DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA

AUTOR: ALUISA CHALÁ, PAUL SEBASTIÁN

TEMA: DESARROLLO DE UN SISTEMA DE IDENTIFICACIÓN DE PROCESOS INDUSTRIALES EN LÍNEA USANDO LA PLATAFORMA OPEN-SOURCE ARDUINO Y MATLAB/SIMULINK

DIRECTOR: ING. GORDILLO ORQUERA, RODOLFO

CO-DIRECTOR: ING. OROZCO BRITO, LUIS

SANGOLQUÍ, MAYO 2014

Certificado de Tutoría

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICADO

Ing. Rodolfo Gordillo

Ing. Luis Orozco

CERTIFICAN

Que el trabajo titulado "DESARROLLO DE UN SISTEMA DE IDENTIFICACIÓN DE PROCESOS INDUSTRIALES EN LÍNEA USANDO LA PLATAFORMA OPEN-SOURCE ARDUINO Y MATLAB/SIMULINK", realizado por Paúl Sebastián Aluisa Chalá, ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la institución, en el Reglamento de Estudiantes de la Universidad de las Fuerzas Armadas – ESPE.

Debido a que se trata de un trabajo de investigación se recomienda su publicación. El mencionado trabajo consta de un documento empastado y un disco compacto el cual contiene los archivos en formato portátil (pdf). Autorizan a Paul Sebastián Aluisa Chalá que lo entregue al Ingeniero Luis Orozco, en su calidad de Coordinador de la Carrera.

Sangolquí, 21 de Mayo de 2014

Ing. Rodolfo Gordillo	Ing. Luis Orozco
DIRECTOR	CO-DIRECTOR

Declaración de Responsabilidad

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

DECLARACIÓN DE RESPONSABILIDAD

PAÚL SEBASTIÁN ALUISA CHALÁ

DECLARO QUE:

El proyecto de grado denominado "DESARROLLO DE UN SISTEMA DE IDENTIFICACIÓN DE PROCESOS INDUSTRIALES EN LÍNEA USANDO LA PLATAFORMA OPEN-SOURCE ARDUINO Y MATLAB/SIMULINK", ha sido desarrollado en base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incluyen en la bibliografía.

Consecuentemente, este trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Sangolquí, 21 de Mayo de 2014

Paúl Sebastián Aluisa Chalá

Autorización de Publicación

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

Yo, Paúl Sebastián Aluisa Chalá,

Autorizo a la Universidad de las Fuerzas Armadas – ESPE la publicación, en la biblioteca virtual de la institución, del trabajo "DESARROLLO DE UN SISTEMA DE IDENTIFICACIÓN DE PROCESOS INDUSTRIALES EN LÍNEA USANDO LA PLATAFORMA OPEN-SOURCE ARDUINO Y MATLAB/SIMULINK", cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y autoría.

Sangolquí, 21 de Mayo de 2014

Paúl Sebastián Aluisa Chalá

DEDICATORIA

Este proyecto está dedicado a toda mi familia, por su constante apoyo a lo largo de mi carrera universitaria.

Paul Aluisa Ch.

AGRADECIMIENTO

En primer lugar gracias a mis padres Gustavo y Lourdes, que con su sacrificio y consejos han sido lo más importante en mi vida y son parte principal de este logro obtenido. A mi abuelita Angélica que ha sido un soporte fundamental. A mis hermanos Andrea y David, que siempre han sido un apoyo en cada momento de mi vida profesional y personal.

A mis amigos y compañeros universitarios que me han acompañado en los buenos y malos momentos de mi carrera universitaria. A los Ingenieros Rodolfo Gordillo y Luis Orozco, por permitirme realizar este proyecto bajo su tutela.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En el presente documento se detalla, la investigación, el diseño y la implementación realizados para desarrollar un sistema de identificación de procesos industriales en línea utilizando Arduino y Matlab/Simulink, en el mismo se muestra el desarrollo teórico del proceso de discretización de sistemas en tiempo continuo para obtener sistemas en tiempo discreto, además muestra algunas características de la identificación de modelos paramétricos a través de técnicas basadas en la minimización del error cuadrático. También se da una breve descripción del hardware usado para la identificación, compuesto principalmente por la tarjeta Arduino Mega 2560 como medio para la adquisición de datos. La segunda parte del proyecto es la del desarrollo de un sistema de emulación de procesos industriales a través de la resolución de las ecuaciones a diferencias de los sistemas discretizados, esta resolución se realiza dentro de un controlador lógico programable (PLC) y así genera las señales de excitación y respuesta de la planta, el sistema es monitoreado en una PC, dicho sistema servirá para probar el funcionamiento de los algoritmos de identificación desarrollados como son: mínimos cuadrados no recursivos, mínimos cuadrados recursivos y lattice, de primer y segundo orden en los 3 casos, en los capítulos siguientes se muestra el desarrollo de los algoritmos de identificación, la implementación en Simulink de dichos algoritmos para su conexión con la tarjeta arduino, la implementación del algoritmo de emulación de procesos industriales que va dentro del PLC, el desarrollo de una interfaz de monitoreo del sistema de emulación y los resultados obtenidos luego de las pruebas realizadas integrando ambos sistemas.

PALABRAS CLAVE: IDENTIFICACIÓN DE PROCESOS INDUSTRIALES, MÍNIMOS CUADRADOS, ARDUINO, MATLAB, PLC.

ABSTRACT

This document is detailed, research, design and implementation made to develop an identification system for industrial process online using Arduino and Matlab/Simulink, the same theoretical development of the quantization process time systems is shown for continuous systems in discrete time, also shows some of the identification of parametric models through based on square error minimization techniques. It also gives a brief description of the hardware used for identification, mainly composed of the Arduino Mega 2560 card as a means for data acquisition. The second part of the project is the development of an emulation system of industrial processes by solving the equations of the discretized systems differences, this resolution is made within a programmable logic controller (PLC) and thus generates the signals excitation and response of the plant, the system is monitored on a PC, the system will test the performance of identification algorithms developed and are least squares recursive, recursive least squares and lattice, first and second order in the 3 patients in the following chapters develop identification algorithms shown in Simulink implementation of these algorithms for connection to the arduino board, the emulation algorithm implementation of industrial processes that goes into the PLC, the development of monitoring interface emulation system and the results of the tests performed after integrating the two systems.

KEYWORDS: IDENTIFICATION OF INDUSTRIAL PROCESSES, LEAST SQUARES, ARDUINO, MATLAB, PLC.

CAPÍTULO 1

DESCRIPCIÓN DEL PROYECTO

1.1. ANTECEDENTES

A menudo los procesos industriales no son accesibles para someterlos a pruebas y hallar su comportamiento o función de transferencia; además, el proceso de sintonización, sin conocer la planta, puede perjudicar el correcto funcionamiento del sistema.

Dentro de este marco se establece como un antecedente importante el artículo de investigación (Tamani, 2007), en el cual se describe el desarrollo de los métodos de identificación paramétrica que permiten obtener el modelo matemático que más se aproxime al funcionamiento real de un sistema y como parte importante se analiza el método recursivo basado en el principio de los mínimos cuadrados.

De igual forma se puede mencionar un importante artículo de investigación (Meizoso López, Piñon Pazos, & Ferreiro Garcia) en el que se presenta como se pueden desarrollar modelos analógicos de plantas industriales y también el uso de Controladores Lógicos Programables (en adelante PLC's) para implementar los mismos modelos, previamente establecidos, y poder utilizarlos en la ampliación de

sus laboratorios de enseñanza como ayuda práctica para mejorar los conocimientos, la emulación de los sistemas físicos ayudará a realizar el análisis de desempeño de los algoritmos de identificación en línea utilizados.

Es necesario resaltar los orígenes del proyecto Arduino, el mismo que fue concebido en Italia en el año 2005 por el zaragozano David Cuartielles, ingeniero electrónico y docente de la Universidad de Mälmo (Suecia) y Massimo Banzi, italiano, diseñador y desarrollador Web.

"Arduino es una plataforma de hardware y software de código abierto, basada en una sencilla placa con entradas y salidas, analógicas y digitales, en un entorno de desarrollo que está basado en el lenguaje de programación *Processing"*. (Martín, 2006).

Esta plataforma servirá para la adquisición de datos y su procesamiento en Matlab/Simulink.

1.2. JUSTIFICACIÓN E IMPORTANCIA

El uso de modelos de los sistemas físicos permite estudiar un problema sin tener la necesidad de experimentar, permitiendo a su vez detectar fallas y errores en los diseños. Así, se pueden evitar los riesgos de experimentos, analizar comportamientos y resultados de forma segura, reducir costos de desarrollo, realizar análisis de fallas, etc.

El proyecto consiste en desarrollar un sistema de identificación de procesos industriales en línea utilizando la plataforma *open-source* Arduino para la adquisición de datos y Matlab/Simulink para su respectivo procesamiento y el desarrollo de las rutinas de identificación paramétrica basadas en algoritmos de mínimos cuadrados.

La importancia del uso de Arduino como plataforma de adquisición de datos para la identificación, radica en varios factores:

- Software y hardware libre.
- Alta disponibilidad en el mercado y diferentes versiones.
- Disponibilidad de librerías para su conexión a múltiples dispositivos.
- Disponibilidad de librerías de comunicaciones para diferentes protocolos y buses de campo.
- Amplio soporte con una comunidad muy extendida y dinámica.
- Muy bajo coste al no requerir ningún pago de licencias y/o royalties tanto en el hardware como en el software de desarrollo.

Se justifica el uso de Arduino debido a que posee una librería completa para su uso en aplicaciones integradas con Matlab/Simulink, esto permitirá afianzar los conocimientos previamente adquiridos, llevándolos a aplicaciones prácticas con hardware potente de bajo costo.

El uso del PLC como emulador de procesos reales encuentra su justificación en que hoy en día la mayoría de ellos soportan lenguajes de programación de alto nivel (texto estructurado), lo cual permite programar fácilmente algoritmos complejos o implementar las ecuaciones a diferencias que rigen un determinado modelo físico.

Además la facilidad de conexión de los PLC’s con un computador permite desarrollar una interfaz gráfica que sirve como marco perfecto para mostrar la evolución del proceso de control, las plantas emuladas ayudarán al análisis de desempeño de los algoritmos de identificación paramétrica desarrollados.

1.3. ALCANCE DEL PROYECTO

En el presente proyecto se desea desarrollar un sistema completo de identificación en línea para obtener un modelo matemático de procesos industriales utilizando arduino y Matlab/Simulink, además se desarrollará un sistema de emulación de plantas industriales, lo que va a permitir realizar las pruebas necesarias para el análisis de los resultados y del desempeño de los algoritmos de identificación paramétrica basados en mínimos cuadrados.

Arduino posee una librería completa para su integración con Matlab/Simulink, lo que va a facilitar la conexión a un computador y un mejor desarrollo de las rutinas de identificación de plantas.

La programación en el lenguaje de texto estructurado que se realizará en el PLC va a permitir, de una forma sencilla, la programación de las ecuaciones a diferencias que representan los modelos de los sistemas físicos a emular, haciendo del PLC, la representación física de una planta industrial, en el cual se genera la señal de excitación de la planta y su respectiva respuesta.

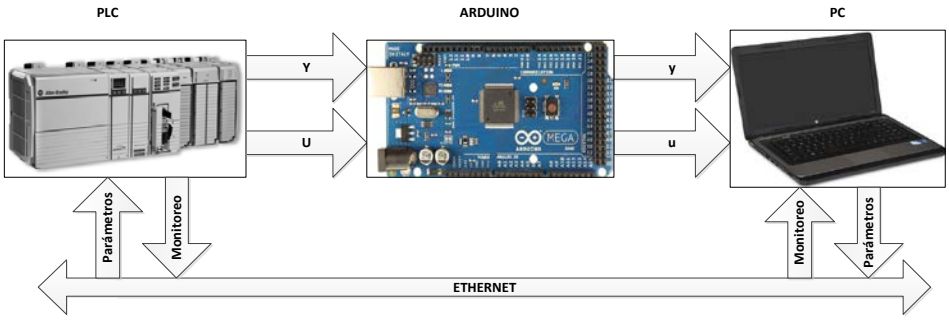

Figura 1.1. Diagrama de bloques del proyecto

La Figura 1.1 Muestra un diagrama de bloques del proyecto el cual se puede dividir en dos etapas: la primera es la de monitoreo y configuración del sistema de emulación de procesos, en esta etapa se muestra una conexión directa entre la PC y el PLC, en la PC se ingresarán los parámetros del modelo a emular y se podrá monitorear la evolución de la salida del sistema, la segunda etapa es la de identificación en línea, una vez ingresados los parámetros al PLC este genera las señales Y (salida del sistema) y U (entrada de excitación al sistema), las mismas que pasan a través de la tarjeta arduino como medio de adquisición de datos hacia la PC, en la cual se utilizan las señales adquiridas, como entradas para los algoritmos de identificación que se va a desarrollar.

Con el desarrollo de este proyecto se espera impulsar el uso de la plataforma Arduino como herramienta educativa, teniendo en cuenta que es un sistema *open-source*, lo que facilita la investigación y el desarrollo de aplicaciones con esta plataforma.

1.4. OBJETIVOS

1.4.1. GENERAL

Utilizar la plataforma *open-source* arduino para desarrollar un sistema de identificación de procesos industriales en línea en conjunto con Matlab/Simulink.

1.4.2. ESPECÍFICOS

- Desarrollar modelos de identificación paramétrica de procesos industriales en línea basados en mínimos cuadrados usando Matlab/Simulink.
- Entender la importancia del uso de prototipos *open-source* para el desarrollo de aplicaciones.
- Desarrollar programación en lenguaje de texto estructurado dentro del PLC para implementar las ecuaciones a diferencias que representan los tipos de modelos de sistemas físicos.
- Integrar la representación de modelos físicos dentro del PLC con los algoritmos de identificación para determinar el grado de funcionalidad de cada algoritmo.

CAPÍTULO 2

FUNDAMENTO TEÓRICO

2.1. INTRODUCCIÓN

En el presente capítulo se hará una exposición detallada de los conceptos necesarios para el desarrollo del proyecto, el cual ha sido descrito en el capítulo anterior. La elaboración de un completo marco teórico permitirá una mejor compresión de las herramientas utilizadas para el diseño e implementación de los algoritmos de identificación y para conocer a fondo el hardware que se utilizara para el análisis de los resultados obtenidos de la implementación de los algoritmos a realizar.

El capítulo está organizado de la siguiente forma: en el Epígrafe 2.2 se describe las formas de representar el modelo de un proceso industrial en tiempo discreto y como se pasa de un modelo de tiempo continuo a uno de tiempo discreto, esto permitirá la emulación de procesos industriales dentro del PLC en tiempo discreto.

En el Epígrafe 2.3 se describen los métodos usados a menudo para la identificación de sistemas dinámicos y se esbozan algunas ideas básicas necesarias para desarrollar un sistema de identificación de procesos.

En el Epígrafe 2.4 se explica detalladamente la plataforma arduino, lo que permite conocer a fondo la tarjeta Arduino Mega 2560 y su lenguaje de programación, para una fácil comprensión del hardware de adquisición de datos a utilizar.

Y por último, en el Epígrafe 2.5 se trata tres puntos, la descripción del PLC que se utilizará para la emulación de procesos industriales, una comparación entre los lenguajes de programación de PLC's, y la forma de programar un PLC en texto estructurado.

2.2. MODELOS DE PROCESOS INDUSTRIALES EN TIEMPO DISCRETO

2.2.1. REPRESENTACIÓN DE SISTEMAS EN TIEMPO DISCRETO

Una señal en tiempo discreto puede representarse como una secuencia de números $\{x[0], x[1], \dots x[N-1]\}$ obtenida a partir del muestreo de una señal definida en tiempo continuo $x_a(t)$. Así, una señal en tiempo discreto o secuencia, es $\{x[n]\}$, donde $x[n]$ es el valor de la señal en el tiempo discreto n.

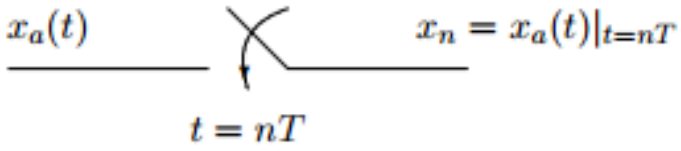

Figura 2.1. Muestreador

La Figura 2.1 representa el proceso de muestreo como un circuito de conmutación, el cual adquiere el valor de la señal continua $x_a(t)$ a intervalos uniformes de tiempo $t = nT$, donde T es el período de muestreo y n es cualquier número entero que también puede ser representado por k. (Miramontes, 2005) Entonces la señal en tiempo discreto puede escribirse de la siguiente manera:

$$x_n = x[n] = x[k] = x_a(t)|_{t=kT} = x_a(kT)$$

Ecuaciones a Diferencias

Los diferentes tipos de sistemas o procesos industriales, muchos de los cuales podemos considerar lineales y en donde los datos que relacionan la entrada con la salida aparecen en forma natural como muestras. Estos sistemas se denominan sistemas de tiempo discreto y se pueden modelar por medio de ecuaciones a diferencias, de la misma manera que los sistemas de tiempo continuo lineales e invariantes con el tiempo se modelan por ecuaciones diferenciales. Las ecuaciones a diferencias que relacionan las secuencias de las señales de entrada y de salida de un sistema discreto muestran las propiedades de los mismos tales como linealidad, invariancia con el tiempo y causalidad, entre otras. Un sistema descrito por una ecuación a diferencias es dinámico porque el valor de la secuencia de la salida, $y[k]$

depende no sólo de la secuencia de la entrada $u[k], u[k-1], u[k-2], \ldots$ sino también de los valores de $y[k-1], y[k-2], \ldots$ (Esquivel)

Por lo tanto el modelo de un sistema discreto puede representarse por su ecuación a diferencias:

$$y(k) + a_n y(k-n) + \cdots a_2 y(k-2) + a_1 y(k-1) = b_m u(k-m) + \cdots + b_1 u(k-m) + b_0 u(k)$$

Donde se puede observar que las muestras para distintos instantes de muestreo desempeñan en la ecuación a diferencias el mismo papel que las derivadas en las ecuaciones diferenciales de los sistemas continuos.

En general la representación de un sistema por su ecuación a diferencias queda de la siguiente manera:

$$y(k) + \sum_{i=1}^{N} a_i y(k-i) = \sum_{j=0}^{M} b_j u(k-j)$$

Función de Transferencia

Para los sistemas en tiempo continuo la función de transferencia $H(s)$se define como la razón de la transformada de Laplace de la salida a la transformada de Laplace de la entrada $H(s) = Y(s)/U(s)$, por lo que se obtiene que:

$$\frac{Y(s)}{U(s)} = \frac{b_m s^m + \cdots + b_1 s + b_0}{a_n s^n + \cdots + a_1 s + a_0}$$

De igual manera para los sistemas en tiempo discreto, la función de transferencia se define como la razón entre la transformada Z de la salida a la transformada Z de la entrada $H(z) = Y(z)/U(z)$, teniendo en cuenta que:

$$y(k-n) \quad \rightarrow \quad z^{-n}Y(z)$$

Podemos describir a la función de transferencia como:

$$\frac{Y(z)}{U(z)} = \frac{b_m z^{-m} + \cdots + b_1 z^{-1} + b_0}{a_n z^{-n} + \cdots + a_1 z^{-1} + 1}$$

En general la representación de un sistema por su función de transferencia en tiempo discreto queda de la siguiente manera (Vega, 2004):

$$\frac{Y(z)}{U(z)} = \frac{\sum_{j=0}^{M} b_j z^{-j}}{1 + \sum_{i=1}^{N} a_i z^{-i}}$$

2.2.2. DISCRETIZACIÓN DE SISTEMAS

Existen distintos procedimientos para obtener sistemas en tiempo discreto que se comporten aproximadamente igual a un sistema en tiempo continuo dado. Esta operación suele denominarse discretización.

Las diferentes aplicaciones de la discretización son:

- Simular con un computador un sistema en tiempo continuo.
- El diseño de un filtro digital basado en un diseño analógico anterior.
- El diseño de un regulador digital basado en un diseño analógico.

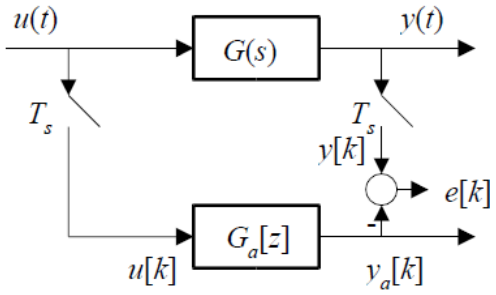

Figura 2.2. Diagrama de bloques de discretización de un sistema de tiempo continuo

La idea de discretizar un sistema es obtener $G_a[z]$ tal que se comporte aproximadamente como $G(s)$, en el sentido de que si se aplican a $G_a[z]$, muestras $u[k]$ de la entrada $u(t)$ aplicada a $G(s)$, resulte que su salida $y_a[k]$ coincida lo suficiente con las muestras de $y(t)$ como se muestra en la Figura 2.2.

Una condición importante para que la discretización de un sistema en tiempo continuo resulte factible es que $G_a[z]$ sea racional, para que el resultado pueda ser traducido en un algoritmo. Para aplicaciones de tiempo real en las que se vayan a aplicar estos algoritmos, normalmente se pide que $G_a[z]$ no exija mucho tiempo de cálculo.

Existen tres técnicas fundamentales para la discretización las cuales se muestran a continuación:

- **Transformaciones**

Consisten en sustituir en G(s) la variable s por una función racional en z. Son sencillas y flexibles de aplicar, en casi cualquier situación. Pueden justificarse como

una aproximación a la derivación, como una aproximación a la integración o como una aproximación racional de $z = e^{Ts}$.

Aproximación	Transformación $s =$	Polos y ceros en: $z =$
Euler (Diferencia en adelanto)	$\frac{z-1}{T}$	$1 + Ts$
Rectangular (Diferencia en atraso)	$\frac{z-1}{Tz}$	$\frac{1}{1-Ts}$
Trapezoidal (Transformada Tustin o bilineal)	$\frac{2(z-1)}{T(z+1)}$	$\frac{2+Ts}{2-Ts}$

Tabla 2.1. Aproximaciones para pasar del plano S al plano Z o viceversa

La Tabla 2.1 muestra una lista de trasformaciones directas entre el plano s y el plano z, teniendo en cuenta que la aproximación mejora de acuerdo a la complejidad de la misma.

- **Simulaciones invariantes**

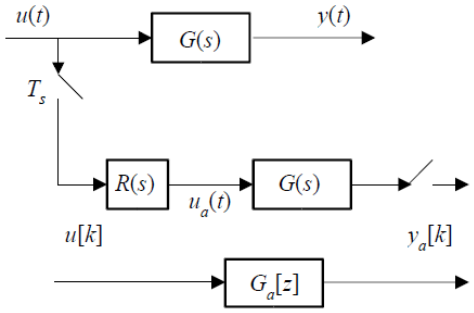

Figura 2.3. Diagrama de bloques de un sistema discretizado con retenedor de datos

Están basadas en la idea de reconstruir $u(t)$ con un retenedor, de lo que resulta una simulación invariante para aquellas formas de $u(t)$ las cuales el retenedor puede

reconstruir exactamente obteniendo del retenedor una señal de entrada u[k] para el sistema discretizado como se puede ver en la Figura 2.3.

Los retenedores van mejorando según el orden de retención que tengan, permitiendo pasar de tiempo discreto a tiempo continuo con la transformada Z directa o inversa, según el caso lo amerite; de la función de trasferencia de cada retenedor como se observa en la Tabla 2.2.

Retenedor	$G_a[z]$
Solamente tiempo de muestreo	$T\ Z\{G(s)\}$
Retenedor de orden 0	$\frac{z-1}{z} Z\left\{\frac{1}{s} G(s)\right\}$
Retenedor triangular	$\frac{(z-1)^2}{Tz} Z\left\{\frac{1}{s^2} G(s)\right\}$

Tabla 2.2. Retenedores de datos

- **Transformación de polos y ceros**

La idea es transformar los ceros y los polos del plano S al plano Z, aunque no hay base matemática que lo justifique estrictamente. Por ejemplo: Si a es un polo o cero en s, b es el polo o cero correspondiente en z usando la siguiente transformación (Pagola, 2002):

$$b = e^{aT}$$

Retenedor de Orden Cero (ZOH)

Al realizar el muestreo de una señal se convierte una señal de tiempo continuo en un tren de pulsos que se representa en los instantes de muestreo $t = 0, T, 2T, \ldots$, donde T es el período de muestreo.

La Figura 2.4 muestra el proceso de retención de datos el cual genera una señal de tiempo continuo a partir de una secuencia de tiempo discreto, que reproduce aproximadamente la señal aplicada al muestreador.

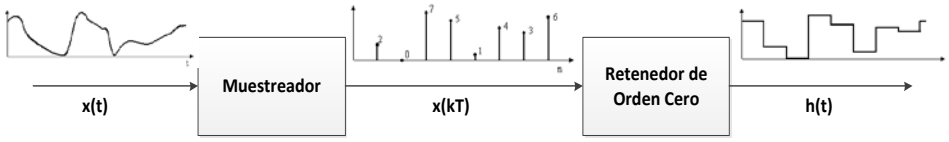

Figura 2.4. Retenedor de orden cero

La señal de entrada se muestrea en instantes discretos, esta se pasa a través del retenedor de orden cero. El circuito del retenedor suaviza la señal muestreada para producir la señal $h(t)$, la cual es constante desde el último valor muestreado hasta que se dispone de la siguiente muestra. (Ogata, Sistemas de Control en Tiempo Discreto Segunda Edición , 1996)

La función de transferencia del retenedor de orden cero está dada mediante:

$$R(s) = \frac{1 - e^{-sT}}{s}$$

Para obtener el equivalente discreto de un sistema continuo se aplica la transformada Z al conjunto Retenedor- Planta de la siguiente manera:

$$G_a[z] = Z\left\{\frac{1 - e^{-sT}}{s} G(s)\right\}$$

$$G_a[z] = (1 - e^{-sT})|_{e^{sT}=z}\ Z\left\{\frac{G(s)}{s}\right\}$$

De donde se obtiene que el equivalente discreto de un sistema continuo utilizando un retenedor de orden cero viene dado por:

$$G_a[z] = (1 - z^{-1})\ Z\left\{\frac{G(s)}{s}\right\} \tag{2.1}$$

O

$$G_a[z] = \frac{z-1}{z} Z\left\{\frac{1}{s} G(s)\right\}$$

La discretización se utilizará para realizar la emulación de plantas continuas, debido a que el PLC se debe utilizar un modelo discreto para realizar la programación.

2.3. IDENTIFICACIÓN DE SISTEMAS

La identificación de un sistema consiste en la obtención de un modelo matemático que caracteriza la dinámica de la planta y con ello se puede predecir su comportamiento. Así se podrán elaborar modelos matemáticos con la suficiente exactitud que permitan la aplicación de técnicas y algoritmos de control conocidos, con un grado adecuado de sintonía de sus parámetros y con un rendimiento superior en las instalaciones donde estos algoritmos hayan sido implementados.

La obtención de un modelo matemático de un proceso o planta se puede realizar de dos maneras:

- **Fuera de línea**

Permite la obtención de un modelo matemático basado en leyes de primeros principios o, indirectamente mediante el tratamiento de datos previamente obtenidos a través de experimentación con la planta o proceso. Ya sea paramétrica o no, este tipo de identificación produce como resultado un modelo susceptible de ser aplicado para el diseño de un controlador óptimo de parámetros fijos, aunque para el reajuste es necesario adquirir nuevos datos de la planta para procesarlos.

- **En línea**

Permite la obtención de un modelo matemático basado en técnicas de identificación paramétrica mediante la actualización, es decir se actualizan los parámetros calculados cada tiempo de muestreo de un modelo cuya estructura ha sido definida previamente. Este método permite que se puedan aplicar métodos de control adaptativos.

2.3.1. MÉTODOS DE IDENTIFICACIÓN

La identificación de procesos “es el conjunto de teorías, estudios y algoritmos que permiten obtener la estructura y los parámetros de un modelo matemático (generalmente dinámico) que reproduce, con suficiente exactitud para los fines de control automático, las variables de salida del proceso o sistema real objeto de estudio ante el mismo conjunto de variables de entrada” (Aguado Behar & Martínez Iranzo, 2003).

Al tener sistemas con comportamiento conocido, es decir con ecuaciones físicas que ligan a sus variables y representan su comportamiento, es posible obtener los modelos matemáticos, llegando de este modo a los conocidos modelos de primeros principios. En la mayoría de los casos estas ecuaciones no son deducibles a través de un comportamiento físico, por lo que la identificación exige realizar experimentación sobre el sistema que permita la determinación indirecta de las ecuaciones que representaran la estructura y los parámetros del modelo. A este tipo de modelos se los conoce como modelos de “caja negra”.

- **Identificación analítica o de primeros principios**

Consiste en desarrollar un modelo basado en relaciones físico-químicas del proceso a identificar, planteando ecuaciones cinemáticas, dinámicas, etc., lo que conduce a modelos generalmente complejos y no-lineales. El inconveniente principal es que se requiere tener un conocimiento muy especializado sobre la tecnología del proceso la cual puede que no esté siempre disponible.

- **Identificación experimental mediante señales especiales**

También denominada como identificación clásica, resulta ser el método más directo y el que permite obtener el modelo de un proceso a más corto plazo. Las señales que se utilizan frecuentemente son los escalones y las secuencias binarias pseudo-aleatorias.

El principal inconveniente de esta solución es la necesidad de introducir señales de prueba que perturban de manera indeseable al proceso y una limitación muy

importante es que, como resultado se obtiene modelos determinísticos, sin considerar los posibles modelos de perturbación.

- **Identificación paramétrica**

La identificación paramétrica está basada en los métodos de minimización del error de predicción, que se obtiene de la diferencia entre la salida real de la planta y el modelo paramétrico estimado, como se observa en la Figura 2.5; derivados de la teoría de mínimos cuadrados. Esta teoría se basa en asumir que el proceso puede ser representado por un modelo de estructura fija, generalmente una ecuación lineal a diferencias, lo que quiere decir que dicho modelo tiene naturaleza discreta.

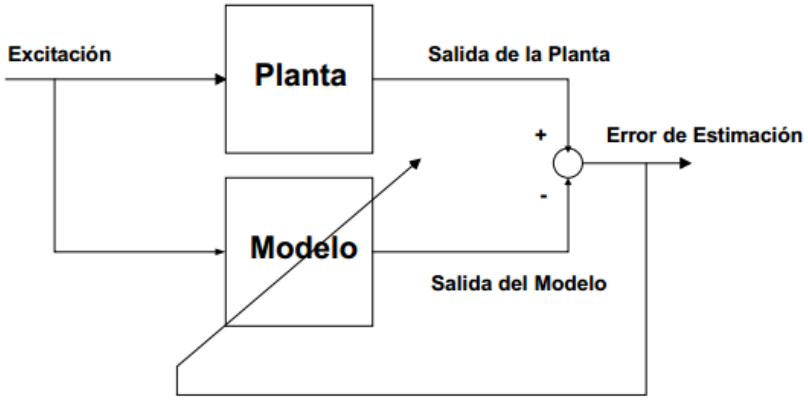

Figura 2.5. Diagrama de bloques de un sistema de identificación paramétrica

Este algoritmo puede aplicarse fuera de línea (no recursiva), a través del uso de toda la información de entrada y salida del proceso para determinar un modelo invariante en el tiempo. También puede aplicarse de forma recursiva, de manera que partiendo de una estimación inicial de los parámetros del modelo se va actualizando

y mejorando con cada nueva información de entrada y salida obtenida, lo que permite obtener un modelo con parámetros variables con el tiempo.

A diferencia de los métodos de identificación clásica, los de identificación paramétrica permiten caracterizar de mejor manera los modelos, ya que contiene modelos de perturbación, lo que los hace aplicables para estructuras de control sofisticadas. (Aguado Behar & Martínez Iranzo, 2003)

2.3.2. IDEAS BÁSICAS SOBRE IDENTIFICACIÓN DE SISTEMAS

El modelo de un sistema es una forma de representar el conocimiento que se tiene sobre su dinámica, por lo que resulta una herramienta importante en el diseño y análisis de sistemas de control.

La identificación consiste en obtener un modelo a partir de observaciones obtenidas del propio sistema que se pretende modelar. La identificación del sistema requiere de varios procedimientos que se deben realizar para que el modelo obtenido reproduzca de la mejor manera la dinámica del proceso:

Planificación de los Experimentos

Debido a la necesidad de experimentar con el proceso real para poder obtener un modelo adecuado del mismo, es necesario tener en cuenta que dicha experimentación puede resultar perjudicial para el proceso. Por esta razón, es necesario elegir una

técnica que sea lo menos invasiva para el proceso desde el punto de vista del tipo de experimentos necesarios.

Algunas técnicas son muy sencillas, pero estas técnicas, requieren que en los experimentos se utilicen señales de entradas preestablecidas de manera muy precisa: pulsos, sinusoides, etc. Y puede darse el caso en que el proceso a modelar no pueda ser sometido a este tipo de entradas por consideraciones de seguridad o motivos económicos.

Otras técnicas de identificación pueden emplear casi cualquier tipo de señal de entrada, pero una vez realizado el experimento es más complicado obtener el modelo. Resulta necesario que en el experimento se utilicen señales de entrada que exciten toda la dinámica o modos del sistema.

Selección del Tipo de Modelo

La selección del tipo de modelo se realiza en base al conocimiento que se tiene del proceso y de las perturbaciones que este pueda llegar a tener. Dependiendo del nivel de conocimiento que poseemos de la estructura del proceso se podrá elegir entre uno u otro modelo descritos a continuación de forma generalizada:

- **Modelos de caja blanca**

Son los obtenidos a partir de las leyes físicas o químicas que rigen al proceso.

- **Modelos de caja negra**

En estos modelos se propone una estructura matemática con una serie de parámetros libres, a los cuales se les da valor a partir de los datos obtenidos en los experimentos.

- **Modelos de caja gris**

Corresponden a un tipo intermedio entre los modelos de caja negra y de caja blanca. Parte del modelo se obtiene mediante leyes físicas y la parte restante se ajusta usando métodos experimentales.

Otra forma de clasificación de los tipos de modelos, puede ser en paramétricos y no paramétricos. En los primeros se tienen una serie de parámetros que hay que estimar, en cambio en los modelos no paramétricos, el modelo no tiene una serie de parámetros que definan la dinámica sino que se compone de una cantidad de información sobre el modelo en sí, por ejemplo los modelos basados en la respuesta en frecuencia de un sistema.

Elección de un Criterio

Al momento de realizar la estimación del modelo es necesario contar con un criterio que exprese las características de ajuste del modelo a los datos. Usualmente, el proceso de ajuste del modelo se realiza de manera que se busca el valor de los parámetros que hacen mínimo al índice o criterio. El método más antiguo que emplea esta estrategia es el de los mínimos cuadrados.

Estimación de los Parámetros

Para realizar la estimación de los parámetros del modelo se requiere de los datos experimentales, un tipo de modelo y un criterio. Estimar los parámetros es resolver un problema de optimización en el cual, el mejor modelo es el que hace mínimo el criterio.

Validación del Modelo

La validación del modelo consiste en comprobar las características del modelo que se ha obtenido como resultado del proceso de identificación. Una técnica muy común para comprobar la bondad de un modelo identificado es la validación cruzada. Tampoco puede descartarse la posibilidad de no usar criterio de validación alguno y efectuar una inspección visual sobre una simulación, en la que se usa el modelo estimado para predecir la salida en base a datos de entradas experimentales.

2.3.3. TIPOS DE MODELOS PARAMÉTRICOS

Generalmente los modelos paramétricos se describen en modelo discreto debido a que los datos que sirven para la identificación se obtienen por muestreo, en el caso que se necesite un modelo continuo, es posible realizar una transformación del dominio discreto al continuo.

Los modelos paramétricos pueden ser generados cuando a un sistema lineal se le inyecta ruido blanco $v(k)$ además de una entrada externa $u(k)$, para generar la respuesta de dicho sistema.

- **Modelo autorregresivo (AR)**

Los modelos autorregresivos se pueden describir como aquellos modelos en los que una variable o conjunto de variables se explican en función de los valores pasados de esa misma variable o conjunto de variables.

En general, se denotan por AR (p). En un modelo AR (p) un valor en el momento k de la serie se expresa como una combinación lineal de las p observaciones anteriores de la serie:

$$y(k) + d_1 y(k-1) + d_2 y(k-2) + \cdots + d_n y(k-n) = v(k)$$

$$D(z^{-1})y(k) = v(k)$$

El término aleatorio $v(k)$ correspondiente a la perturbación tiene una estructura muy simple porque no depende de los valores pasados.

- **Modelo de media móvil (MA)**

En los modelos MA (q), el valor de la serie en el momento k se expresa como una combinación de innovaciones. Es el caso más sencillo y viene descrito por:

$$y(k) = v(k) + c_1 v(k-1) + c_2 v(k-2) + \cdots + c_n v(k-n)$$

$$y(k) = C(z^{-1})v(k)$$

Con este modelo se pueden describir muchos tipos de perturbaciones aleatorias. Sin embargo, no incluye a los valores pasados de la salida por lo que no sirve para modelar procesos que tengan dinámica.

