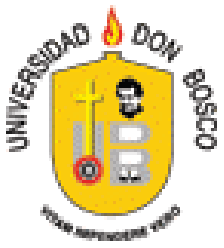
UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERÍA
ESCUELA DE ELECTRÓNICA



INTERFAZ GRÁFICA PARA LA OBTENCIÓN DEL MODELO MATEMÁTICO, DISEÑO DE CONTROLADORES, Y CONTROL DE UNA PLANTA

TRABAJO DE GRADUACIÓN

PRESENTADO POR

Renato Alberto Leiva Gutiérrez

Carlos René Vásquez Ortiz

PARA OPTAR AL GRADO DE

Ingeniero en Electrónica

Asesor:

Ing. Edgardo Cruz Zeledón

Julio de 2004

Soyapango – El Salvador – Centro América

UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERÍA
ESCUELA DE ELECTRÓNICA

AUTORIDADES:

RECTOR
ING. FEDERICO HUGUET RIVERA

VICERRECTOR ACADÉMICO
PBRO. VÍCTOR BERMÚDEZ, sdb

SECRETARIO GENERAL
LIC. MARIO RAFAEL OLMOS

DECANO DE LA FACULTAD DE INGENIERÍA
ING. GODOFREDO GIRÓN

DIRECTOR DE ESCUELA DE ELECTRÓNICA
ING. OSCAR DURÁN VIZCARRA

ASESOR DEL TRABAJO DE GRADUACIÓN
ING. EDGARDO CRUZ ZELEDÓN

JURADO EVALUADOR
ING. ALEXANDER GUZMÁN
ING. RICARDO MEDRANO
ING. EDUARDO RIVERA

UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERÍA
ESCUELA DE ELECTRÓNICA

JURADO EVALUADOR DEL TRABAJO DE GRADUACIÓN

Ing. Alexander Guzmán
JURADO

Ing. Ricardo Medrano
JURADO

Ing. Eduardo Rivera
JURADO

Ing. Edgardo Cruz Zeledón
ASESOR

A mi madre, a quien dedico este trabajo, por tanto apoyo brindado, quien compartió conmigo los altibajos vividos en este esfuerzo de superación profesional, a esa noble persona que con infinita paciencia me alentó a continuar en los momentos difíciles e inciertos de mi carrera, a ella, que con su amor, llenó mi alma de fe, para acercarme a la fuente de lo divino y superar las adversidades. Por todo esto y por mucho mas:

Gracias, Amada Madre

“ La integral de la vida se encuentra definida por el diferencial de nuestros días y los limites entre nacer y morir ”

Renato Alberto Leiva Gutiérrez

A su gran misericordia y amor para conmigo, gracias por darme este regalo en mi vida, sin ti, hubiese sido imposible este triunfo, a **Dios mi Padre**.

A quien con tanto sacrificio me dió todo su apoyo en todas las áreas, en las tristezas y alegrías estuvo a mi lado, no tengo con que pagarte todo lo que haz hecho por mi, **Gracias, Querida Madre**.

En este proyecto que aquí concluye, existe una lista muy grande de gente involucrada que sería imposible nombrar, espero abarcarlos a todos: a mi núcleo familiar, a mi familia espiritual y a mis amigos.

¡Gracias a todos, de todo corazón!

"Mas la senda de los justos es como la luz de la aurora, que va en aumento hasta que el día es perfecto"
Proverbios 4:18.

Carlos René Vásquez Ortíz.

Indice.

1 Introducción.

El presente documento recopila la información concerniente al trabajo de graduación denominado: "Interfaz Gráfica Para la Obtención del Modelo Matemático y Control de una Planta".

El marco teórico expone los métodos para la obtención de un modelo matemático, para lo cual se desarrollan los tipos de estructuras en la identificación de sistemas, se establecen técnicas para la identificación paramétrica y la no paramétrica, también se establecen criterios y recomendaciones prácticas en la realización de experimentos que tengan como fin el modelado matemático.

Posterior a esto se expone la estructura estándar de un PID, se estudian las propiedades de cada una de sus partes como son las acciones: P, PI, PD, PID; los métodos empíricos para calibrar un controlador PID, se describe el método de la curva de reacción de Ziegler-Nichols, se desarrolla la modificación a este método con el cálculo de áreas bajo la curva de reacción.

El diseño del controlador PID discreto inicia con la expresión continua como ecuación diferencial hasta transformarlo en una ecuación en diferencias.

Luego se presenta el diseño de la aplicación, en esta parte se explica la lógica de cada una de las partes que componen la aplicación desarrollada, como son: comunicación con la PCI, la captura y presentación de la información, el análisis y tratamiento de los datos, la interacción con el usuario.

Aunado a lo anterior, la determinación de los parámetros del PID, como el control a lazo cerrado.

También se ha incluido en la aplicación, la implementación de un filtro digital para la captura de los datos.

Para finalizar se muestran los resultados de pruebas realizadas en una planta electrónica, se evalúa comparativamente el modelo matemático teórico de la planta, contra modelos obtenidos experimentalmente con la aplicación desarrollada.

Se compara el controlador discreto diseñado, con un controlador continuo del laboratorio.

2 Objetivos

2.1 Objetivo General.

Obtención del modelado matemático a través de un adquisidor de datos y técnicas de muestreo; análisis, con la ayuda de las cajas de herramientas de MATLAB, en especial la de Control Automático; diseño de controladores y control en lazo cerrado de una planta industrial.

2.2 Objetivos Específicos.

- Desarrollo de una interfaz gráfica amigable al usuario.

- Vinculación de las cajas de herramientas de control automático existentes en MATLAB, para lo siguiente:
  - Análisis:
    - Determinación de la Función de Transferencia.
    - Respuesta en el tiempo al escalón unitario.
    - Diagrama del lugar de las raíces.
    - Diagramas de Bode.
    - Diagrama de Nyquist.
  - Determinación de los parámetros de configuración de los controladores PID
    - Tiempo Integral.
    - Tiempo Derivativo.
    - Ganancia Proporcional.
- Desarrollar la comunicación entre el software (MATLAB) y el Hardware (adquisidor) a través del puerto serial.

- Diseño e implementación de un PID discreto, para controlar a lazo cerrado una planta.

3 Alcances y Limitantes.

- El desarrollo de la interfaz gráfica será en una GUI del programa MATLAB, donde se podrá observar la información y monitorear la planta de control.

- La función de transferencia, diagramas de Bode, respuesta temporal, lugar de las raíces y la obtención de los parámetros de ajustes de los controladores PID se determinarán haciendo uso de las librerías de control automático de MATLAB mediante un enlace con la interfaz gráfica desarrollada.

- El adquisidor de señal que se utilizaría, es el que está disponible en los laboratorios del CITT, el PC/I-1 LM8912.

- En el sistema de lazo cerrado se va a limitar a controles tipo: PID, cuyos parámetros podrán ser configurados por el usuario.

- La respuesta será un modelo aproximado cuya exactitud dependerá de:
    1. La frecuencia de muestreo del adquisidor de datos.
    2. La resolución de los bits en la conversión de análogo-digital y viceversa.
    3. El algoritmo empleado para el modelado.

- El modelo matemático se obtendrá aplicando una señal tipo escalón a la entrada del sistema y analizando las muestras temporales a su salida.

4 Marco Teórico.

4.1 Definiciones.[1]

El diseño de un controlador continuo o discreto, ya sea mediante técnicas clásicas o en variables de estado, requiere de un modelo de la planta a controlar que caracterice su comportamiento dinámico. Este modelo permite al diseñador realizar y validar mediante simulación el ajuste de los parámetros del controlador que permiten obtener una respuesta que satisfaga las especificaciones de diseño. En este tema se estudian diferentes alternativas para obtener el modelo de un sistema como paso previo al diseño de un controlador.

4.1.1 Concepto de sistema.

Un ***sistema*** es toda realidad en la que interactúan variables de diferentes tipos para producir señales observables. Las señales observables que son de interés para el observador se denominan ***salidas*** del sistema, mientras que las señales que pueden ser manipuladas libremente por dicho observador son las ***entradas*** del mismo. El resto de señales que influyen en la evolución de las salidas pero no pueden ser manipuladas por el observador se denominan ***perturbaciones***.

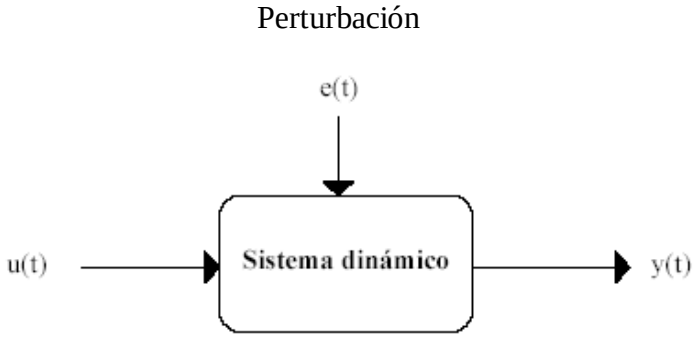


Figura 4.1. Sistema dinámico con entrada u(t), perturbación e(t) y salida y(t).

4.1.2 Modelo de un sistema

Cuando se hace necesario conocer el comportamiento de un sistema en unas determinadas condiciones y ante unas determinadas entradas, se puede recurrir a la experimentación sobre dicho sistema y a la observación de sus salidas. Sin embargo, en muchos casos la experimentación puede resultar compleja o incluso imposible de llevar a

cabo, lo que hace necesario trabajar con algún tipo de representación que se aproxime a la realidad, y a la que se conoce como *modelo*. Básicamente, un modelo es una herramienta que permite predecir el comportamiento de un sistema sin necesidad de experimentar sobre él.

4.1.3 Tipos de modelos

Los modelos de sistemas físicos pueden ser de muy diversos tipos. Una clasificación, en función del grado de formalismo matemático que poseen, es la siguiente:

1. *Modelos mentales, intuitivos o verbales.* Estos modelos carecen de formalismo matemático. Para conducir un coche, por ejemplo, se requiere un modelo mental o intuitivo sobre el efecto que produce el movimiento del volante, pero no es necesario caracterizar dicho efecto mediante ecuaciones matemáticas exactas.

