71

7.1 Algoritmo mínimos cuadrados recursivos, aplicación modelo de dos tanques de agua concatenados 75
7.2 Generación de la base de datos y controlador Truxal, sistema de dos tanques concatenados 78
7.3 Controlador Data Drive, sistema de dos tanques concatenados 81

Bibliografía

[1] J. R. Salvador, D.R. Ramirez, T. Alamo, D. Muñoz de la Peña: '*Offset free data driven control: Application to a process control trainer*', *IET Research Journal*, 2015, pp. 1-11.
[2] Stephen Boyd: '*Convex Optimization*', 2009.
[3] D. R. Ramirez, C. Bordóns: '*Análisis y control de sistemas en espacio de estado. Identificación de sistemas. Control Adaptativo*', 2005.
[4] S. Sotoca: '*EI problema de las condiciones íniciales en los algoritmos de estimación recursiva de modelos lineales*', 1993.
[5] T. Alamo: '*Representación en Espacio de Estados de Sistemas Lineales*', 2017.
[6] Luis Carlos Ríos, Nicolás Toro: '*ESTIMACIÓN DE PARÁMETROS EN MODELOS ARMA POR EL CRITERIO DE MÍNIMOS CUADRADOS*', 2006.
[7] J. R. Salvador, D.R. Ramirez, T. Alamo, D. Muñoz de la Peña: '*Data Driven Control: An Offset Free Approach*', *18th European Control Conference (ECC)*, 2019, 23-28.
[8] ANDRÉS GARCÍA, OSVALDO AGAMENONI, JOSÉ FIGGUEROA: '*TUTORIAL DE LINEALIZACIÓN*'.
[9] Manuel Vargas Villanueva: '*Tutorial de análisis y control de sistemas usando matlab*'.
[10] Bruno Vargas Tamani: '*Obtención de Modelos de Procesos Mediante Métodos de Identificación Recursiva*', 2007.
[11] Rafael Telles: '*MODELADO DE SISTEMAS LINEALES Y NO LINEALES*', 2015.
[12] Felipe Andrés Fernández: '*Modelado y Simulación de Sistemas de Control de Llenado de Estanques mediante Simulink*', 2015.
[13] J. R. Salvador, D.R. Ramirez, T. Alamo, D. Muñoz de la Peña: '*Reconciliación de Datos y Monitorización de Redes de Abastecimiento de Agua*'.
[14] D.R. Ramirez, T. Alamo: '*Diseño de controladores discretos*'.
[15] D.R. Ramirez, T. Alamo: '*Identificación mediante el método de los mínimos cuadrados recursivos*'.
[16] T.Alamo: '*Método de los Mínimos Cuadrados Recursivos*'.