- **Modelo autorregresivo de media móvil (ARMA)**

Es la combinación de los dos anteriores, por lo que toma la forma:

$$y(k) + d_1 y(k-1) + \cdots + d_n y(k-n) = v(k) + c_1 v(k-1) + \cdots + c_n v(k-n)$$

$$D(z^{-1})y(k) = C(z^{-1})v(k)$$

Este modelo permite describir procesos más completos. Sin embargo, desde el punto de vista del control es interesante poder considerar el efecto de una entrada externa.

- **Modelo autorregresivo de media móvil con una entrada exógena (ARMAX)**

También llamado modelo CARMA (Controlled ARMA). Viene descrito por:

$$\begin{aligned} y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) \\ = b_1 u(k-1) + \cdots + b_n u(k-n) + v(k) + c_1 v(k-1) + \cdots \\ + c_n v(k-n) \end{aligned}$$

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

Esto da lugar a un sistema de ecuaciones donde las incógnitas son los coeficientes del modelo discreto, cuyas soluciones se obtienen por predicción del error.

- **Modelo autorregresivo con entrada exógena para mínimos cuadrados (ARX-LS)**

Este modelo surge como una versión simplificada del anterior, para el caso en el que no se necesita que la fuente de perturbaciones tenga una estructura tan compleja. Viene descrito por:

$$y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) = b_1 u(k-1) + \cdots + b_n u(k-n) + v(k)$$

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

Se utiliza en la identificación por el método de los mínimos cuadrados.

- **Modelo autorregresivo de media móvil integrada y con una entrada exógena (ARIMAX o CARIMA).**

Este modelo incorpora un integrador en la fuente de perturbaciones, por lo que viene descrito por:

$$\begin{aligned} y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) & \\ = b_1 u(k-1) + \cdots + b_n u(k-n) & \\ + \frac{v(k) + c_1 v(k-1) + \cdots + c_n v(k-n)}{\Delta} & \end{aligned}$$

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

Donde $\Delta = 1 - z^{-1}$. Este tipo de modelos es útil en esquemas de control predictivo para formular leyes de control que incorporen un efecto integral, de manera que sean capaces de rechazar perturbaciones en escalón. (Rodríguez Ramírez & Bordóns Alba, 2005)

2.3.4. IDENTIFICACIÓN POR MÍNIMOS CUADRADOS

Es una técnica de análisis numérico que se encuentra dentro de la optimización matemática, en la cual, dados un conjunto de pares ordenados: variable independiente, variable dependiente, y una familia de funciones, se intenta encontrar la función que mejor se aproxime a los datos, de acuerdo con el criterio de mínimo error cuadrático.

La razón principal para el uso del método de mínimos cuadrados es que a parte de su visión intuitiva, posee una serie de propiedades estadísticas simples y posibilita la implementación de una forma recursiva lo suficientemente simple. El método de mínimos cuadrados, define una estrategia, bastante utilizada para obtener un buen ajuste de los parámetros a encontrar. Este método permite la identificación de parámetros en los modelos lineales.

Mínimos Cuadrados No Recursivos

Considérese el siguiente modelo paramétrico lineal monovariable:

$$y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) = b_1 u(k-1) + \cdots + b_n u(k-n) + e(n) \tag{2.1}$$

Obteniendo la transformada z obtenemos la siguiente función de transferencia:

$$G(z^{-1}) = \frac{b_1 z^{-1} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}}$$

La ecuación (2.1) se puede escribir como:

$$y(k) = \emptyset(k)\theta \tag{2.2}$$

Donde el vector:

$$\emptyset(k) = [-y(k-1) \ \dots \ -y(k-n) \ \ u(k-1) \dots \ \ u(k-n)] \tag{2.3}$$

Es llamado regresor y:

$$\theta = [a_1 \dots \ a_n \ \ b_1 \dots \ b_n]^T$$

Es el vector de parámetros.

Asumiendo un valor del vector parámetros $\hat{\theta}$, el error de estimación será:

$$e(k) = y(k) - \hat{y}(k) = y(k) - \emptyset(k)\theta$$

No se puede hallar directamente el vector de parámetros, debido a que, se necesitaría que el modelo de la planta corresponda exactamente al modelo de la ecuación (2.1), de este modo se tendría un sistema de ecuaciones compatible. Pero en la práctica el proceso no se puede describir a la perfección mediante un modelo del tipo de la ecuación (2.1) por lo que se tiene que el sistema de ecuaciones no tiene solución.

Para resolver este inconveniente se puede encontrar un valor del vector de parámetros que haga mínimo el error de estimación, de manera más precisa que haga mínima la suma de los cuadrados de los errores de estimación del conjunto de estimación. Esta es la estrategia utilizada en el método de mínimos cuadrados.

Se define la función de costo, la cual es la medida que se desea optimizar para encontrar los parámetros:

$$J = \frac{1}{n}\sum_{k=1}^{n} e^2(k) = \frac{1}{2}[e^2(1) + e^2(2) + \cdots + e^2(n)]$$

$$J = \frac{1}{n}[e(1)\ \ e(2)\ \ \ldots\ \ e(n)] \begin{bmatrix} e(1) \\ e(2) \\ \ldots \\ e(n) \end{bmatrix}$$

$$J = \frac{1}{n}\ E^T\ E \tag{2.4}$$

Donde E es el vector de error de estimación. Se puede reescribir la función de costo de la ecuación (2.4) reemplazando el vector de error de estimación:

$$J = \frac{1}{n}\ (Y - \emptyset\theta)^T(Y - \emptyset\theta)$$

Se obtendrá el mínimo valor de $J(\theta)$ cuando se cumpla que:

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

De donde se obtiene que el valor del vector de parámetros que hace mínima la función de costo:

$$\theta = (\emptyset^T\emptyset)^{-1}(\emptyset^T Y) \tag{2.5}$$

Se debe tomar en cuenta que para que el problema de identificación tenga solución la matriz $(\emptyset^{\mathrm{T}}\emptyset)^{-1}(\emptyset^{\mathrm{T}}\mathrm{Y})$ tiene que ser invertible. Dicha condición se verifica cuando la entrada cumple las condiciones de excitación persistente del sistema. Se deberá acudir por tanto a señales de entrada parecidas al ruido blanco.

Es posible obtener este tipo de señales de entrada a través de secuencias de valores pseudoaleatorios, en la práctica se recurre a secuencias pseudoaleatorias binarias (PRBS). Este tipo de señal tiene amplitud unitaria, el concepto de binario se refiere solamente a dos niveles de entrada distintos y la parte aleatoria quiere decir que el ancho y la frecuencia de los pulsos van a variar indistintamente.

Mínimos Cuadrados Recursivos

La ecuación (2.5) implica que se deba invertir una matriz que puede llegar a ser de alto orden, intentar realizar estos cálculos en línea puede resultar complejo para el hardware, por lo tanto este algoritmo está destinado para identificación fuera de línea.

Para la identificación mínimo cuadrática recursiva, que es la que se utiliza en línea, se realiza el siguiente procedimiento.

La estimación para un instante k usando las medidas obtenidas vendrá dada por:

$$\hat{\theta} = [\Phi^T \Phi]^{-1}[\Phi^T Y]$$

$$\hat{\theta}(k) = P(k)[\Phi^T(k)Y(k)]$$

$$\hat{\theta}(k) = P(k)[\Phi^T(k-1)Y(k-1) + \emptyset^T(k)y(k)] \quad (2.6)$$

Dónde:

$$P(k) = [\emptyset^T(k)\ \emptyset(k)]^{-1} = \left[\sum_{i=1}^{k} \emptyset^T(i)\emptyset(i)\right]^{-1}$$

Es llamada matriz de covarianza. Se puede comprobar que:

$$P^{-1}(k-1) = P^{-1}(k) - \emptyset^T(k)\,\emptyset(k)$$

Por otra parte también se puede obtener que:

$$\emptyset^T(k-1)Y(k-1) = P^{-1}(k-1)\hat{\theta}(k-1)$$

$$\emptyset^T(k-1)Y(k-1) = P^{-1}(k)\hat{\theta}(k-1) - \emptyset^T(k)\,\emptyset(k)\hat{\theta}(k-1)$$

Combinando las dos últimas expresiones con la ecuación (2.6) se obtiene:

$$\hat{\theta}(k) = \hat{\theta}(k-1) - P(k)\emptyset^T(k)\emptyset(k)\hat{\theta}(k-1) + P(k)\emptyset^T(k)y(k)$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P(k)\emptyset^T(k)\left[y(k) - \emptyset(k)\hat{\theta}(k-1)\right]$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)\left[y(k) - \emptyset(k)\hat{\theta}(k-1)\right] \qquad (2.7)$$

Por lo tanto $\hat{\theta}(k)$ se puede expresar de forma recursiva, es decir en función del valor del estimador en el instante anterior más un término corrector siendo K(k) una ganancia de adaptación. (Rodríguez Ramírez & Bordóns Alba, 2005)

2.4. DESCRIPCIÓN DE LA PLATAFORMA ARDUINO

Figura 2.6. Logotipo de arduino

Arduino (Figura 2.6) es una fusión de tres elementos fundamentales: el hardware, el software y una amplia comunidad de desarrolladores de prototipos electrónicos, ya sea a manera profesional, investigación o simple entretenimiento. Para sacar el máximo provecho de ella es necesario tener un conocimiento básico de los tres elementos. Es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar, basada en una sencilla placa con entradas y salidas, analógicas y digitales.

Esta placa puede interactuar con el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros dispositivos. El microcontrolador de la placa se programa usando el Arduino Programming Language (basado en Wiring) y el Arduino Development Environment (basado en Processing).

Las placas arduino se pueden ensamblar a mano o adquirir pre-ensambladas, el software se puede descargar gratuitamente. Los diseños de referencia del hardware

están disponibles bajo licencia open-source, por lo que existe la libertad de adaptar el hardware a cualquier tipo de necesidad. (Enríquez Herrador, 2009)

2.4.1. HARDWARE ARDUINO

Arduino es un Hardware con el cual se puede realizar implementaciones personalizadas de una manera sencilla. Las placas han sido diseñadas para ser fácilmente extensible al estar basadas en estándar y poderosos componentes. El Hardware de Arduino se basa en un microcontrolador AVR, en particular el ATmega8, ATmega168, ATmega328 y el ATmega1280, en la Tabla 2.3 se detallan los modelos de placas arduino disponibles en el mercado actual.

Modelo	Características
Arduino Uno	Microcontrolador ATmega328. 14 entradas / salidas digitales (6 para PWM). 6 entradas analógicas.
Arduino Leonardo	Microcontrolador ATmega32u4. 20 entradas / salidas digitales (7 para PWM). 12 entradas analógicas
Arduino Due	Microcontrolador Atmel SAM3X8E ARM Cortex-M3 CPU. Núcleo ARM de 32-bit. 54 entradas / salidas digitales (12 para PWM). 12 entradas analógicas. 4 UARTs.
Arduino Esplora	Microcontrolador Atmega32U4. Oscilador de 16 MHz Puerto USB capaz de actuar como un

	dispositivo de cliente USB, como un ratón o un teclado.
Arduino Mega 2560	Microcontrolador Atmega2560. 54 entradas / salidas digitales (14 para PWM). 16 entradas analógicas. 4 UARTs.
Arduino Mega ADK	Microcontrolador Atmega2560. Interfaz USB para conectar con los teléfonos basados en Android. 54 entradas / salidas digitales (14 para PWM). 16 entradas analógicas.
Arduino Ethernet	Microcontrolador ATmega328. 14 entradas / salidas digitales. 6 entradas analógicas. Conexión RJ45.
Arduino Mini	Microcontrolador ATmega168. 14 entradas / salidas digitales (6 para PWM). 8 entradas analógicas.
Arduino LilyPad	Es una placa electrónica diseñada para usarla sobre textiles. Microcontrolador ATMEGA168V o el ATmega328V.
Arduino LilyPadUSB	Microcontrolador ATmega32u4. 9 entradas / salidas digitales (4 para PWM). 4 entradas analógicas Conexión USB.
Arduino Micro	Microcontrolador ATmega32u4. 20 entradas / salidas digitales (7 para PWM). 12 entradas analógicas
Arduino Nano	Microcontrolador ATmega328 o ATmega168. Funciona con un cable mini-B USB en lugar de una normal.
Arduino Pro Mini	Microcontrolador ATmega168. 14 entradas / salidas digitales (6 para PWM). 6 entradas analógicas.
Arduino Pro	Microcontrolador ATmega168 o ATmega328. Versiones de 3,3 V / 8 MHz y 5 V / 16 MHz. 14 entradas / salidas digitales (6 para

	PWM). 6 entradas analógicas.
Arduino Fio	Microcontrolador ATmega328P. 14 entradas / salidas digitales (6 para PWM). 8 entradas analógicas. Zócalo XBee disponible en la parte inferior de la placa.

Tabla 2.3. Modelos disponibles de placas arduino

2.4.2. ARDUINO MEGA 2560

Visión General

El Arduino Mega 2560 es una placa electrónica basada en el microprocesador Atmega2560. Cuenta con 54 entradas/salidas digitales de los cuales 14 se pueden utilizar como salidas PWM, 16 entradas analógicas, 4 UARTs (puertos seriales asíncronos), un oscilador de 16 MHz, conexión USB, conector de alimentación, tal y como se muestra en la Figura 2.7. Contiene todo lo necesario para apoyar el microcontrolador, basta con conectarlo a un computador con un cable USB, a un adaptador de CA a CC o a una batería para empezar.

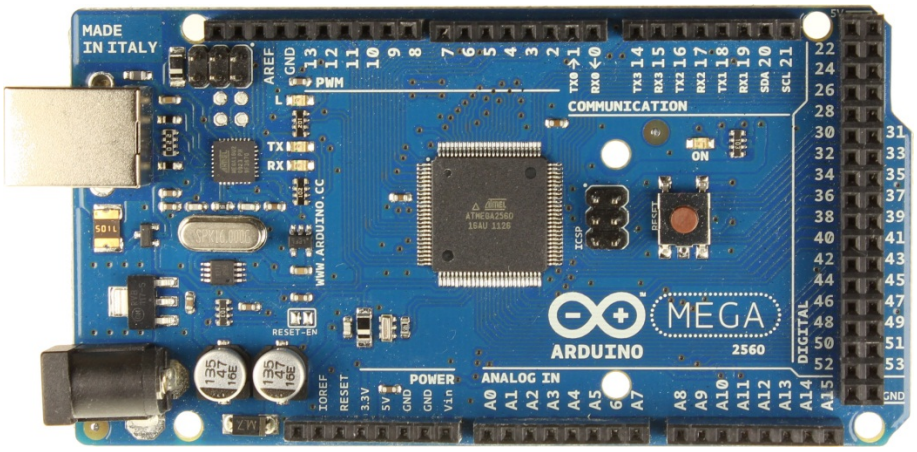

Figura 2.7. Arduino Mega2560

Resumen

Característica	Descripción
Microcontrolador	ATmega 2560
Voltaje de Operación	5V
Tensión de Entrada (Recomendada)	7-12V
Tensión de Entrada (límite)	6-20V
Pines Digitales de E/S	54 (15 para PWM)
Pines de Entrada Analógicos	16
Corriente DC por pin E/S	40mA
Corriente DC para pin 3.3V	50mA
Memoria Flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Frecuencia de Reloj	16 MHz

Tabla 2.4. Características de arduino Mega 2560

Alimentación

La tarjeta arduino mega 2560 puede ser alimentada a través de conexión USB o con una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente. La alimentación externa puede venir con un adaptador de AC/DC o desde una batería. El adaptador puede ser conectado mediante un enchufe centro-positivo de 2.1mm en el conector de alimentación de la placa. Los cables de la

batería pueden insertarse en las cabeceras de los pines GND y Vin del conector POWER.

La placa puede operar con un suministro externo de 6 a 20 voltios. Si se proporcionan menos de 7V, el pin de 5V puede proporcionar menos voltaje y la placa puede ser inestable. Si se utiliza más de 12V, el regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- **VIN**. Es la entrada de tensión a la placa Arduino cuando está usando una fuente de alimentación externa. Se puede suministrar tensión a través de este pin, o, si suministra tensión a través del conector de alimentación, se puede acceder a esta tensión a través de este pin.
- **5V**. Es el suministro regulado de energía usado para alimentar al microcontrolador y otros componentes de la placa. Este puede venir o desde el pin VIN o ser suministrado por USB.
- **3.3V.** Es un suministro de 3,3 voltios generado por un regulador en la placa. El consumo de corriente máxima es de 50 mA.
- **GND**. Pines de tierra.

Memoria

El ATmega2560 tiene 256 KB de memoria Flash para almacenar código. Tiene 8 KB de SRAM y 4 KB de EEPROM.

Entrada y Salida

Cada uno de los 54 pines digitales del Mega2560 puede ser usado como entrada o salida. Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna pull-up de 20-50 KOhms. Además, algunos pines tienen funciones especiales:

- **Serial: 0 (RX) y 1 (TX); Serial 1: 19 (RX) y 18 (TX); Serial 2: 17 (RX) y 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Usados para recibir (Rx) y transmitir (Tx) datos TTL en serie.
- **Interrupciones externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2).** Estos pines pueden ser configurados para provocar una interrupción en un valor bajo, un margen creciente o decreciente, o un cambio de valor.
- **PWM: del pin 2 al 4 y del 44 al 46.** Proporcionan salida PWM de 8 bits.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** Estos pines soportan comunicación SPI (*Serial Peripheral Interface*) el cual es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
- **LED: 13**. Hay un *LED* empotrado conectado al pin digital 13. Cuando el pin está a valor *HIGH*, el *LED* está encendido, cuando el pin está a *LOW*, está apagado.

El Mega2560 tiene 16 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución. Por defecto miden de 0 a 5 voltios.

- **AREF**. Voltaje de referencia para las entradas analógicas.
- **Reset**. Pone esta línea a LOW para resetear el microcontrolador.

Comunicación

El Arduino Mega2560 tiene un número de comodidades para comunicarse con un computador, otro Arduino, u otros microcontroladores. El ATmega2560 provee comunicación serie UART TTL (5 V). El software Arduino incluye un monitor serie que permite a datos de texto simple ser enviados hacia y desde la placa Arduino. El ATmega2560 también soporta comunicación TWI y SPI. (Arduino)

2.4.3. LENGUAJE DE PROGRAMACIÓN DE ARDUINO

Estructura de un Sketch

La estructura básica del lenguaje de programación en Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias encierran bloques que contienen declaraciones, sentencias o instrucciones:

```
void setup ( )
{
sentencias;
}
void loop ( )
```

{

sentencias;

}

Ambas funciones son requeridas para que el programa funcione.

- **Setup ( )**

Esta función se invoca una sola vez al inicio del programa, se usa para inicializar los modos de trabajo de los pines o el puerto serie y además contiene la declaración de cualquier variable, esta función debe ser incluida en el programa aunque no haya declaración que ejecutar. Aquí también se puede establecer el valor inicial de las salidas de la placa.

- **Loop( )**

Después de llamar a la función setup( ), la función loop( ) se ejecuta de manera cíclica, lo que permite que el programa responda continuamente ante los eventos que se produzcan en la placa, es decir, leyendo entradas, activando salidas, etc.

- **Funciones**

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas cuando se llama a la función. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y reducir el tamaño y desorden de un programa.

Las funciones se declaran asociadas a un tipo de valor, este valor será el que va a devolver la función (*int, void,…*). Después de declarar el tipo de dato que se

devolverá se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

- **Comentarios**

Los bloques de comentarios o comentarios multi – línea "/*…*/" son áreas de texto ignoradas por el programa y se usan para grandes descripciones de código o comentarios que ayudan a otras personas a entender partes del programa. Una línea de comentario empieza con / / y terminan con la siguiente línea de código.

- **Variables**

Una variable es una forma de llamar y almacenar un valor numérico para usarse después por el programa. Una variable necesita ser declarada y, opcionalmente, asignada al valor que necesita para ser almacenada. Una variable puede ser cualquier nombre o palabra que no sea una palabra reservada en el entorno de arduino.

- **Declaración de una variable**

Todas las variables tienen que ser declaradas antes de que puedan ser usadas. Declarar una variable significa establecer su tipo de valor, como *int, long, float*, etc., y definir un nombre específico, opcionalmente se puede asignar un valor inicial.

- **Uso de una variable**

Una variable global es aquella que puede ser vista y utilizada por cualquier función y sentencia de un programa. Esta variable se declara al comienzo del programa, antes del setup (). Una variable local es aquella que se define dentro de

una función o como parte de un bucle. Solo es visible y puede utilizarse únicamente dentro de la función en la que se declaró.

Tipos de Datos

Tipo de Dato	Tipo de Valor	Rango
Byte	Entero	0 a 255
Int	Entero	-32767 a 32767
Long	Entero	-2147483647 a 2147483647
Float	Flotante	-3.4028235E+38 a 3.4028235E+38
Array	Arreglo	--

Tabla 2.5. Tipos de datos que soporta arduino

Aritmética

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente de dos operandos.

- **Operadores de comparación**

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo *if* para testear si una condición es verdadera, las comparaciones permitidas por la placa arduino se enlistan en la Tabla 2.6.

Comparación	Símbolo
Igual	==
Menor que	<
Menor o igual que	<=
Mayor que	>
Mayor o igual que	>=
Diferente	!=

Tabla 2.6. Tipos de Comparaciones

- **Operadores lógicos**

Los operadores lógicos son normalmente una forma de comparar dos expresiones y devuelven *true* o *false* dependiendo del operador. Hay tres operadores lógicos, AND, OR y NOT, que se usan a menudo en declaraciones *if*.

- **Constantes**

El lenguaje de programación de arduino tiene ciertos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos:

- ***True/False***

Son constantes booleanas que definen niveles lógicos. *False* se define como 0 mientras *true* es 1 o un valor distinto de 0.

- ***High/Low***

Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital en los pines. *High* está definido como 1 lógico, *ON*, o 5V, mientras que *low* es el 0 lógico, OFF, o 0 voltios.

- ***Input/Output***

Estas constantes son utilizadas para definir el modo de funcionamiento de los pines mediante la instrucción pinMode, de tal manera que el pin puede ser una entrada *INPUT* o una salida *OUTPUT*.

Control de Flujo

- ***If* (Si condicional)**

Las sentencias *if* comprueban si cierta condición ha sido alcanzada y ejecutan todas las sentencias dentro de las llaves si la declaración es cierta. Si es falsa el programa ignora la sentencia.

- ***If…else* (Si…sino)**

Es una estructura que se ejecuta en respuesta a la idea "si esta condición no se cumple haga lo que sigue". *Else* puede preceder a otra comprobación *if*, por lo que múltiples y mutuas comprobaciones exclusivas pueden ejecutarse al mismo tiempo.

- ***For***

La declaración *for* se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración *for* tiene tres partes separadas por (;).

- ***While***

Un bucle del tipo *while* se ejecutará continuamente mientras se cumpla la expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada.

- ***Do…While***

El bucle *do…while* es un bucle que trabaja de la misma forma que el bucle *while*, con la excepción de que la condición es revisada al final del bucle, por lo que el bucle *do…while* siempre se ejecutará al menos una vez.

E/S Digitales

- **pinMode(#pin, mode)**

Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser *INPUT* (entrada) u *OUTPUT* (salida). Los pines configurados como *INPUT* se dice que están en un estado de alta impedancia.

- **digitalRead(pin)**

Lee el valor de un pin (definido como digital) dando un resultado *HIGH* (alto) o *LOW* (bajo). El pin se puede especificar ya sea como una variable o una constante numérica.

- **digitalWrite(pin, value)**

Hace que un pin digital especificado se ponga en un valor determinado, ya sea *HIGH* (alto) o *LOW* (bajo), el pin puede ser especificado como una variable o constante.

E/S Analógicas

- **analogRead(pin)**

Lee el valor desde un pin analógico especificado con una resolución de 10 bits. Esta función sólo trabaja en los pines analógicos. Los valores enteros devueltos están en el rango de 0 a 1023.

- **analogWrite(pin, value)**

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pines de arduino marcados como “PWM”. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre dentro de un rango de 0 a 255.

Control de Tiempo

- **delay(ms)**

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la instrucción.

- **millis()**

Devuelve el número de milisegundos desde que la placa arduino empezó a ejecutar el programa actual, como un valor *long* sin signo.

Matemática

- **min(x,y)**

Esta función obtiene el valor mínimo entre dos números, devolviendo el más pequeño, funciona para cualquier tipo de dato.

- **max(x, y)**

Esta función obtiene el valor máximo entre dos números, devolviendo el número con el valor más grande, funciona para cualquier tipo de dato. (Evans, 2011)

2.5. DESCRIPCIÓN DEL CONTROLADOR LÓGICO PROGRAMABLE (PLC)

Según la IEC 61131, “Un controlador programable es un sistema electrónico programable diseñado para ser utilizado en un entorno industrial, que utiliza una memoria programable para el almacenamiento interno de instrucciones orientadas al usuario, para implantar soluciones específicas tales como funciones lógicas, secuencia, temporización, recuento y funciones aritméticas con el fin de controlar mediante entradas y salidas, digitales y analógicas diversos tipos de máquinas o procesos”.

El PLC tiene un campo de aplicación muy extenso, principalmente un PLC se utiliza para las siguientes funciones:

- **Control de procesos**

El PLC se encarga de que cada fase de proceso sea efectuado en el orden cronológico correcto y sincronizado.

- **Visualización de instalaciones**

El PLC verifica automáticamente ciertas condiciones de la instalación como temperaturas, presiones, niveles y al detectar un exceso en los coeficientes máximos o mínimos de los parámetros, actúa de dos formas; adopta las medidas necesarias para evitar errores o emite señales de aviso para el personal.

- **Control de Puesta a Punto para Maquinas CNC**

Las máquinas herramientas modernas casi siempre están dotadas de un control numérico computarizado (CNC). Pero para que el CNC y la máquina herramienta se entiendan, es preciso integrar un PLC, que se encarga de la comunicación entre ambos equipos. (Mendoza Jiménez & Guillén Garcia)

2.5.1. CONTROLADOR COMPACTLOGIX 1768-L43

Los controladores CompactLogix 1768 son ideales para aplicaciones de tamaño pequeño y mediano que requieren, movimiento o comunicaciones complejas. Este tipo de controladores ofrecen canales en serie, EtherNet/IP o ControlNet integrados y comunicaciones DeviceNet modulares. Son compatibles con hasta 30 módulos de E/S y hasta 16 ejes de movimiento.

Figura 2.7. CompactLogix 1768-L43

Características

- Permite controlar E/S distribuidas mediante EtherNet/IP, ControlNet o DeviceNet.
- Soporta la interconexión entre los datos de control y recopilarlos sobre la misma red.
- Permite edición en línea, forzado de entradas y salidas y otras actividades estándar aisladas de las operaciones de seguridad.
- Posee integración con módulos E/S de los controladores CompactLogix 1769.
- Proporciona puerto serie integrado.
- Posee opciones de memoria de usuario de 2MB. (Rockwell Automation)

Característica	1768-L43
Memoria Disponible	2MB
Tarjeta de Memoria	1784-CF128 (128MB)
Opciones de Comunicación	EtherNet/IP ControlNet DeviceNet
Comunicación por puerto serie	1 puerto RS-232
Número máximo de módulos 1768	2
Batería	Ninguna
Fuente de alimentación eléctrica	1768-PA3, 1768-PB3

Tabla 2.7. Características de CompactLogix 1768-L43

2.5.2. LENGUAJES DE PROGRAMACIÓN DE PLC'S

Diagrama de Bloques Funcionales (FBD)

Es un lenguaje de programación orientado a gráficos. Trabaja con una lista de redes, cada una de las cuales contiene una estructura que representa una expresión lógica o aritmética como se muestra en la Figura 2.8. Se clasifican en dos grupos en función de su forma de operar y su disponibilidad en el programa.

- **Bloques secuenciales Básicos**

Aquellos que son de uso generalizado en todo tipo de controladores (contadores, biestables, temporizadores y registros de desplazamiento).

- **Bloques de expansión o funciones**

Son los que hacen posible el tratamiento de variables numéricas y el registro de datos, con sentencias aritméticas (comparación, transferencias, etc.), aumentando así la potencia del lenguaje.

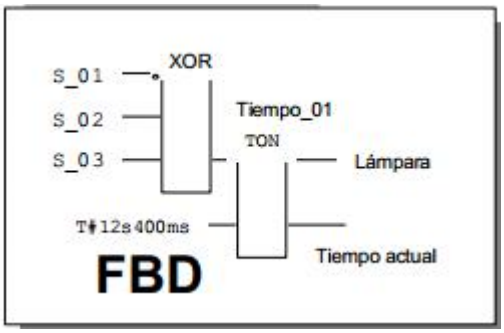

Figura 2.8. Programación FBD

Diagrama de Escalera (*Ladder*)

Este lenguaje de programación es un lenguaje parecido a los diagramas eléctricos empleados para representar los esquemas de lógica cableada utilizados para controlar procesos. En la Figura 2.9 se muestra un ejemplo de programación en LADDER.

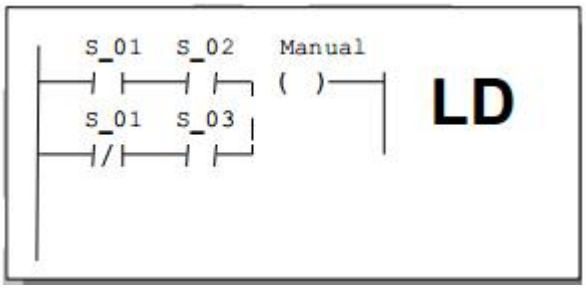

Figura 2.9. Programación LADDER

Lista de Instrucciones (IL)

Este lenguaje se basa en el uso de un mnemónico que representa la instrucción seguido del operando u operandos sobre los que se aplica. El resultado de la operación puede ser almacenado sobre alguno de los registros que emplea el equipo. Cada línea del programa contiene una única instrucción y su ejecución es secuencial comenzando por la primera de la lista como se muestra en la Figura 2.10. Todos los programas escritos en cualquiera de los otros lenguajes pueden ser finalmente traducidos a IL.

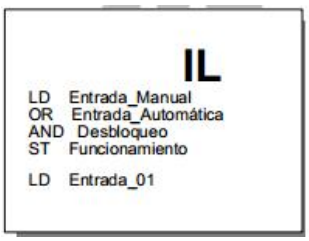

Figura 2.10. Programación IL

Diagramas Funcionales Secuenciales (SFC)

El lenguaje de los diagramas funcionales secuenciales (*Sequential Function Chart*) surge como una evolución del lenguaje de modelado de sistemas secuenciales GRAFCET. SFC proporciona un potente lenguaje para la representación de procesos secuenciales.

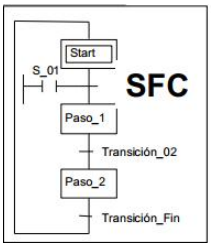

Figura 2.11. Programación SFC

Texto Estructurado (ST)

El texto estructurado (Structured Text) es un lenguaje literal de alto nivel que surge de adaptar el lenguaje pascal empleado en la programación de computadores a las necesidades propias del control de procesos. Es un lenguaje muy potente especialmente indicado para la representación de algoritmos de control complejos en los que sea necesario emplear bucles, condicionales, etc. (Sirgo Blanco, 2008)

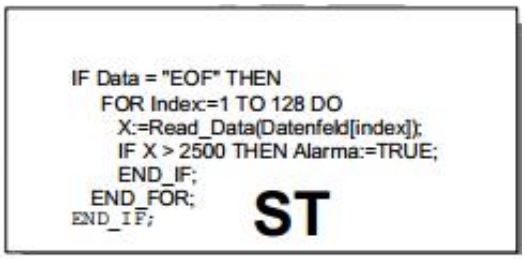

Figura 2.12. Programación ST

2.5.3. PROGRAMACIÓN DE PLC'S EN TEXTO ESTRUCTURADO (ST)

El texto estructurado es un lenguaje de programación textual que usa enunciados para definir lo que se va a ejecutar. El texto estructurado trata indistintamente las mayúsculas y las minúsculas.

Los componentes básicos de programación en texto estructurado se muestran en la Tabla 2.8.

Término	Definición
Asignación	El enunciado de asignación se utiliza para asignar valores a los tags. Se usa ":=" como operador de asignación.
Expresión	Una expresión es parte de una asignación completa o de un enunciado de construcción. Una expresión evalúa según un número (expresión numérica) o según un estado de verdadero o falso (expresión BOOL).
Instrucción	Una instrucción es un enunciado autónomo, usa paréntesis para contener sus operandos. Según la instrucción, puede haber, cero, uno o múltiples operandos. Cuando se ejecuta, una instrucción produce uno o más valores que son parte de una estructura de datos. Aunque su sintaxis es similar, las instrucciones difieren de las funciones en que las instrucciones no pueden usarse en expresiones.
Construcción	Es un enunciado condicional usado para activar el código de texto estructurado.
Comentario	Texto que explica o aclara lo que hace una sección del texto estructurado.

Tabla 2.8. Componentes de programación en ST

Asignaciones

Se usa una asignación para cambiar el valor almacenado dentro de un tag, con la siguiente sintaxis.

tag:=expresion;

El tag retiene el valor asignado hasta que otra asignación cambia el valor, la expresión puede ser simple, como un valor inmediato u otro nombre de tag, o la expresión puede ser compleja e incluir varios operadores y/o funciones.

Expresiones

Una expresión es el nombre de un tag, una ecuación o una comparación, para escribir una expresión se debe utilizar los siguientes elementos:

- Nombre del tag que almacene el valor.
- Nombre que se introduce directamente en una expresión.
- Funciones.
- Operadores.

Al momento de escribir una expresión, también es recomendable seguir las siguientes reglas:

- Usar cualquier combinación de mayúsculas y minúsculas.
- Para expresiones más complejas, es recomendable usar paréntesis para agrupar las expresiones.

En texto estructurado se puede utilizar dos tipos de expresiones:

- **Expresión BOOL**

Es una expresión que produce ya sea el valor BOOL de 1(verdadero) o 0(falso). Este tipo de expresiones usan tags booleanos, operadores con relaciones (Tabla 2.11) y operadores lógicos (Tabla 2.12) para verificar si las condiciones son verdaderas o falsas.

- **Expresión numérica**

Es una expresión que calcula un valor entero o de punto flotante. Una expresión numérica usa operadores aritméticos (Tabla 2.9), funciones aritméticas (Tabla 2.10) y operadores bit a bit.

Operador	Símbolo	Tipo de dato
Suma	+	DINT, REAL
Resta	-	DINT, REAL
Multiplicación	*	DINT, REAL
Exponente	**	DINT, REAL
División	/	DINT, REAL
Modulo-división	MOD	DINT, REAL

Tabla 2.9. Operadores Aritméticos ST

Función	Símbolo	Tipo de Dato
Valor absoluto	ABS (expresion_numerica)	DINT, REAL
Coseno	COS (expresion_numerica)	REAL
Radianes a grados	DEG (expresion_numerica)	DINT, REAL
Logaritmo natural	LN (expresion_numerica)	REAL
Logaritmo base 10	LOG (expresion_numerica)	REAL
Grados a radianes	RAD (expresion_numerica)	DINT, REAL
Seno	SIN (expresion_numerica)	REAL
Raíz cuadrada	SQRT (expresion_numerica)	DINT, REAL
Tangente	TAN (expresion_numerica)	REAL
Truncar	TRUNC (expresion_numerica)	DINT, REAL

Tabla 2.10. Funciones Aritméticas ST

Los operadores con relaciones comparan dos valores o cadenas para proporcionar un resultado verdadero o falso, es decir, el resultado de esta operación es un valor booleano.

Comparación	Símbolo	Tipo de Dato
Igual	=	DINT, REAL, string
Menor que	<	DINT, REAL, string
Menor o igual que	<=	DINT, REAL, string
Mayor que	>	DINT, REAL, string
Mayor o igual que	>=	DINT, REAL, string
Diferente	<>	DINT, REAL, string

Tabla 2.11. Operadores de Relación ST

Los operadores lógicos permiten verificar si múltiples condiciones son verdaderas o falsas, obviamente, su resultado será un valor booleano.

Operador	Símbolo	Tipo de dato
Y lógico	&, AND	BOOL
O lógico	OR	BOOL
O lógico exclusivo	XOR	BOOL
Complemento lógico	NOT	BOOL

Tabla 2.12. Operadores Lógicos

Instrucciones

Los enunciados de texto estructurado también pueden ser instrucciones. Una instrucción de texto estructurado se ejecuta cada vez que se escanea, Una instrucción dentro de una construcción se ejecuta cada vez que las condiciones de la construcción son verdaderas. Si las condiciones de la construcción son falsas, los enunciados dentro de la construcción no se escanean.

Construcciones

Las construcciones pueden programarse individualmente o anidadas dentro de otras construcciones.

- **If…Then**

Esta construcción se utiliza para hacer algo si o cuando ocurra una condición especifica.

IF bool_expression1 THEN

<enunciado >;

ELSIF bool_expression2 THEN

<enunciado>;

ELSE

<enunciado>;

END_IF;

ELSIF se utiliza para seleccionar entre varios posibles grupos de enunciados, un enunciado dentro de *ELSIF* representa una ruta alternativa. El controlador ejecuta el primer enunciado *IF* o *ELSIF* verdadero y se salta el resto de enunciados. *ELSE* se utiliza cuando todas las condiciones *IF* o *ELSIF* son falsas.