2. *Modelos no paramétricos.* Muchos sistemas quedan perfectamente caracterizados mediante un gráfico o tabla que describa sus propiedades dinámicas mediante un número no finito de parámetros. Por ejemplo, un sistema lineal queda definido mediante su respuesta al impulso o al escalón, o bien mediante su respuesta en frecuencia.

3. *Modelos paramétricos o matemáticos.* Para aplicaciones más avanzadas, puede ser necesario utilizar modelos que describan las relaciones entre las variables del sistema mediante expresiones matemáticas como pueden ser ecuaciones diferenciales (para sistemas continuos) o en diferencias (para sistemas discretos). En función del tipo de sistema y de la representación matemática utilizada, los sistemas pueden clasificarse en:
    - Determinísticos o estocásticos. Se dice que un modelo es determinístico cuando expresa la relación entre entradas y salidas mediante una ecuación exacta. Por contra, un modelo es estocástico si posee un cierto grado de incertidumbre. Estos últimos se definen mediante conceptos probabilísticos o estadísticos.

    - Dinámicos o estáticos. Un sistema es estático cuando la salida depende únicamente de la entrada en ese mismo instante (un resistor, por ejemplo, es

un sistema estático). En estos sistemas existe una relación directa entre entrada y salida, independiente del tiempo. Un sistema dinámico es aquél en el que las salidas evolucionan con el tiempo tras la aplicación de una determinada entrada (por ejemplo, una red RC). En estos últimos, para conocer el valor actual de la salida es necesario conocer el tiempo transcurrido desde la aplicación de la entrada.

- Continuos o discretos. Los sistemas continuos trabajan con señales continuas, y se caracterizan mediante ecuaciones diferenciales. Los sistemas discretos trabajan con señales muestreadas, y quedan descritos mediante ecuaciones en diferencias.

Todo modelo matemático o paramétrico, por tanto, consta de una o varias ecuaciones que relaciona/n la/s entrada/s y salida/s (en los modelos dinámicos la variable t -tiempo- juega también un papel primordial).

Un ejemplo de modelo matemático de un sistema dinámico con dos entradas $\mathbf{u_1}$ y $\mathbf{u_2}$ y una salida **y** puede ser el siguiente:

$$y(t) = 5 \cdot \frac{du_1(t)}{dt} + 3 \cdot u_1(t) - 2 \cdot u_2(t) \qquad \text{ec 4.1}$$

Donde hay que distinguir:

1. la *estructura* del modelo: $y(t) = k_1 \cdot \frac{du_1(t)}{dt} + k_2 \cdot u_1(t) + k_3 \cdot u_2(t)$ ec 4.2

2. los *parámetros* del modelo: $k_1 = 5$; $k_2 = 3$; $k_3 = -2$.

De ahí que a los modelos matemáticos se les conozca más comúnmente como *modelos paramétricos*, ya que pueden definirse mediante una *estructura* y un número finito de *parámetros*.

4.1.4 Métodos de obtención de modelos.

Existen dos métodos principales para obtener el modelo de un sistema:

1. *Modelado teórico*. Se trata de un método analítico, en el que se recurre a leyes básicas de la física para describir el comportamiento dinámico de un fenómeno o proceso.

2. *Identificación del sistema*. Se trata de un método experimental que permite obtener el modelo de un sistema a partir de datos reales recogidos de la planta bajo estudio. El modelado teórico tiene un campo de aplicación restringido a procesos muy sencillos de modelar, o a aplicaciones en que no se requiera gran exactitud en el modelo obtenido. En muchos casos, además, la estructura del modelo obtenido a partir del conocimiento físico de la planta posee un conjunto de parámetros desconocidos y que sólo se pueden determinar experimentando sobre el sistema real. De ahí la necesidad de recurrir a los métodos de *identificación de sistemas.*

Los modelos obtenidos mediante técnicas de identificación tienen, sin embargo, las siguientes desventajas:

1. Su rango de validez suele ser limitado (sólo son aplicables a un determinado punto de trabajo, un determinado tipo de entrada o un proceso concreto).

2. En muchos casos es difícil dar significado físico al modelo obtenido, puesto que los parámetros identificados no tienen relación directa con ninguna magnitud física. Estos parámetros se utilizan sólo para dar una descripción aceptable del comportamiento conjunto del sistema.

En la práctica, lo ideal es recurrir a una mezcla de ambos métodos de modelado para obtener el modelo final. El uso de datos reales para identificar los parámetros del modelo provee a éste de una gran exactitud, pero el proceso de identificación se ve tanto más facilitado cuanto mayor sea el conocimiento sobre las leyes físicas que rigen el proceso.

4.2 Identificación de sistemas.[2]

Se entiende por identificación de sistemas a la obtención de forma experimental de un modelo que reproduzca con suficiente exactitud, para los fines deseados, las características dinámicas del proceso objeto de estudio.

4.2.1 El proceso de identificación.

En términos generales, el proceso de identificación comprende los siguientes pasos:

1. *Obtención de datos de entrada - salida*. Para ello se debe excitar el sistema mediante la aplicación de una señal de entrada y registrar la evolución de sus entradas y salidas durante un intervalo de tiempo.

2. *Tratamiento previo de los datos registrados*. Los datos registrados están generalmente acompañados de ruidos indeseados u otro tipo de imperfecciones que puede ser necesario corregir antes de iniciar la identificación del modelo. Se trata, por tanto, de 'preparar' los datos para facilitar y mejorar el proceso de identificación.

3. *Elección de la estructura del modelo.* Si el modelo que se desea obtener es un modelo paramétrico, el primer paso es determinar la estructura deseada para dicho modelo. Este punto se facilita en gran medida si se tiene un cierto conocimiento sobre las leyes físicas que rigen el proceso.

4. *Obtención de los parámetros del modelo.* A continuación se procede a la estimación de los parámetros de la estructura que mejor ajustan la respuesta del modelo a los datos de entrada-salida obtenidos experimentalmente.

5. *Validación del modelo*. El último paso consiste en determinar si el modelo obtenido satisface el grado de exactitud requerido para la aplicación en cuestión. Si se llega a la conclusión de que el modelo no es válido, se deben revisar los siguientes aspectos como posibles causas:

a) El conjunto de datos de entrada-salida no proporciona suficiente información sobre la dinámica del sistema.
b) La estructura escogida no es capaz de proporcionar una buena descripción del modelo.
c) El criterio de ajuste de parámetros seleccionado no es el más adecuado.

Dependiendo de la causa estimada, deberá repetirse el proceso de identificación desde el punto correspondiente. Por tanto, el proceso de identificación es un proceso iterativo, cuyos pasos pueden observarse en el organigrama de la figura 4.2.

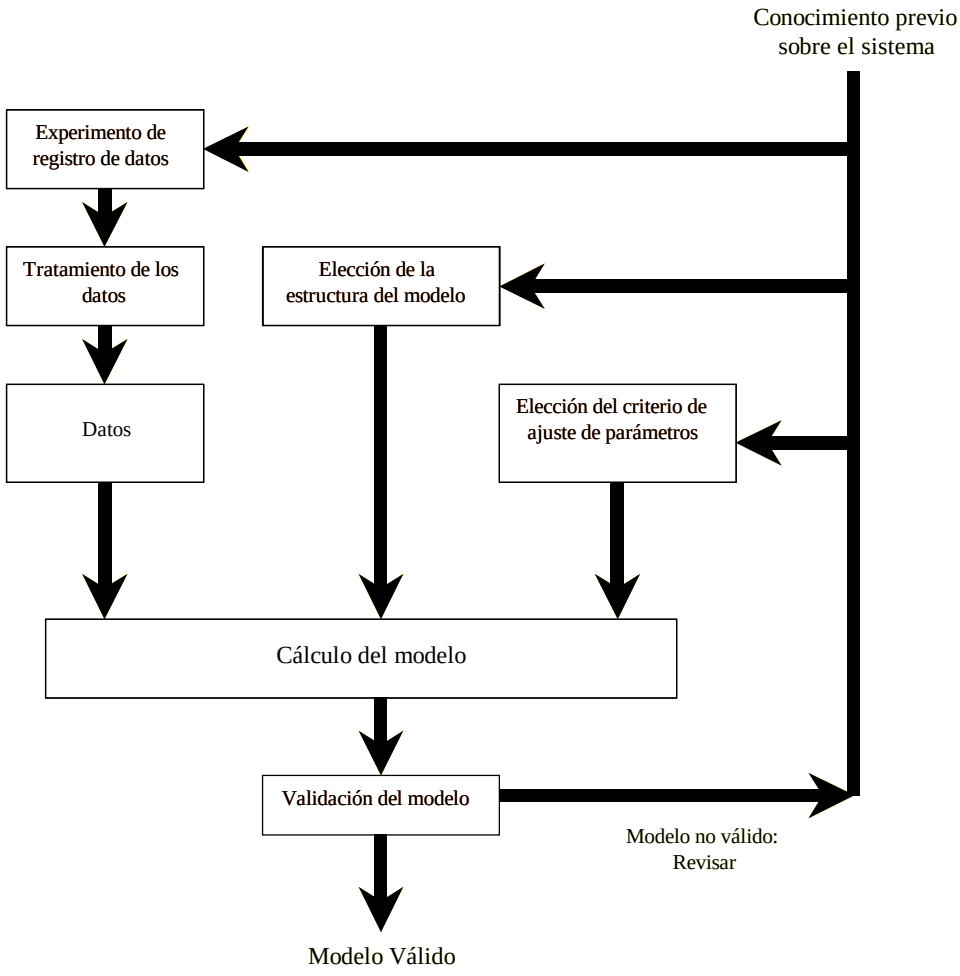


Figura 4.2. El proceso de identificación

4.2.2 Métodos de identificación.

Existen diversos métodos de identificación, que pueden clasificarse según distintos criterios:

- ***Dependiendo del tipo de modelo obtenido:***
  1. Métodos no paramétricos, que permiten obtener modelos no paramétricos del sistema bajo estudio. Algunos de estos métodos son: análisis de la respuesta transitoria, análisis de la respuesta en frecuencia, análisis de la correlación, análisis espectral, análisis de Fourier, etc.