- **Case…Of**

Se utiliza CASE para seleccionar que hacer en base a un valor numérico.

CASE numeric_expression OF

selector1 : <enunciado>;

selector2 : <enunciado>;

selector3 : <enunciado>;

ELSE

<enunciado>;

END_CASE;

La construcción CASE es similar a un enunciado de interruptor en los lenguajes de programación C o C++. Sin embargo, con la construcción CASE el controlador ejecuta sólo los enunciados asociados con el primer valor de selector coincidente. La ejecución siempre se interrumpe después de los enunciados de dicho selector y va al enunciado END_CASE.

- **For…Do**

Se utiliza el lazo FOR…DO para hacer algo un número específico de veces, antes de hacer otra cosa.

FOR count := initial_value

TO final_value

BY increment

DO

<enunciado>;

IF bool_expression THEN

EXIT;

END_IF;

END_FOR;

Para detener el lazo antes de que el conteo llegue al último valor, se utiliza en enunciado *EXIT.*

- **While…Do**

Se usa el lazo WHILE…DO para continuar haciendo algo, siempre y cuando ciertas condiciones sean verdaderas.

WHILE bool_expression1 DO

<statement>;

IF bool_expression2 THEN

EXIT;

END_IF;

END_WHILE;

Al igual que en el lazo FOR…DO, el enunciado EXIT, hace que el lazo se detenga antes de que las condiciones sean falsas.

- **Repeat…Until**

Se utiliza el lazo REPEAT…UNTIL para continuar haciendo algo hasta que las condiciones sean verdaderas.

REPEAT

<statement>;

IF bool_expression2 THEN

EXIT;

END_IF;

UNTIL bool_expression1

END_REPEAT;

Al igual que los dos lazos anteriores, en la construcción REPEAT…UNTIL se pude utilizar la sentencia EXIT para salir anticipadamente.

CAPÍTULO 3

DISEÑO DEL SISTEMA DE IDENTIFICACIÓN DE PROCESOS INDUSTRIALES EN LÍNEA

3.1. INTRODUCCIÓN

En el Capítulo 2 se expuso ideas básicas sobre la identificación de sistemas y en especial un desarrollo teórico del método de los mínimos cuadrados para estimar modelos paramétricos. Esta teoría se utilizará en el presente capítulo para realizar el diseño y simulación de un sistema de identificación de procesos industriales en línea.

El capítulo está organizado de la manera siguiente: en el Epígrafe 3.2 se desarrolla el modelamiento de dos procesos o sistemas físicos reales, los cuales serán utilizados en la simulación para comprobar el buen funcionamiento de las rutinas de identificación.

En el Epígrafe 3.3 se realiza la implementación del algoritmo de identificación LMS en Matlab para comprobar su correcto funcionamiento y su posterior simulación en Simulink como herramienta para emular un entorno de ejecución en línea.

En el Epígrafe 3.4 se implementa el algoritmo RLS como otra alternativa de identificación, en este caso recursiva, y de la misma manera que el algoritmo anterior se realiza las pruebas necesarias en Matlab y en Simulink para analizar y comparar los resultados de simulación, además de implementar el desarrollo para acoplar el algoritmo de identificación a sistemas de múltiples entradas y salidas.

En el Epígrafe 3.5 se implementa en algoritmo LS-Lattice para la identificación de modelos AR (Auto-Regresivos) y así obtener más recursos de comparación entre los métodos de identificación anteriores para observar la incidencia de varios aspectos en la implementación final del sistema de identificación de procesos industriales en línea como son: período de muestreo, algoritmo utilizado y el orden del modelo a identificar.

3.2. MODELADO DE PROCESOS INDUSTRIALES

En el Epígrafe 2.2 se discutió los modelos de procesos industriales en tiempo discreto y sus formas de representar, ahora se desarrollará el modelamiento de dos procesos industriales específicos, uno de primer orden y el otro de segundo orden, asumiendo diferentes valores de sus parámetros para analizar la respuesta de los

algoritmos ante la presentación de diferentes funciones de transferencia, los que serán utilizados para comprobar el funcionamiento de los algoritmos que se implementarán más adelante. Como se menciona anteriormente se necesita el modelo discreto del sistema para que sea utilizado por el algoritmo de identificación, pero inicialmente se debe obtener la función de transferencia en tiempo continuo para posteriormente obtener la función de transferencia en tiempo discreto a través de la retención de orden cero (ZOH).

3.2.1. MODELADO DE UN SISTEMA TÉRMICO DE PRIMER ORDEN

En la Figura 3.1 se muestra un sistema térmico que consta de un tanque aislado para eliminar las pérdidas de calor hacia el aire, el tanque posee una entrada por donde ingresa liquido frio y se calienta a través del calefactor y el mezclador en el interior del tanque, el tanque está provisto de una salida por donde se vierte el líquido caliente.

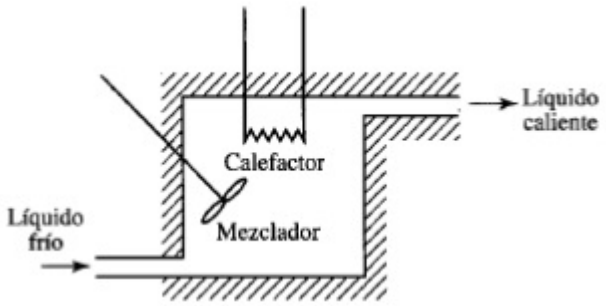

Figura 3.1. Sistema térmico

Obtención de la Función de Transferencia en Tiempo Continuo

Para el sistema térmico de la Figura 3.1 se puede definir los parámetros mostrados en la Tabla 3.1, los que permitirán obtener la función de transferencia en tiempo continuo del proceso.

Símbolo	Parámetro
$\overline{\Theta_i}$	Temperatura del líquido que entra
$\overline{\Theta_o}$	Temperatura del líquido que sale
G	Velocidad del flujo del líquido
M	Masa del líquido en el tanque
c	Calor especifico del líquido
R	Resistencia térmica
C	Capacitancia térmica
$\overline{H}$	Entrada del flujo de calor

Tabla 3.1. Parámetros del sistema térmico

El sistema está diseñado para cambiar la temperatura del líquido frio que ingresa al tanque, es decir que la temperatura del líquido cambia repentinamente de $\overline{\Theta_i}$ a $\overline{\Theta_i} + \theta_i$, se puede asumir que el flujo de calor de entrada H y el flujo de líquido G se conservan constantes, el flujo de calor de salida cambiará de $\overline{H}$ a $\overline{H} + h_o$ y la temperatura del líquido que sale cambia a $\overline{\Theta_o} + \theta$. La ecuación diferencial para este caso es:

$$C\frac{d\theta}{dt} = Gc\theta_i - h_o \tag{3.1}$$

Teniendo en cuenta que $h_o = Gc\theta$ y $R = \frac{1}{Gc}$ la ecuación 3.1 puede reescribirse como:

$$RC\frac{d\theta}{dt} + \theta = \theta_i \tag{3.2}$$

Como se sabe que la temperatura del líquido de entrada θ_i y la del líquido de salida θ son funciones del tiempo, se puede aplicar la transformada de Laplace a ambos lados de la ecuación 3.2, de donde se obtiene la temperatura de salida $\Theta(s)$ en función de la temperatura de entrada $\Theta_i(s)$, constituyendo así la función de transferencia en tiempo continuo del sistema térmico como se muestra a continuación. (Ogata, Ingeniería de Control Moderna 3ed., 1998)

$$\frac{\Theta(s)}{\Theta_i(s)} = \frac{1}{RCs + 1} \tag{3.3}$$

Para un tratamiento general de la función de transferencia de primer orden se realiza el siguiente reemplazo:

$$T = RC$$

De esta forma la función de transferencia final queda de la siguiente manera:

$$\frac{\Theta(s)}{\Theta_i(s)} = \frac{1}{Ts + 1}$$

Donde T se considera la constante de tiempo del sistema.

Obtención de la Función de Transferencia en Tiempo Discreto

Para obtener la función de transferencia de un sistema en tiempo discreto a partir de la función en tiempo continuo se debe pasar del plano S al plano Z, para este fin se utilizará el retenedor de orden cero (ZOH) previamente desarrollado.

Para realizar la trasformación de S a Z se debe utilizar la función de transferencia ZOH de la ecuación 2.1 que se muestra a continuación:

$$G_a[z] = (1 - z^{-1})\ Z\left\{\frac{G(s)}{s}\right\}$$

Donde G(s) es la función de transferencia en tiempo continuo que se quiere discretizar y $G_a[z]$ es la función resultante. Reescribiendo la ecuación 3.3 se obtiene:

$$G(s) = \frac{\Theta(s)}{\Theta_i(s)} = \frac{1/T}{s + 1/T}$$

Se aplica la función de transferencia ZOH de lo que resulta:

$$G_a[z] = (1 - z^{-1})\ Z\left\{\frac{1}{s}\left(\frac{1/T}{s + 1/T}\right)\right\}$$

$$G_a[z] = \frac{1}{T}(1 - z^{-1})\ Z\left\{\frac{1}{s}\left(\frac{1}{s + 1/T}\right)\right\}$$

Para facilitar la transformación se aplica fracciones parciales a todo el conjunto que se encuentra expresado en función de S de lo que se obtiene:

$$G_a[z] = \frac{1}{T}(1 - z^{-1})\ Z\left\{\frac{T}{s} - \left(\frac{T}{s + 1/T}\right)\right\}$$

$$G_a[z] = (1 - z^{-1})\ Z\left\{\frac{1}{s} - \left(\frac{1}{s + 1/T}\right)\right\}$$

Aplicando la transformada Z obtenemos:

$$G_a[z] = (1 - z^{-1})\left(\frac{1}{1 - z^{-1}} - \frac{1}{1 - (e^{-\frac{T_m}{T}})z^{-1}} \right)$$

$$G_a[z] = \frac{(1 - e^{-\frac{T_m}{T}})z^{-1}}{1 - (e^{-\frac{T_m}{T}})z^{-1}}$$

(3.4)

La ecuación 3.4 es la función de transferencia en tiempo discreto del sistema térmico de la Figura 3.1, siendo el resultado de aplicar el retenedor de orden cero (ZOH) a la función de transferencia del sistema en tiempo continuo. Donde T_m es el período de muestreo que se va a aplicar para la discretización y T es la constante de tiempo del sistema definido previamente.

Ejemplos de Funciones de Transferencia de Primer Orden a Identificar

Una vez obtenida la función de transferencia en tiempo discreto del sistema térmico de primer orden, se procede a reemplazar valores en la ecuación (3.4) para obtener algunos ejemplos que sean objeto de simulación con el sistema de identificación en línea y así poder analizar los resultados de la implementación.

Para los ejemplos se utilizará dos períodos de muestreo distintos y dos constantes de tiempo diferentes para obtener más fuentes de comparación, la Tabla 3.2 contiene las funciones de transferencia resultantes de reemplazar los valores de período de muestreo (Tm) y de Constante de tiempo (T), las mismas que se utilizaran para simular el sistema de identificación en línea.

Período de Muestreo [Tm]	Constante de Tiempo [T]	$G(s)$	$G_a[z]$
0.09s	1s	$\frac{1}{s+1}$	$\frac{0.0861z^{-1}}{1-0.914z^{-1}}$
	2s	$\frac{0.5}{s+0.5}$	$\frac{0.044z^{-1}}{1-0.956z^{-1}}$
0.05s	1s	$\frac{1}{s+1}$	$\frac{0.049z^{-1}}{1-0.951z^{-1}}$
	2s	$\frac{0.5}{s+0.5}$	$\frac{0.0247z^{-1}}{1-0.975z^{-1}}$

Tabla 3.2. Funciones de transferencia de primer orden a identificar

3.2.2. MODELADO DE UN SISTEMA TÉRMICO DE SEGUNDO ORDEN

En la industria es común encontrar sensores de temperatura que se hallan instalados dentro de una funda protectora o vaina como se muestra en la Figura 3.2, debido a que el sensor necesita ser aislado del medio al que se mide temperatura, se observará que la influencia de la funda protectora hace que el sistema pase de ser de primer orden a ser de segundo ya que no se puede despreciar al aislamiento.

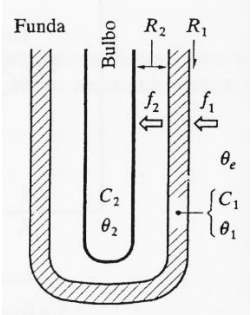

Figura 3.2. Sensor de Temperatura con funda protectora

Obtención de la Función de Transferencia en Tiempo Continuo

Para el sistema térmico de la Figura 3.2 se puede definir los parámetros mostrados en la Tabla 3.3, los que permitirán obtener la función de transferencia en tiempo continuo del sistema.

Símbolo	Parámetro
U	Coeficiente de transmisión de calor
A	Superficie de conducción de calor
M	Masa del material
c	Calor especifico
R	Resistencia térmica
C	Capacitancia térmica
$\Theta(\theta)$	Temperatura
F	Flujo de calor
H	Cantidad de calor

Tabla 3.3. Parámetros del sensor de temperatura con funda protectora

Las ecuaciones de equilibrio térmico de este sistema son las siguientes:

Para el sensor de temperatura se tendrá:

$$M_2 c_2 \frac{d\theta_2}{dt} = U_2 A_2 (\theta_1 - \theta_2)$$

Tomando en cuenta que $Mc = C$ y que $UA = {}^{1}/_{R}$, queda:

$$R_2 C_2 \frac{d\theta_2}{dt} + \theta_2 = \theta_1$$

Asumimos la variación de temperatura como función del tiempo por lo que se puede aplicar la transformada de Laplace a ambos lados de la ecuación, obteniendo:

$$\Theta_2 (R_2 C_2 s + 1) = \Theta_1 \tag{3.5}$$

Para la funda protectora, el balance térmico cumplirá la siguiente ecuación:

$$M_1 c_1 \frac{d\theta_1}{dt} = U_1 A_1(\theta_e - \theta_1) - U_2 A_2(\theta_1 - \theta_2)$$

Recordamos que $Mc = C$ y que $UA = {}^{1}/_{R}$, queda:

$$C_1 \frac{d\theta_1}{dt} = \frac{\theta_e - \theta_1}{R_1} - \frac{\theta_1 - \theta_2}{R_2}$$

Aplicando la transformada de Laplace a ambos lados se obtiene:

$$C_1 s\Theta_1 + \Theta_1 \left(\frac{1}{R_1} + \frac{1}{R_2}\right) = \frac{\Theta_e}{R_1} + \frac{\Theta_2}{R_2} \tag{3.6}$$

Reemplazando la ecuación 3.5 en la ecuación 3.6 se obtiene:

$$[R_1 C_1 R_2 C_2 s^2 + (R_1 C_1 + R_2 C_2 + R_1 C_2)s + 1]\Theta_2 = \Theta_e \tag{3.7}$$

Con el fin de facilitar el tratamiento de las funciones se simplifica la ecuación 3.7 haciendo los siguientes reemplazos:

$$T_1 = R_1 C_1$$

$$T_2 = R_2 C_2$$

$$T^2 = T_1 T_2$$

$$2\varepsilon T = T_1 + T_2 + R_1 C_2$$

Obteniendo la ecuación siguiente:

$$\frac{\Theta_2(s)}{\Theta_e(s)} = \frac{1}{T^2 s^2 + 2\varepsilon T s + 1} \tag{3.8}$$

La ecuación 3.8 representa la función de transferencia del sensor de temperatura con funda protectora de tal manera que $\Theta_e\,(s)$es la temperatura que se desea medir y $\Theta_2(s)$ es la temperatura que en realidad va adquiriendo el sensor a través del tiempo. (Roca Cusidó, 2002)

Ejemplos de Funciones de Transferencia de Segundo Orden a Identificar

Una vez obtenida la función de transferencia de tiempo continuo del sensor de temperatura con funda protectora, se utilizará la función c2d de Matlab para obtener la función de transferencia en tiempo continuo, debido a la complejidad del cálculo literal que se debe utilizar para realizarlo manualmente. Para esto se reescribe la ecuación 3.8 quedando lo siguiente:

$$\frac{\Theta_2(s)}{\Theta_e\,(s)} = \frac{{}^{1}/_{T^2}}{s^2 + \frac{2\varepsilon}{T}s + \frac{1}{T^2}} \tag{3.9}$$

De igual manera que la sección anterior se asumirá valores de período de muestreo (Tm) y de constantes de tiempo para el sistema de segundo orden de la ecuación (3.9), además se debe establecer valores para el factor de amortiguamientoε, se asumirá un valor único para la constante de tiempo T=1s, y el valor de épsilon será considerado menor que uno para un sistema subamortiguado y mayor que 1 para un sistema sobre amortiguado. Los resultados del reemplazo y del uso de la función c2d de Matlab se muestran en la Tabla 3.4.

Período de Muestreo [Tm]	Factor de Amortiguamiento [ε]	$G(s)$	$G_a[z]$
0.09s	0.85	$\frac{1}{s^2+1.7s+1}$	$\frac{0.003849z^{-1}+0.003657z^{-2}}{1-1.851z^{-1}+0.8581z^{-2}}$
	1.15	$\frac{1}{s^2+2.3s+1}$	$\frac{0.003782z^{-1}+0.00353z^{-2}}{1-1.806z^{-1}+0.813z^{-2}}$
0.05s	0.85	$\frac{1}{s^2+1.7s+1}$	$\frac{0.001215z^{-1}+0.001181z^{-2}}{1-1.916z^{-1}+0.9185z^{-2}}$
	1.15	$\frac{1}{s^2+2.3s+1}$	$\frac{0.001203z^{-1}+0.001158z^{-2}}{1-1.889z^{-1}+0.8914z^{-2}}$

Tabla 3.4. Funciones de transferencia de segundo orden a identificar

3.3. IMPLEMENTACIÓN DEL ALGORITMO DE IDENTIFICACIÓN LMS

En este apartado se muestra el desarrollo matemático del algoritmo LMS adaptado a la identificación en línea, lo mostrado en el epígrafe 2.3.4 es un desarrollo teórico de la minimización del error cuadrático y para ello se utilizan bloques de datos obtenidos de los procesos, por lo que; si se aplica directamente la ecuación 2.5 no se obtendrían los resultados requeridos ya que no es un método iterativo.

3.3.1. DESARROLLO DEL ALGORITMO LMS

Para obtener un método iterativo basado en mínimos cuadrados se debe mencionar el método del descenso más rápido (*Steepest Descent*), de donde se desprende la siguiente ecuación:

$$\hat{\theta}(k) = \hat{\theta}(k-1) - \mu\nabla_{\hat{\theta}}[\varepsilon(k)] \tag{3.10}$$

Donde $\hat{\theta}(k)$ representa el vector que contiene los parámetros a estimados, μ es una constante que se definirá directamente tomando en cuenta que no puede ser menor o igual a cero ya que provocaría inestabilidad en el sistema, $\nabla_{\hat{\theta}}$ representa el gradiente, que se puede definir como la dirección en la cual la función varia más rápidamente, para nuestro caso la función a evaluar es $\varepsilon(k)$ que representa la evolución del MSE, que es el parámetro a minimizar por la vía más rápida, que viene dada por el cálculo de su gradiente. El método del descenso más rápido posee un desarrollo mayor con el cual se puede estimar directamente parámetros, pero en este punto se debe recurrir a su integración con el método de mínimos cuadrados, el cual se basa en la minimización del error cuadrático de la estimación de la salida del sistema; debido a que este método se implementa mejor de manera práctica y el método del descenso más rápido viene a ser un método puramente teórico.

Para poder utilizar la ecuación 3.10 en la estimación de parámetros se debe calcular el gradiente del MSE a continuación:

$$\nabla_{\hat{\theta}}[\varepsilon(k)] = \frac{\partial}{\partial\hat{\theta}_N}E\{e^2(k)\} = 2E\left\{e(k)\frac{\partial}{\partial\hat{\theta}_N}e(k)\right\} \tag{3.11}$$

Para desarrollar la derivada parcial $\frac{\partial}{\partial\hat{\theta}_N}e(k)$, se debe calcular el error de estimación de la salida como sigue:

$$e(k) = y(k) - \emptyset(k)\hat{\theta}(k-1) \tag{3.12}$$

Donde $\emptyset(k)$ es el vector que contiene una muestra hasta el valor actual de la entrada y la salida del sistema, el número de valores que contenga el vector

dependerá del orden del sistema que se desea identificar, ya que depende directamente del número de parámetros que forman el sistema. Aplicando derivada parcial a la ecuación 3.12 se obtiene:

$$\frac{\partial}{\partial \hat{\theta}_N} e(k) = -\emptyset(k) \tag{3.13}$$

Sustituyendo la ecuación 3.13 en la ecuación 3.11, el gradiente queda:

$$\nabla_{\hat{\theta}}[\varepsilon(k)] = -2E\{e(k)\emptyset(k)\} \tag{3.14}$$

Reemplazando la ecuación 3.14 en la ecuación 3.10 dada por el método *Steepest Descent* queda una ecuación iterativa para la estimación de los parámetros del sistema:

$$\hat{\theta}(k+1) = \hat{\theta}(k) - \alpha E\{e(n)\emptyset(k)\} \tag{3.15}$$

Donde $\alpha = 2\mu$ y $E\{\,.\,\}$ es el operador esperanza, es decir, representa el valor que se desea obtener de una función.

Para hacer de este un algoritmo útil, se necesita simplemente aproximar la esperanza $E\{e(n)\emptyset(k)\}$ con el valor instantáneo dentro de los corchetes, dejando a la ecuación 3.15 como:

$$\hat{\theta}(k+1) = \hat{\theta}(k) - \alpha e(n)\emptyset(k) \tag{3.16}$$

La ecuación 3.16 representa el estimador LMS, en la Tabla 3.5 se muestra un resumen muy útil para la implementación del algoritmo en Matlab, así como también se muestra la manera adecuada de inicializar las variables. (S. Thomas, 1986)

Inicialización:
En k=0: $$\widehat{\boldsymbol{\theta}}(\mathbf{0}) = \emptyset(\mathbf{0}) = \mathbf{0}_{\boldsymbol{N}}$$ $$\boldsymbol{\alpha} > \mathbf{0}$$
Operación:
Para k=1 en adelante: 1. Adquirir y(k) y u(k) 2. Para el error de estimación de 3.12. $$\boldsymbol{e}(\boldsymbol{k}) = \boldsymbol{y}(\boldsymbol{k}) - \emptyset(\boldsymbol{k})\widehat{\boldsymbol{\theta}}(\boldsymbol{k}-\mathbf{1})$$ 3. Actualizar el estimador LMS de 3.16 $$\widehat{\boldsymbol{\theta}}(\boldsymbol{k}+\mathbf{1}) = \widehat{\boldsymbol{\theta}}(\boldsymbol{k}) - \boldsymbol{\alpha e}(\boldsymbol{k})\emptyset(\boldsymbol{k}).$$ 4. Actualizar el vector de datos para la siguiente iteración $$\emptyset(\boldsymbol{k}) = [-\boldsymbol{y}(\boldsymbol{k})\ \emptyset_{\mathbf{1}}(\boldsymbol{k}-\mathbf{1})\ \boldsymbol{u}(\boldsymbol{k})\ \ \emptyset_{\mathbf{3}}(\boldsymbol{k}-\mathbf{1})].$$ Lazo completo. Regresar a (1).

Tabla 3.5. Algoritmo LMS

3.3.2. SIMULACIÓN DEL ALGORITMO DE IDENTIFICACIÓN LMS EN SISTEMAS DE PRIMER ORDEN

Implementación y Simulación en Matlab

Para la simulación del algoritmo se utilizan dos de las funciones de transferencia de primer orden descritas en la Tabla 3.2. La primera se desarrolló con una constante

de tiempo T=1 y un período de muestreo Tm=0.09, obteniendo las siguientes funciones de transferencia en tiempo continuo y en tiempo discreto:

$$G(s) = \frac{1}{s+1} \tag{3.17}$$

$$G_a[z] = \frac{0.0861z^{-1}}{1 - 0.914z^{-1}} \tag{3.18}$$

Los resultados obtenidos de la simulación del sistema de identificación aplicando una entrada de ruido blanco a la función de transferencia descrita en la ecuación 3.18 son los siguientes:

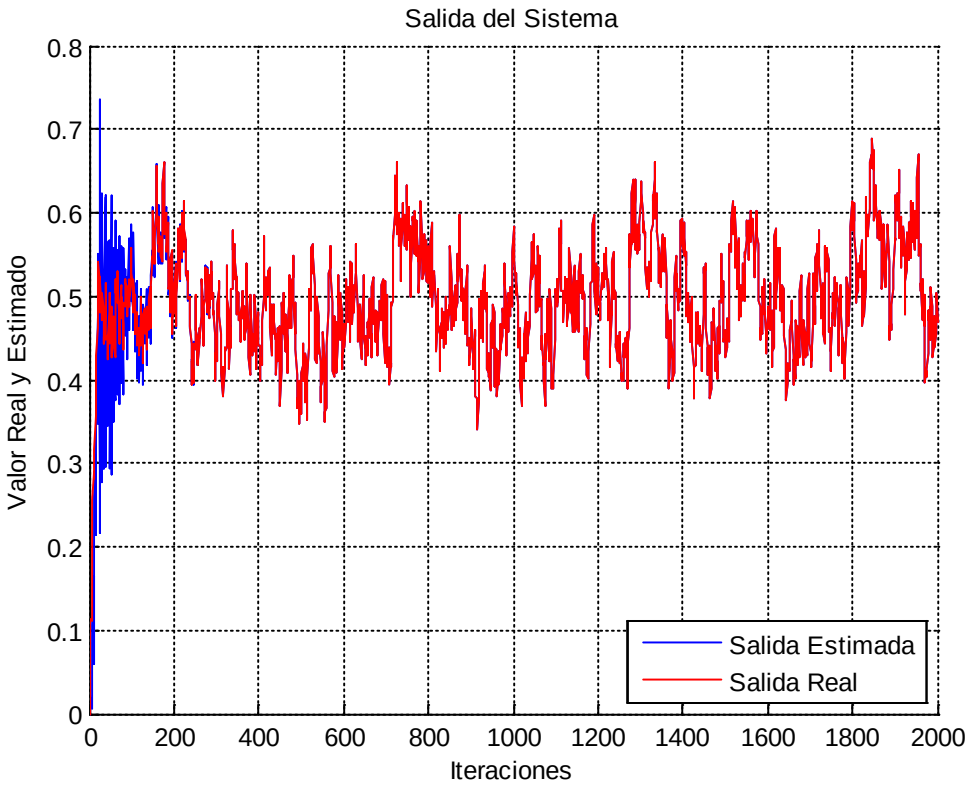

Figura 3.3. Salida del sistema, real y estimada (ec. 3.18)

La Figura 3.3 muestra gráficamente la comparación entre la salida real del sistema y la salida obtenida con los parámetros estimados por el identificador, se puede observar que con el paso de las iteraciones la salida estimada se aproxima a la salida real, esto sirve como verificación del correcto funcionamiento del algoritmo.

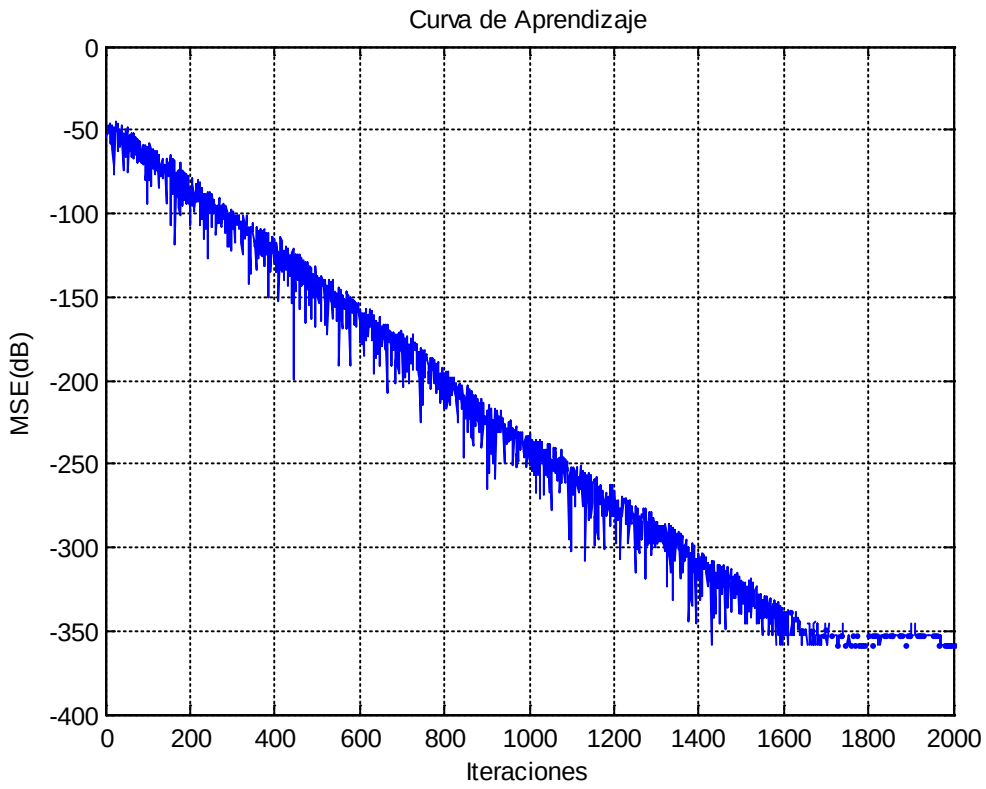

Figura 3.4. Curva de Aprendizaje (ec. 3.18)

La Figura 3.4 representa la curva de aprendizaje o la gráfica del MSE, obtenido del error entre la salid real y la salida estimada, los valores de la curva se dan en dB, en la misma se puede observar que el error desciende hasta establecerse en un valor constante donde el error tiende a ser cero, lo que quiere decir que el algoritmo converge a los valores de parámetros esperados.

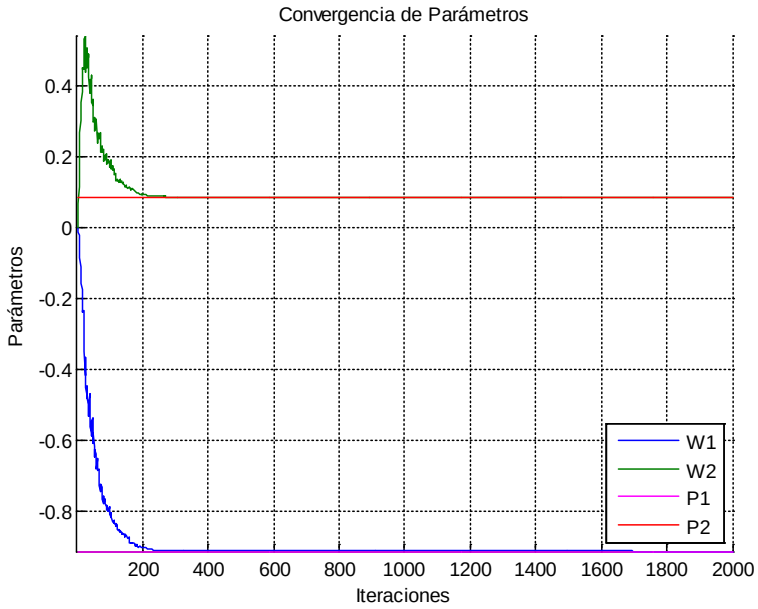

Figura 3.5. Convergencia de Parámetros (ec. 3.18)

La Figura 3.5 muestra la variación de los parámetros en cada iteración, W1 y W2 representan el valor que toman los parámetros estimados a través de las iteraciones, P1 y P2 representan el valor real de los parámetros de la función de transferencia establecida para la identificación. Aquí se muestra claramente la forma en que los parámetros estimados tienden a adquirir el valor de los parámetros reales, llegando a un punto en el que se establecen en el mismo valor.

La Tabla 3.6 indica los valores finales de los parámetros estimados en una comparación con los valores de los parámetros reales, y así comprobar el buen funcionamiento durante la simulación del algoritmo con la primera función de transferencia de la ecuación 3.18.

Parámetros Reales		Parámetros Estimados	
P1	-0.913931185271228	W1	-0.913931185271228
P2	0.086068814728772	W2	0.086068814728772

Tabla 3.6. Comparación entre parámetros reales y parámetros estimados (ec. 3.18)

En esta parte se puede aseverar que, en primera instancia el algoritmo de identificación funciona adecuadamente para estimar procesos industriales de primer orden, no obstante, no se puede asegurar su funcionamiento en línea, pero la implementación en Matlab sirve como antesala a la implementación del algoritmo en Simulink.

Implementación y Simulación en Simulink

La Figura 3.6 muestra el diagrama de bloques del sistema de identificación simulado, donde se puede observar el un generador de ruido blanco, el cual sirve para mantener la función de transferencia con excitación persistente, el bloque que contiene la función de transferencia del proceso, para el primer caso se establece la función de transferencia de la ecuación 3.17. Los 3 bloques ZOH discretizan todas las señales en tiempo continuo que se puede tener, como es el caso del ruido de excitación de la entrada, la salida de la función de transferencia y el tiempo del reloj.

Los bloques mencionados previamente ingresan al bloque función de Matlab, el mismo que contiene el algoritmo de identificación LMS para sistemas de primer orden, del cual se desprenden los resultados tales como los parámetros estimados y la salida estimada del sistema, además es necesario obtener la salida real del sistema por motivos de comparación. Al lado izquierdo de la Figura 3.6 se observa los bloques que van a permitir observar los resultados, un display que muestra en valor de los parámetros estimador y dos *scope* que ejercen como medio de exportación de datos hacia el espacio de trabajo de Matlab, para el análisis de los datos obtenidos.

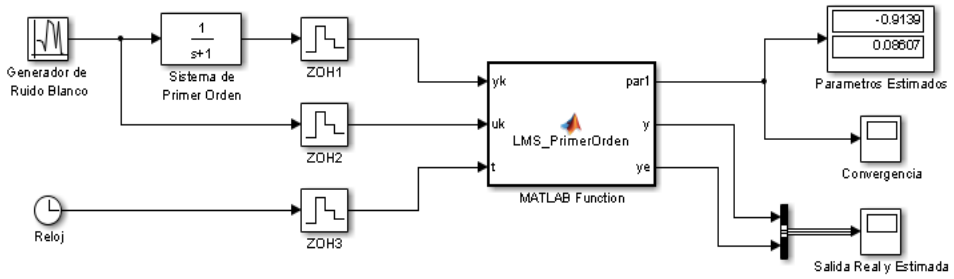

Figura 3.6. Diagrama de bloques de sistema de identificación en línea (ec. 3.17)

La Figura 3.7 muestra la comparación entre la salida real y la salida estimada del sistema de identificación en línea LMS, donde se puede observar que luego de algunos segundos la salida estimada tiende a ser la misma que la salida real.

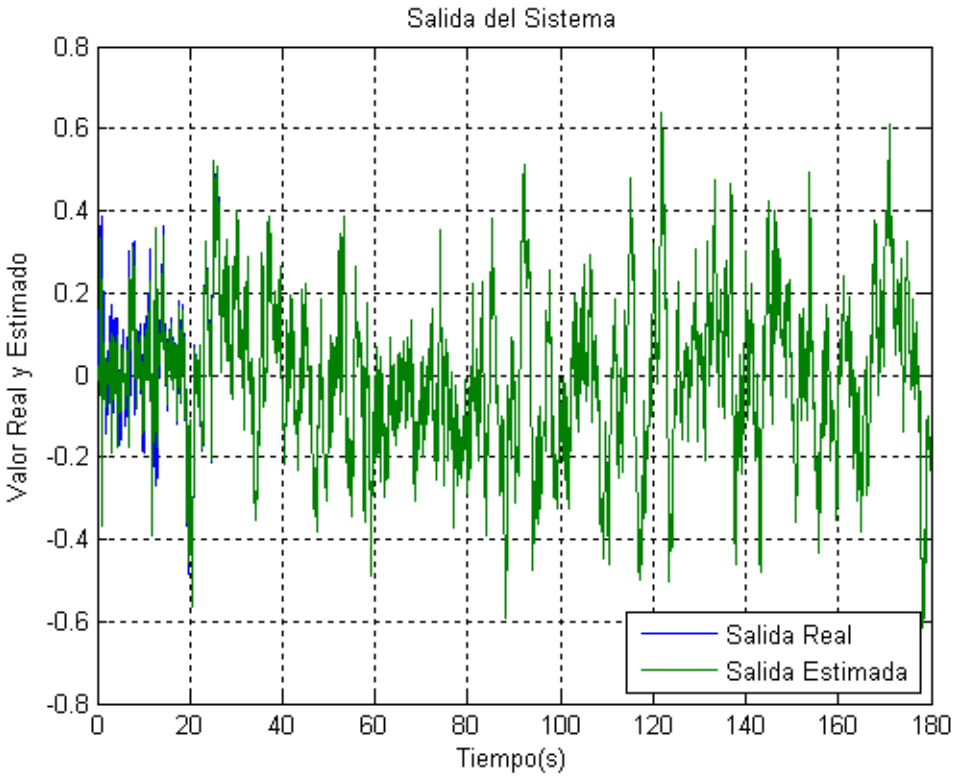

Figura 3.7. Salida del sistema, real y estimada (sist. En línea ec. 3.17)

La Figura 3.8 indica la evolución de la curva de aprendizaje, mostrando el MSE en dB a través del tiempo, se puede ver el descenso del valor del error cuadrático, cumpliendo así el objetivo del algoritmo LMS que consiste en reducir el error obteniendo una estimación de parámetros lo más cerca posible al valor real.

Y la Figura 3.9 muestra la el valor que van tomando los parámetros estimados (W1 y W2) del sistema de primer orden a través del tiempo, en la gráfica se puede observar como estos parámetros después de algunos segundos tienden a igualar el valor de P1 y P2, los cuales representan el valor real de los parámetros del proceso a identificar.

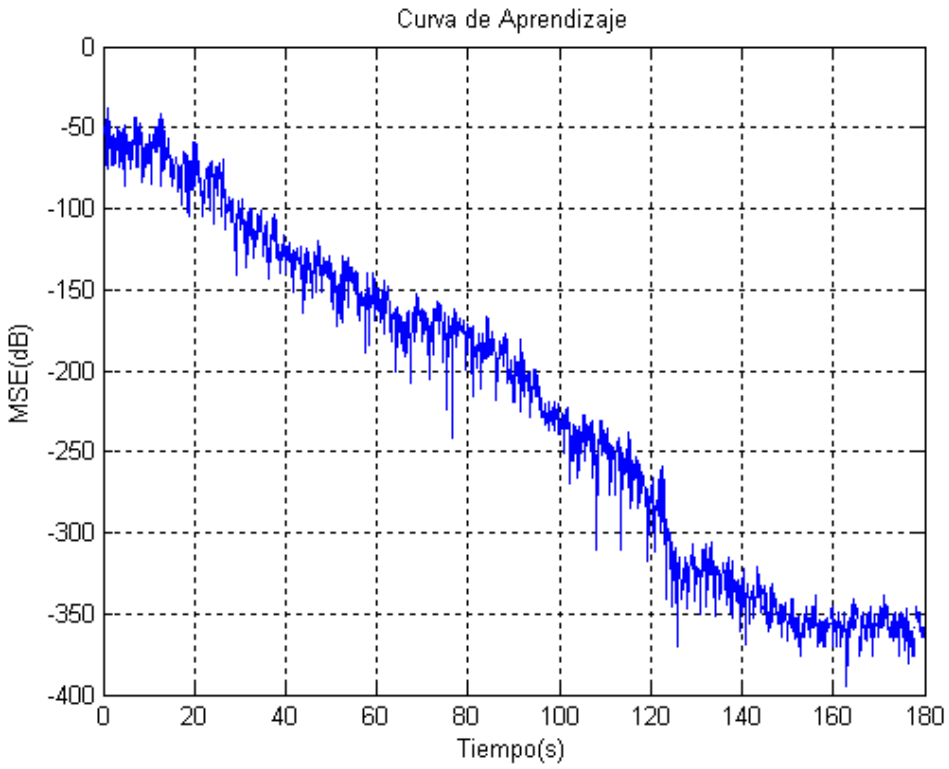

Figura 3.8. Curva de Aprendizaje (sist. En línea ec. 3.17)

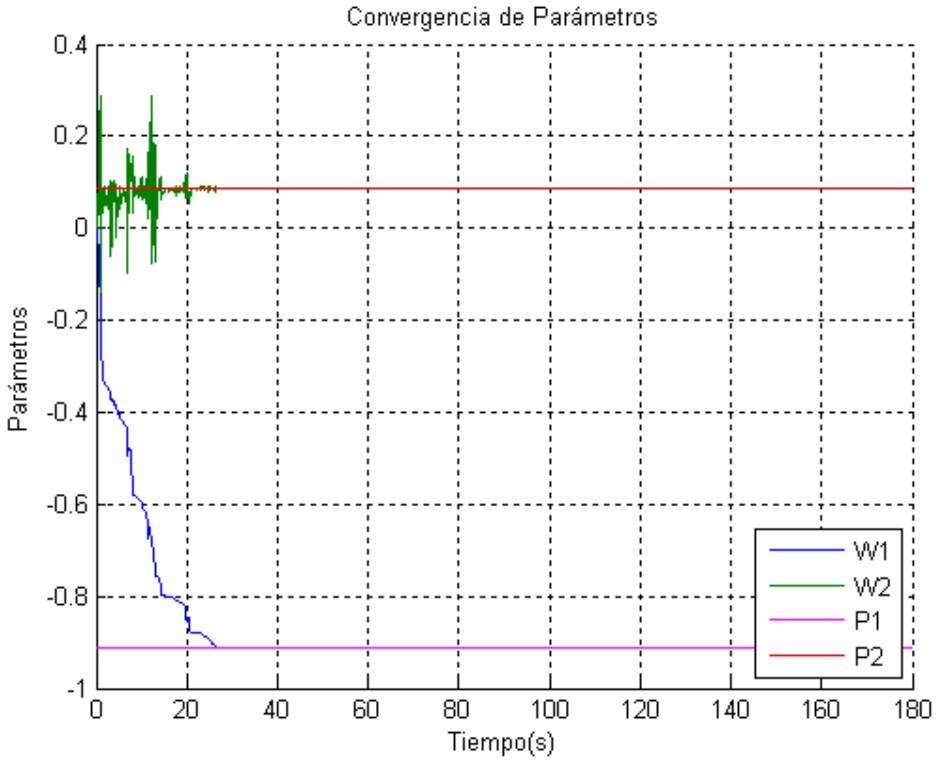

Figura 3.9. Convergencia de Parámetros (sist. En línea ec. 3.17)

Parámetros Reales		Parámetros Estimados	
P1	-0.913931185271228	W1	-0.9139285
P2	0.086068814728772	W2	0.0860715

Tabla 3.7. Comparación entre parámetros reales y estimados (sist. en línea ec. 3.18)

En la Tabla 3.7 se realiza una comparación entre los parámetros reales del sistema y los parámetros obtenidos de la estimación del algoritmo LMS implementado en Simulink, donde se observa que la tendencia a igualar a los parámetros reales.

3.3.3. SIMULACIÓN DEL ALGORITMO DE IDENTIFICACIÓN LMS EN SISTEMAS DE SEGUNDO ORDEN

Implementación y Simulación en Matlab

Para el caso en que se tenga un sistema de segundo orden, se debe tomar en cuenta que el número de parámetros a estimar pasa a ser cuatro, de esta forma el vector de datos debe aumentar su tamaño, para completarlo se aumenta un valor de salida del sistema en un estado anterior, y un valor de entrada del sistema en un estado anterior. Para esta simulación se utilizan dos de las funciones de transferencia de segundo orden descritas en la Tabla 3.4. La primera se desarrolló con una constante de tiempo T=1s, un período de muestreo Tm=0.09 y un factor de amortiguamiento $\varepsilon = 0.85$ las siguientes funciones de transferencia en tiempo continuo y en tiempo discreto:

$$G(s) = \frac{1}{s^2 + 1.7s + 1} \tag{3.21}$$

$$G_a[z] = \frac{0.003849z^{-1} + 0.003657z^{-2}}{1 - 1.851z^{-1} + 0.8581z^{-2}} \tag{3.22}$$

Los resultados obtenidos de la simulación del sistema de identificación aplicando una entrada de ruido blanco a la función de transferencia descrita en la ecuación 3.22 son los siguientes:

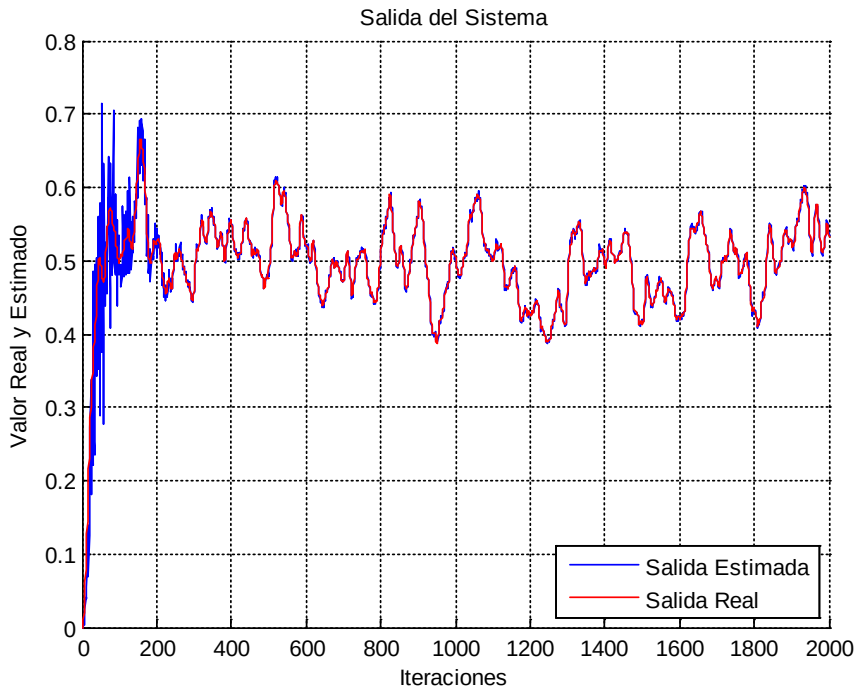

Figura 3.10. Salida del sistema, real y estimada (ec. 3.22)

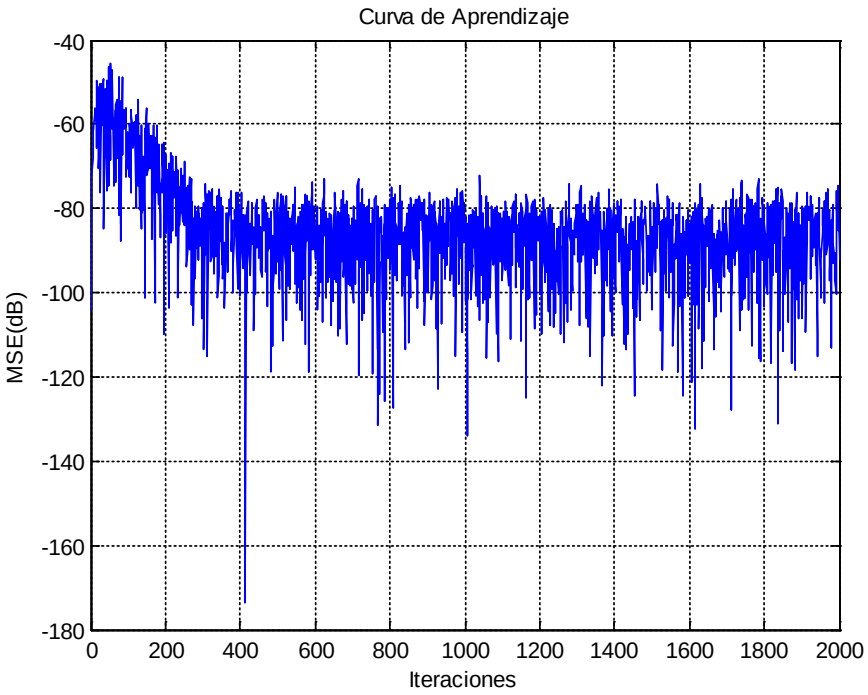

Figura 3.11. Curva de Aprendizaje (ec. 3.22)

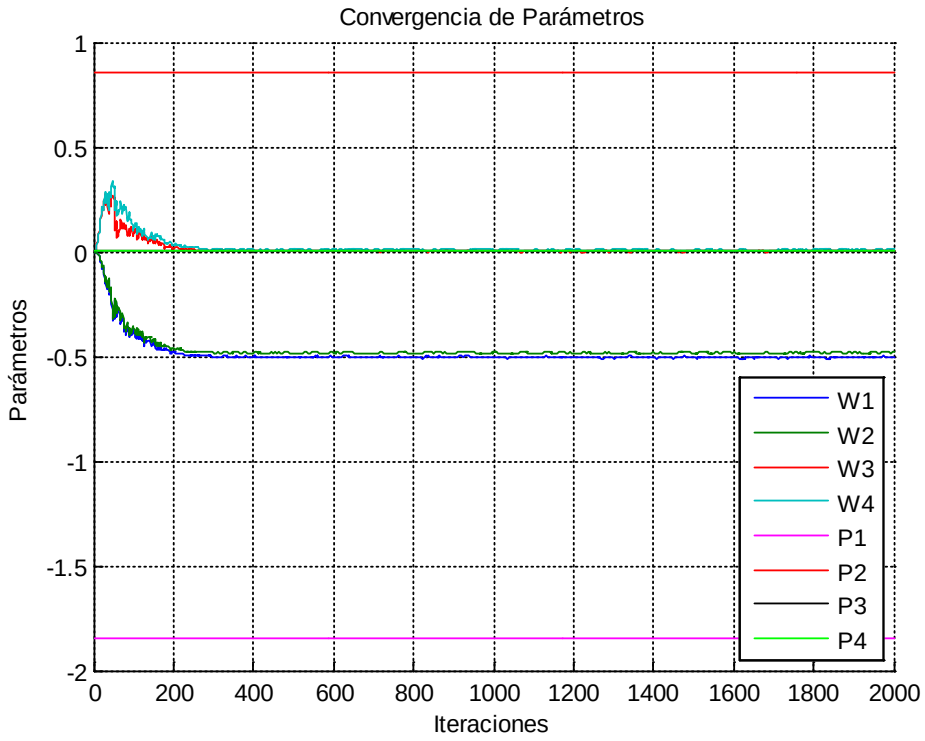

Figura 3.12. Convergencia de Parámetros (ec. 3.22)

Para el análisis de los resultados de la simulación en Matlab se han realizado tres gráficas: la primera (Figura 3.10) muestra una comparativa entre la salida real del sistema y la salida estimada, donde al igual que el caso de primer orden; se observa que con el paso de las iteraciones la salida estimada intenta seguir a la salida real, pero también se puede notar que no logra seguirla de manera fiel. En la segunda gráfica (Figura 3.11) se muestra la evolución de la curva de aprendizaje, en la que se puede observar la minimización del error, cabe mencionar que el error no llega a minimizarse en un alto grado, como se vio en el caso de primer orden.

Por último la tercera gráfica obtenida como resultado de la simulación (Figura 3.12) indica la tendencia que toman los parámetros estimados (W1, W2, W3, W4), en relación a los parámetros reales (P1, P2, P3, P4), la gráfica muestra una falsa tendencia de W1 y W2 hacia los valores de P1 y P2, pero esto se debe a que son

valores muy pequeños y no se puede aseverar que en realidad los parámetros estimados convergen correctamente, también se debe mencionar que los valores de W1 y W2 no tienden a los valores adecuados, esto es debido a que el algoritmo no está orientado a la recursividad es decir a depender de valores anteriores de los parámetros, y si bien esto no se notó en el caso en el caso de primer orden, en este caso al aumentar el número de parámetros a estimar se ve la necesidad de recursividad para lograr que el error se minimice a un mayor grado y así obtener la tendencia adecuada de los parámetros estimados.

Parámetros Reales		Parámetros Estimados	
P1	-1.850624008809246	W1	-0.509196523811949
P2	0.858129721811394	W2	-0.483697701831646
P3	0.003848542843238	W3	0.006674238027292
P4	0.003657170158909	W4	0.009611858167576

Tabla 3.8. Comparación entre parámetros reales y parámetros estimados (ec. 3.22)

La Tabla 3.8 muestra una comparativa entre los parámetros reales y parámetros estimados del sistema de segundo orden, donde se observa que los valores no tienden a ser los adecuados. Es necesario probar otra función de transferencia de segundo orden para obtener una mejor referencia del algoritmo LMS para este caso.

Implementación y Simulación en Simulink

La Figura 3.13 muestra el diagrama de bloques para la simulación del sistema de identificación en línea LMS de un sistema de segundo orden, el diagrama posee los mismos elementos que el diagrama de la Figura 3.6 para el caso de primer orden, para este caso los scope ayudaran a la exportación de datos hacia el espacio de trabajo de Matlab para su análisis.

Cabe mencionar que para este caso se hará solo la simulación del proceso de segundo orden descrito por la ecuación 3.21, debido a que con el antecedente del rendimiento negativo del algoritmo LMS para sistemas de segundo orden obtenido de la simulación en Matlab, se da por descartado el algoritmo para el caso de segundo orden, pero vale la pena realizar la simulación en línea.

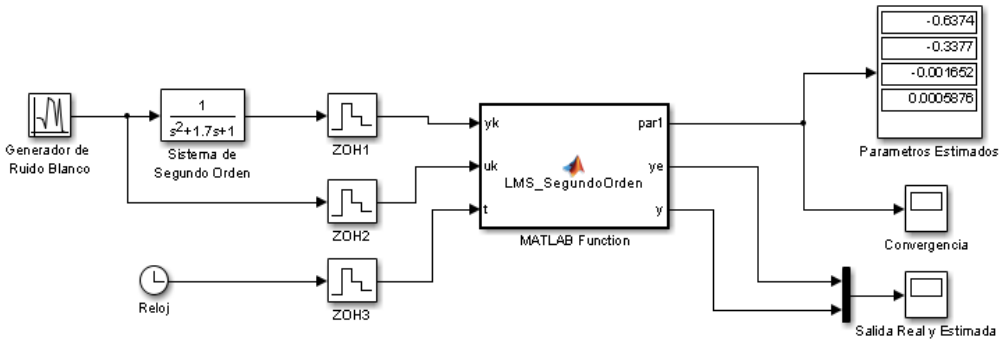

Figura 3.13. Diagrama de bloques de sistema de identificación en línea (ec. 3.21)

Los resultados obtenidos de la ejecución del diagrama en Simulink son los siguientes:

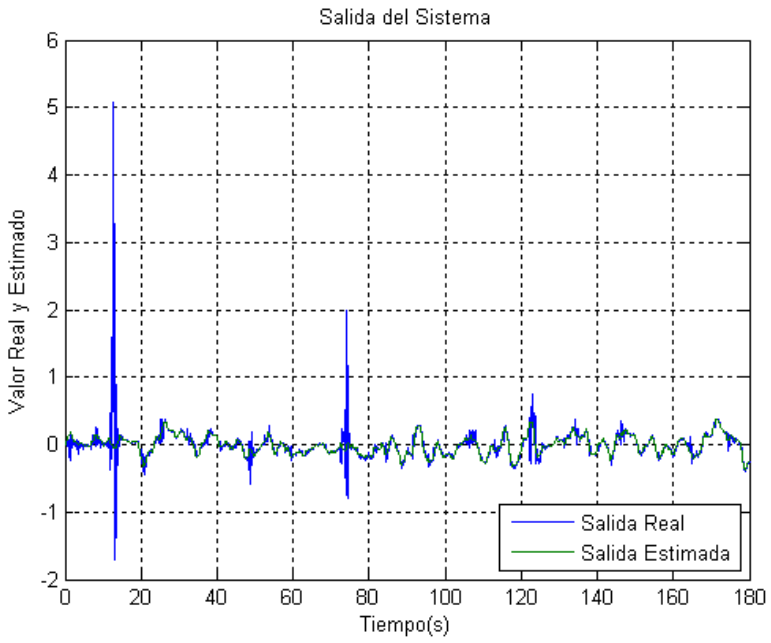

Figura 3.14. Salida del sistema, real y estimada (sist. En línea ec. 3.21)

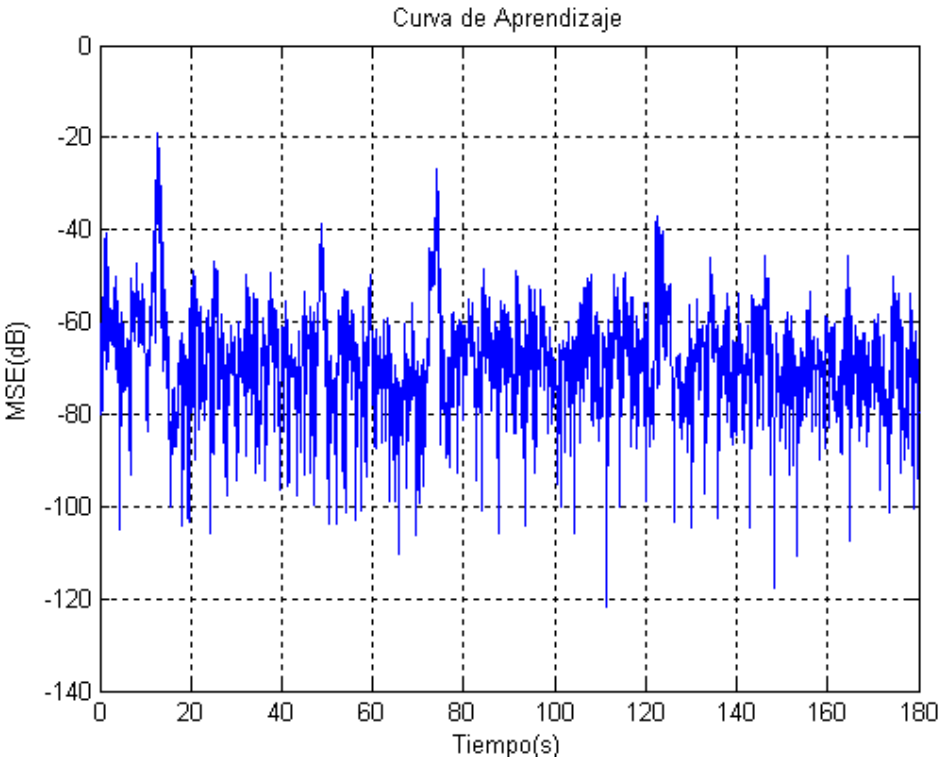

Figura 3.15. Curva de Aprendizaje (sist. En línea ec. 3.21)

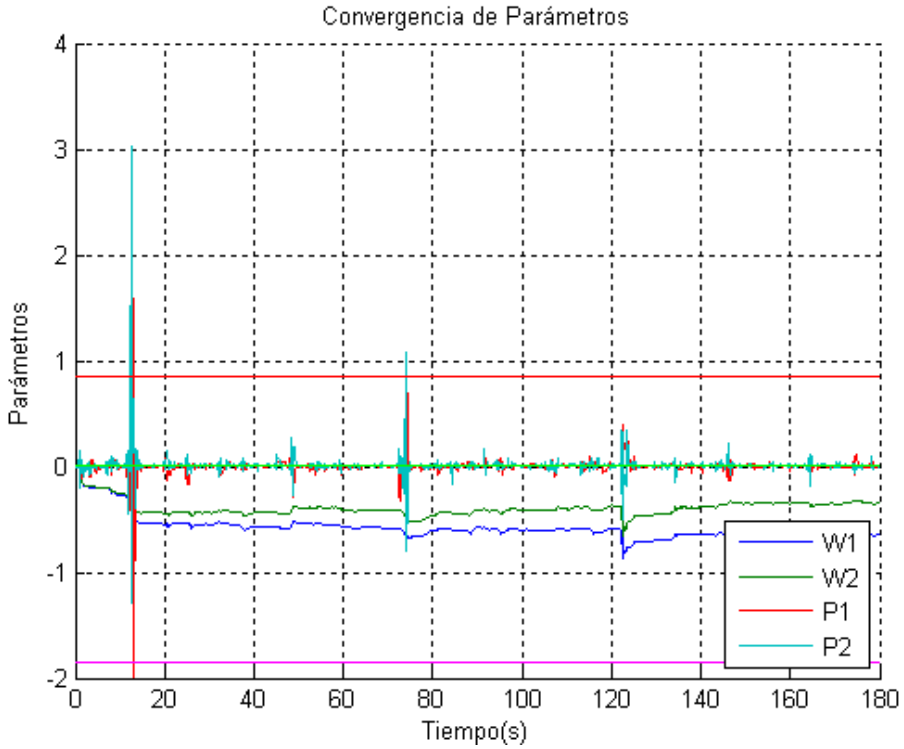

Figura 3.16. Convergencia de Parámetros (sist. En línea ec. 3.21)

La Figura 3.14 muestra inestabilidad en el sistema de identificación debido a que puede se puede observar que en ciertos tramos la salida estimada sigue a la salida real, pero en otros el valor estimado toma valores no relacionados a la salida real, esto se ve reflejado en la Figura 3.16 donde se ve que los parámetros no tienen una tendencia uniforme, y lo que es más, los valores de los parámetros estimados no se acercan a los valores reales.

Para descartar definitivamente el algoritmo Figura 3.15 muestra que el error cuadrático no tiende a minimizarse y más bien se nota una tendencia constante del MSE, por lo que no se cumple el objetivo del algoritmo LMS para sistemas de segundo orden en la minimización del error cuadrático.

La comparación realizada en la Tabla 3.9, nos da una idea del bajo rendimiento del algoritmo LMS para sistemas de segundo orden, imponiendo la necesidad de buscar un algoritmo recursivo para la estimación de los parámetros.

Parámetros Reales		Parámetros Estimados	
P1	-1.850624008809246	W1	-0.637424771490477
P2	0.858129721811394	W2	-0.337676058468429
P3	0.003848542843238	W3	-0.001652180253680
P4	0.003657170158909	W4	0.000587573595845

Tabla 3.9. Comparación entre parámetros (sist. en línea ec. 3.22)

3.4. IMPLEMENTACIÓN DEL ALGORITMO DE IDENTIFICACIÓN RLS

Ya se ha visto el desarrollo de matemático de las ecuaciones que rigen el algoritmo RLS en el epígrafe 2.3.4; y en el presente apartado se utilizará dichas ecuaciones como base para desarrollar el algoritmo que se debe implementar en el estimador RLS ya sea de primer o de segundo orden.

3.4.1. DESARROLLO DEL ALGORITMO RLS

Luego de haber obtenido la ecuación 2.7 en el epígrafe 2.3.4, se debe realizar el cálculo de K(k) y el de la matriz de covarianza $P(k)$ para la resolución de la ecuación reescrita a continuación:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)\left[y(k) - \emptyset(k)\hat{\theta}(k-1)\right] \tag{3.25}$$

Como el error de estimación viene dado por:

$$e(k) = y(k) - \emptyset(k)\hat{\theta}(k-1) \tag{3.26}$$

Reemplazando la ecuación 3.26 en la ecuación 3.25 se obtiene:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)e(k) \tag{3.27}$$

Donde K(k), denominada ganancia de adaptación, se calcula así:

$$K(k) = \frac{P(k-1)\emptyset(k)}{\lambda + \emptyset(k)'P(k-1)\emptyset(k)}$$

(3.28)

Para simplificar la ecuación 3.27 se hace el reemplazo $\mathrm{VK(k)} = \mathrm{P(k-1)\emptyset(k)}$ obteniendo lo siguiente:

$$K(k) = \frac{VK(k)}{\lambda + \emptyset\prime(k)VK(k)} \tag{3.29}$$

La llamada matriz de covarianza P(k) se calcula de manera iterativa dependiendo de sus valores anteriores con la siguiente ecuación:

$$P(k) = \frac{1}{\lambda}[P(k-1) - K(k)VK'(k)] \tag{3.30}$$

El valor de λ llamado factor de olvido debe escogerse intuitivamente, de acuerdo a la frecuencia o la velocidad con la que cambian los parámetros del sistema, una elección adecuada es de valores entre 0.9 y 1, teniendo en cuenta que los valores próximos a 0.9 corresponden a un olvido más rápido, es decir, deben escogerse cuando el sistema cambia de parámetros de una forma rápida, mientras que tomar el valor de 1 corresponde al caso de la regresión normal, sin olvido.

La Tabla 3.10 muestra paso a paso la manera de implementar el algoritmo RLS. La inicialización de la matriz de covarianza $P(k)$ se hace con una matriz identidad multiplicada por una ganancia α, es recomendable que α sea mucho mayor 1, aunque puede tomar cualquier valor.

Inicialización:

En k=0:

$\mathbf{0.9} \leq \boldsymbol{\lambda} \leq \mathbf{1}$
$\boldsymbol{\alpha} \gg \mathbf{1}$
$\boldsymbol{P}(\mathbf{0}) = \boldsymbol{\alpha I}$
$\widehat{\boldsymbol{\theta}}_{\boldsymbol{N}}(\mathbf{0}) = \mathbf{0}_{\boldsymbol{N}}$
$\emptyset_{\boldsymbol{N}}(\mathbf{0}) = \mathbf{0}_{\boldsymbol{N}}$

Operación:

Para k=1 en adelante:

1. Adquirir y(k) y u(k)

2. Para el error de estimación de 3.26.

$$\boldsymbol{e}(\boldsymbol{k}) = \boldsymbol{y}(\boldsymbol{k}) - \emptyset(\boldsymbol{k})\widehat{\boldsymbol{\theta}}(\boldsymbol{k}-\mathbf{1})$$

3. Calcular el vector de ganancia de 3.29

$$\mathbf{VK}(\mathbf{k}) = \mathbf{P}(\mathbf{k}-\mathbf{1})\emptyset(\mathbf{k})$$
$$\boldsymbol{K}(\boldsymbol{k}) = \frac{\boldsymbol{VK}(\boldsymbol{k})}{\boldsymbol{\lambda} + \emptyset'(\boldsymbol{k})\boldsymbol{VK}(\boldsymbol{k})}$$

4. Actualizar el estimador RLS de 3.27

$$\widehat{\boldsymbol{\theta}}(\boldsymbol{k}) = \widehat{\boldsymbol{\theta}}(\boldsymbol{k}-\mathbf{1}) + \boldsymbol{K}(\boldsymbol{k})\boldsymbol{e}(\boldsymbol{k})$$

5. Actualizar la matriz de covarianza para la siguiente iteración de 3.30

$$\boldsymbol{P}(\boldsymbol{k}) = \frac{\mathbf{1}}{\boldsymbol{\lambda}}[\boldsymbol{P}(\boldsymbol{k}-\mathbf{1}) - \boldsymbol{K}(\boldsymbol{k})\boldsymbol{VK}'(\boldsymbol{k})]$$

6. Actualizar el vector de datos

$$\emptyset(\boldsymbol{k}) = [-\boldsymbol{y}(\boldsymbol{k})\ \emptyset_{\mathbf{1}}(\boldsymbol{k}-\mathbf{1})\ \boldsymbol{u}(\boldsymbol{k})\ \ \emptyset_{\mathbf{3}}(\boldsymbol{k}-\mathbf{1})].$$

Lazo completo.
Regresar a (1).

Tabla 3.10. Algoritmo RLS

3.4.2. SIMULACIÓN DEL ALGORITMO DE IDENTIFICACIÓN RLS EN SISTEMAS DE PRIMER ORDEN

Implementación y Simulación en Matlab

Para la simulación del algoritmo se utilizan dos de las funciones de transferencia de primer orden descritas en la Tabla 3.2. Reescritas a continuación:

$$G(s) = \frac{1}{s+1} \tag{3.31}$$

$$G_a[z] = \frac{0.0861z^{-1}}{1-0.914z^{-1}} \tag{3.32}$$

Los resultados obtenidos de la simulación del sistema de identificación aplicando una entrada de ruido blanco a la función de transferencia descrita en la ecuación 3.31 son los siguientes:

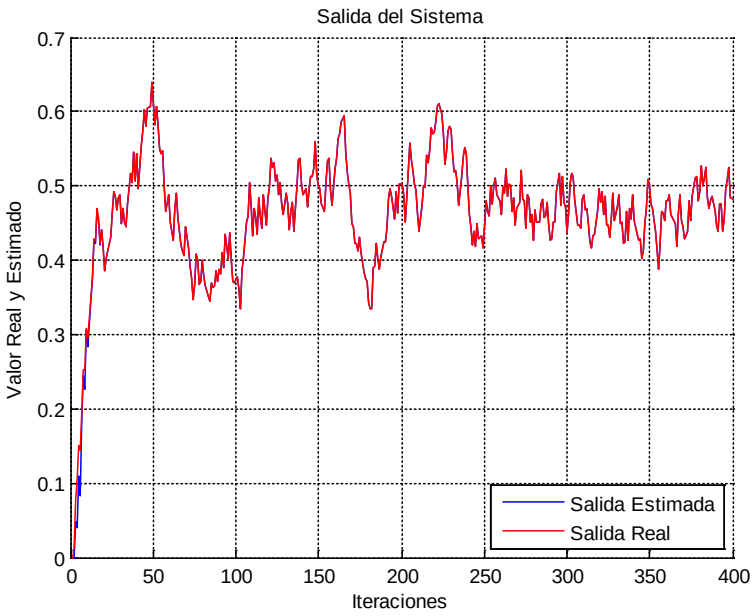

Figura 3.17. Salida del sistema, real y estimada (ec. 3.32)

La Figura 3.17 muestra gráficamente la comparación entre la salida real del sistema y la salida obtenida con los parámetros estimados por el identificador RLS, verificando la tendencia de los valores estimados hacia los valores reales.

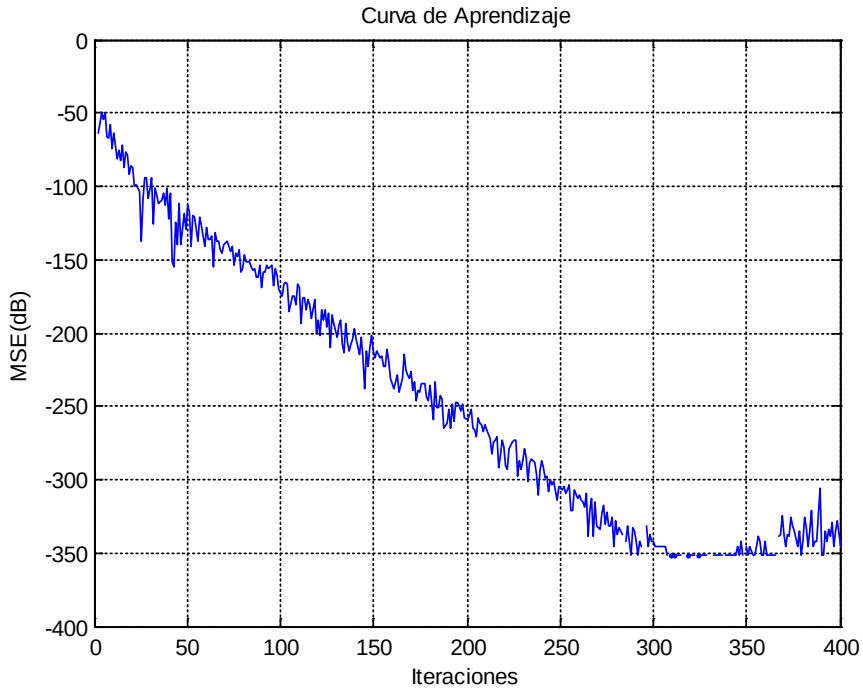

Figura 3.18. Curva de Aprendizaje (ec. 3.32)

La Figura 3.18 representa la curva de aprendizaje o la gráfica del MSE, en la gráfica se puede observar que el error desciende hasta establecerse en un valor constante donde el error tiende a ser cero.

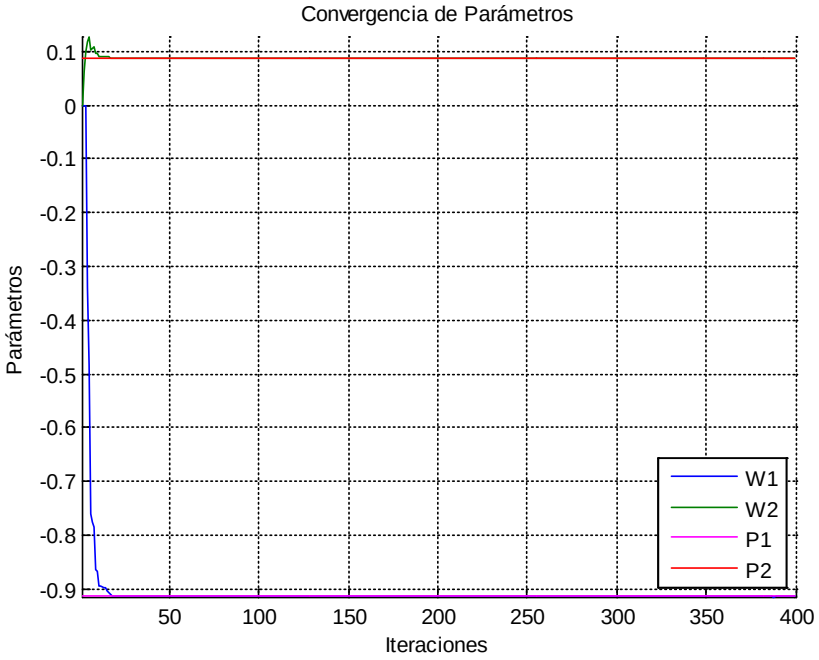

Figura 3.19. Convergencia de Parámetros (ec. 3.32)

La Figura 3.19 muestra la variación de los parámetros en cada iteración, W1 y W2 representan el valor que toman los parámetros estimados a través de las iteraciones, P1 y P2 representan el valor real de los parámetros de la función de transferencia establecida para la identificación. Se puede decir que los parámetros estimados convergen de una forma rápida a su valor real.

La Tabla 3.11 indica los valores finales de los parámetros estimados y de los parámetros reales, como comparación de los resultados de simulación.