  2. Métodos paramétricos, que permiten obtener modelos paramétricos. Estos métodos requieren la elección de una posible estructura del modelo, de un criterio de ajuste de parámetros, y por último de la estimación de los parámetros que mejor ajustan el modelo a los datos experimentales.

- ***Dependiendo de la aplicación:***
  1. Métodos de identificación off-line (a posteriori), utilizados en aquellas aplicaciones en que no se requiera un ajuste continuado del modelo. En estos casos, se realiza la identificación previa de la planta, considerándose que la validez de los parámetros obtenidos no se verá alterada con el paso del tiempo.

  2. Métodos de identificación on-line (identificación recursiva), en los que los parámetros se van actualizando continuamente a partir de los nuevos datos de entrada-salida obtenidos durante la evolución del proceso. Estos métodos son muy utilizados en sistemas de control adaptativo.

- ***Dependiendo del criterio de ajuste de los parámetros***. Existen diversos métodos matemáticos para ajustar los parámetros de una estructura a un conjunto de datos de entrada-salida. Algunos de los más utilizados en el campo de la identificación son el método de mínimos cuadrados y el método de las variables instrumentales.

4.3 Técnicas de identificación no paramétrica.[2]

Los métodos de identificación no paramétricos permiten obtener modelos o representaciones no paramétricas de la planta bajo estudio.

4.3.1 Conceptos Previos.

Supóngase el siguiente sistema:

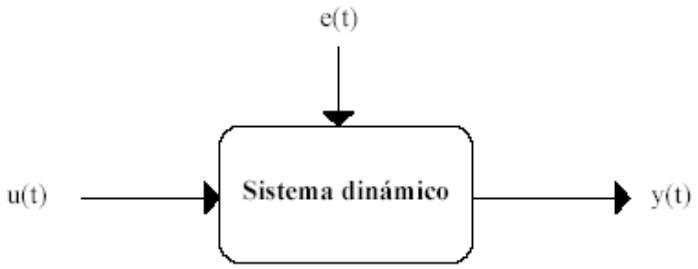


Figura 4.3. Señales de interés para el modelado no paramétrico

Suponiendo que el sistema es lineal, la relación entre la salida del sistema y(t), su entrada u(t) y el ruido e(t) puede expresarse del siguiente modo:

$$y(t) = G(q^{-1}) \cdot u(t) + e(t) \qquad \text{ec. 4.3}$$

Donde q^{-1} es el operador retardo y el producto $G(q^{-1})u(t)$ representa la siguiente secuencia:

$$G(q^{-1}) \cdot u(t) = \sum_{k=1}^{\infty} g(k)u(t-k) \qquad \text{ec. 4.4}$$

Y por tanto:

$$G(q^{-1}) = \sum_{k=1}^{\infty} g(k)q^{-k} \qquad \text{ec. 4.5}$$

La secuencia g(k) se conoce como *respuesta al impulso* del sistema, y coincide con la salida del mismo cuando a la entrada se aplica un impulso unitario. Por otro lado, la función G(z) es la *función de transferencia* del sistema. Evaluando esta última a lo largo del círculo unidad ($z=e^{j\omega}$) se obtiene la llamada *respuesta en frecuencia* del sistema, $G(e^{j\omega})$. La respuesta al impulso es un modelo no paramétrico que se define en el dominio del tiempo, mientras que la respuesta en frecuencia es una descripción no paramétrica en el dominio de la frecuencia.

4.3.2 Identificación no paramétrica en el dominio del tiempo.

Mediante esta técnica de identificación se pretende obtener la respuesta al impulso del sistema, o bien la respuesta al escalón del mismo (pudiendo obtenerse esta última mediante una integración de la primera). Para ello, debe registrarse la evolución temporal de la salida del sistema tras la aplicación de una señal impulso o escalón. Obviamente, la imposibilidad de conseguir este tipo de señales en la práctica lleva a utilizar un método indirecto para obtener la respuesta impulsiva, conocido como *análisis de la correlación.* Si se escoge como entrada al sistema u(t) un ruido blanco, cuya función de covarianza es:

$$R_u(\tau) = E\left[u(t+\tau)u(t)\right] = \begin{cases} \lambda & \text{si} \quad \tau = 0 \\ 0 & \text{si} \quad \tau \neq 0 \end{cases} \qquad \text{ec. 4.6}$$

Entonces la correlación cruzada entre la entrada y la salida puede ponerse del siguiente modo:

$$R_{yu}(\tau) = E\left[y(t+\tau)u(t)\right] = \lambda g(\tau) \qquad \text{ec. 4.7}$$

Y por tanto la respuesta al impulso puede obtenerse a partir de las N muestras registradas de la entrada y la salida del sistema del siguiente modo:

$$g(\tau) = \frac{1}{\lambda N}\sum_{t=1}^{N} y(t+\tau)u(t) \qquad \text{ec. 4.8}$$

Si la entrada al sistema no es un ruido blanco puro, puede considerarse como el resultado de la aplicación de un filtro $L(q^{-1})$ a un ruido blanco. Aplicando dicho filtro a los datos de salida del sistema puede realizarse la misma operación indicada anteriormente para obtener los coeficientes de g(k).

Este método es muy apropiado para obtener una idea rápida de la relación entre distintas señales del sistema, retardos, constantes de tiempo y ganancias estáticas del mismo.

4.3.3 Identificación no paramétrica en el dominio de la frecuencia

En este caso, el modelo resultante es una representación de la respuesta en frecuencia del sistema, obtenida mediante la aplicación de señales de entrada sinusoidales de distintas frecuencias. Cuando no sea posible aplicar este tipo de entradas, puede recurrirse a la aplicación de un ruido blanco, que permite obtener la respuesta en frecuencia mediante el conocido ***análisis espectral***. Este análisis se basa en la realización de la transformada de Fourier de las funciones de covarianza de la entrada y la salida y la correlación entre la entrada y la salida. Por tanto, definiendo las siguientes funciones de correlación:

$$R_u(\tau) = E\left[u(t+\tau)\cdot u(t)\right] = {}^{1}\!/_{N}\cdot\sum_{t=1}^{N} u(t+\tau)\cdot u(t)$$
$$R_{yu}(\tau) = E\left[y(t+\tau)\cdot u(t)\right] = {}^{1}\!/_{N}\cdot\sum_{t=1}^{N} y(t+\tau)\cdot u(t) \qquad \text{ec. 4.9}$$

Y sus transformadas de Fourier:

$$\Phi_u(\omega) = \sum_{\tau=-\infty}^{\infty} R_u(\tau)\cdot e^{-j\omega\tau}$$
$$\Phi_{yu}(\omega) = \sum_{\tau=-\infty}^{\infty} R_{yu}(\tau)\cdot e^{-j\omega\tau} \qquad \text{ec. 4.10}$$

Se demuestra que puede obtenerse la respuesta en frecuencia del sistema mediante la siguiente expresión:

$$G(e^{j\omega}) = \frac{\Phi_{yu}(\omega)}{\Phi_u(\omega)} \qquad \text{ec. 4.11}$$

Las principales ventajas de este método son el no requerir un procesamiento complejo de los datos, ni ningún tipo de conocimiento previo sobre la planta, a excepción de que ésta sea lineal. Además, permite concentrar los datos obtenidos en torno al margen de frecuencias de interés. El principal inconveniente es que el modelo resultante no puede usarse directamente para simulación.

4.4 Técnicas de identificación paramétrica.[2]

Los modelos paramétricos, a diferencia de los anteriores, quedan descritos mediante una estructura y un número finito de parámetros que relacionan las señales de interés del sistema (entradas, salida y perturbaciones). En muchas ocasiones es necesario realizar la identificación de un sistema del cual no se tiene ningún tipo de conocimiento previo. En estos casos, se suele recurrir a modelos estándar, cuya validez para un amplio rango de sistemas dinámicos ha sido comprobada experimentalmente. Generalmente estos modelos permiten describir el comportamiento de cualquier sistema lineal. La dificultad radica en la elección del tipo de modelo (orden del mismo, número de parámetros, etc.) que se ajuste satisfactoriamente a los datos de entrada - salida obtenidos experimentalmente.

4.4.1 Tipos de modelos paramétricos

Generalmente los modelos paramétricos se describen en el dominio discreto, puesto que los datos que sirven de base para la identificación se obtienen por muestreo. En el caso de que se requiera un modelo continuo, siempre es posible realizar una transformación del dominio discreto al continuo. La expresión más general de un modelo discreto es del tipo:

$$s(t) = \eta(t) + w(t) \qquad \text{ec. 4.12}$$

Donde w(t) es el término que modela la salida debida a las perturbaciones, $\eta(t)$ la salida debida a la entrada, y s(t) la salida medible del sistema. Cada uno de estos términos puede desarrollarse de la siguiente forma:

$$\eta(t) = G(q^{-1},\theta)\cdot u(t) \qquad \text{ec. 4.13}$$

$$w(t) = H(q^{-1},\theta)\cdot e(t) \qquad \text{ec. 4.14}$$

$$s(t) = A(q^{-1},\theta)\cdot y(t) \qquad \text{ec. 4.15}$$

Donde q^{-1} es el operador retardo, θ representa un vector de parámetros, u(t) y e(t) son la entrada al sistema y el ruido de entrada al mismo respectivamente e y(t) es la salida de interés del sistema (que puede no coincidir con la salida medible). Tanto $G(q^{-1},\theta)$ como $H(q^{-1},\theta)$ son cocientes de polinomios del tipo:

$$G(q^{-1},\theta)=\frac{B(q^{-1})}{F(q^{-1})}=\frac{b_1\cdot q^{-nk}+b_2\cdot q^{-nk-1}+\cdots+b_{nb}\cdot q^{-nk-nb+1}}{1+f_1\cdot q^{-1}+\cdots+f_{nf}\cdot q^{-nf}}$$
$$H(q^{-1},\theta)=\frac{C(q^{-1})}{D(q^{-1})}=\frac{1+c_1\cdot q^{-1}+\cdots+c_{nc}\cdot q^{-nc}}{1+d_1\cdot q^{-1}+\cdots+d_{nd}\cdot q^{-nd}} \qquad \text{ec. 4.16}$$

Y $A(q^{-1},\theta)$ un polinomio del tipo:

$$A(q^{-1},\theta)=1+a_1\cdot q^{-1}+\ldots+a_{na}\cdot q^{-na} \qquad \text{ec. 4.17}$$

El vector de parámetros θ contiene los coeficientes a_i, b_i, c_i, d_i y f_i de las funciones de transferencia anteriores. La estructura genérica de estos modelos es por tanto:

$$A(q^{-1})\cdot y(t)=G(q^{-1},\theta)\cdot u(t)+H(q^{-1},\theta)\cdot e(t)=\frac{B(q^{-1})}{F(q^{-1})}\cdot u(t)+\frac{C(q^{-1})}{D(q^{-1})}\cdot e(t) \qquad \text{ec. 4.18}$$

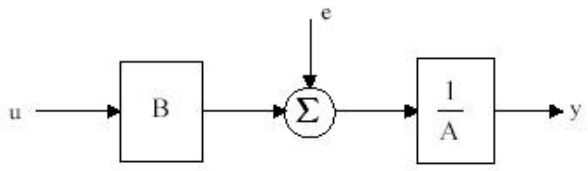
Para elegir la ***estructura*** de este tipo de modelos hay que determinar el orden de cada uno de los polinomios anteriores, es decir na, nb, nc, nd, nf y el retardo entre la entrada y la salida nk. Una vez elegidos estos valores, sólo queda determinar el vector de coeficientes θ (a_i, b_i, c_i, d_i y f_i) que hacen que el modelo se ajuste a los datos de entrada - salida del sistema real.