Parámetros Reales		Parámetros Estimados	
P1	-0.913931185271228	W1	-0.913931185271228
P2	0.086068814728772	W2	0.086068814728772

Tabla 3.11. Comparación entre parámetros reales y parámetros estimados (ec. 3.32)

En primera instancia, según la Tabla 3.11; el estimador RLS funciona correctamente para sistemas lineales de primer orden, pero se debe tener en cuenta

que la identificación paramétrica en línea puede variar sus resultados dependiendo del estimador, con este precedente, se debe analizar el caso RLS en línea.

Implementación y Simulación en Simulink

Previo a la implementación final del sistema utilizando la tarjeta arduino, se debe simular el comportamiento de los estimadores, en este caso el del estimador RLS. En la Figura 3.20. Se muestra el diagrama de bloques del sistema listo para simulación en línea, desarrollado en simulink, para este caso lo único que se ha cambiado es la función de Matlab donde se hace la estimación en los parámetros, esto en comparación a los diagramas utilizados para el caso del algoritmo LMS, es necesario tener en cuenta que para el algoritmo RLS también se necesita la excitación persistente del sistema provocada por una entrada de ruido blanco.

Para la simulación se utiliza la función de transferencia de la ecuación 3.31, extraída de la Tabla 3.2, para simular un entorno en línea del sistema de identificación de funciones de transferencia de primer orden, basado en el algoritmo RLS.

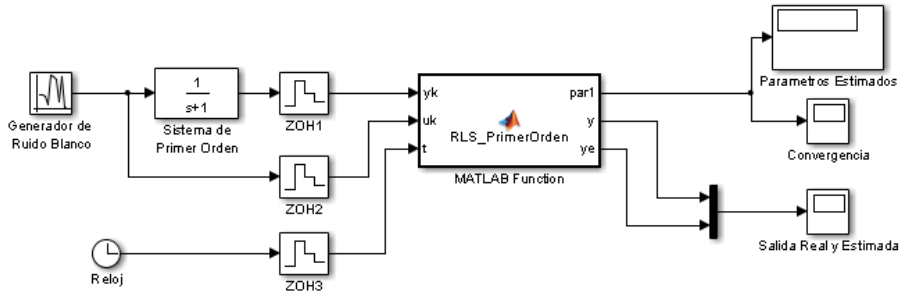

Figura 3.20. Diagrama de bloques de sistema de identificación en línea (ec. 3.31)

La Figura 3.21 muestra una comparación entre la salida real y la salida estimada del sistema de identificación en línea RLS implementado en simulink, donde se puede observar la rápida tendencia del valor estimado hacia el valor real, llegando a sobreponerse en la gráfica.

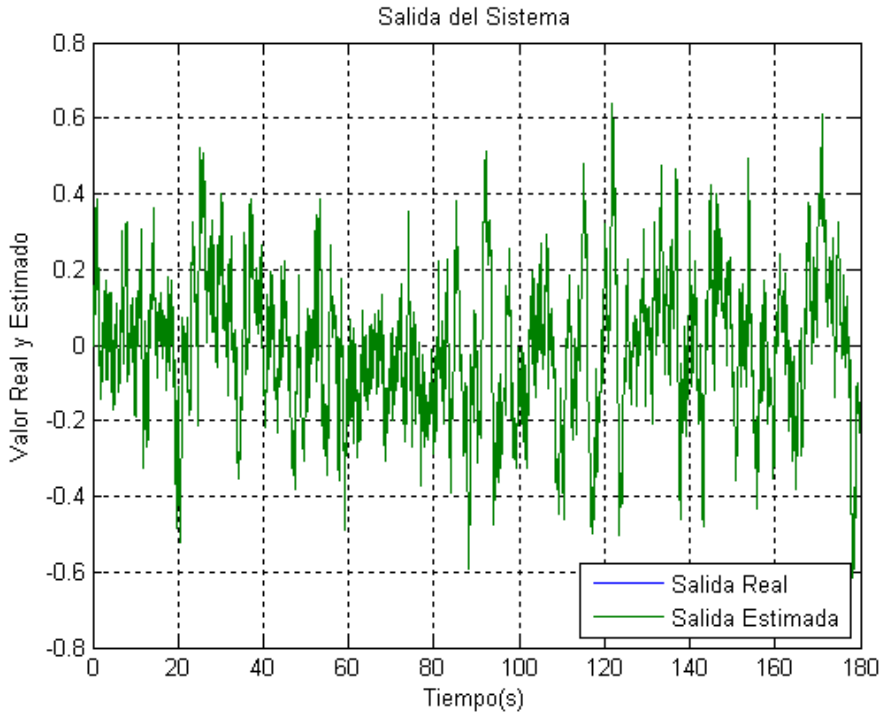

Figura 3.21. Salida del sistema, real y estimada (sist. En línea ec. 3.31)

La gráfica de la Figura 3.22 muestra la tendencia de la curva de aprendizaje a minimizar el error en dB a través del tiempo transcurrido de la simulación, se puede ver el descenso del valor del error cuadrático, lo que se ve reflejado en la Figura 3.23 donde se puede observar la evolución del valor de los parámetros estimados (W1 y W2) del sistema de primer orden, con una tendencia hacia los valores reales del sistema (P1 y P2) que finalmente igualan sus valores respectivos.

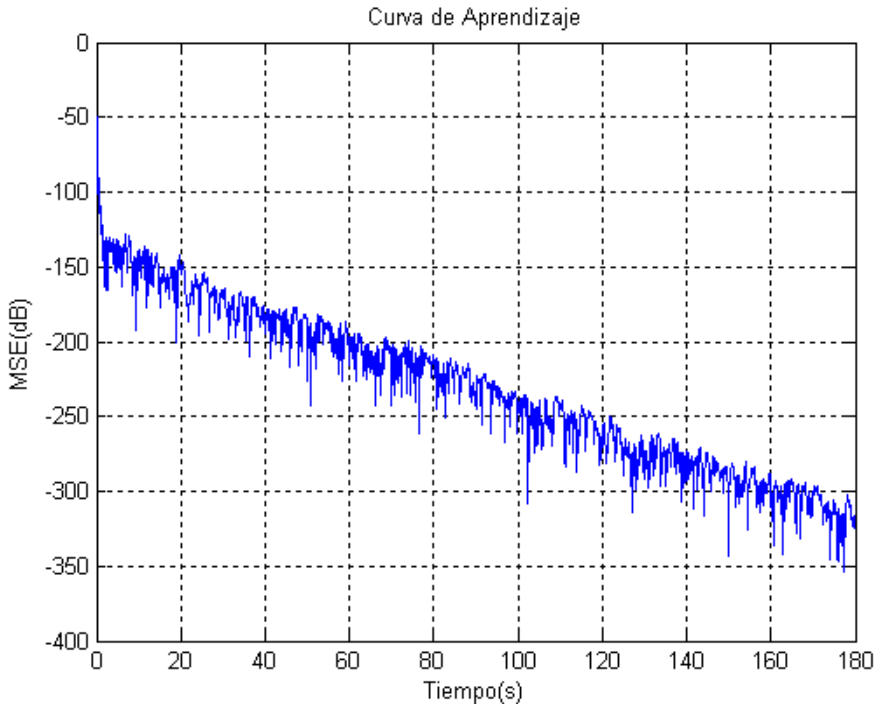

Figura 3.22. Curva de Aprendizaje (sist. En línea ec. 3.31)

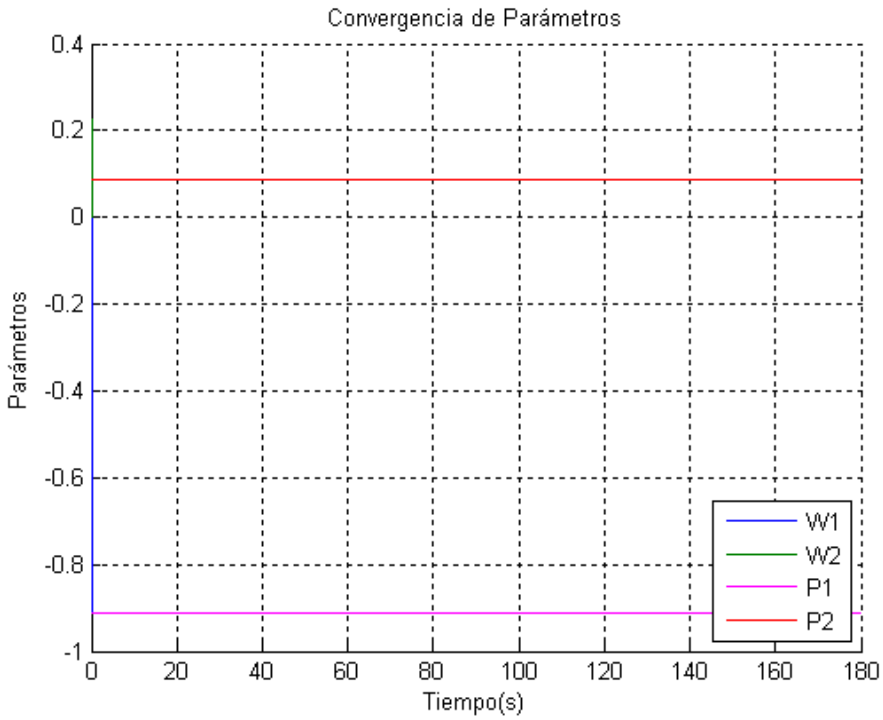

Figura 3.23. Convergencia de Parámetros (sist. En línea ec. 3.31)

Parámetros Reales		Parámetros Estimados	
P1	-0.913931185271228	W1	-0.913928499999971
P2	0.086068814728772	W2	0.0860715

Tabla 3.12. Comparación entre parámetros reales y estimados (sist. en línea ec. 3.32)

En la Tabla 3.12 se realiza una comparación entre los parámetros reales del sistema y los parámetros obtenidos de la estimación del algoritmo RLS implementado en Simulink, donde se observa que la tendencia a igualar a los parámetros reales.

3.4.3. SIMULACIÓN DEL ALGORITMO DE IDENTIFICACIÓN RLS EN SISTEMAS DE SEGUNDO ORDEN

Implementación y Simulación en Matlab

Para esta simulación se utiliza la función de transferencia de segundo orden en tiempo continuo y discreto descritas en la Tabla 3.4, las cuales están reescritas a continuación:

$$G(s) = \frac{1}{s^2 + 1.7s + 1} \tag{3.33}$$

$$G_a[z] = \frac{0.003849z^{-1} + 0.003657z^{-2}}{1 - 1.851z^{-1} + 0.8581z^{-2}} \tag{3.34}$$

Los resultados obtenidos de la simulación del sistema de identificación aplicando una entrada de ruido blanco a la función de transferencia descrita en la ecuación 3.33 son los siguientes:

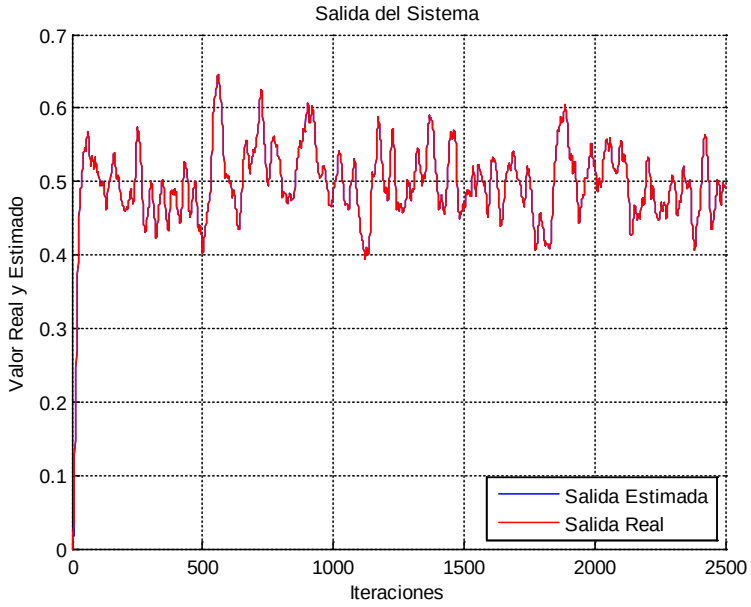

Figura 3.24. Salida del sistema, real y estimada (ec. 3.34)

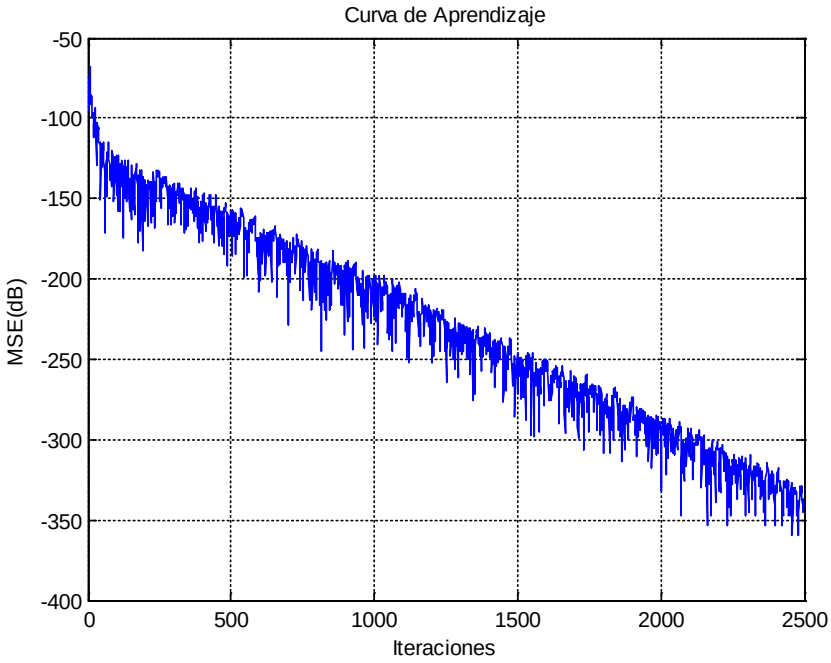

Figura 3.25. Curva de Aprendizaje (ec. 3.34)

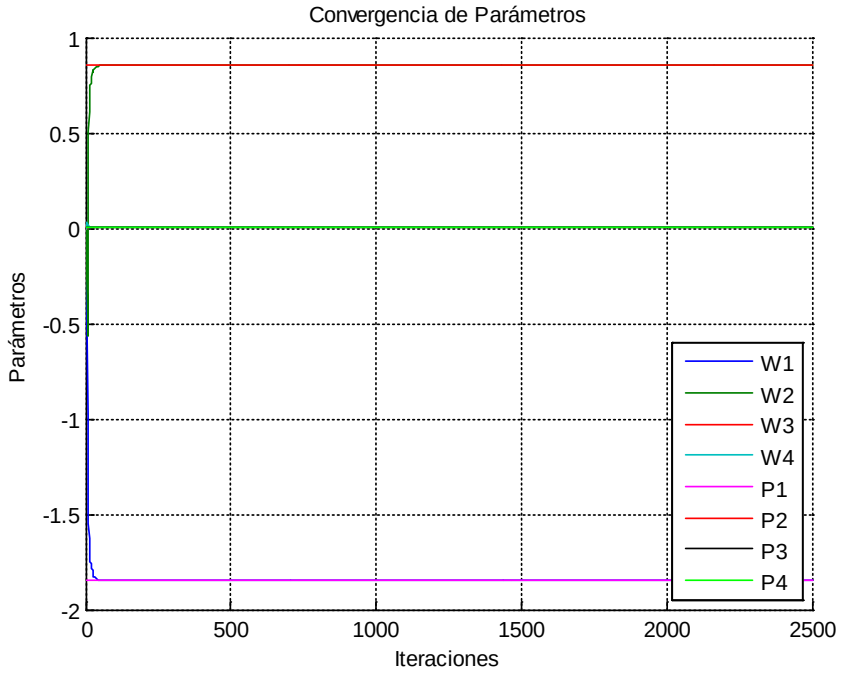

Figura 3.26. Convergencia de Parámetros (ec. 3.34)

Para el análisis de los resultados de la simulación en Matlab se han realizado tres gráficas: la primera (Figura 3.24) muestra la comparación entre la salida real del sistema y la salida estimada por el algoritmo RLS, en dicha gráfica se observa que con el paso de las iteraciones la salida estimada iguala a la salida real. En la segunda gráfica (Figura 3.25) se muestra la evolución de la curva de aprendizaje, en la que se puede observar la minimización del error con el paso de las iteraciones.

Por último la tercera gráfica obtenida como resultado de la simulación (Figura 3.26) indica la tendencia que toman los parámetros estimados (W1, W2, W3, W4), en relación a los parámetros reales (P1, P2, P3, P4), se puede observar una rápida tendencia de los parámetros estimados hacia los valores reales, cabe mencionar que para el caso del algoritmo LMS para sistemas de segundo orden, esta convergencia

no se dio, por lo que el algoritmo RLS muestra notables ventajas en la velocidad de convergencia y el valor que toman los parámetros estimados.

Parámetros Reales		Parámetros Estimados	
P1	-1.850624008809246	W1	-1.850624008808909
P2	0.858129721811394	W2	0.858129721811057
P3	0.003848542843238	W3	0.003848542843238
P4	0.003657170158909	W4	0.003657170158910

Tabla 3.13. Comparación entre parámetros reales y parámetros estimados (ec. 3.34)

La Tabla 3.13 muestra los valores finales de los parámetros estimados y los valores reales que deben tomar, y como se observa en la Figura 3.26 los valores tienden a ser iguales respectivamente, cumpliendo el objetivo del algoritmo RLS para la estimación paramétrica de sistemas lineales.

Implementación y Simulación en Simulink

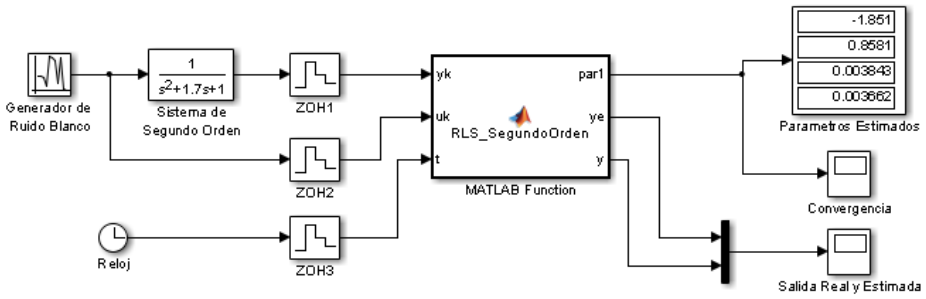

Figura 3.27. Diagrama de bloques de sistema de identificación en línea (ec. 3.33)

La Figura 3.27 muestra el diagrama de bloques implementado para la simulación del sistema de identificación en línea RLS de un sistema de segundo orden, el diagrama posee los mismos elementos que los diagramas de Simulink anteriores, el cambio se encuentra en el bloque función de Matlab, en el cual se encuentra el algoritmo del estimador RLS para sistemas de segundo orden, la función de

transferencia utilizada es la mostrada en la ecuación 3.33, obteniendo los siguientes resultados:

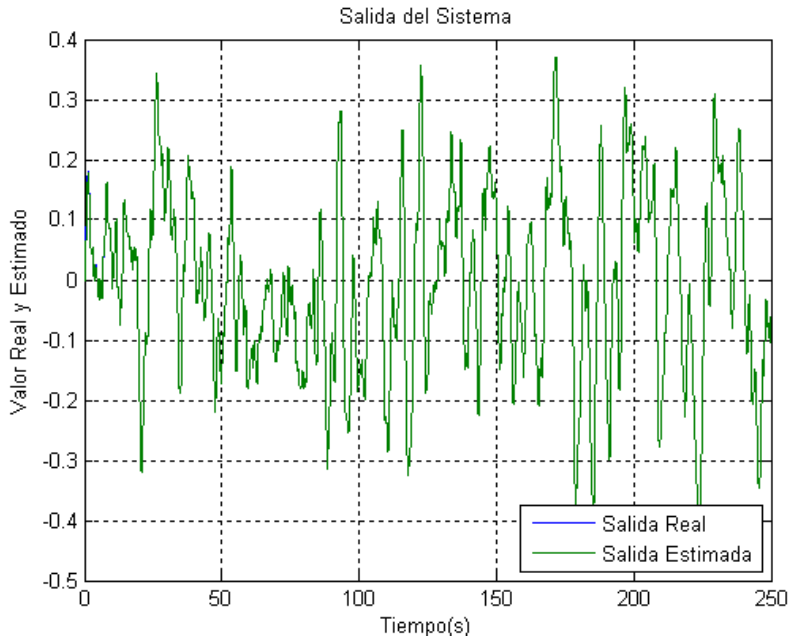

Figura 3.28. Salida del sistema, real y estimada (sist. En línea ec. 3.33)

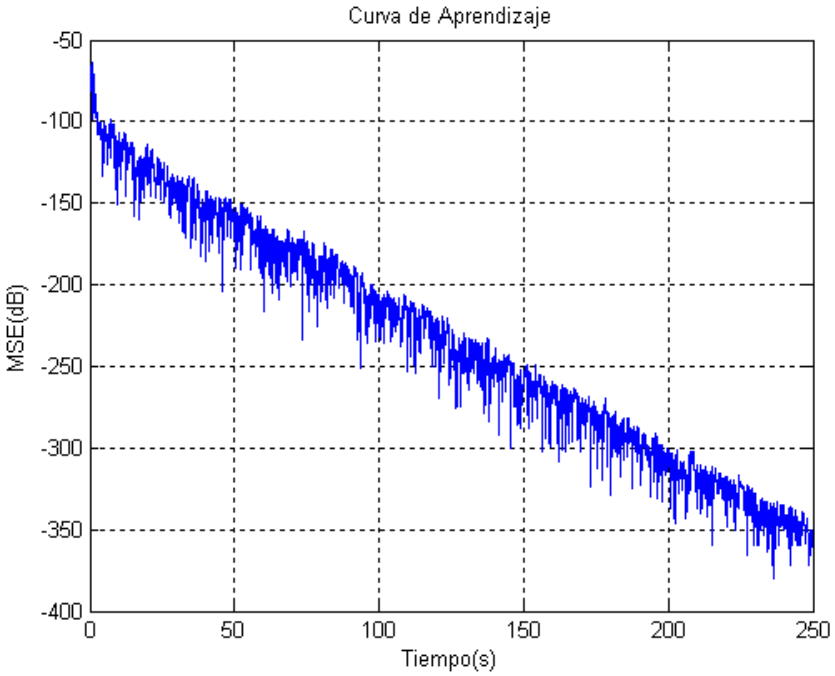

Figura 3.29. Curva de Aprendizaje (sist. En línea ec. 3.33)

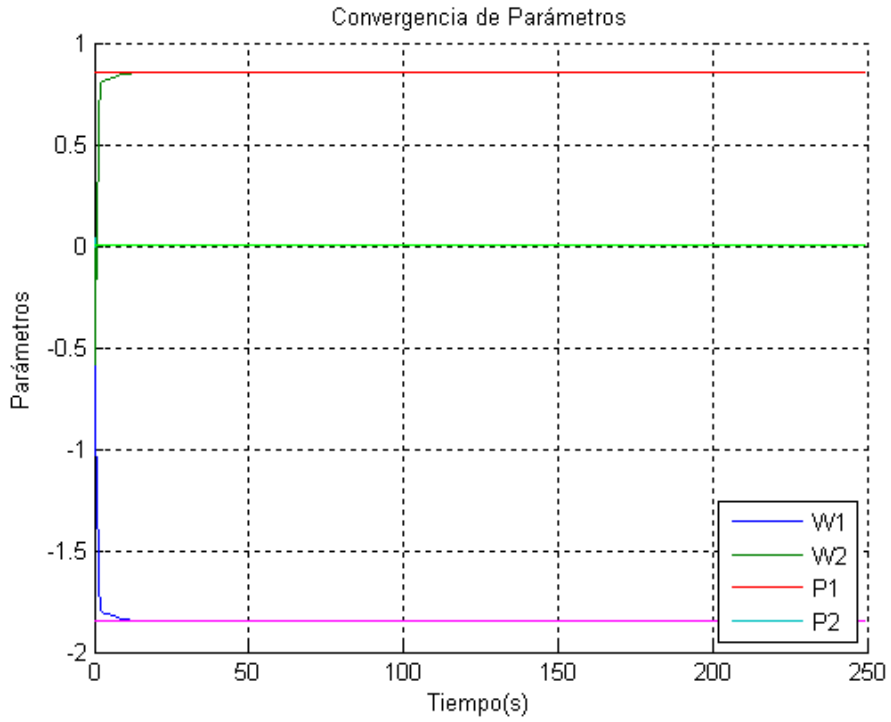

Figura 3.30. Convergencia de Parámetros (sist. En línea ec. 3.33)

La Figura 3.28 muestra el buen funcionamiento del estimador RLS donde se observa que la salida estimada sigue fielmente a la salida real del sistema de segundo orden. La curva de aprendizaje de la Figura 3.29 muestra la minimización del error cuadrático (MSE) a través del tiempo, con lo que se cumple el objetivo del algoritmo para sistemas de segundo orden y mostrando una ventaja sobre el algoritmo LMS en velocidad y convergencia de parámetros como se muestra en la Figura 3.30 donde los valores de W1, W2, W3 y W4 tienden a los valores de P1, P2, P3 y P4 respectivamente, verificando el buen funcionamiento del estimador implementado en línea con el algoritmo RLS.

La comparación realizada en la Tabla 3.14, muestra la eficiencia de la versión recursiva del algoritmo de mínimos cuadrados, donde los parámetros estimados

tienden a ser iguales a los valores reales del sistema de segundo orden de la ecuación 3.34.

Parámetros Reales		Parámetros Estimados	
P1	-1.850624008809246	W1	-1.850627220499987
P2	0.858129721811394	W2	0.858132883384738
P3	0.003848542843238	W3	0.00384345
P4	0.003657170158909	W4	0.003662212884750

Tabla 3.14. Comparación entre parámetros reales y parámetros estimados

3.4.4. IMPLEMENTACIÓN DEL ALGORITMO DE IDENTIFICACIÓN RLS PARA SISTEMAS DE MÚLTIPLES ENTRADAS Y SALIDAS (MIMO)

Una vez desarrollados tres tipos de algoritmos para la identificación de procesos industriales, que por defecto se establecieron de tipo SISO (Single-Input/Single-Output), se debe seleccionar el algoritmo más adecuado para la identificación de sistemas MIMO (Multiple-Input/ Multiple -Output), de esta manera, ampliar el uso del algoritmo escogido.

El algoritmo LMS, resulta ineficiente para sistemas de segundo orden en línea, lo que lo descarta para realizar la identificación de un sistema más complejo. El algoritmo LS-Lattice se lo diseño para que identifique los parámetros del denominador de la función de transferencia que representa al sistema, por lo que si se intenta aumentar el número de parámetros a identificar, el algoritmo automáticamente va a asumir que es un sistema SISO y lo hará que los parámetros

estimados se agrupen en el denominador del sistema. De esta forma se ha determinado que el algoritmo de identificación RLS resulta ser el más adecuado para la identificación de sistemas MIMO en línea.

El sistema MIMO a identificar va a ser del modelo de la Figura 3.31, el cual posee dos entradas que son filtradas por cuatro funciones de transferencia distintas, produciendo dos salidas.

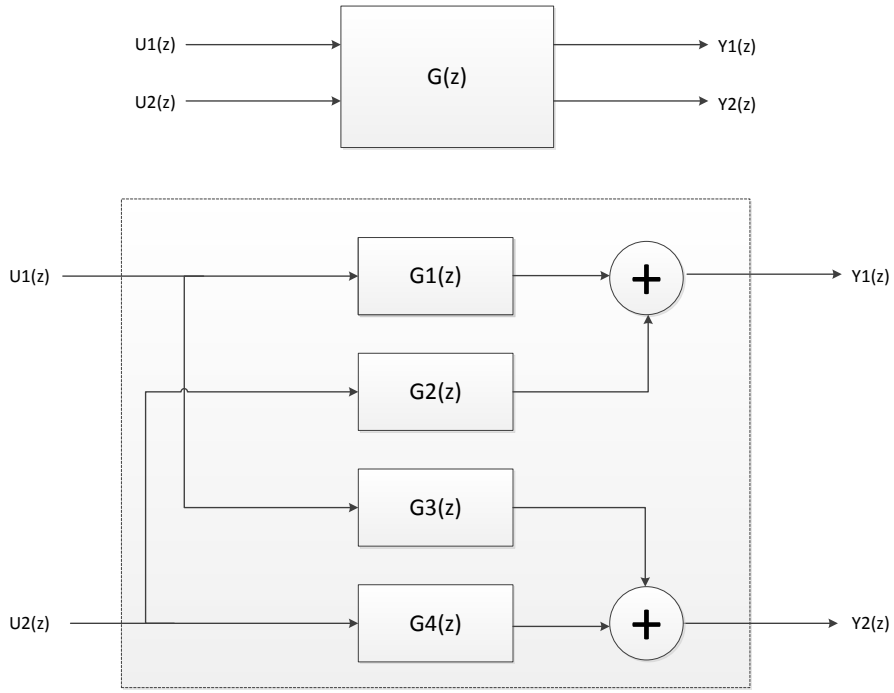

Figura 3.31. Sistema MIMO a identificar

Simulación del Algoritmo de Identificación RLS en Sistemas MIMO

El algoritmo RLS resumido en la Tabla 3.10, debe ser modificado para que el vector de parámetros se convierta en matriz, para el efecto se toma en cuenta que se tiene un vector de salidas y uno de entradas, para el caso cada vector tiene dos elementos.

$$\boldsymbol{Y} = [y_1 \ y_2] \quad \boldsymbol{U} = [u_1 \ u_2]$$

Para desarrollar una ecuación matricial del sistema, se utiliza el diagrama de la Figura 3.31, de donde se puede desprender las ecuaciones a diferencias de ambas salidas en función de las entradas, las funciones de transferencia que componen el sistema para el caso de primer orden tienen la siguiente forma:

$$G_1(z) = \frac{k_1}{1 + k_2 z^{-1}}$$

$$G_2(z) = \frac{k_3}{1 + k_2 z^{-1}}$$

$$G_3(z) = \frac{k_4}{1 + k_5 z^{-1}}$$

$$G_4(z) = \frac{k_6}{1 + k_5 z^{-1}}$$

De la Figura 3.31 se puede obtener la ecuación a diferencias para ambas salidas de la siguiente manera:

$$y_1(k) = G_1(z)u_1(k) + G_2(z)u_2(k) \tag{3.35}$$

$$y_2(k) = G_3(z)u_1(k) + G_4(z)u_2(k) \tag{3.36}$$

Para la inicialización de las matrices que son parte del algoritmo RLS, se necesita saber las dimensiones que deben tener, para el efecto, se hace necesario desarrollar una ecuación matricial del sistema. A partir de las ecuaciones 3.35 y 3.36 se obtiene la siguiente ecuación:

$$\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} k_1 k_3 k_2 \ 0 \\ k_4 k_6 \ 0 \ k_5 \end{bmatrix} \begin{bmatrix} u_1(k) \\ u_2(k) \\ y_1(k-1) \\ y_2(k-1) \end{bmatrix} \tag{3.37}$$

De la ecuación 3.37 se puede concluir que, el vector de parámetros $\hat{\theta}(k)$ se convierte en matriz de 2x8 dimensiones, y también se obtienen los datos que actualizan el vector de regresión $\emptyset(k)$.

Una vez modificado el algoritmo para que las dimensiones matriciales concuerden en las operaciones de la Tabla 3.10 se realiza la simulación del algoritmo, el diagrama de bloques mostrado en la Figura 3.32 representa el sistema MIMO, al que se le han asignado valores aleatorios de parámetros para su uso dentro de la simulación, el algoritmo de identificación está dentro del bloque función de Matlab, con las entradas y salidas requeridas.

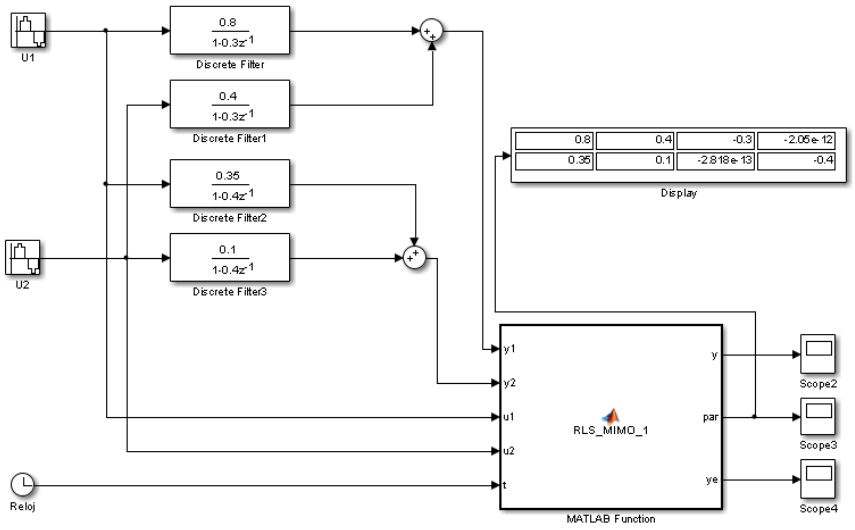

Figura 3.32. Simulación del Algoritmo RLS para identificación de un sistema MIMO

Una vez ejecutada la simulación se obtiene la matriz de parámetros de acuerdo a la ecuación 3.37, obteniendo la convergencia mostrada en la Figura 3.33.

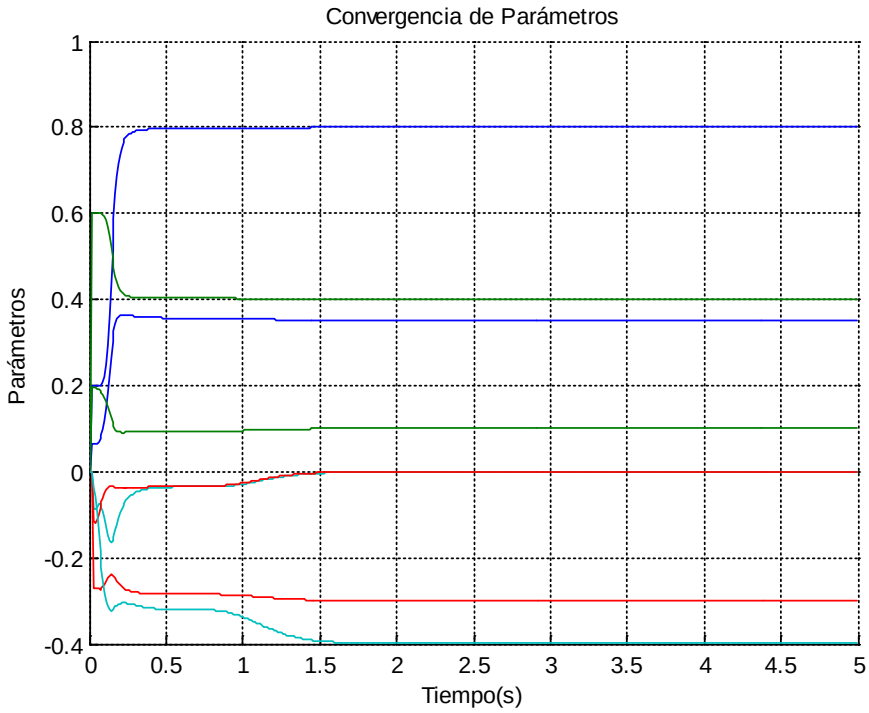

Figura 3.33. Convergencia de Parámetros

Para una comparación más objetiva, la Tabla 3.15 muestra los valores reales del sistema y los parámetros estimados con el algoritmo RLS.

Parámetros Reales		Parámetros Estimados	
K1	0.8	W1	0.79999923103556
K2	-0.3	W2	-0.299998642560705
K3	0.4	W3	0.40000011685338
K4	0.35	W4	0.350000820320797
K5	-0.4	W5	-0.399992852792866
K6	0.1	W6	0.099999512558119

Tabla 3.15. Comparación parámetros reales y parámetros estimados

3.5. IMPLEMENTACIÓN DEL ALGORITMO DE IDENTIFICACIÓN LS LATTICE

Este apartado muestra el desarrollo matemático del algoritmo LS LATTICE adaptado a la identificación en línea, como un método alterno a los algoritmos LMS y RLS. Este algoritmo es básicamente una estructura en cascada para la identificación, basado en filtros de predicción forward y backward. El algoritmo se basa en el desarrollo de mínimos cuadrados para reducir el error cuadrático, de los filtros mencionados anteriormente, e ir actualizando los coeficientes de reflexión en función de tiempo y así calcular los parámetros del sistema identificado, el orden del sistema va a determinar el número de etapas que tendrá el identificador colocados en cascada.