En muchos casos, alguno de los polinomios anteriores no se incluye en la descripción del modelo, dando lugar a los siguientes casos particulares, entre otros:

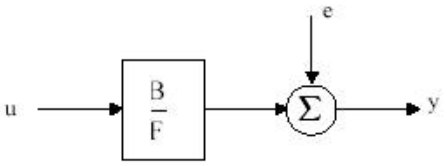
Tipo de modelo	Condición	Estructura resultante
Modelo ARX	$F(q^{-1})=D(q^{-1})=C(q^{-1})=1$	$A(q^{-1})y(t)=B(q^{-1})\cdot u(t)+e(t)$
Modelo Output Error (OE)	$C(q^{-1})=D(q^{-1})=A(q^{-1})=1$	$y(t)=\frac{B(q^{-1})}{F(q^{-1})}\cdot u(t)+e(t)$
Modelo ARMAX	$F(q^{-1})=D(q^{-1})=1$	$A(q^{-1})\cdot y(t)=B(q^{-1})\cdot u(t)+C(q^{-1})\cdot e(t)$
Modelo Box Jenkins (BJ)	$A(q^{-1})=1$	$y(t)=\frac{B(q^{-1})}{F(q^{-1})}\cdot u(t)+\frac{C(q^{-1})}{D(q^{-1})}\cdot e(t)$

Tabla 4.1. Diferentes estructuras de modelos paramétricos.

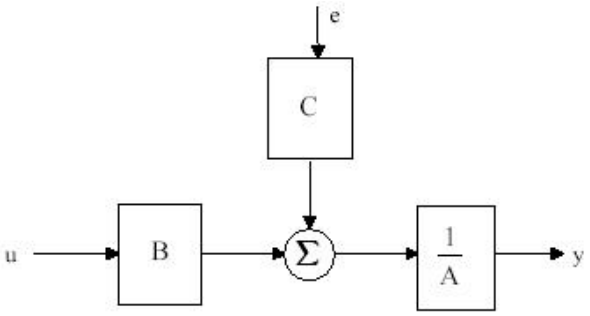
En la figura 4.4 se muestra el diagrama de bloques equivalente para cada uno de los modelos anteriores.



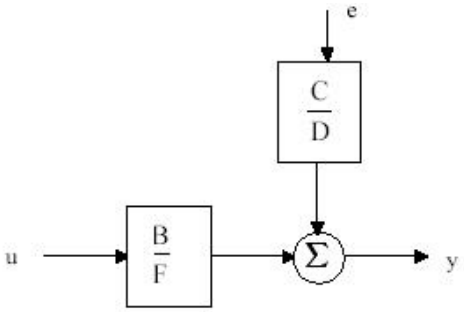
a) Estructura ARX



b) Estructura OE



c) Estructura ARMAX



d) Estructura BJ

Figura 4.4. Diagramas de bloques de las estructuras de la tabla 4.1.

La anulación de alguno de los polinomios, resultando estructuras simplificadas, facilita el proceso de ajuste de parámetros. Cada una de las estructuras (ARX, ARMAX, OE o BJ) tiene sus propias características y debe ser elegida fundamentalmente en función del punto en el que se prevé que se añade el ruido en el sistema. En cualquier caso, puede ser necesario ensayar con varias estructuras y con varios órdenes dentro de una misma estructura hasta encontrar un modelo satisfactorio.

4.4.1.1 Ejemplo de estructura ARMAX.

El modelo ARMAX es:

$$A(q^{-1}) \cdot y(t) = B(q^{-1}) \cdot u(t) + C(q^{-1}) \cdot e(t) \qquad \text{ec. 4.19}$$

Donde:

$$A(q^{-1},q) = 1 + a_1 \cdot q^{-1} + \ldots + a_{na} \cdot q^{-na} \qquad \text{ec. 4. 20}$$

$$B(q^{-1}) = b_1 \cdot q^{-nk} + b_2 \cdot q^{-nk-1} + \cdots + b_{nb} \cdot q^{-nk-nb+1} \qquad \text{ec. 4. 21}$$

$$C(q^{-1}) = 1 + c_1 \cdot q^{-1} + ... + a_{nc} \cdot q^{-nc} \qquad \text{ec. 4. 22}$$

Tomando la función discreta de segundo orden: $y(q) = u^2(q)$

y un vector de datos de 10 muestras, tenemos:

$$u(q) = [\ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10\ ] \quad \text{(entrada)}$$

$$y(q) = u^2(q) = [\ 1 \quad 4 \quad 9 \quad 16 \quad 25 \quad 36 \quad 49 \quad 64 \quad 81 \quad 100\ ] \quad \text{(salida)}$$

Encontrando el orden y el retraso para este ejemplo en ec. 3.20, ec.3.21 y ec.3.22; tenemos:

$$A(q^{-1}) = 1 + a_1 \cdot q^{-1} + a_2 \cdot q^{-2} \qquad \text{ec. 4.23}$$

$$B(q^{-1}) = b_1 \cdot q^{-2} + b_2 \cdot q^{-3} \qquad \text{ec. 4.24}$$

$$C(q^{-1}) = 1 + c_1 \cdot q^{-1} + a_2 \cdot q^{-2} \qquad \text{ec. 4.25}$$

Donde: na = 2, nb = 2, nc = 2 y nk = 2.

Determinaremos el vector de parámetros θ que contiene los coeficientes a_i , b_i y c_i ; valiéndonos de los comandos de la "Caja de Herramientas de Identificación de Sistemas" de MATLAB digitando los siguientes comandos:

Generamos los vectores de entrada y salida ("u" e "y", respectivamente).

```
>> u = 1:1:10
u =
     1     2     3     4     5     6     7     8     9    10

>> y = u.^2
y =
     1     4     9    16    25    36    49    64    81   100
```

Los pasamos a vectores columnas para ser tratados por el comando IDDATA de MATLAB.

```
>> u = u'

>> y = y'
```

Para aplicar el comando ARMAX de MATLAB, habrá que transformar la entrada y salida a un formato de objeto IDDATA, además agregar un Ts, 1 seg. (Tiempo de Muestreo por defecto):

```
>> datos = iddata(y,u,1)

Time domain data set with 10 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y

Inputs       Unit (if specified)
   u
```

Aplicamos el comando ARMAX:

```
>> th_armax = ARMAX(datos,[2 2 2 2]);
```

[2 2 2 2] : Representa na, nb, nc y nk respectivamente que es el orden y el retardo de dicha función.

Hacemos la presentación del modelo paramétrico.

```
>> present(th_armax)

Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + C(q)e(t)
```

$$\mathbf{A(q) = 1 - 2.417\ (+\text{-}0)\ q^{-1} + 1.417\ (+\text{-}0)\ q^{-2}} \qquad \text{ec. 4.26}$$

$$\mathbf{B(q) = 0.75\ (+\text{-}0)\ q^{-2} - 1.583\ (+\text{-}0)\ q^{-3}} \qquad \text{ec. 4.27}$$

$$\mathbf{C(q) = 1 - 0.994\ (+\text{-}14.16)\ q^{-1} - 0.004351\ (+\text{-}0.2294)\ q^{-2}} \qquad \text{ec. 4.28}$$

Para el sistema en el siguiente ejemplo, dado que el ruido se suma directamente a la salida, el tipo de estructura más apropiada para identificación debe ser del tipo “Output Error” (OE). Dentro de este tipo, el número de parámetros óptimo de los polinomios B y F son los que coinciden con la estructura real del sistema, es decir:

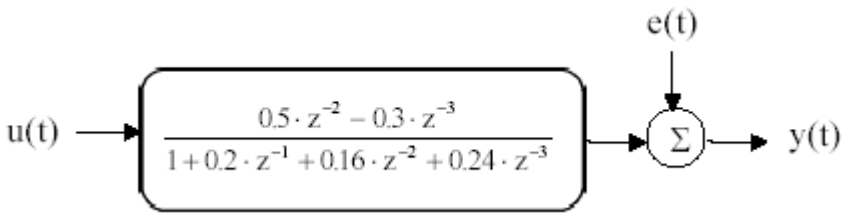


Figura 4.5. Ejemplo de estructura OE

$$y(t) = \frac{B(q^{-1})}{F(q^{-1})} \cdot u(t) + e(t) = \frac{b_1 \cdot q^{-2} + b_2 \cdot q^{-3}}{1 + f_1 \cdot z^{-1} + f_2 \cdot z^{-2} + f_3 \cdot z^{-3}} \cdot u(t) + e(t) \qquad \text{ec..4.29}$$

Y por tanto nb=2, nf=3 y nk=2.