3.5.1. DESARROLLO DEL ALGORITMO LS LATTICE

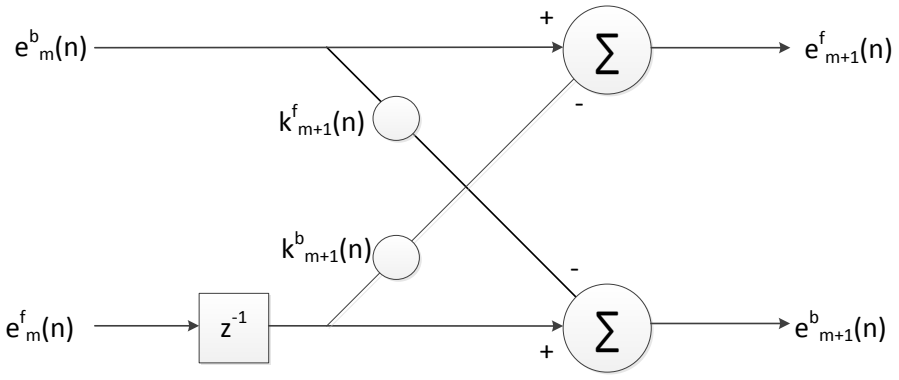

Figura 3.34. Etapa del estimador LS Lattice

La Figura 3.34 muestra el diagrama de bloques que ilustra como la estructura lattice calcula los errores obtenidos de los estimadores forward y backward, cabe mencionar que para el uso de la estructura lattice como filtro estimador, solo es necesario actualizar los coeficientes de reflexión $k^{f}_{m+1}(n)$ y $k^{b}_{m+1}(n)$, dentro de un estimador de m etapas cada instante n de tiempo discreto, dichos coeficientes se utilizan para calcular los errores de predicción forward y backward para la siguiente etapa, las etapas de la estructura lattice vienen conectadas en cascada tal y como se muestra en la Figura 3.35.

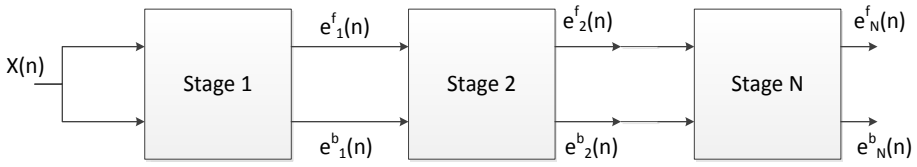

Figura 3.35. Cascada de etapas para formar un filtro de estimación lineal de orden N.

Para realizar el cálculo de los errores de predicción forward y backward se utiliza el diagrama de bloques de la Figura 3.34, del que se obtiene las siguientes formulas:

$$e^{f}_{m+1}(n) = e^{f}_{m}(n) - k^{b}_{m+1}e^{b}_{m}(n-1) \tag{3.38}$$

$$e^{b}_{m+1}(n) = e^{b}_{m}(n-1) - k^{f}_{m+1}e^{f}_{m}(n) \tag{3.39}$$

Dónde:

e^{f}_{m}: Error de predicción forward.

e_m^b: Error de predicción backward.

k_m^f y k_m^b: Coeficientes de reflexión.

m: Representa la etapa actual en la que se está realizando el cálculo.

Los coeficientes de reflexión vienen dados por:

$$k_{m+1}^b(n) = \frac{\Delta_{m+1}(n)}{\varepsilon_m^b(n-1)} \tag{3.40}$$

$$k_{m+1}^f(n) = \frac{\Delta_{m+1}(n)}{\varepsilon_m^f(n)} \tag{3.41}$$

El producto punto es muy importante para el desarrollo del estimador LS Lattice, también llamado coeficiente de correlación parcial $\Delta_{\mathrm{m+1}}(\mathrm{n})$ entre los errores de predicción forward y backward, como se muestra en la ecuación (3.42).

$$\Delta_{\mathrm{m+1}}(\mathrm{n}) = \langle z^{-1} e_m^b(n).\, e_m^f(n) \rangle \tag{3.42}$$

Para acoplar el cálculo del coeficiente de correlación parcial al algoritmo de identificación, se debe buscar una alternativa recursiva para su cálculo, que vendrá dada por:

$$\Delta_{\mathrm{m+1}}(\mathrm{n}) = \Delta_{\mathrm{m+1}}(\mathrm{n}-1) + \frac{e_m^f(n) e_m^b(n-1)}{\gamma_m(n-1)} \tag{3.43}$$

La forma recursiva depende directamente del ángulo formado por el producto vectorial entre los errores de predicción $\gamma_{\mathrm{m}}(\mathrm{n}-1)$, y se puede calcular con la ecuación (3.44).

$$\gamma_{m+1}(n-1) = \gamma_m(n-1) - \frac{[e_m^b(n-1)]^2}{\varepsilon_m^b(n-1)} \tag{3.44}$$

Para el cálculo de los coeficientes de reflexión; ec. 3.40 y ec. 3.41, y para el cálculo del ángulo en la ec. 3.44, aparecen dos nuevos términos $\varepsilon_m^b(n-1)$ y $\varepsilon_m^f(n)$ que son los errores cuadráticos BPE y FPE, físicamente representan la energía en cada etapa del estimador.

$$\varepsilon_m^f(n) = \langle e_m^f(n), e_m^f(n) \rangle \tag{3.45}$$

$$\varepsilon_m^b(n) = \langle e_m^b(n), e_m^b(n) \rangle \tag{3.46}$$

Para la actualización de los errores cuadráticos dentro de cada iteración del algoritmo de identificación y dentro de cada etapa se utiliza las siguientes fórmulas:

$$\varepsilon_{m+1}^f(n) = \varepsilon_m^f(n) - \frac{\Delta_{m+1}^2(n)}{\varepsilon_m^b(n-1)} \tag{3.47}$$

$$\varepsilon_{m+1}^b(n) = \varepsilon_m^b(n-1) - \frac{\Delta_{m+1}^2(n)}{\varepsilon_m^f(n)} \tag{3.48}$$

La Tabla 3.16 muestra paso a paso la manera de implementar el algoritmo de identificación LS Lattice.

Inicialización:

En n=0:

$$e_m^b(0) = \Delta_m(0) = 0$$
$$\gamma_m(0) = 1$$
$$\varepsilon_m^f(0) = \varepsilon_m^b(0) = \delta$$

Operación:

Para n=1 en adelante:

1. Inicializar los valores de la etapa 0 del estimador para cada iteración

$$e_0^b(n) = e_0^f(n) = x(n)$$
$$\varepsilon_0^f(n) = \varepsilon_0^b(n) = \varepsilon_0^b(n-1) + x^2(n)$$
$$\gamma_0(n) = 1$$

Calcular los parámetros para cada etapa m representa la etapa actual y N el número de etapas:

Para $0 \leq m \leq N-1$

2. Calculo del coeficiente de correlación parcial de ec. 3.43

$$\Delta_{m+1}(n) = \Delta_{m+1}(n-1) + \frac{e_m^f(n) e_m^b(n-1)}{\gamma_m(n-1)}$$

3. Calculo del error Forward de ec. 3.38 y ec. 3.40

$$e_{m+1}^f(n) = e_m^f(n) - \frac{\Delta_{m+1}(n)}{\varepsilon_m^b(n-1)} e_m^b(n-1)$$

4. Calculo del error Backward de ec. 3.39 y ec. 3.41

$$e_{m+1}^b(n) = e_m^b(n-1) - \frac{\Delta_{m+1}(n)}{\varepsilon_m^f(n)} e_m^f(n)$$

5. Calculo del Error cuadrático Forward de ec. 3.47

$$\boldsymbol{\varepsilon_{m+1}^{f}(n) = \varepsilon_{m}^{f}(n) - \frac{\Delta_{m+1}^{2}(n)}{\varepsilon_{m}^{b}(n-1)}}$$

6. Calculo del Error cuadrático Backward de ec. 3.48

$$\boldsymbol{\varepsilon_{m+1}^{b}(n) = \varepsilon_{m}^{b}(n-1) - \frac{\Delta_{m+1}^{2}(n)}{\varepsilon_{m}^{f}(n)}}$$

7. Calculo del Angulo de ec. 3.44

$$\boldsymbol{\gamma_{m+1}(n-1) = \gamma_{m}(n-1) - \frac{\left[e_{m}^{b}(n-1)\right]^{2}}{\varepsilon_{m}^{b}(n-1)}}$$

Etapa finalizada Regresar a (2) hasta completar todas las etapas

Lazo completo.
Regresar a (1).

Tabla 3.16. Algoritmo LS Lattice

3.5.2. SIMULACIÓN DEL ALGORITMO LS LATTICE PARA IDENTIFICACIÓN DE SISTEMAS

Implementación y Simulación en Matlab

Debido a que el diseño del algoritmo solo permite identificar los valores de las constantes que se encuentran en el denominador de la función de transferencia $G_a[z]$, no se puede hacer uso de las funciones desarrolladas en la Tabla 3.2 y en la Tabla 3.4, para este caso se va a proponer directamente la función de trasferencia discreta $G_a[z]$, para el efecto será:

$$G_a[z] = \frac{1}{1-2z^{-1}+z^{-2}} \tag{3.49}$$

Se justifica el uso de una función de transferencia discreta de segundo orden, debido a que esto permite mostrar directamente la respuesta que va a tener el algoritmo ante una función un tanto compleja.

Para obtener la respuesta de la función de transferencia de la ecuación (3.49), se utiliza el modelo en Simulink mostrado en la Figura 3.36.

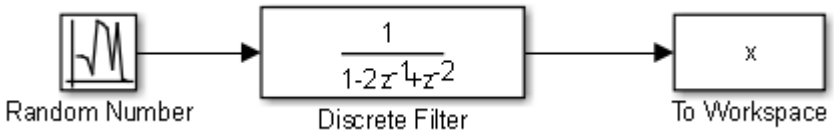

Figura 3.36. Simulación del sistema discreto para LS-Lattice

La variable x mostrada en la Figura 3.36 representa la respuesta del sistema ante una entrada de ruido blanco, esta señal es exportada al workspace de Matlab, para utilizarse en el algoritmo LS-Lattice desarrollado a partir de la Tabla 3.16, y la convergencia de las variables identificadas es la siguiente con respecto a los valores reales se muestra en Figura 3.37:

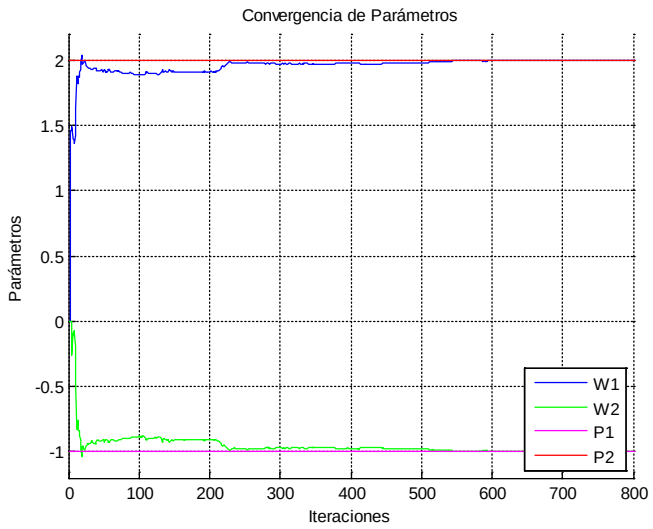

Figura 3.37. Convergencia de Parámetros (ec. 3.46)

La Tabla 3.17 indica los valores finales de los parámetros estimados y de los parámetros reales, como comparación de los resultados de simulación.

Parámetros Reales		Parámetros Estimados	
P1	2	W1	1.999985651466035
P2	-1	W2	-0.999985650383154

Tabla 3.17. Comparación entre parámetros reales y parámetros estimados (ec. 3.46)

Se puede observar la excelente aproximación que se obtiene entre los parámetros estimados y los reales, para un sistema de segundo orden discreto, se asume de esta manera, que para sistemas de primer orden se obtendrá el mismo o un mejor resultado.

Implementación y Simulación en Simulink

Previo a la implementación final del sistema utilizando la tarjeta arduino, se debe simular el comportamiento de los estimadores, en este caso el del estimador LS-Lattice. En la Figura 3.38. Se muestra el diagrama de bloques del sistema listo para simulación en línea, desarrollado en Simulink, se utiliza directamente un sistema discreto de segundo orden, tal y como se desarrolló en Matlab, debido a que el algoritmo no está diseñado para calcular la salida estimada de la planta, solo se hace referencia a la forma en que convergen los parámetros estimados en comparación a los reales, no se puede calcular un MSE y hacer una comparación con los algoritmos anteriores.

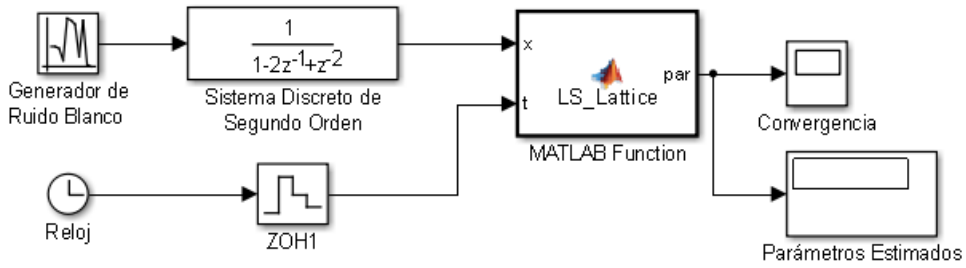

Figura 3.38. Diagrama de bloques de sistema de identificación en línea (ec. 3.46)

La Figura 3.39 se puede observar la evolución del valor de los parámetros estimados (W1 y W2) del sistema de segundo orden discreto, con una tendencia hacia los valores reales del sistema (P1 y P2) que finalmente igualan sus valores respectivos.

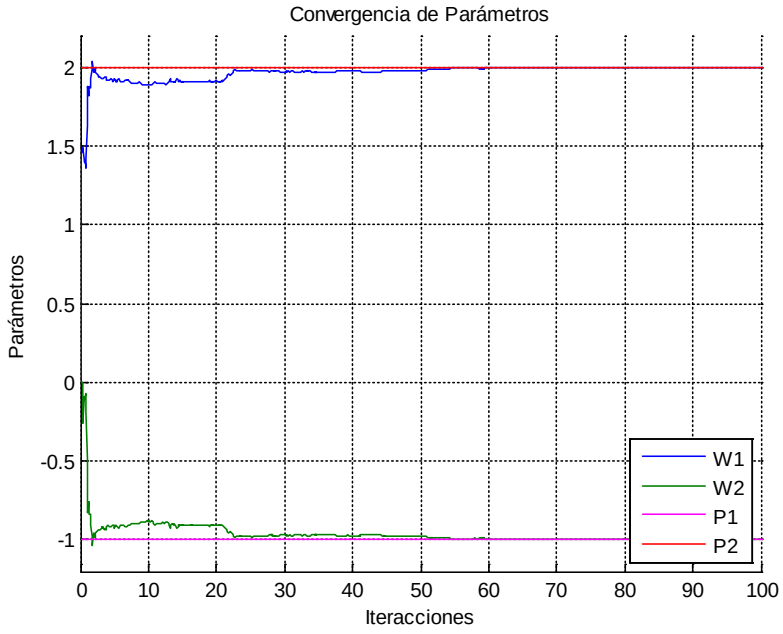

Figura 3.39. Convergencia de Parámetros (sist. En línea ec. 3.46)

En la Tabla 3.18 se realiza una comparación entre los parámetros reales del sistema y los parámetros obtenidos de la estimación del algoritmo LS-Lattice implementado en Simulink, donde se observa que la tendencia a igualar a los parámetros reales.

Parámetros Reales		Parámetros Estimados	
P1	2	W1	1.998614792722284
P2	-1	W2	-0.998613593550578

Tabla 3.18. Comparación entre parámetros reales y estimados (sist. en línea ec. 3.46)

CAPÍTULO 4

IMPLEMENTACIÓN Y PRUEBAS

4.1. INTRODUCCIÓN

En el Capítulo 1 se explica la forma de desarrollar el proyecto, donde se expresa la necesidad de crear un sistema de emulación de plantas para realizar las pruebas correspondientes del sistema de identificación en línea, tal y como se muestra en el diagrama de la Figura 1.1.

El presente capítulo está organizado de la manera siguiente: en el Epígrafe 4.2 se desarrolla la implementación y pruebas del sistema de emulación de plantas o procesos industriales, obteniendo la generación de las señales de entrada y de salida de la planta desde un PLC, el cual se programa en el lenguaje de texto estructurado (ST) y monitoreado con la ayuda de Matlab, a su vez este último envía los parámetros en tiempo discreto hacia el PLC, para que se desarrolle la ecuación a diferencias, como alternativa se ha desarrollado el mismo sistema de emulación con la tarjeta Arduino DUE y monitoreado por una aplicación desarrollada en Java™.

En el Epígrafe 4.3 se desarrolla la implementación final del sistema de identificación de procesos industriales en línea con el uso de Simulink y la librería Arduino para facilitar la conexión a la tarjeta Arduino Mega 2560, y también se realizan las pruebas necesarias para verificar el correcto funcionamiento del sistema final, integrando el emulador de procesos industriales y el sistema de identificación, en el capítulo siguiente se hace un desglose de todas las conclusiones y recomendaciones obtenidas del proyecto.

4.2. DESARROLLO DEL SISTEMA DE EMULACIÓN DE PROCESOS INDUSTRIALES

Como ya se ha mencionado, es necesario del desarrollo de un sistema de emulación de procesos industriales para poder realizar las pruebas finales del proyecto, para su implementación se utiliza el diagrama de bloques de la Figura 4.1.

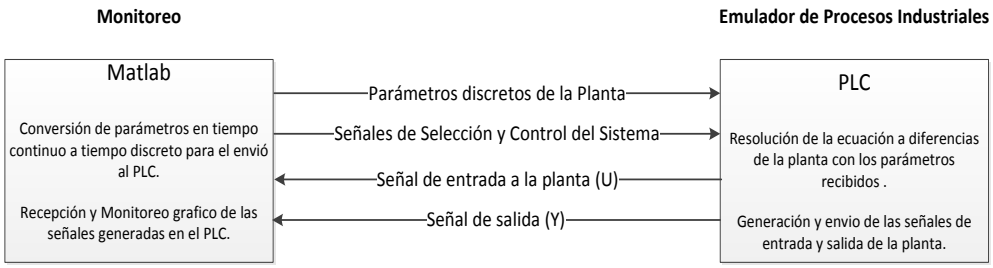

Figura 4.1. Diagrama del Sistema de Emulación de Procesos Industriales

Una vez distribuidas las funciones del sistema, se debe implementar los algoritmos de emulación de plantas dentro del PLC, y una interfaz gráfica de monitoreo y control del sistema en Matlab.

4.2.1. DESARROLLO DEL ALGORITMO DE GENERACIÓN DE SEÑALES DE ENTRADA/SALIDA

El emulador debe cumplir con las funciones de generar la señal de entrada hacia la planta u(k) y la de resolver la ecuación a diferencias que representa el proceso industrial, generando la señal de salida y(k); además de enviar dichas señales para su monitoreo mediante comunicación DDE entre el PLC y Matlab, y también mandar las salidas a través del módulo analógico del PLC para integrar estos datos con el sistema de identificación en línea mediante su adquisición con la tarjeta Arduino Mega 2560.

La primera función del emulador, la cual corresponde a la generación de la señal de entrada, se desarrolla de tal manera que pueda ser seleccionada de entre tres opciones:

- **Escalón Unitario**
- **Sinusoidal**
- **PRBS**

Para la generación del escalón unitario simplemente se debe establecer el valor de $u(k) = 1$ antes del desarrollo de la ecuación a diferencias y sin que sufra algún cambio durante cada iteración.

Para generar una función seno se utiliza la instrucción SIN del PLC, la variable tiempo se establece con la ayuda del valor del acumulador de un timer (T2.ACC) ubicado en el programa principal el cual inicia su cuenta al momento en que se da

esa orden desde la interfaz de monitoreo, siendo necesaria la transformación a radianes con la instrucción RAD, se aumenta un valor el cual se puede modificar para cambiar la frecuencia de la sinusoide, la ecuación 4.1 muestra la función $u(k)$ que genera una sinusoide a la entrada.

$$u(k) = SIN(RAD(\frac{T2.ACC}{24})) \tag{4.1}$$

Debido a que la señal PRBS puede ser tomada como de naturaleza digital, se puede utilizar registros de desplazamiento para generar la variación del bit de salida y obtener una variación en el ancho del pulso, lo que hace que se obtenga un buen rango de frecuencias y la similitud a una señal de ruido blanco.

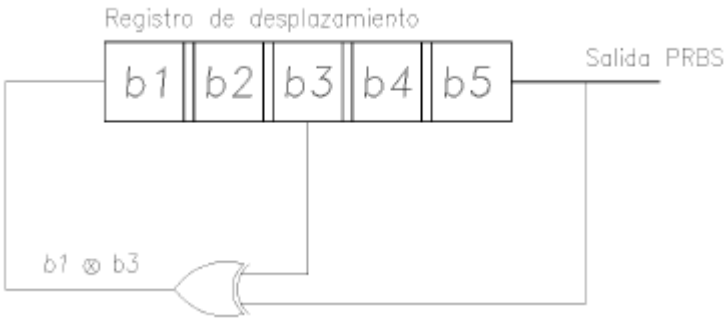

Figura 4.2. Generación de una señal PRBS con registro de 5 bits

La Figura 4.2 muestra el proceso de generación de la señal PRBS con un registro de desplazamiento de 5 bits, el cual debe ser inicializado antes de comenzar las iteraciones, la naturaleza aleatoria de la generación de la señal viene dada por la operación XOR entre el bit de salida b5 y el bit intermedio b3, el valor de u(k) dependerá del estado de b5, los valores de cada bit se van desplazando uno a

continuación de otro con cada iteración, es decir, b5 pasa a ser b4, b4 será b3 y así sucesivamente hasta que b1 toma el valor resultante de la operación XOR. En la Tabla 4.2 se muestra el algoritmo de la generación PRBS dentro del PLC.

La segunda función principal del sistema emulador de procesos o plantas industriales es la de resolver la ecuación a diferencias del proceso a emular y obtener la señal de salida y(k) a partir de la entrada u(k), para dicho efecto se establecen cuatro tipos principales de procesos a emular, todos en tiempo discreto debido a que el PLC va a ejecutar las iteraciones en los intervalos de tiempo establecidos por el período de muestreo.

Debido a que el sistema de emulación tiene el fin específico de servir como modelo de pruebas para el sistema de identificación, debe acoplarse a los requerimientos finales, es decir, proveer modelos a identificarse con el algoritmo de mínimos cuadrados ya sea normal o recursivo, y con el algoritmo LS-Lattice, desarrollados en el Capítulo 3, en ambos casos se da la posibilidad de identificar modelos de primer y segundo orden, por lo que el emulador tendrá la misma funcionalidad, dando así los cuatro modelos que se van a generar.

El primer modelo es de primer orden y se genera para ser identificado por el algoritmo LMS o RLS, de lo que se obtiene la función genérica en tiempo discreto de la ecuación 4.2.

$$G_a[z] = \frac{Y(z)}{U(z)} = \frac{k_1 z^{-1}}{1 + k_2 z^{-1}} \tag{4.2}$$

Pasando la función de transferencia de la ecuación 4.2 del dominio de la frecuencia al dominio del tiempo se obtiene la ecuación a diferencias que debe resolver el programa del PLC para obtener la salida y(k).

$$y(k) = k_1 u(k-1) - k_2 y(k-1) \tag{4.3}$$

El segundo modelo es de segundo orden y se genera para ser identificado por el algoritmo LMS o RLS, de lo que se obtiene la función genérica en tiempo discreto de la ecuación 4.4.

$$G_a[z] = \frac{Y(z)}{U(z)} = \frac{k_1 z^{-1} + k_2 z^{-2}}{1 + k_3 z^{-1} + k_4 z^{-2}} \tag{4.4}$$

Pasando la función de transferencia de la ecuación 4.4 del dominio de la frecuencia al dominio del tiempo se obtiene la ecuación a diferencias de la planta o proceso, este modelo y el anterior se basan en la aplicación de un retenedor de orden cero (ZOH) para pasar la función de transferencia del tiempo continuo al tiempo discreto.

$$y(k) = k_1 u(k-1) + k_2 u(k-2) - k_3 y(k-1) - k_4 y(k-2) \tag{4.5}$$

Para los dos últimos modelos de procesos a emular se determina los parámetros discretos directamente, y se usa con el algoritmo de identificación LS-Lattice para funciones de transferencia de primer y segundo orden.

$$G_a[z] = \frac{Y(z)}{U(z)} = \frac{1}{1 + k_1 z^{-1}} \tag{4.6}$$

$$y(k) = u(k) - k_1 y(k-1) \tag{4.7}$$

Las ecuaciones 4.6 y 4.7 muestran la tercera posibilidad de modelo a emular por el sistema, cabe notar que para el caso LS-Lattice solo se establece los parámetros en el denominador.

Las ecuaciones 4.8 y 4.9 muestran la función de transferencia y la ecuación a diferencias de la última posibilidad de modelo a emular, que corresponde a la identificación con el algoritmo LS-Lattice de sistemas de segundo orden.

$$G_a[z] = \frac{Y(z)}{U(z)} = \frac{1}{1 + k_1 z^{-1} + k_2 z^{-2}} \tag{4.8}$$

$$y(k) = u(k) - k_1 y(k-1) - k_2 y(k-2) \tag{4.9}$$

El programa del PLC posee una rutina principal y 4 subrutinas las que contienen la generación de las señales de entrada y salida para los cuatro diferentes tipos de funciones de transferencia que serán sujeto de emulación, la Tabla 4.1 muestra el algoritmo de la rutina principal del PLC. Cabe mencionar que el programa posee un bit de control "*start*" que indica el inicio de la emulación y otros cuatro bits de selección de subrutina llamados: "*ord1*", "*ord2*", "*lat1*", "*lat2*", que seleccionan la subrutina que genera las señales para las funciones de transferencia a emular.

Inicialización:
Si start = 0: $\boldsymbol{y(k) = 0}$ $\boldsymbol{u(k) = 0}$
Operación:
Si start = 1: 1. Iniciar conteo del timer T2 (generación de período de la señal sinuoidal) 2. Iniciar conteo del timer T1 (generación de período de muestreo) 3. Verificar bits de selección: Si ord1 = 1: Ir a subrutina de generación de señales para le ec. 4.2 Si ord2 = 1: Ir a subrutina de generación de señales para le ec. 4.4 Si lat1 = 1: Ir a subrutina de generación de señales para le ec. 4.6 Si lat2 = 1: Ir a subrutina de generación de señales para le ec. 4.8 4. Escribir las señales generadas y(k) y u(k) en las salidas del PLC 5. Retornar a (3) cada período de muestreo

Tabla 4.1. Rutina principal del PLC

Dentro de cada subrutina el procedimiento a seguir es similar, la única variación que se presenta en cada una es la de la ecuación a diferencias que se resuelve para cada tipo de función de transferencia seleccionada para emular, la Tabla 4.2 muestra el algoritmo general de las subrutinas para la generación de las señales y(k) y u(k) resultantes de la emulación de procesos industriales. Dentro de las subrutinas existen bits de selección de la señal de entrada u(k) denominadas "*escalon*", "*seno*" y "*prbs*", que seleccionan forma de u(k) de entre las señales tratadas previamente, también existe un bit de control denominado "*bandera*", para establecer la inicialización de los parámetros.

Inicialización:

Si bandera = 0:

Asignar los parámetros de la función de transferencia.

Inicializar bits del registro para PRBS (ver Figura 4.2):
b1=1
b2=0
b1=1
b1=0
b1=1

Operación:

Si bandera = 1:

1. Seleccionar señal de entrada u(k):

   Si *escalon*= 1:

   $$\boldsymbol{u(k) = 1}$$

   Si *seno* = 1:

   $$\boldsymbol{u(k) = SIN(RAD(\frac{T2.ACC}{24}))}$$

   Si prbs = 1: (Algoritmo generador PRBS ver Figura 4.2)

   salida = b5
   b1 = salida XOR b3
   b5 = b4
   b4 = b3
   b3 = b2
   b2 = b1

2. Resolver la ecuación a diferencias (dependerá de la función de transferencia a emular):

   Para ec. 4.2

   $$\boldsymbol{y(k) = k_1 u(k-1) - k_2 y(k-1)}$$

   Para ec. 4.4

   $$\boldsymbol{y(k) = k_1 u(k-1) + k_2 u(k-2) - k_3 y(k-1) - k_4 y(k-2)}$$

   Para ec. 4.6

   $$\boldsymbol{y(k) = u(k) - k_1 y(k-1)}$$

   Para ec. 4.8

   $$\boldsymbol{y(k) = u(k) - k_1 y(k-1) - k_2 y(k-2)}$$

3. Almacenar valores de u(k) y y(k) para la siguiente iteración
4. Escalar las señales generadas
5. Retornar a (1) cada período de muestreo

Tabla 4.2. Algoritmo general de las subrutinas de generación de señales

4.2.2. DESARROLLO DE LA INTERFAZ DE MONITOREO DEL SISTEMA DE EMULACIÓN

Una vez cargado el PLC con los algoritmos desarrollados en el epígrafe anterior se hace necesaria la creación de un sistema de control y monitoreo de sistema de emulación de procesos industriales, el mismo que tendrá las funciones de: enviar los valores de los parámetros de las funciones de transferencia en tiempo discreto (ecuaciones 4.1 a 4.7), seleccionar el tipo de señal de entrada a la planta u(k), dar la señal de inicio al programa del PLC y mostrar una gráfica de y(k) y u(k) generadas en el PLC. El intercambio de datos entre la interfaz de monitoreo y el PLC se da a través de comunicación DDE.

Para la creación de la interfaz de monitoreo se ha utilizado la herramienta GUIDE de Matlab, la Figura 4.3 muestra el diagrama de distribución de la interfaz gráfica general de monitoreo del sistema de emulación, la misma que consta de dos niveles, el primero es el de presentación y selección del tipo de planta a emular y el segunda que consta de 4 interfaces independientes que ayudaran a la configuración de parámetros y la visualización de las señales generadas por el PLC para cada uno de los tipos de función de transferencia seleccionables.

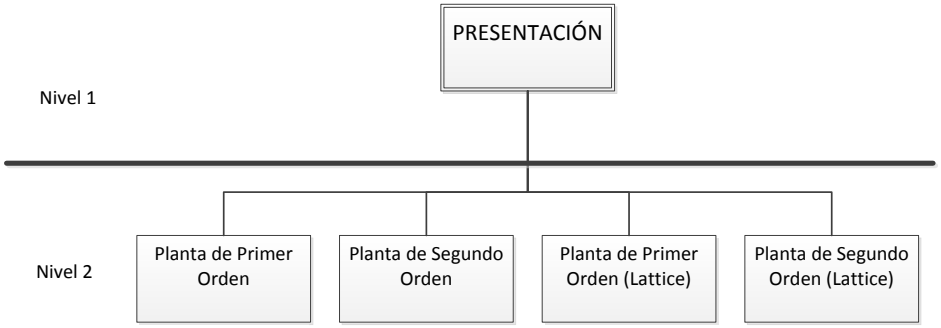

Figura 4.3. Diagrama de la interfaz general

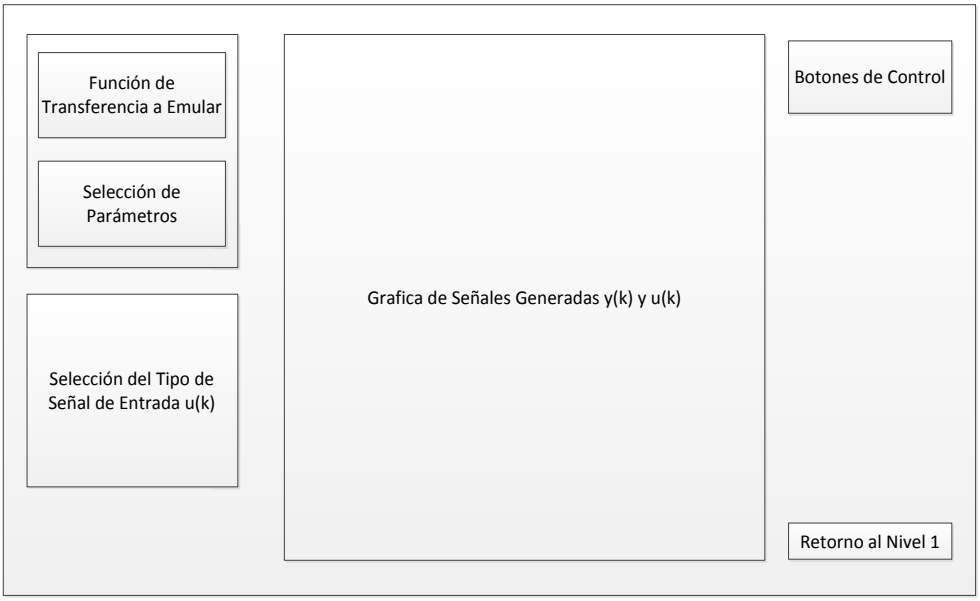

Figura 4.4. Esquema general de la interfaz de monitoreo del sistema de emulación

En la Figura 4.4 se puede observar la disposición general y la distribución de las funciones que tendrá el sistema de emulación dentro de una interfaz gráfica de usuario, la misma que se generaliza para los cuatro tipos de funciones de trasferencia

que se puede emular, este modelo va dentro del nivel 2 del diagrama general de la Figura 4.3.

La interfaz que se encuentra en el nivel 1 del diagrama general es bastante simple, gráficamente posee la capacidad de selección del tipo de función de transferencia a emular y dentro de su programación se debe incluir la inicialización de la comunicación DDE entre la interfaz y el PLC, además de configurar los bits de control del programa del PLC a su valor inicial. La Figura 4.5 muestra la interfaz gráfica de presentación.

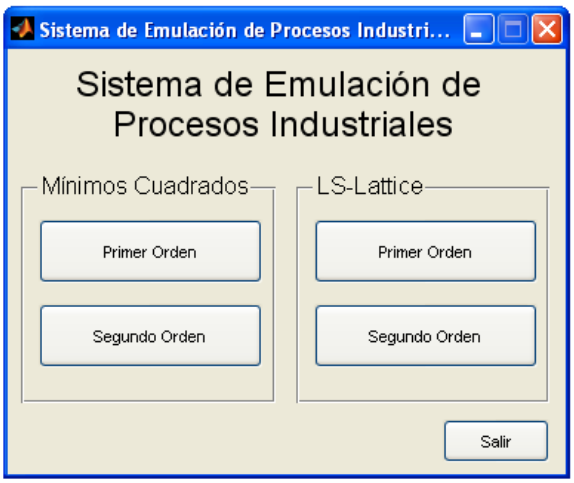

Figura 4.5. Interfaz gráfica de presentación

Las interfaces que se encuentran en el nivel 2, aparte de tener las características graficas de la Figura 4.4, deben ser capaces de enviar los parámetros discretos de las funciones de transferencia al PLC, para el caso de mínimos cuadrados se recibe los parámetros de la función de transferencia en tiempo continuo y se utiliza Matlab para discretizar la planta con el período de muestreo ingresado, para el caso de LS-Lattice

se recibe los parámetros directamente de la interfaz. Luego de seleccionar el tipo de entrada u(k) y de enviar la señal de inicio al PLC, la interfaz debe escalar los datos de las señales y(k) y u(k) generadas por el PLC para que puedan ser graficadas y así verificar su correcto funcionamiento.

El sistema de emulación completo viene de acuerdo al diagrama de la Figura 4.1, para simular el sistema se utiliza un emulador de PLC's, en el que se encuentra cargado el programa desarrollado en el epígrafe 4.2.1.

Los resultados obtenidos de la emulación de un sistema de primer orden a identificar por los métodos de mínimos cuadrados se pueden apreciar a través de la interfaz gráfica de monitoreo, en la

Figura 4.6 se muestra la reacción del sistema ante una entrada escalón generado por el PLC, la Figura 4.7 la reacción ante una entrada sinusoidal y la Figura 4.8 ante una señal aleatoria PRBS.