Sin embargo, es importante tener en cuenta que en la mayoría de los casos el diseñador no dispone de esta información sobre el sistema real, por lo que será necesario ensayar con varios tipos de estructuras y número de parámetros hasta dar con el modelo que mejor se ajusta a los datos de entrada-salida registrados.

4.4.2 Métodos para el ajuste de parámetros.

Una vez elegida la estructura del modelo (tanto el tipo - ARX, ARMAX, BJ, OE...- como los órdenes de cada polinomio), es necesario determinar el valor de los parámetros del mismo que ajustan la respuesta del modelo a los datos de entrada - salida experimentales. Es importante destacar, sin embargo, que esta etapa del proceso de identificación se ve facilitada por la existencia de herramientas software que proporcionan diferentes algoritmos para el ajuste de parámetros. Una de estas herramientas es el *Toolbox de Identificación* de Matlab.

Existen varios métodos o criterios para realizar este ajuste de parámetros, entre los que cabe destacar el método de mínimos cuadrados y el de variables instrumentales.

4.4.2.1 Errores de predicción o residuos de un modelo.

Todo modelo matemático es capaz de predecir el valor de la salida del sistema en función de las entradas y salidas en instantes anteriores. Se llama error de predicción $\varepsilon(t,\theta)$ a

la diferencia entre la salida estimada por el modelo y la salida real del sistema en un determinado instante de tiempo:

$$\varepsilon(t, \theta)=y(t)-y_e(t, \theta) \qquad \text{ec. 4.30}$$

Donde $y_e(t, \theta)$ es la salida estimada por el modelo en el instante t.

4.4.2.2 Regresión lineal

Se dice que una estructura posee regresión lineal cuando la salida estimada puede expresarse como:

$$y_e(t,\theta) = \varphi^T(t)\cdot\theta \qquad \text{ec.4.31}$$

Donde $\varphi^T(t)$ es un vector columna formado por las salidas y entradas anteriores (conocido como *vector de regresión*), y θ es el vector de parámetros del modelo.
El modelo ARX es un claro ejemplo de estructura con regresión lineal, definiendo:

$$\theta = \begin{bmatrix} a_1 & a_2 & \cdots & a_{na} & b_1 & \cdots & b_{nb} \end{bmatrix}^T \qquad \text{ec. 4.32}$$

$$\varphi^T(t) = \begin{bmatrix} -y(t-1) & \cdots & -y(t-na) & u(t-nk) & \cdots & u(t-nk-nb+1) \end{bmatrix} \qquad \text{ec. 4.33}$$

4.4.2.3 Método de mínimos cuadrados (LSE)

Aplicando los criterios fijados en los dos apartados anteriores, la expresión del error de predicción es:

$$\varepsilon(t,\theta) = y(t) - \varphi^T(t)\cdot\theta \qquad \text{ec. 4.34}$$

Se define la siguiente función del error:

$$V_N(\theta) = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{2}\cdot\left[y(t) - \varphi^T(t)\cdot\theta\right]^2 \qquad \text{ec. 4.35}$$

Conocida como *criterio de mínimos cuadrados* para una regresión lineal.

Existe un valor de θ que minimiza la función anterior y que constituye la *estimación del modelo por mínimos cuadrados*:

$$\theta_{LSE} = sol\left\{\frac{1}{N}\cdot\sum_{t=1}^{N}\varphi^{T}(t)\cdot\left[y(t)-\varphi^{T}(t)\cdot\theta\right]=0\right\} \qquad \text{ec. 4.36}$$

Para este vector de parámetros, la función de error V_N toma su valor mínimo, siendo éste la *función de pérdidas* del modelo estimado.

Una variante del método anterior, conocido como *Criterio de Akaike* consiste en minimizar otra función de pérdidas distinta a la anterior, que puede obtenerse a partir de ésta del siguiente modo:

$$V_{AIC}(\theta) = V_N(\theta)\cdot(1+2\cdot d/N) \qquad \text{ec. 4.37}$$

Siendo d el número de parámetros del modelo y N el número de muestras de los datos de entrada-salida utilizados para su identificación.

4.4.2.4 Método de variables instrumentales

Mediante este método, el vector de parámetros debe cumplir la relación:

$$\theta_{IV} = sol\left\{\frac{1}{N}\cdot\sum_{t=1}^{N}\xi(t)\left[y(t)-\varphi^{T}(t)\cdot\theta\right]=0\right\} \qquad \text{ec. 4.38}$$

Donde los elementos del vector ξ(t) son las llamadas *variables instrumentales*, que resultan de aplicar algún tipo de filtro lineal al vector de regresión lineal $\varphi_T(t)$. Este método es en realidad una generalización del método de mínimos cuadrados, que proporciona mejores resultados en aquellos casos en que existe algún tipo de correlación entre el ruido y la salida del sistema.

4.5 Consideraciones prácticas sobre identificación.[2]

En este apartado se revisan algunas consideraciones prácticas a tener en cuenta durante el proceso de identificación.

4.5.1 De la obtención de los datos

El primer paso dentro del proceso de identificación es realizar algún tipo de experimento sobre el sistema bajo estudio para obtener los datos de entrada-salida que servirán de base para la obtención del modelo final.

Para que el proceso de identificación sea satisfactorio, es necesario que los datos utilizados para tal fin contengan información significativa sobre el sistema. Esto implica un cuidadoso diseño del experimento de adquisición de datos, debiéndose tomar una serie de decisiones respecto a las señales que deben ser medidas, el periodo de muestreo a utilizar, el tipo de entrada más adecuada, el número de datos a almacenar, etc.

4.5.1.1 Elección de las señales a medir

La primera decisión es qué señales se deben registrar (mediante algún tipo de sistema de adquisición y el correspondiente sistema de almacenamiento de datos), y qué señales deben ser manipuladas para excitar al sistema durante el experimento. Se debe tener en cuenta que pueden existir señales que, aunque afecten a la evolución de la salida, no pueden considerarse como entradas debido a la imposibilidad de actuar sobre ellas. En el caso de que estas señales puedan ser medidas, pueden considerarse también como entradas al sistema (midiéndose sus valores durante el experimento). En caso contrario, deben ser consideradas como perturbaciones.

4.5.1.2 Elección del tipo de entrada/s

La/s entrada/s al sistema deben ser cuidadosamente elegidas de forma que los datos recogidos proporcionen toda la información posible sobre el sistema. A este respecto, conviene tener en cuenta los siguientes aspectos:

La señal de entrada debe contener el mayor número de frecuencias posibles. Por ejemplo, una señal senoidal pura no es adecuada en un experimento de identificación, puesto que sólo se obtendrá la respuesta del sistema para la frecuencia de dicha señal. Por el contrario, las señales escalonadas (con cambios bruscos) son muy utilizadas, puesto que contienen un espectro suficientemente amplio de frecuencias.

Para sistemas lineales, basta con utilizar dos niveles de entrada, preferiblemente barriendo todo el rango de variación permitido. En este tipo de sistemas se suelen utilizar señales binarias de duración aleatoria (conocidas como señales binarias aleatorias o pseudoaleatorias), como la mostrada en la figura 4.6.a). Sin embargo, para sistemas no lineales es necesario trabajar con más de dos niveles de entrada, como se muestra en la figura 4.6.b).

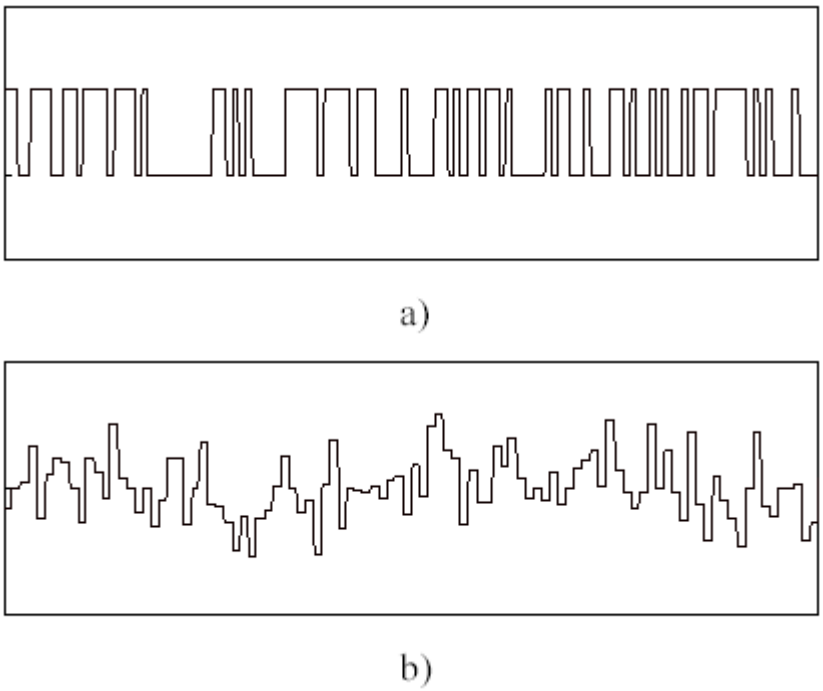


Figura 4.6. a) Entrada binaria aleatoria para sistemas lineales. b) Entrada escalonada aleatoria para sistemas no lineales.

Si se sabe que el sistema va a trabajar preferentemente en torno a un determinado punto de trabajo, es conveniente realizar el registro de datos en ese mismo entorno. Este aspecto adquiere especial importancia si el sistema no es lineal.

4.5.1.3 Elección del periodo de muestreo

La elección del periodo de muestreo está directamente relacionada con las constantes de tiempo del sistema, y tiene una influencia decisiva en el experimento de identificación. Así, un periodo de muestreo muy pequeño puede llevar a la obtención de datos redundantes, que no aportan información sobre el sistema (pero sí ocupan espacio en la memoria del dispositivo de almacenamiento de datos), mientras que un periodo de muestreo demasiado grande provoca grandes dificultades a la hora de identificar la dinámica del sistema.

Una regla comúnmente usada consiste en escoger una frecuencia de muestreo alrededor de diez veces el ancho de banda del sistema. Esto corresponde aproximadamente a muestrear en torno a cinco u ocho valores del tiempo de subida de la respuesta al escalón del sistema.

4.5.1.4 Elección del número de muestras a tomar

En principio, cuanta más información se tenga sobre el sistema, más exacto será el proceso de identificación. En la práctica, el número de muestras a recoger durante el experimento de identificación viene limitado por la capacidad del dispositivo de memoria utilizado. Por tanto, es importante llegar a un buen compromiso en la elección del periodo de muestreo y el número de muestras a tomar.

4.5.2 Del pretratamiento de los datos

Los datos registrados pueden tener deficiencias que implican efectos devastadores en el resto del proceso de identificación, como son las siguientes:

- Presencia de perturbaciones de alta frecuencia, por encima de las frecuencias de interés en la respuesta del sistema.
- Datos claramente erróneos, producidos por fallos en el hardware o software utilizados en el experimento de recogida de muestras.
- Desviaciones, desplazamientos o perturbaciones de baja frecuencia.

A continuación, se verá la forma de tratar cada una de estas deficiencias para conseguir unos datos adecuados para el proceso de identificación.

4.5.2.1 Eliminación de perturbaciones de alta frecuencia

Estas perturbaciones se producen por fuentes de ruido ajenas al sistema y pueden ser evitadas mediante una correcta elección del período de muestreo. Si, tras el experimento, se observa que el período de muestreo escogido era innecesariamente pequeño (captándose por tanto estas perturbaciones indeseadas), se puede recurrir al diezmado de los datos, para evitar repetir el experimento con un período de muestreo mayor.

4.5.2.2 Eliminación de datos erróneos

Estos datos suelen presentarse de forma aislada, pero pueden tener un efecto muy negativo en el proceso de identificación. Por tanto, es fundamental eliminarlos antes de iniciar el proceso.

Esto se realiza generalmente manualmente, eliminando dicho dato y aproximando su nuevo valor mediante interpolación. Para aplicaciones más avanzadas, existen algoritmos de detección de fallos que permiten corregir estos datos de forma casi automática.

4.5.3 De la validación del modelo

En todo proceso de identificación es conveniente probar varias estructuras y diferentes órdenes dentro de cada estructura hasta dar con el modelo que mejor se ajuste a los datos obtenidos experimentalmente de la planta real. En definitiva, se trata de determinar cuándo un determinado modelo es lo suficientemente exacto para la aplicación requerida, proceso que se conoce habitualmente como *validación del modelo*.

En general, la mayoría de los métodos de validación tratan de determinar si la respuesta del modelo se ajusta con suficiente exactitud a los datos de entrada-salida obtenidos mediante experimentación. A continuación se exponen algunos criterios típicos a la hora de descartar o elegir unos modelos respecto a otros.

a) Validación en base a la aplicación del modelo

Puesto que en la práctica es imposible determinar si un modelo responde exactamente al comportamiento de un sistema real, suele ser suficiente comprobar que el modelo es capaz de resolver el problema para el cual ha sido hallado (simulación, predicción, diseño de un

controlador, etc.). Así, por ejemplo, si el controlador que ha sido ajustado por medio del modelo da buen resultado sobre el sistema real, se puede asegurar que el modelo era 'válido' para esta aplicación.

b) Comprobación de parámetros físicos

Para una determinada estructura que haya sido parametrizada en función de magnitudes físicas, un método importante de validación consiste en comparar el valor estimado de dichos parámetros y el que sería de esperar mediante el conocimiento previo que se tiene de la planta.

c) Coherencia con el comportamiento de entrada-salida

Para determinar si el comportamiento de entrada-salida está suficientemente caracterizado, puede ser necesario recurrir a diferentes métodos de identificación y comparar los resultados obtenidos. Por ejemplo, comparando los diagramas de Bode de los modelos obtenidos mediante identificación paramétrica de diferentes estructuras, por el método de variables instrumentales y por análisis espectral, se puede determinar si la dinámica del sistema ha quedado suficientemente caracterizada.

d) Reducción del modelo

Un procedimiento para determinar si un modelo proporciona una descripción simple y apropiada de un sistema consiste en aplicarle algún método de reducción de modelos. Si una reducción en el orden del modelo no produce alteraciones apreciables en el comportamiento de entrada-salida del mismo, entonces el modelo original era innecesariamente complejo.

e) Intervalos de fiabilidad de parámetros

Otro método para determinar si el modelo bajo estudio contiene demasiados parámetros consiste en comparar los parámetros estimados con su desviación estándar. Si el intervalo de confianza de un parámetro contiene el valor cero, se debe considerar la posibilidad de eliminar dicho parámetro.

f) Simulación

Un procedimiento muy habitual que puede ser considerado como otra técnica de validación de modelos consiste en simular el modelo con un conjunto de entradas distintas a las utilizadas para identificación, y comparar la respuesta del modelo con la obtenida del sistema real.

g) Análisis de residuos

Se conocen como *residuos* de un sistema a los errores de predicción obtenidos según la expresión:

$$\varepsilon(t) = \varepsilon(t,\theta) = y(t) - y_e(t,\theta) \qquad \text{ec.4.39}$$

Siendo θ el vector de parámetros del modelo, y(t) la respuesta real del sistema e $y_e(t)$ la respuesta estimada por el modelo para la misma entrada.

Idealmente, estos residuos deben ser independientes de la entrada. Si no sucede así, significa que hay componentes en ε(t) que proceden de la entrada u(t), lo cual a su vez significa que el modelo no es capaz de describir completamente la dinámica del sistema.

Para realizar el estudio anterior, suele comprobarse la correlación entre el error de predicción y la entrada al sistema, según la expresión:

$$R_{\varepsilon u} = \frac{1}{N}\sum_{t=1}^{N}\varepsilon(t+\tau)\cdot u(t) \qquad \text{ec. 4.40}$$

El modelo será tanto más exacto cuanto más se acerquen a cero los términos de la correlación anterior. Puede demostrarse que si ε(t) y u(t) son realmente independientes, la expresión anterior (para valores grandes de N) es una distribución normal, con media cero y varianza.

$$P_r = \frac{1}{N}\sum R\varepsilon(k)\cdot R_u(k) \qquad \text{ec. 4.41}$$

Donde R_ε y R_u son las covarianzas de ε(t) y u(t) respectivamente.

Generalmente, $R_{\varepsilon u}(\tau)$ se representa en un diagrama junto con las líneas $\pm 3\sqrt{Pr}$. Si $R_{\varepsilon u}(\tau)$ sobrepasa dichas líneas para algún valor de τ, significa que $\varepsilon(t+\tau)$ y $u(t)$ probablemente no son independientes para ese valor de τ.

A la hora de examinar la función de correlación, es importante tener en cuenta los siguientes aspectos:

- Si existe correlación para valores negativos de τ, esto indica que existe realimentación de la salida hacia la entrada, no que el modelo sea deficiente.
- Si la estimación de un modelo y su expresión de correlación entre residuos y entrada $R_{\varepsilon u}$ se han determinado utilizando los mismos datos de entrada, entonces $R_{\varepsilon u}(\tau)=0$ para $\tau=nk,...,nk+nb-1$.
- Si $R_{\varepsilon u}(\tau)$ es considerablemente distinto de cero para un valor τ_0, esto indica que el término $u(t-\tau_0)$ debería ser incluido en el modelo. Éste es un buen método para ajustar el orden más apropiado de la estructura del modelo.
- Obviamente, el análisis de los residuos será un método de validación más eficaz si el conjunto de datos utilizados para realizar la correlación es distinto que el usado para la identificación del modelo.

4.6 Método de reducción de polos y ceros no dominantes.[3]

El modelo de las plantas en la práctica supera a los sistemas de segundo orden. Sin embargo, la influencia de los polos de la cadena cerrada no son todos de igual importancia. Aquellos que están más cerca del semiplano positivo son más lentos en su evolución temporal que otros orientados hacia el $-\infty$ del semiplano negativo.

A los polos más próximos al semiplano positivo se les llama dominantes y a los otros polos, insignificantes.

La regla práctica de clasificación de unos sobre otros depende de si es el polo dominante complejo conjugado o de primer orden. Si es complejo conjugado, debe de haber una distancia sobre el eje real de 5 a 10 veces el valor de la constante de amortiguamiento, entre el polo dominante y el resto de los polos. Para los polos dominantes de primer orden, el valor de la constante del polo dominante debe de ser al menos 5 ó 10 veces mayor que el de los polos insignificantes.

La reducción del orden del sistema simplifica tanto la fase de análisis como de diseño. En la práctica, se emplean las características dinámicas de los sistemas de primer o de segundo orden para definir los requisitos de diseño, aunque el sistema sea de mayor orden. Desde luego no tiene sentido hablar del coeficiente de amortiguamiento o de la frecuencia natural de un sistema si es de tercer, cuarto o de orden superior.

El comportamiento de los sistemas de orden elevado puede aproximarse por otro equivalente de segundo o primer orden. La respuesta del equivalente no es idéntica, no tiene tantos matices, pero se aproximan y se hace factible aplicar reglas sencillas tanto para la predicción de su comportamiento como para el diseño.

Hay dos formas de reducir el orden de un sistema:

1. Aplicación de la teoría de polos dominantes.
2. El efecto de un polo y un cero próximo entre sí quedan cancelados.

Una vez reducido el grado del polinomio característico se ajustará la ganancia estática para que el comportamiento en el régimen permanente sea idéntico. Esta condición requiere que sean idénticas las ganancias estáticas, tanto la del reducido como la del modelo de la planta:

$$\lim_{s \to 0} G(s) = \lim_{s \to 0} G_{eq}(s) \qquad \text{ec. 4.42}$$

Ejemplo:

Dibujar la respuesta aproximada al escalón unitario del siguiente sistema:

$$G(s) = \frac{30(s+1)}{(s+1)(s^2 + 20s + 15)} \qquad \text{ec. 4.43}$$

Al descomponerlo en polos y ceros la FDT, se observa que el efecto del cero se puede compensar con el polo de constante de tiempo 1/0.78s:

$$G(s) = \frac{30(s+1)}{(s+1)(s^2+20s+15)} = \frac{30(s+1)}{(s+0.1)(s+19.22)(s+0.78)} \qquad \text{ec. 4.44}$$

El resultado puede ser simplificado al analizar la ubicación del polo –19.22 respecto de -0.1, hay suficiente distancia como para aplicar el concepto de polo dominante. Igualando las ganancias estáticas se tendrá el equivalente reducido:

$$G_{eq}(s) = \frac{k}{s+0.1} \Rightarrow G_{eq}(0) = \frac{k}{0.1} = \frac{30}{0.1*19.22*0.78} = 20 \Rightarrow G_{eq} = \frac{2}{s+0.1} \qquad \text{ec. 4.45}$$

La respuesta ante una entrada en escalón se corresponderá a un sistema de primer orden, con ganancia 20 y con un establecimiento de 30s. Las gráficas muestran que no hay casi diferencias entre la planta y su equivalente reducida.