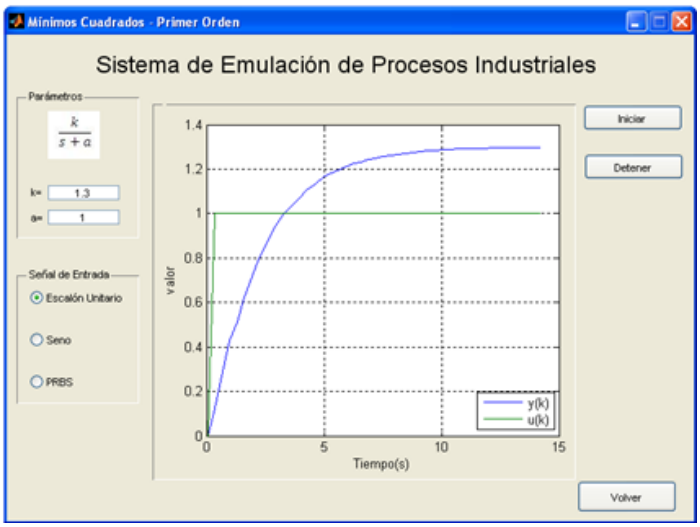

Figura 4.6. Emulación de planta para mínimos cuadrados - primer orden, entrada escalón

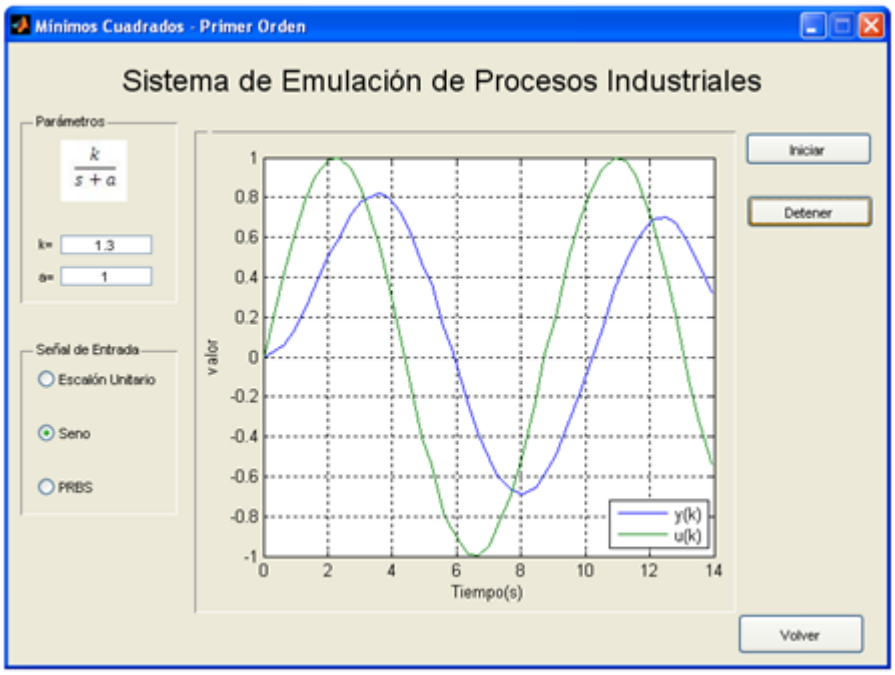

Figura 4.7. Emulación de planta para mínimos cuadrados - primer orden, entrada sinusoidal

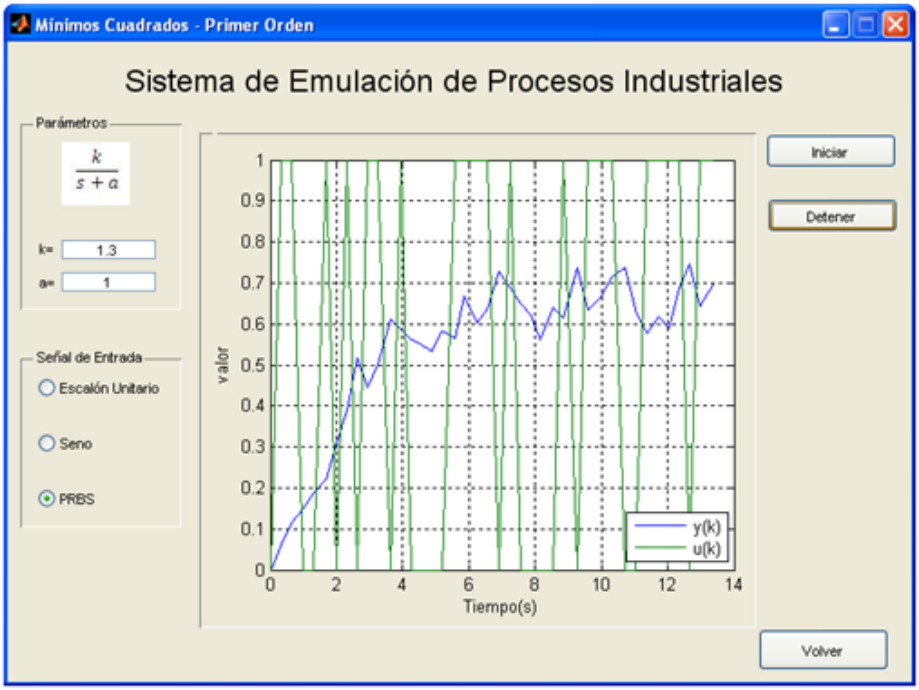

Figura 4.8. Emulación de planta para mínimos cuadrados - primer orden, entrada PRBS

Para un sistema de segundo orden a identificar por los métodos de mínimos cuadrados LMS y RLS se tienen las siguientes señales generadas, la Figura 4.9 muestra la reacción de la planta emulada ante una entrada escalón, la Figura 4.10 ante una entrada sinusoidal y la Figura 4.11 ante una señal aleatoria PRBS.

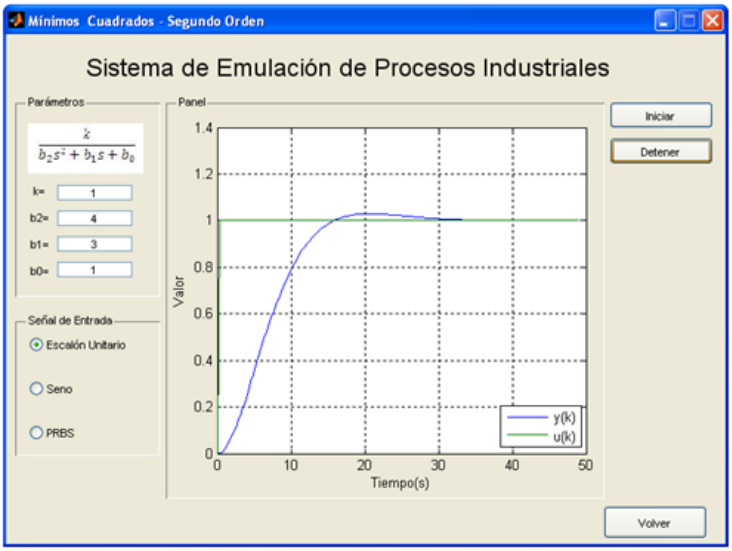

Figura 4.9. Emulación de planta para mínimos cuadrados - segundo orden, entrada escalón

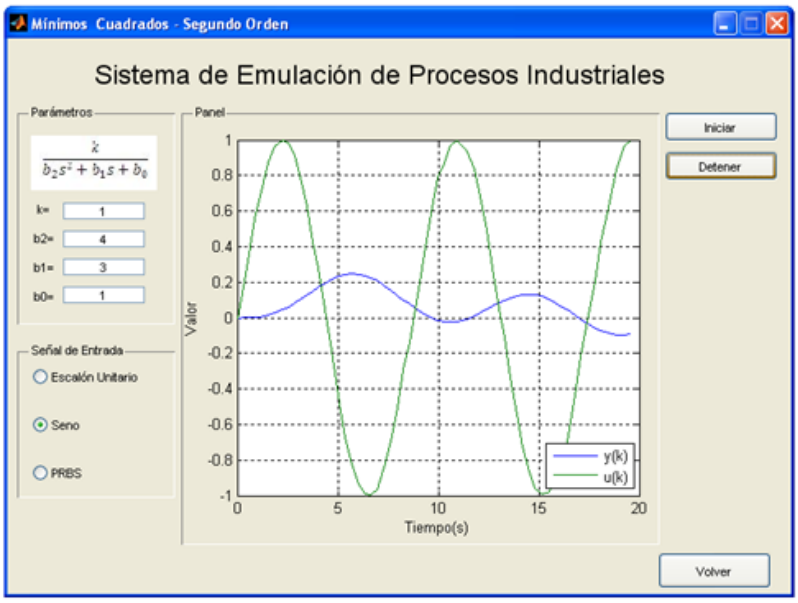

Figura 4.10. Emulación de planta para mínimos cuadrados - segundo orden, entrada sinusoidal

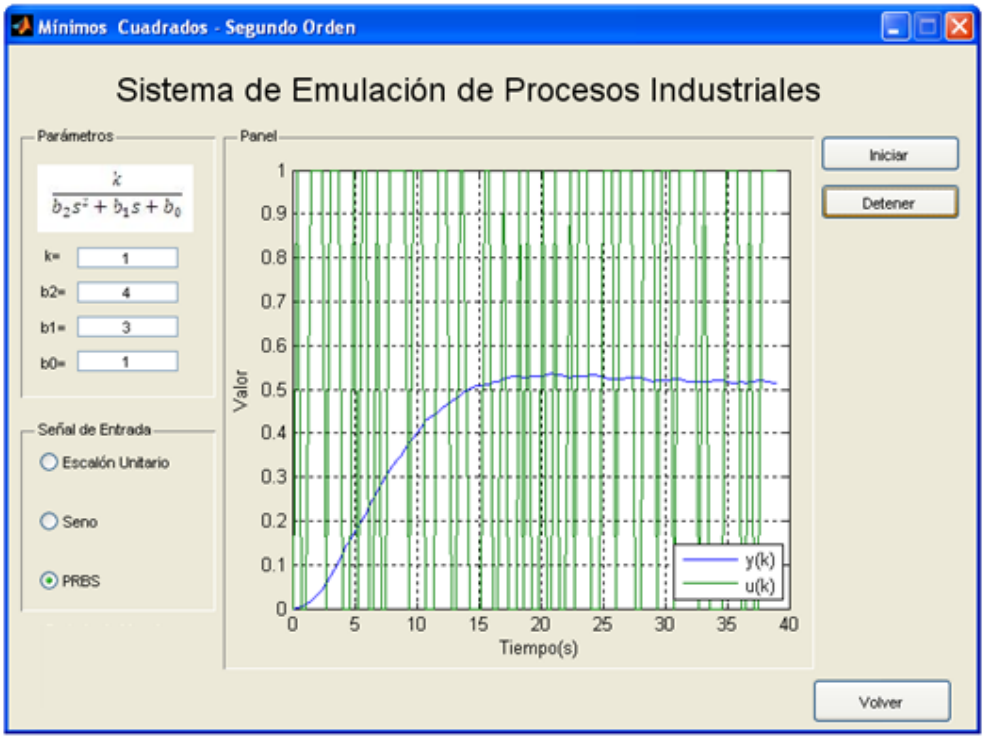

Figura 4.11. Emulación de planta para mínimos cuadrados - segundo orden, entrada PRBS

Para un sistema de primer orden a identificar por el algoritmo LS-Lattice se tienen las siguientes señales generadas, la Figura 4.12 muestra la reacción de la planta emulada ante una entrada escalón, la Figura 4.13 ante una entrada sinusoidal y la Figura 4.14 ante una señal aleatoria PRBS.

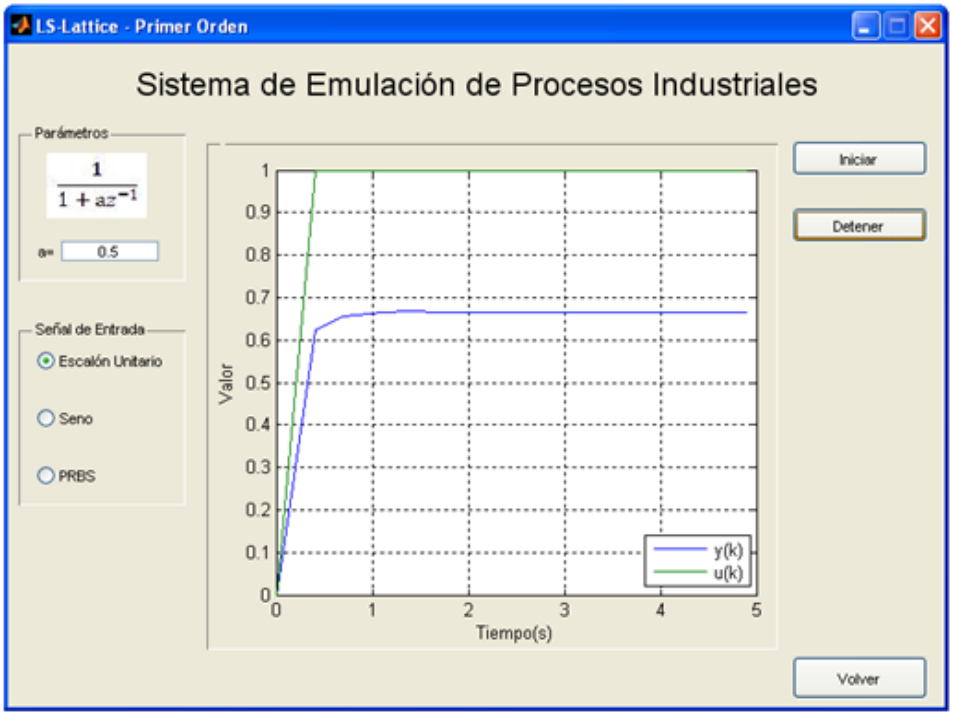

Figura 4.12. Emulación de planta para Lattice - primer orden, entrada escalón

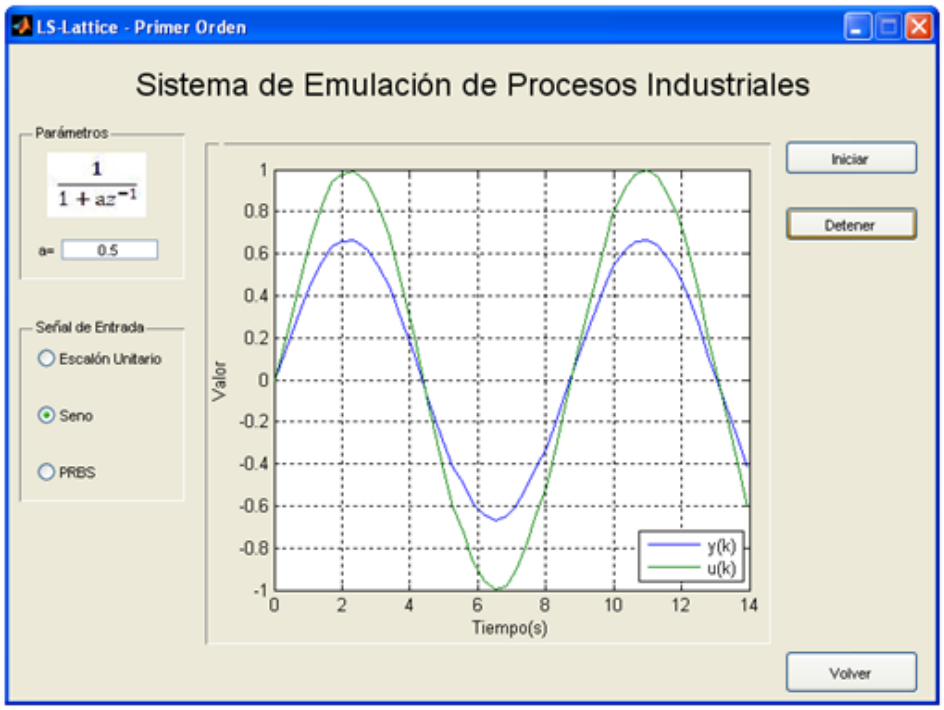

Figura 4.13. Emulación de planta para Lattice - primer orden, entrada sinusoidal

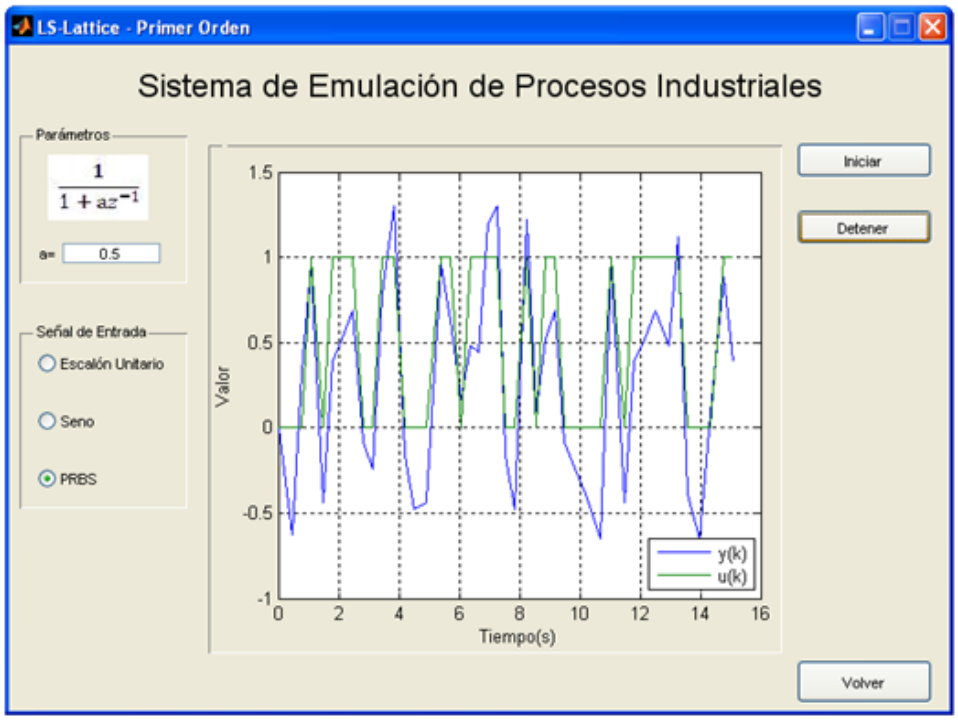

Figura 4.14. Emulación de planta para Lattice - primer orden, entrada PRBS

Para un sistema de segundo orden a identificar por el algoritmo LS-Lattice se tienen las siguientes señales generadas, la Figura 4.15 muestra la reacción de la planta emulada ante una entrada escalón, la Figura 4.16 ante una entrada sinusoidal y la Figura 4.17 ante una señal aleatoria PRBS.

Figura 4.15. Emulación de planta para Lattice - segundo orden, entrada escalón

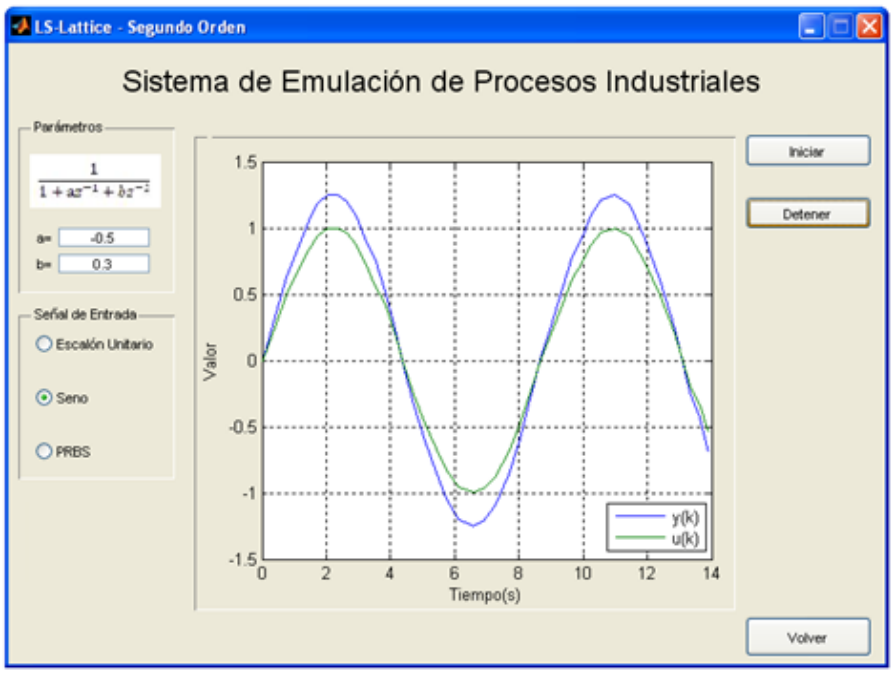

Figura 4.16. Emulación de planta para Lattice - segundo orden, entrada sinusoidal

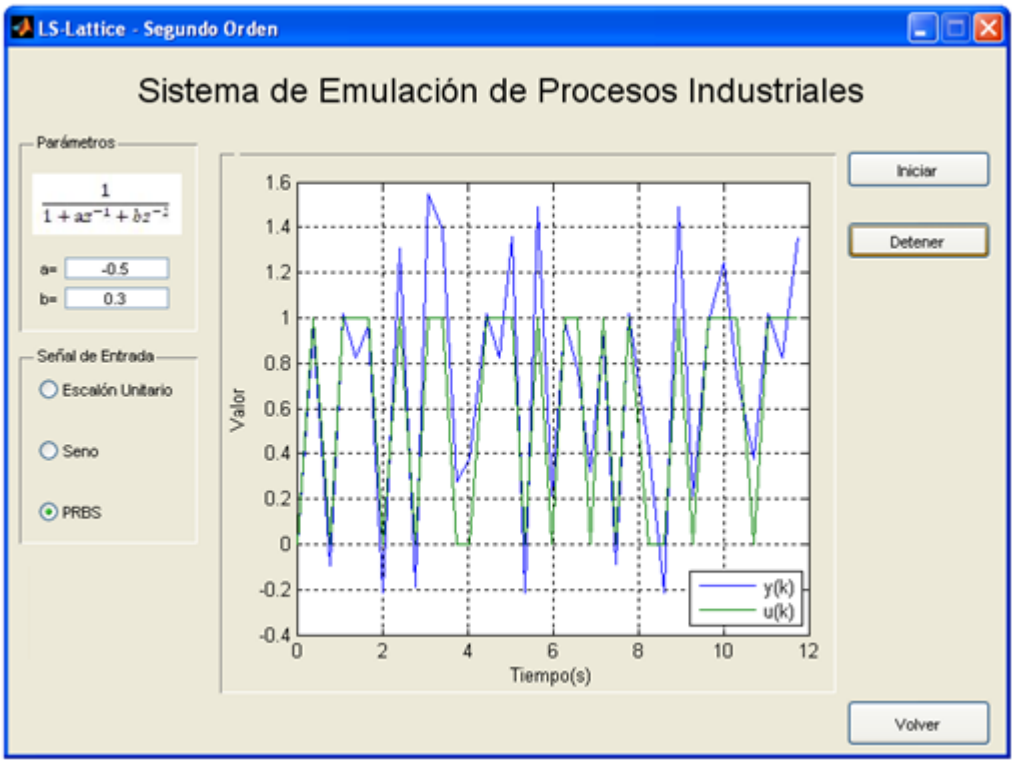

Figura 4.17. Emulación de planta para Lattice - segundo orden, entrada PRBS

4.2.3. DESARROLLO DE LA INTERFAZ DE MONITOREO DEL SISTEMA DE EMULACIÓN EN LA PLATAFORMA JAVA™

Como alternativa a la generación de señales de entrada y salida de procesos industriales, se ha optado por el uso de la tarjeta Arduino DUE sustituyendo al PLC, esto para dar otra solución al hardware de emulación. Se seleccionó la tarjeta Arduino DUE debido a que posee dos conversores digital/análogo (DAC's) para obtener las salidas analógicas directamente, sin necesidad de adicionar hardware al sistema.

El programa que se encuentra dentro de la tarjeta arduino DUE se encuentra basado en el algoritmo desarrollado en el Epígrafe 4.2.1 para generar las señales de entrada/salida.

Para realizar el monitoreo y configuración de parámetros que van a llegar al algoritmo de emulación se ha desarrollado una aplicación en lenguaje JavaTM, la misma que permite seleccionar el tipo de planta que se va a emular, establecer los parámetros que van a regir el comportamiento de la planta, seleccionar la señal de entrada que va a excitar la planta y monitorear la señal de entrada y salida generada. La aplicación se comunica con la tarjeta arduino DUE a través de comunicación serial, la misma que es posible, ya que la tarjeta posee un conversor USB-Serial embebido.

Para el uso de la aplicación se debe configurar inicialmente el puerto serial que representa la tarjeta arduino DUE seleccionándolo en la ventana que se muestra en la Figura 4.18.

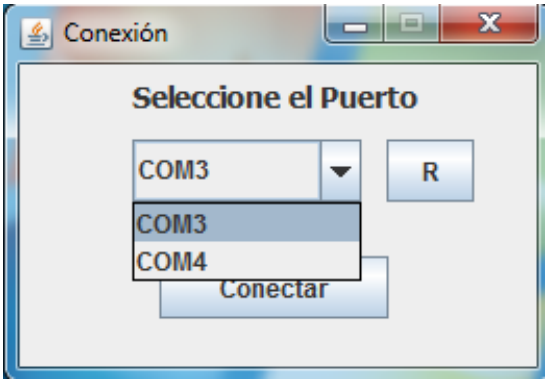

Figura 4.18. Selección del Puerto Serie

Una vez seleccionado el puerto serie se despliega una ventana para seleccionar el tipo de planta a emular, establecer los parámetros, seleccionar el tipo de entrada y visualizar las señales generadas como se muestra en la Figura 4.19.

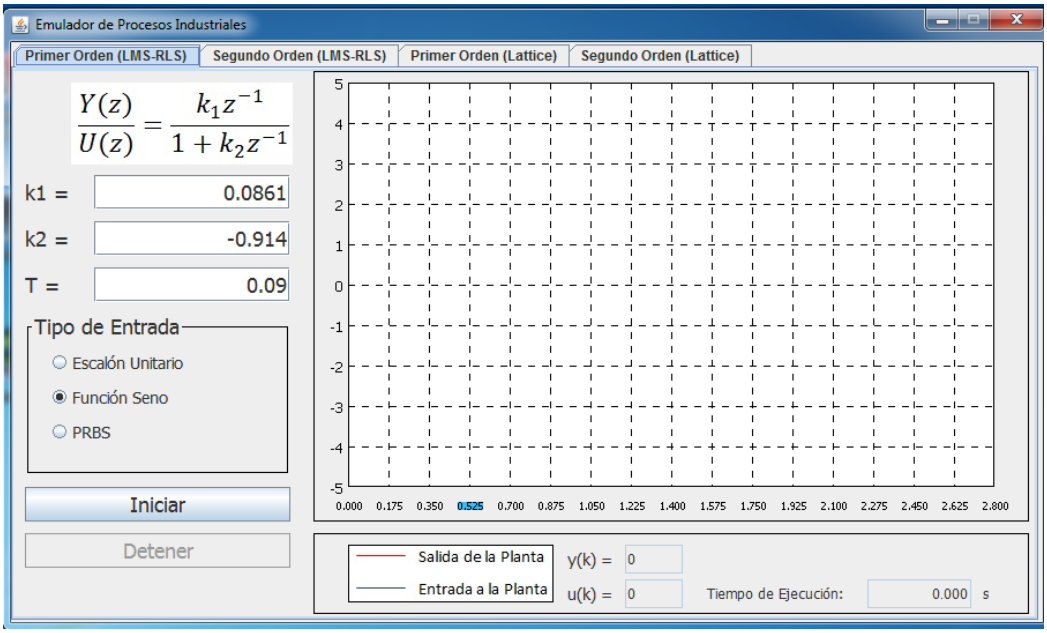

Figura 4.19. Pantalla principal del Sistema de Emulación

La Figura 4.20 muestra el sistema en ejecución, en la misma se ha seleccionado un sistema de primer orden a ser identificado con el algoritmo LMS o RLS, se puede observar que se han dejado establecidos los parámetros por defecto con una función seno como señal de entrada a la planta.

La Figura 4.21 muestra otro ejemplo el sistema en ejecución, en la misma se ha seleccionado un sistema de segundo orden a ser identificado con el algoritmo LMS o RLS, se puede observar que se han dejado establecidos los parámetros por defecto con una función pseudo-aleatoria como señal de entrada a la planta.

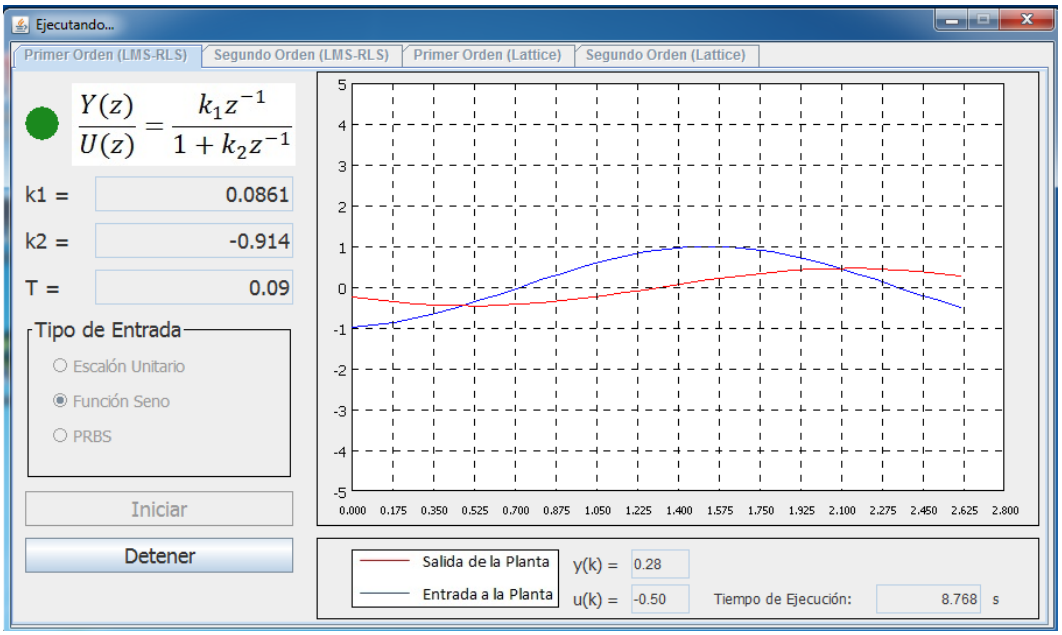

Figura 4.20. Emulación planta de primer orden

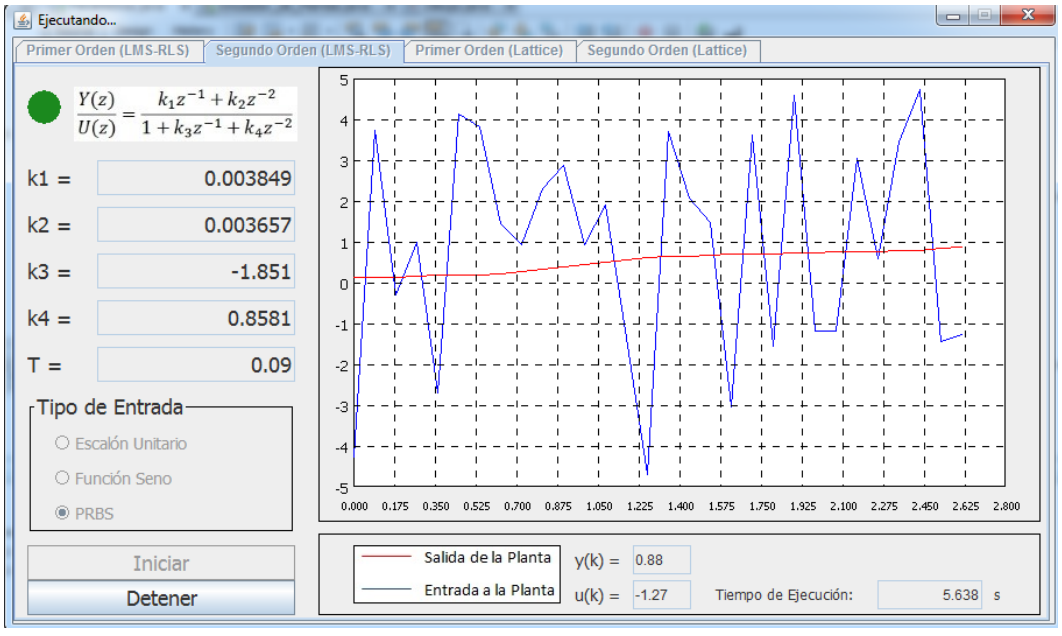

Figura 4.21. Emulación planta de segundo orden

El sistema de emulación desarrollado en java con la tarjeta arduino DUE funciona de la misma manera que el sistema desarrollado en al epígrafe anterior en Matlab para el PLC.

4.3. PRUEBAS DEL SISTEMA DE IDENTIFICACIÓN USANDO EL EMULADOR DE PROCESOS INDUSTRIALES

Una vez desarrollada toda la parte teórica y de diseño de los elementos necesarios para la elaboración del proyecto es momento de presentar los resultados de la implementación final y la ejecución del programa en tiempo real con el sistema de emulación de procesos industriales en línea con el sistema de identificación de acuerdo al esquema de la

Figura 1.1, donde el PLC hace las veces de proceso industrial con los algoritmos de emulación de plantas cargados, la tarjeta Arduino Mega 2560 adquiere los datos de las señales y(k) y u(k) generadas hacia la PC donde se ejecutan los algoritmos de identificación con la ayuda de Simulink.

La librería de Arduino para Simulink permite una adquisición rápida de los datos desde la tarjeta hacia la PC, además a la capacidad de lectura en tiempo real de la Arduino Mega 2560, lo que permite la actualización de datos en los bloques de Simulink mientras dura la ejecución de los diagramas.

4.3.1. IMPLEMENTACIÓN Y PRUEBAS DEL ALGORITMO DE IDENTIFICACIÓN LMS

Función de Transferencia de Primer Orden

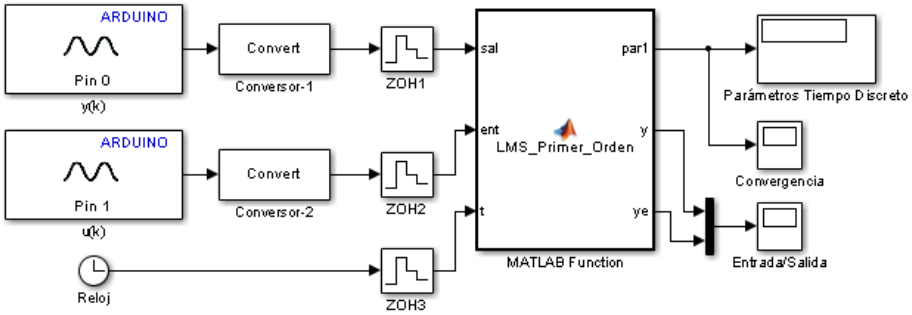

Figura 4.22. Diagrama de bloques sistema de identificación de primer orden-LMS

La Figura 4.22 muestra el diagrama de bloques del sistema de identificación de procesos industriales basado en el algoritmo LMS para identificar funciones de transferencia de primer orden, a la derecha del diagrama se observan los bloques de lectura de entradas analógicas de la tarjeta arduino, estas señales deben convertirse a datos que puedan ser leídos por el bloque función de Matlab que contiene el algoritmo identificador, los parámetros en tiempo discreto obtenidos y la salida real y estimada de la planta son exportados al *workspace* de Matlab para ser graficados y ser sujeto de análisis.

Los resultados obtenidos de la ejecución del diagrama de bloques de Simulink de la Figura 4.22 pueden ser analizados gráficamente con la ayuda de Matlab, la Figura 4.23 muestra la comparación entre el valor real y el estimado de la salida del sistema, en la gráfica se puede observar la tendencia de la salida estimada hacia la salida real

con ciertos picos que demuestran la vulnerabilidad de la tarjeta de adquisición de datos al ruido.

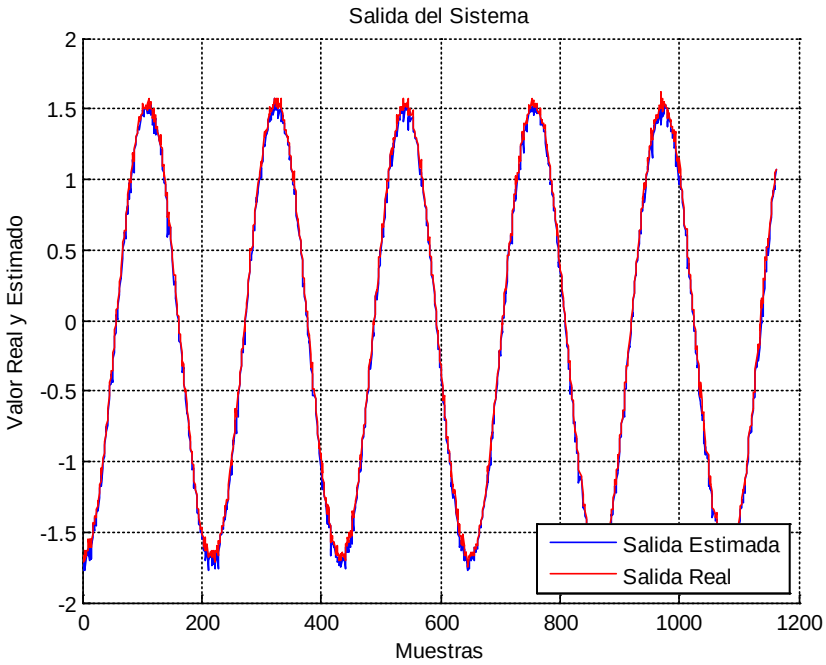

Figura 4.23. Salida del sistema, real y estimada (sist. primer orden-LMS)

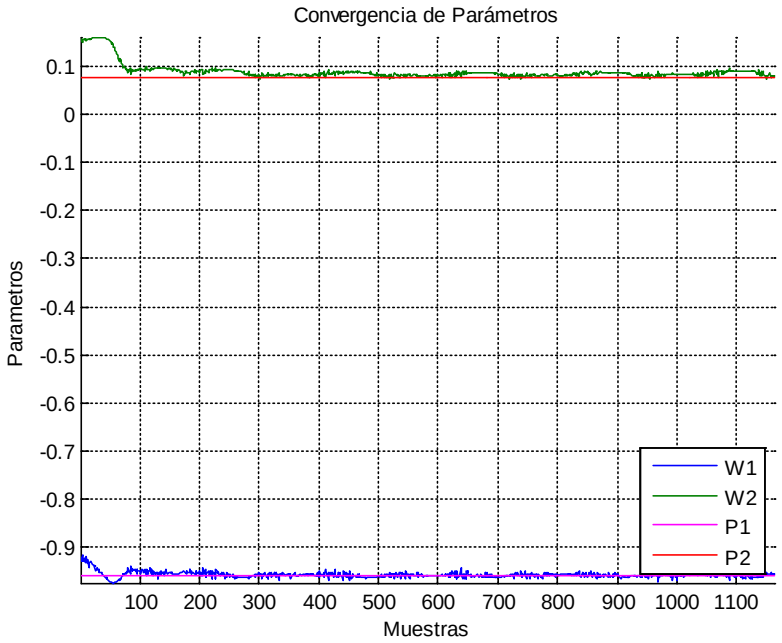

Figura 4.24. Convergencia de Parámetros (sist. primer orden-LMS)

La Figura 4.24 muestra la tendencia que toman los parámetros estimados en comparación a los parámetros reales a lo largo del tiempo de ejecución del sistema de identificación, para la comparación, los parámetros reales vienen de la discretización de la función de transferencia en tiempo continuo de la ecuación 4.10, a un período de muestreo T=0.04s:

$$G(s) = \frac{2}{s+1} \tag{4.10}$$

La Tabla 4.3 muestra el valor real del sistema a identificar y el valor de los parámetros obtenidos de la estimación una vez se haya detenido ejecución del programa, notándose una gran proximidad entre los valores.

Parámetros Reales		Parámetros Estimados	
P1	-0.960789439152323	W1	-0.955245614051819
P2	0.078421121695354	W2	0.086068814728772

Tabla 4.3. Comparación entre parámetros reales y parámetros estimados (sist. primer orden-LMS)

Función de Transferencia de Segundo Orden

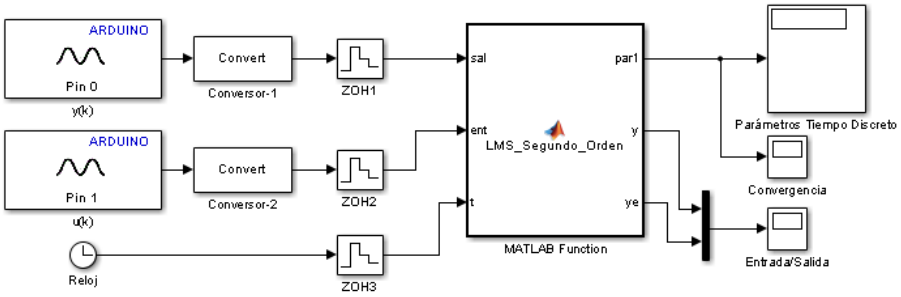

Figura 4.25. Diagrama de bloques sistema de identificación de segundo orden-LMS

El diagrama de bloques de la Figura 4.25 muestra las mismas características que el diagrama de la Figura 4.22 en donde destacan los bloques de la librería de arduino para Simulink en lo que tiene que ver con la adquisición de datos, la única variación se encuentra en el bloque función de Matlab, en el mismo que se modifica la inicialización de los vectores para adecuarse con el número de parámetros a identificar, que para este caso son cuatro.

Luego de la simulación del algoritmo de identificación LMS para sistemas de segundo orden desarrollado en el capítulo anterior, se observó que resultaba inestable, en la Figura 4.26 y la Figura 4.27 se demuestra que la inestabilidad se mantiene en la ejecución del programa en tiempo real, dado que pasado un número relativamente bajo de muestras tomadas la salida estimada del sistema alcanza valores muy altos, lo mismo que resulta consecuencia del valor elevado que van tomando los parámetros estimados.

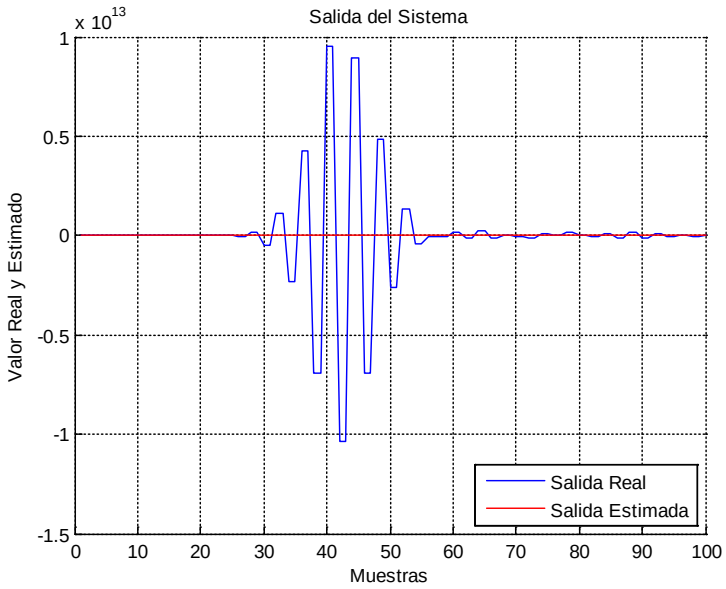

Figura 4.26. Salida del sistema, real y estimada (sist. segundo orden-LMS)

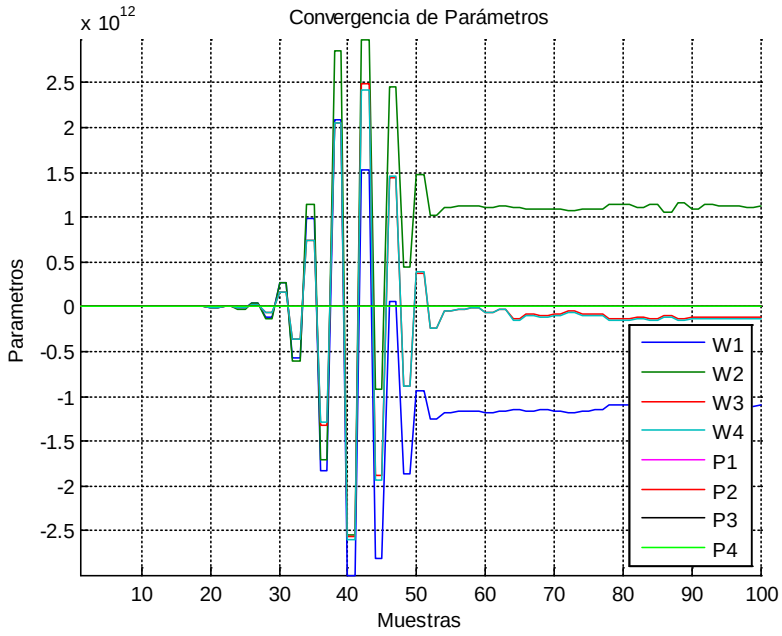

Figura 4.27. Convergencia de Parámetros (sist. segundo orden-LMS)

4.3.2. IMPLEMENTACIÓN Y PRUEBAS DEL ALGORITMO DE IDENTIFICACIÓN RLS

Función de Transferencia de Primer Orden

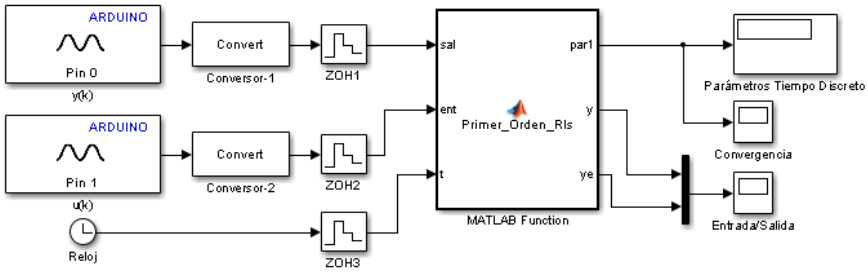

Figura 4.28. Diagrama de bloques sistema de identificación de primer orden-RLS

La Figura 4.28 muestra el diagrama de bloques del sistema de identificación de procesos industriales basado en el algoritmo RLS para identificar funciones de transferencia de primer orden, el mismo que es similar a los diagramas anteriores con la única modificación en el bloque función de Matlab que para esta caso contiene el algoritmo de identificación RLS desarrollado en la Tabla 3.10.

Los resultados obtenidos de la ejecución del diagrama de bloques de Simulink de la Figura 4.28 pueden ser analizados gráficamente con la ayuda de Matlab, la Figura 4.29 muestra la comparación entre el valor real y el estimado de la salida del sistema, en la gráfica se puede observar la tendencia de la salida estimada hacia la salida real.

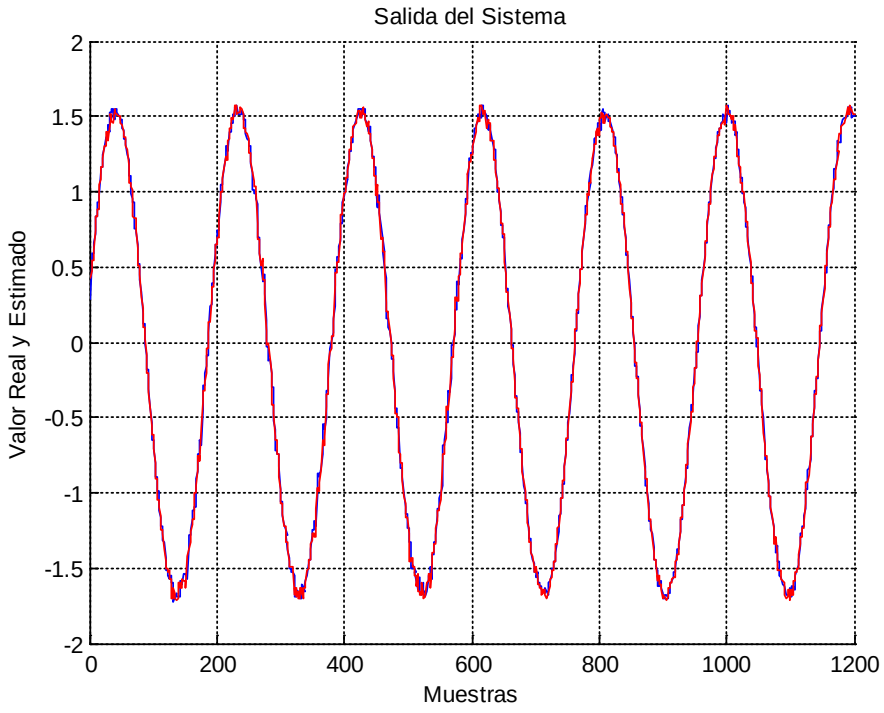

Figura 4.29. Salida del sistema, real y estimada (sist. primer orden-RLS)

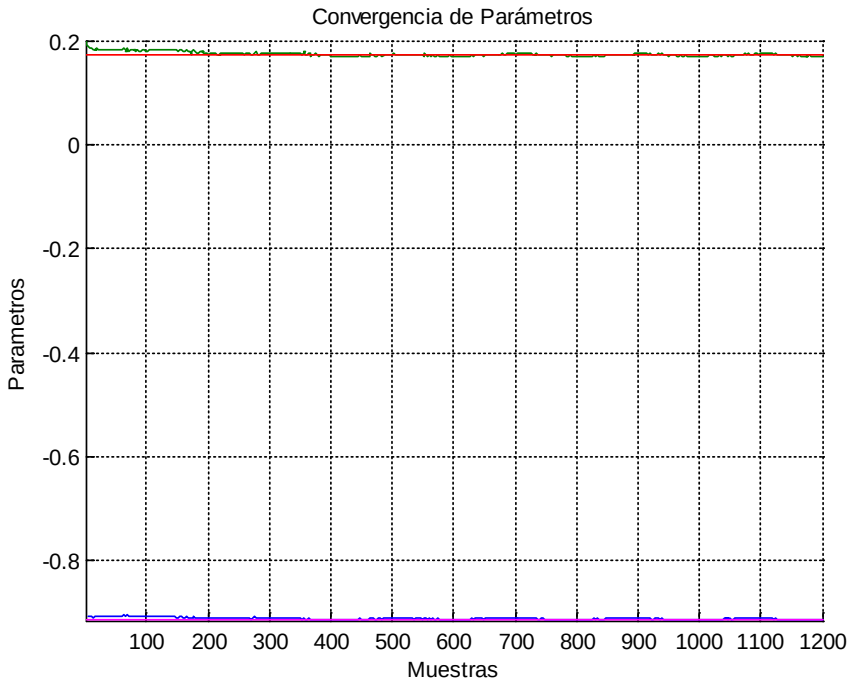

Figura 4.30. Convergencia de Parámetros (sist. primer orden-RLS)

La Figura 4.30 muestra la tendencia que toman los parámetros estimados en comparación a los parámetros reales a lo largo del tiempo de ejecución del sistema de identificación, para la comparación, los parámetros reales vienen de la discretización de la función de transferencia en tiempo continuo de la ecuación 4.10, a un período de muestreo T=0.09s:

La Tabla 4.4 muestra el valor real del sistema a identificar y el valor de los parámetros obtenidos de la estimación una vez se haya detenido ejecución del programa.

Parámetros Reales		Parámetros Estimados	
P1	-0.913931185271228	W1	-0.914778828620911
P2	0.172137629457544	W2	0.171220958232880

Tabla 4.4. Comparación entre parámetros reales y parámetros estimados (sist. primer orden-RLS)

Función de Transferencia de Segundo Orden

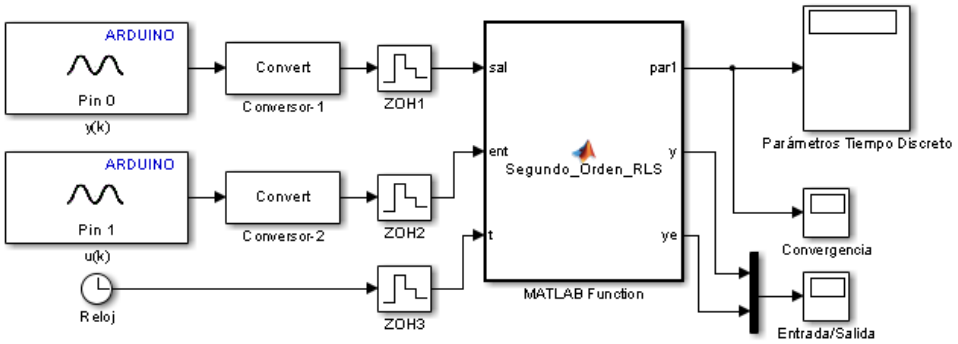

Figura 4.31. Diagrama de bloques sistema de identificación de segundo orden-RLS

Para la ejecución del sistema de identificación de procesos industriales en sistemas de segundo orden basado en el algoritmo RLS se utiliza el diagrama de Simulink de la Figura 4.31, en el bloque función de Matlab se encuentra cargado el algoritmo de la Tabla 3.10 para funciones de transferencia de segundo orden.

Luego de la ejecución del diagrama de bloques de Simulink y observando los resultados obtenidos en la Figura 4.32 y de la Figura 4.33, se puede concluir que el sistema diseñado no tiene un buen desempeño en sistemas de segundo orden ya que se vuelve inestable con el paso del tiempo de ejecución, lo que se puede justificar por la falta de implementación de un algoritmo para hacer el factor de olvido variable de acuerdo al estado del identificador lo que puede provocar el apagado del algoritmo antes de la convergencia de los parámetros.

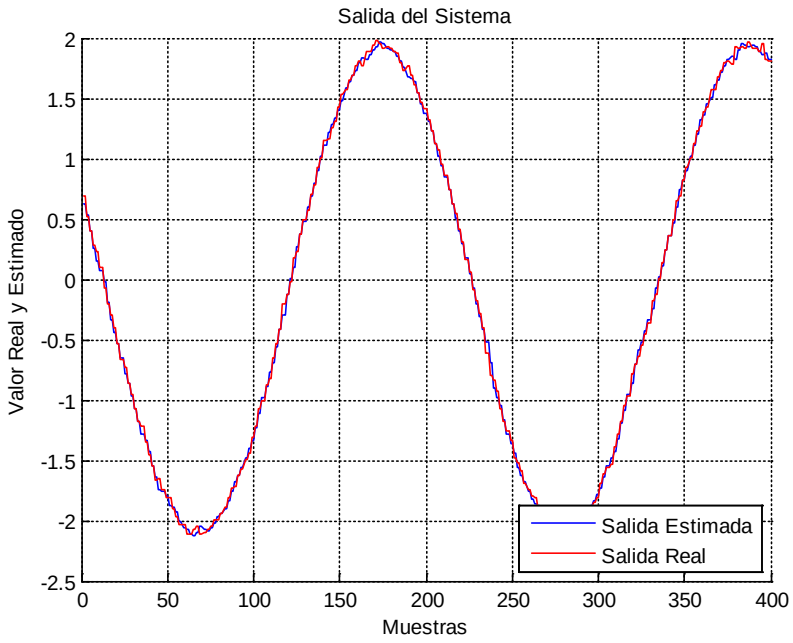

Figura 4.32. Salida del sistema, real y estimada (sist. segundo orden-RLS)

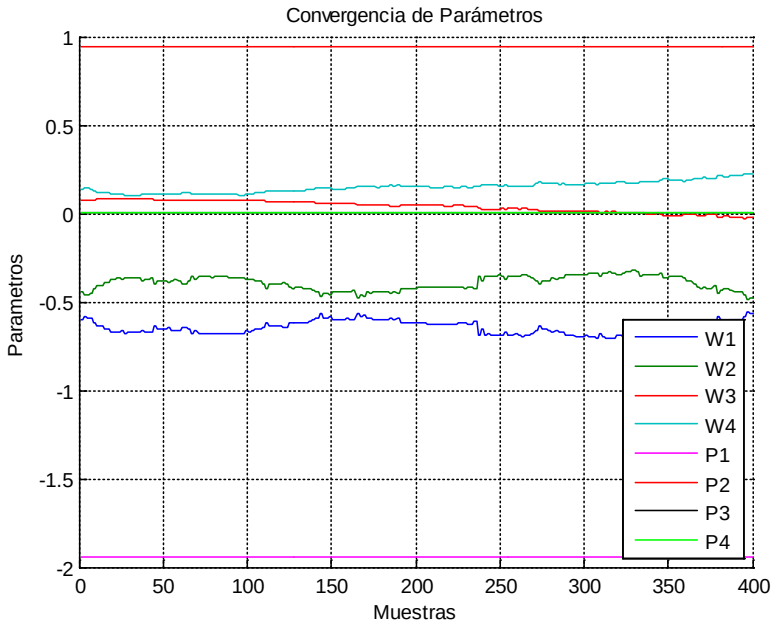

Figura 4.33. Convergencia de Parámetros (sist. segundo orden-RLS)

4.3.3. IMPLEMENTACIÓN Y PRUEBAS DEL ALGORITMO DE IDENTIFICACIÓN LS-LATTICE

Función de Transferencia de Primer Orden

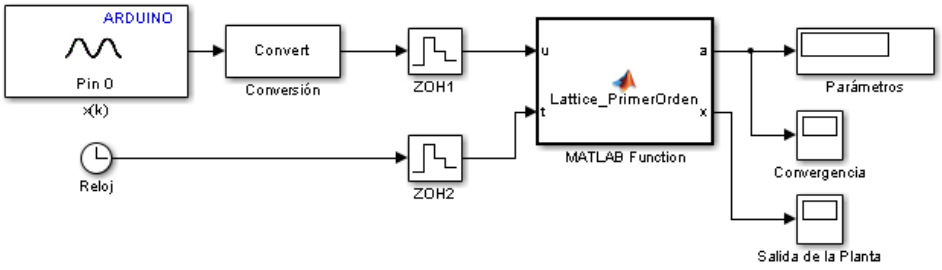

Figura 4.34. Diagrama de bloques sistema de identificación de primer orden-Lattice

La Figura 4.34 muestra el diagrama de bloques del sistema de identificación de procesos industriales basado en el algoritmo LS-Lattice para identificar funciones de transferencia de primer orden de la forma de la ecuación 4.6, las modificaciones del diagrama con respecto a los diagramas anteriores se ven reflejadas en el uso de una sola entrada analógica de la tarjeta arduino, ya que el algoritmo que se encuentra dentro del bloque función de Matlab solo requiere de la señal de salida de la planta generada por el PLC, tal y como se muestra en la Tabla 3.16 que resume el algoritmo LS-Lattice.

Los resultados obtenidos de la ejecución del diagrama de bloques de Simulink de la Figura 4.34 pueden ser analizados gráficamente con la ayuda de Matlab, la Figura

4.35 muestra la tendencia que toma el parámetro a estabilizarse en una valor cercano al valor real.

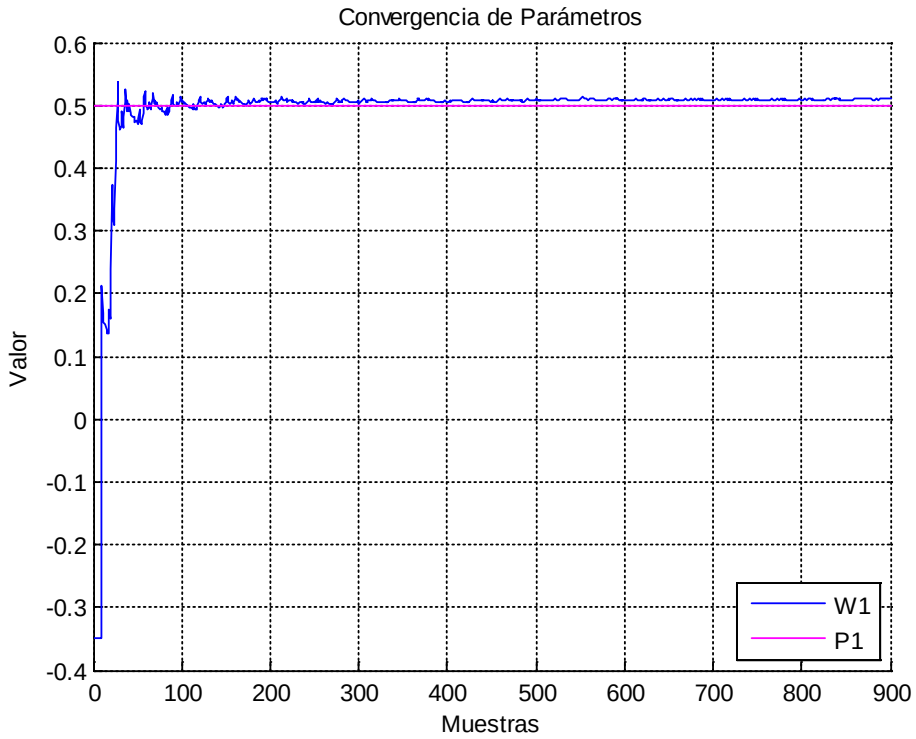

Figura 4.35. Convergencia de Parámetros (sist. primer orden-Lattice)

La Tabla 4.5 muestra el valor real del sistema a identificar y el valor de los parámetros obtenidos de la estimación una vez se haya detenido ejecución del programa.

Parámetro Real		Parámetros Estimados	
P1	0.5	W1	0.510706663131714

Tabla 4.5. Comparación entre parámetros reales y parámetros estimados (sist. primer orden-Lattice)

Función de Transferencia de Segundo Orden

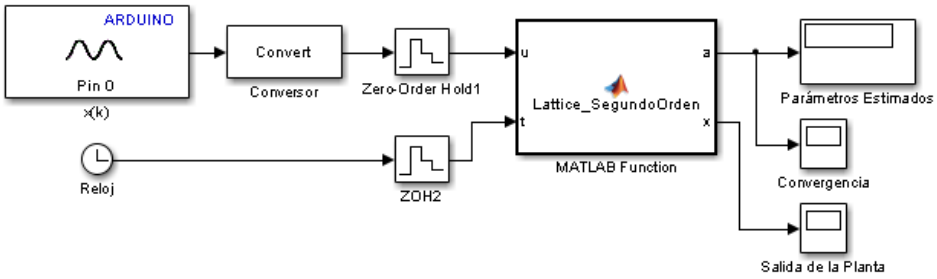

Figura 4.36. Diagrama de bloques sistema de identificación de segundo orden-Lattice

Para la ejecución del sistema de identificación de procesos industriales en sistemas de segundo orden basado en el algoritmo LS-Lattice se utiliza el diagrama de Simulink de la Figura 4.36, en el bloque función de Matlab se encuentra cargado el algoritmo de la Tabla 3.16 en el que se ha aumentado una etapa debido al aumento de un parámetro a identificar como se muestra en la función de transferencia de la ecuación 4.8.

Luego de la ejecución del diagrama de bloques de Simulink, la Figura 4.37 muestra la estabilidad que van tomando los parámetros estimados justificando algunas variaciones por el ruido al que la tarjeta arduino es susceptible, ambos parámetros se muestran aproximados al valor esperado de la identificación y no presentan inestabilidad como con los métodos anteriores, la Tabla 4.6 muestra los valores que alcanzaron los parámetros estimados hasta el momento final de la ejecución del sistema identificador.

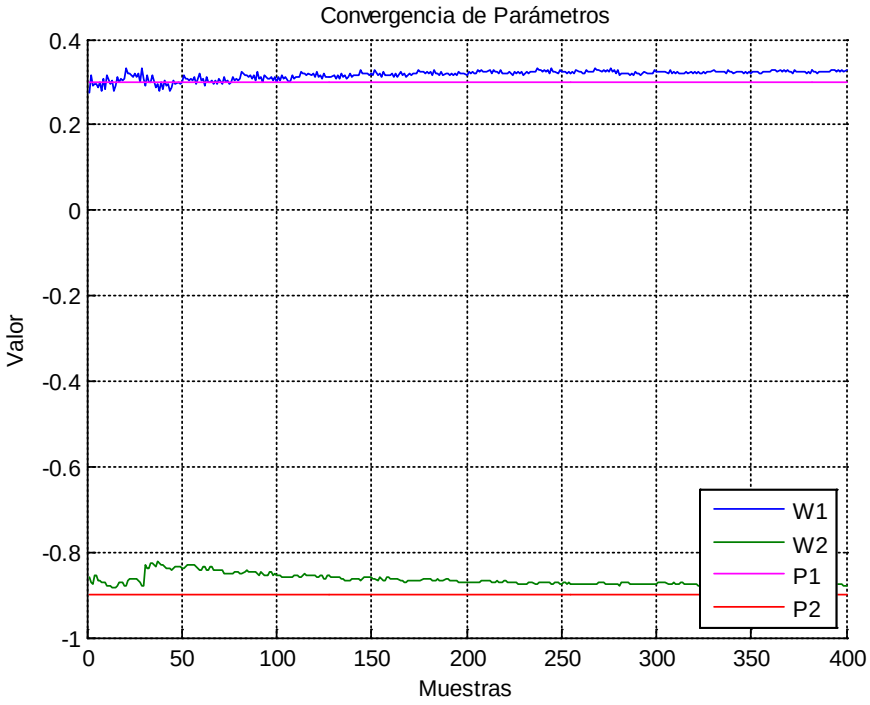

Figura 4.37. Convergencia de Parámetros (sist. segundo orden-Lattice)

Parámetros Reales		Parámetros Estimados	
P1	0.3	W1	0.330002099275589
P2	-0.9	W2	-0.878717124462128

Tabla 4.6. Comparación entre parámetros reales y parámetros estimados (sist. segundo orden-Lattice)

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- El proyecto desarrollado permite realizar la identificación de varios tipos de procesos industriales de primer y segundo orden generados por el sistema de emulación de procesos industriales destacando el algoritmo RLS como el más óptimo para la emulación de plantas en línea, por su rapidez a la hora de converger hacia los parámetros deseados.
- La plataforma *open-source* arduino permite la fácil interacción entre la tarjeta Arduino Mega 2560 para la adquisición de datos y Simulink para el desarrollo de los algoritmos de identificación y su ejecución en tiempo real en línea con el PLC, a través de una librería dedicada para arduino instalada en Simulink.
- El proyecto desarrollado muestra otra funcionalidad del PLC, como es el tratamiento y la generación de señales, pudiendo reemplazar el uso de DSP's para el tratamiento de señales, siempre y cuando el PLC sea de gama alta y soporte texto estructurado como lenguaje de programación.

- Para sistemas de segundo orden, la simulación del algoritmo LMS resulta ineficiente, en la ejecución en Matlab se obtiene convergencia de los parámetros pero no a los valores requeridos, en el entorno de Simulink el algoritmo tiende a ser inestable a lo largo de la ejecución, esto se debe a que la naturaleza no recursiva del algoritmo LMS no permite identificar sistemas con mayor complejidad.
- Con el antecedente de la simulación, las pruebas del algoritmo LMS dentro del sistema en tiempo real confirman su funcionamiento en sistemas de primer orden y su ineficiencia para sistemas de orden superior, se puede notar la tendencia de los parámetros estimados hacia los parámetros deseados, con ciertos períodos de tiempo en los que no coinciden los valores debido al ruido al que la tarjeta arduino se vuelve susceptible, en sistemas de segundo orden el algoritmo resulta totalmente inestable.
- En comparación con el algoritmo LMS, el algoritmo RLS presenta mayores ventajas en la simulación por su naturaleza recursiva, mayor rapidez de convergencia y más exactitud en la estimación de los parámetros, pero se ve aumentada la complejidad computacional por las operaciones matriciales que se realiza a lo largo de la ejecución del algoritmo.
- La ventaja en la rapidez del algoritmo RLS sobre el algoritmo LMS se la puede visualizar en los gráficos del error cuadrático medio (Figura 3.4 y Figura 3.18), mientras el algoritmo LMS necesita de 1600 iteraciones hasta alcanzar el valor mínimo del error cuadrático, el algoritmo RLS solo necesita de aproximadamente 300 iteraciones, verificando una ventaja en la velocidad de convergencia siendo el algoritmo RLS aproximadamente 5,3 veces más rápido que el algoritmo LMS.

- La desventaja de algoritmo RLS implementado se ve reflejada en la ejecución del sistema en tiempo real para sistemas de segundo orden donde el algoritmo se vuelve inestable, esto se debe al ruido al que es susceptible la tarjeta arduino, dicho ruido debería verse opacado con la variación del factor de olvido para no tomar en cuenta algunos valores pasados del par entrada/salida, la desventaja es la falta de implementación de un algoritmo de actualización del factor de olvido y así evitar el efecto de apagado antes de la convergencia de los parámetros.
- Los algoritmos LMS y RLS pueden adaptarse a sistemas multi-entrada y multi-salida, acoplando los vectores o matrices, dependiendo del caso, al número de entradas, salidas y parámetros a estimar, no se ha realizado la simulación del algoritmo LMS para sistema MIMO debido a que se ha comprobado su ineficiencia para sistemas de segundo orden, descartándolo para sistemas multi-variables.
- Como una opción adicional el algoritmo LS-Lattice es otra opción para la estimación de parámetros que, en comparación con el algoritmo RLS, muestra una mejor estabilidad ya que no existe la desventaja de necesitar un factor de olvido variable, sino que, en este caso los coeficientes de reflexión, se van actualizando con cada iteración y para cada etapa del estimador.
- El algoritmo LS-Lattice implementado no puede ser acoplado para sistemas multi-variable, el algoritmo está diseñado para asumir el aumento de parámetros a estimar en aumento de etapas, y cada etapa se ve reflejada en el aumento del orden del denominador de un modelo SISO.

5.2. RECOMENDACIONES

- La tarjeta arduino mega 2560 se ha utilizado como una solución para la adquisición de datos por ser open-source y de bajo costo, pero para este tipo de proyectos donde se necesita que el ruido sea mínimo, se requiere realizar las conexiones cableadas de la forma más adecuada ya que la tarjeta está diseñada para varios tipos de prototipos basados en microcontroladores y no solo en la adquisición de datos.
- Para sistemas o procesos industriales lentos y de primer orden se recomienda el uso del algoritmo LMS para la estimación de parámetros ya que tiene una buena convergencia y no exige muchos recursos computacionales.
- El algoritmo RLS es recomendable para sistemas de donde se necesita una convergencia de parámetros más rápida, y aunque puede ser preferido su uso para cualquier tipo de sistema se debe tomar en cuenta que su costo computacional es mayor en comparación al algoritmo LMS.
- El uso del algoritmo LS-Lattice es recomendado en modelos netamente discretos, en los cuales se requeriría el diseño discreto de un controlador, ya que la disposición de los parámetros que se obtienen de la estimación no es la misma disposición que la de los parámetros de una planta previamente discretizada.
- Al momento de la emulación, se recomienda utilizar un método para sincronizar el período de muestreo de la generación de la señal en el PLC y el período de muestreo que se usa en la adquisición de datos.
- Para un futuro queda abierta la idea de desarrollar un controlador STR(Self Tunning Regulator) dentro del PLC en el cual se desearía integrar la resolución de

la ecuación a diferencias con el proceso de identificación en línea para identificar los parámetros del controlador a través de la estimación de los parámetros de la planta.

- Se propone también explorar algoritmos derivados del RLS como por ejemplo algoritmos que implementen una actualización del factor de olvido en cada iteración y mejorar la velocidad en la convergencia de los parámetros identificados.

GLOSARIO

- **Processing**: Plataforma de código abierto para prototipos electrónicos.
- **Matlab/Simulink**: Marca registrada de Mathworks (http://www.mathworks.com/).
- **Open-source**: Hardware y software desarrollado libremente.
- **PRBS**: Pseudo-Random Binary Sequence.
- **IEC**: Comisión Internacional Electrotecnia
- **CompactLogix 1768**: Marca registrada de Allen-Bradley (ab.rockwellautomation.com/).
- **LMS**: *Least Mean Square*
- **RLS:** *Recursive Least Square*
- **LS-Lattice:** *Least Square Lattice*
- **MSE:** *Mean Square Error*
- **Scope**: Visualizador gráfico de datos
- **DDE**: *Dynamic Data Exchange*
- **DSP:** *Digital Signal Processor*

BIBLIOGRAFÍA

Aguado Behar, A., & Martínez Iranzo, M. (2003). *Identificación y Control Adaptativo.* Madrid: Pearson Educación.

Arduino. (s.f.). *Arduino*. Recuperado el 2012, de Productos Arduino: http://arduino.cc/en/

Enríquez Herrador, R. (2009). *Guia de Usuario de Arduino.* Córdoba: Universidad de Córdoba.

Esquivel, V. M. (s.f.). *Análisis de Sistemas y Señales con Cómputo Avanzado.* Mexico: Unidad de Apoyo Editorial UNAM.

Evans, B. (2011). *Arduino Programming Notebook.* Recuperado el 2012, de Ardumania: http://www.ardumania.es/

Martín, I. G. (02 de 11 de 2006). *Introduccion a Arduino.* Recuperado el 2012, de Lagunak: http://lagunak.gisa-elkartea.org/

Meizoso López, M. d., Piñon Pazos, A., & Ferreiro Garcia, R. (s.f.). *Implementación de Modelos de Sistemas Físicos Mediante Autómatas Programables y Circuitos Analógicos.* La Coruña: Universidad de La Coruña.

Mendoza Jiménez , S., & Guillén Garcia, Y. (s.f.). *Controladores Lógico Programables (PLC'S).* Expressa.

Miramontes, G. (2005). *Procesamiento Digital de Señales Introducción con Teoría y Práctica.* México: Cuerpo Académico de Procesamiento Digital de Señales.

Ogata, K. (1996). *Sistemas de Control en Tiempo Discreto Segunda Edición* . Mexico: Prentice Hall.

Ogata, K. (1998). *Ingeniería de Control Moderna 3ed.* México: Prentice Hall.

Pagola, L. (2002). *Control Digital.* Madrid: Universidad Pontificia Comillas.

Roca Cusidó, A. (2002). *Control de Procesos, 2ed.* México: Alfaomega.

Rockwell Automation. (s.f.). *Controladores CompactLogix L4x*. Recuperado el 2013, de Rockwell Automation: http://ab.rockwellautomation.com/es/

Rockwell Software. (2008). *Texto Estructurado de los controladores Logix 5000.*

Rodríguez Ramírez , D., & Bordóns Alba, C. (2005). *Apuntes de Ingeniería de Control.*

S. Thomas, A. (1986). *Adaptive Signal Processing (Theory and Applications).* New York: Springer-Verlag.

Sirgo Blanco, J. A. (2008). *Autómatas Programables* . Asturias: Universidad de Oviedo.

Tamani, B. V. (2007). *Obtención de Modelos de Procesos Mediante Métodos de Identificación Recursiva* . Lima: Universidad Nacional Mayor de San Marcos.

Vega, A. A. (2004). *Control Digital* .

FECHA DE ENTREGA

El proyecto de titulación "DESARROLLO DE UN SISTEMA DE IDENTIFICACIÓN DE PROCESOS INDUSTRIALES EN LÍNEA USANDO LA PLATAFORMA OPEN-SOURCE ARDUINO Y MATLAB/SIMULINK" fue entregado al Departamento de Eléctrica y Electrónica y reposa en la Universidad de las Fuerzas Armadas-ESPE.

Sangolquí,…..de…….. de 2014

Elaborado por:

_____

Paul Sebastián Aluisa Chalá

Autoridad:

_____

Ing. Luis Orozco Brito

Director de Carrera

Ingeniería en Electrónica, Automatización y Control