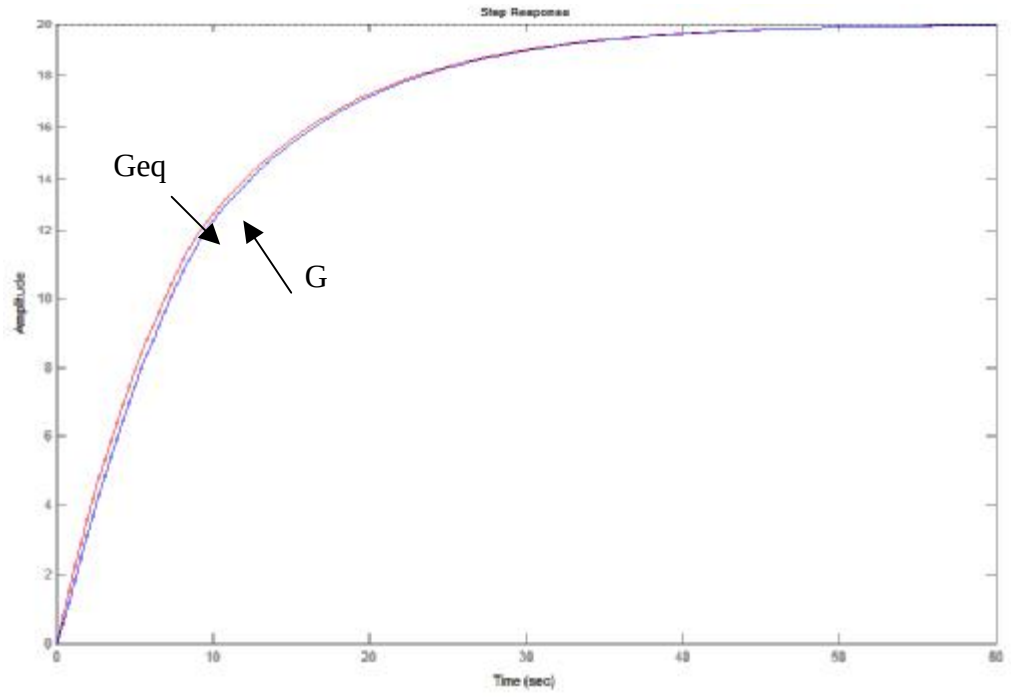


Figura 4.7. Comparación de la respuesta al escalón entre la función de transferencia con reducción (Geq) y la función original (G).

4.7 Determinación de los parámetros PID. [4]

La estructura de control a implementar es utilizada ampliamente en la industria. Se trata de la familia de controladores de estructura fija llamada **familia de controladores PID**.

Estos controladores han mostrado ser robustos y extremadamente beneficiosos en el control de muchas aplicaciones de importancia en la industria.

PID significa:

Proporcional,
Integral
Derivativo.

Históricamente, ya las primeras estructuras de control usaban las ideas del control PID. Sin embargo, no fue hasta el trabajo de Minorsky de 1922, sobre conducción de barcos, que el control PID cobró verdadera importancia teórica.

Hoy en día, a pesar de la abundancia de sofisticadas herramientas y métodos avanzados de control, el controlador PID es aún el más ampliamente utilizado en la industria moderna, controlando más del 95% de los procesos industriales en lazo cerrado.

Consideramos el lazo básico de control SISO.

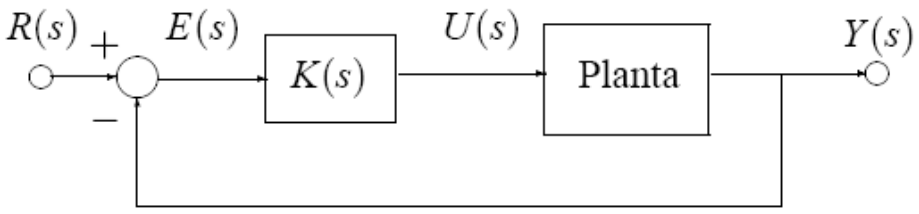


Figura 4.8. Diagrama de bloques del sistema con controlador.

Las formas estándar de controladores PID:

Proporcional $K_P(s) = K_P$ ec.4.46

Proporcional e Integral $K_{PI}(s) = K_p\left(1+\frac{1}{T_r s}\right)$ ec.4.47

Proporcional y Derivativo $K_{PD} = K_p\left(1+\frac{T_d s}{t_d s+1}\right)$ ec.4.48

Proporcional, Integral y Derivativo $K_{PID}(s) = K_p\left(1+\frac{1}{T_r s}+\frac{T_d s}{t_d s+1}\right)$ ec.4.49

A continuación se detallará las acciones de cada uno de los términos PID:

P: Acción de control proporcional, da una salida del controlador que es proporcional al error, es decir: $u(t) = K_P.e(t)$, que descrita desde su función de transferencia, queda:

$$C_p(s) = K_p \qquad \text{ec.4.50}$$

Donde Kp es una ganancia proporcional ajustable.

Un controlador proporcional puede controlar cualquier planta estable, pero posee desempeño limitado y error en régimen permanente (off-set).

I: acción de control integral: da una salida del controlador que es proporcional al error acumulado, lo que implica que es un modo de controlar lento.

$$u(t) = K_i\int_0^t e(t)dt \qquad C_i(s) = \frac{K_i}{s} \qquad \text{ec.4.51}$$

La señal de control u(t) tiene un valor diferente de cero cuando la señal de error e(t) es cero. Por lo que se concluye que dada una referencia constante, o perturbaciones, el error en régimen permanente es cero.

PI: acción de control proporcional-integral, se define mediante

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)d(t) \qquad \text{ec.4.52}$$

Donde T_i se denomina tiempo integral y es quien ajusta la acción integral. La función de transferencia resulta:

$$C_{PI}(s) = K_p\left(1 + \frac{1}{T_i s}\right) \qquad \text{ec.4.53}$$

Con un control proporcional, es necesario que exista error para tener una acción de control distinta de cero. Con acción integral, un error pequeño positivo siempre nos dará una acción de control creciente, y si fuera negativo la señal de control será decreciente. Este razonamiento sencillo muestra que el error en régimen permanente será siempre cero.

PD: acción de control proporcional-derivativa, se define mediante:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt} \qquad \text{ec.4.54}$$

Donde T_d es una constante denominada tiempo derivativo. Esta acción tiene carácter de previsión, lo que hace más rápida la acción de control, aunque tiene la desventaja importante que amplifica las señales de ruido y puede provocar saturación en el actuador. La acción de control derivativa nunca se utiliza por sí sola, debido a que sólo es eficaz durante períodos transitorios. La función de transferencia de un controlador PD resulta:

$$C_{PD}(s)=K_p+sK_pT_d \qquad \text{ec.4.55}$$

Cuando una acción de control derivativa se agrega a un controlador proporcional, permite obtener un controlador de alta sensibilidad, es decir que responde a la velocidad del cambio del error y produce una corrección significativa antes de que la magnitud del error se vuelva demasiado grande. Aunque el control derivativo no afecta en forma directa al error en estado estacionario, añade amortiguamiento al sistema y, por tanto, permite un valor más grande que la ganancia K, lo cual provoca una mejora en la precisión en estado estable.

PID: acción de control proporcional-integral-derivativa, esta acción combinada reúne las ventajas de cada una de las tres acciones de control individuales. La ecuación de un controlador con esta acción combinada se obtiene mediante:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int e(t)dt + K_p T_d \frac{de(t)}{dt} \qquad \text{ec.4.56}$$

y su función de transferencia resulta:

$$C_{PID}(s) = K_p\left(1+\frac{1}{T_i s}+T_d s\right) \qquad \text{ec.4.57}$$

4.7.1 Ajuste empírico de controladores PID.

Se expondrá el método de la "Curva de Reacción de Ziegler-Nichols", ya que es el método aplicado en el proyecto, debido a su facilidad y confiabilidad para determinar los parámetros PID con los datos obtenidos de la planta. Cabe mencionar sin embargo que existen numerosas alternativas para llevar a cabo esta tarea.

Algunos de los métodos empíricos tradicionales comenzaron a usarse por los años 1950. Algunos de estos métodos son:

- El método de oscilación de Ziegler-Nichols.
- El método de la curva de reacción de Ziegler-Nichols.
- El método de la curva de reacción de Cohen-Coon.

Es importante saber cuál es la estructura (estándar, serie o paralelo) del PID al que se aplica el ajuste propuesto por Ziegler y Nichols. Existe cierta controversia respecto a cuál fue la estructura originalmente usada por Ziegler y Nichols; las reglas dadas aquí se proponen para la estructura estándar[1].

4.7.1.1 Método de la curva de reacción de Ziegler-Nichols

Muchas plantas en la práctica pueden describirse satisfactoriamente con un modelo de la forma:

$$H(s) = m\frac{e^{-sL}}{1+sT} \qquad \text{ec.4.58}$$

Una versión linealizada cuantitativa de este modelo puede obtenerse mediante un experimento a lazo abierto con el siguiente procedimiento:

[1] Vease Ec.4.57

1. Se lleva manualmente la planta a lazo abierto a un punto de operación normal manipulando $u(t)$ (ver figura 3.1). Supóngase que la planta se estabiliza en $y(t) = y_0$ para $u(t) = u_0$.
2. En un instante inicial t_0 se aplica un cambio al escalón en la entrada, de u_0 a u_∞ (el salto debe estar entre un 10 a 20% del valor nominal.
3. Se registra la respuesta de la salida hasta que se estabilice en el nuevo punto de operación. La figura 3.2 muestra una curva típica.
4. Se calculan los parámetros del modelo de las fórmulas:

$$m = \frac{y_{max} - y_0}{U_{ref} - U_0} \qquad \text{ec.4.59}$$

$$L = t_1 - t_2 \qquad \text{ec. 4.60}$$

$$T = t_2 - t_1 \qquad \text{ec. 4.61}$$

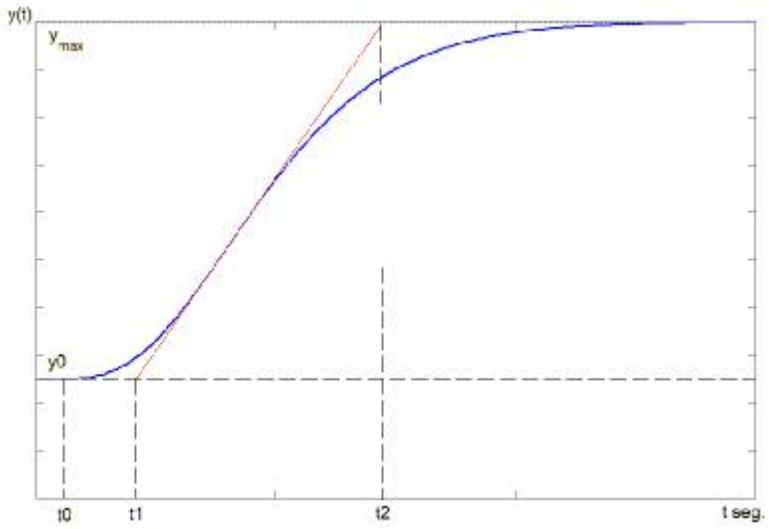


Figura 4.9: Respuesta al escalón (**curva de reacción**) en lazo abierto de la planta.

Los parámetros del controlador PID propuestos por Ziegler y Nichols a partir de la curva de reacción se determinan de Tabla 4.2.

	K_P	t_i	t_d
P	$\frac{T}{m\,L}$		
PI	$\frac{0.9T}{mL}$	$3L$	
PID	$\frac{1.2T}{mL}$	$2L$	$0.5L$

Tabla 4.2. Parámetros de controladores PID según el método de la curva de reacción de Ziegler-Nichols

4.7.1.1.1 Alternativa al método de la curva de reacción de Z-N.

El método descrito en la sección anterior presenta algunos inconvenientes, para lograr determinar los parámetros mediante programa de computadora, dicha curva tendrá que mostrar una forma de "S" perfectamente definida, sin que existan sinuosidades ni presencia de ruido ya que podría identificar un falso punto de inflexión y por lo tanto generar tiempos incorrectos.

Para superar estas limitantes, se escogió para el desarrollo de esta aplicación una variante de la expuesta anteriormente, que se describe a continuación:

4.7.1.1.2 Método de áreas. [5] y [6]

Este método pertenece al llamado "*Descripción de procesos basados en el modelo"* ya que identifica el modelo de la planta a través de un experimento simple realizado en ella. El modelo del proceso determinará un regulador que puede ser descrito con una función de transferencia estrictamente propia y asintóticamente estable, la cual puede ser identificada como un modelo de primer orden con tiempo muerto.[1]

El procedimiento se realiza según los pasos:

1. Se registran los valores de la salida de la planta y_s *(t)* ante la entrada de tipo escalón.

[1] Ec. 4.58

2. Se normaliza el vector y_s *(t)* dividiéndolo entre el nivel del escalón de referencia R_s, y de esta forma se obtiene la respuesta al escalón unitario $y_{us(t)} = y_s(t) / R$.
3. Tomando en cuenta el tiempo final del experimento tf_{in} esto es cuando y_{us} $(t_{fin}) = \mu$. y la planta ha alcanzado la estabilidad, por tanto μ es la ganancia del sistema, se determinan las tres cantidades siguientes:

$$A_0 = \int_0^{t_{fin}} \left(m - y_{us}(t)\right)dt \qquad \text{ec.4.62}$$

$$t_p = \frac{A_0}{m} \qquad \text{ec.4.63}$$

$$A_1 = \int_0^{t_p} y_{us}(t)dt \qquad \text{ec.4.64}$$

Donde A_0 y A_1 representan las áreas que se muestran en la figura 4.10.

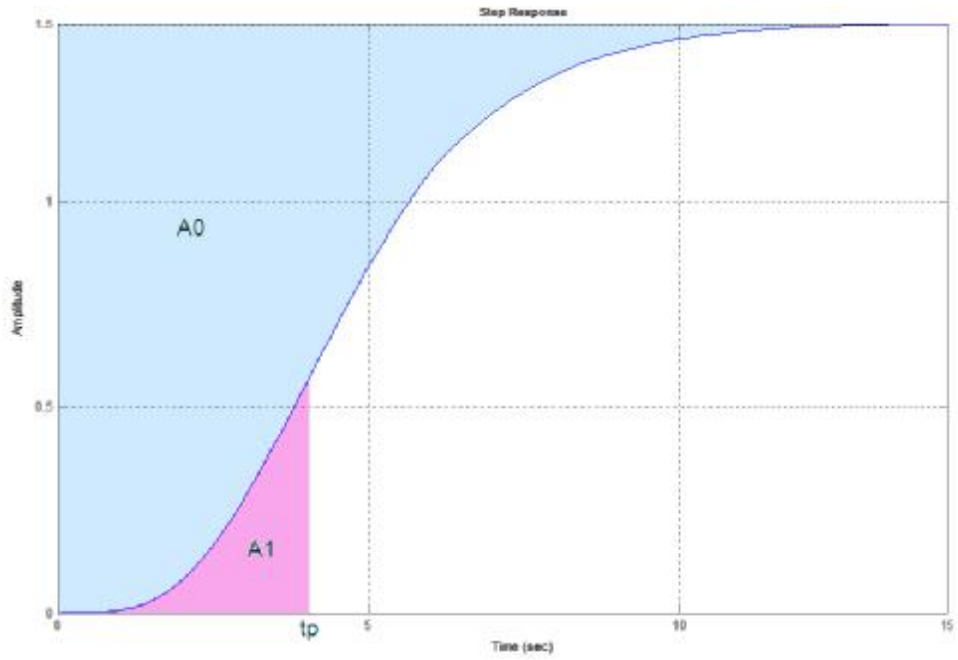


Figura 4.10. Áreas consideradas para el cálculo de los parámetros del PID.

En la aplicación a desarrollar se empleará la forma discreta para calcular dicha área:

$$A_0 = \sum_{k=1}^{k=n} \left(m - y(k)\right) * T_s \qquad \text{ec.4.65}$$

$$A_1 = \sum_{k=1}^{k=k_p} y(k) * T_s \qquad \text{ec.4.66}$$

Los parámetros restantes del modelo se obtienen como:

$$T = \frac{e \cdot A_1}{m} \qquad \text{ec.4.67}$$

$$L = \frac{A_0 - e \cdot A_1}{m} \qquad \text{ec.4.68}$$

El método de áreas es bastante eficaz y preciso, presenta una alta inmunidad al ruido por su método de integración.

Sin embargo existen casos donde el escalón no puede ser aplicado a una planta ya que su respuesta corresponderá al de una rampa, este caso se conoce como un modelo que tiene un polo en el origen del plano "S" o dicho de otra forma contiene un integrador.

Los parámetros del PID se obtienen utilizando la tabla de Ziegler-Nichols[1] de la curva de reacción.

4.7.1.1.3 Ejemplo de método de áreas.

Dada la función de transferencia:

$$H(s) = \frac{1.5}{(1+s)^5} \qquad \text{ec.4.69}$$

Por medio de simulación se ha obtenido la respuesta al escalón unitario alcanzando así los vectores de amplitud "*Y*" y el vector de tiempo "*t*" el tiempo total para lograr la estabilización es:

$$T_{fin} = 15 \text{ Seg.} \qquad \text{ec. 4.70}$$

Teniéndose un total de 1172 muestras se determina un tiempo de muestreo de:

$$Ts = Tfin / (\text{No. muestras}) \qquad \text{ec. 4.71}$$

[1] Véase Tabla 4.2.

$$T_s = 0.0128 \text{ Seg.} \qquad \text{ec. 4.72}$$

La amplitud en el estado estacionario es:

$$Y_{final} = 1.0471 \qquad \text{ec. 4.73}$$

Ya que el valor del escalón es unitario, este valor representa la ganancia, de modo que:

$$Y_{final} = \mu = 1.0471 \qquad \text{ec. 4.74}$$

Calculando el área superior por medio de ec.3.19, se logra:

$$A_0 = \sum_{k=1}^{k=1172} \left(1.0471 - y(k)\right) * 0.0128 \qquad \text{ec. 4.75}$$

$$\boldsymbol{A_0 = 4.3276}$$

$$t_p = \frac{A_0}{m} = \frac{4.3276}{1.0471} = 4.13 Seg. \qquad \text{ec. 4.76}$$

Ahora se determinará cuantas muestras originan este tiempo:

$$K_p = \frac{t_p}{T_s} = \frac{4.13}{0.0128} = 322.65 muestras \qquad \text{ec. 4.77}$$

Ya que las muestras son enteras se aproximan a 323, aplicando la regla de redondeo donde explica que si el valor seguido del punto decimal es igual o mayor a 5, se toma el entero superior, de lo contrario se toma el entero inferior.

Determinando el área inferior, con la ec.4.64, se tiene:

$$A_1 = \sum_{k=1}^{k=323} y(k) * 0.0128 \qquad \text{ec. 4.78}$$

$$\boldsymbol{A_1 = 0.5219}$$

En la figura 4.10, se muestran las áreas de la respuesta al escalón unitario de H(s).

Determinando los parámetros de un modelo de primer orden con tiempo muerto, definidas por las ecuaciones 4.67 – 68 se tiene:

$$g_0 = \frac{e \cdot A_1}{m} = \frac{2.7183 * 0.5219}{1.0471} = 1.3150 \qquad \text{ec. 4.79}$$

$$t_0 = \frac{A_0 - e \cdot A_1}{m} = \frac{4.3276 - 2.7183 * 0.5219}{1.0471} = 2.7781 \qquad \text{ec. 4.80}$$

Utilizando la tabla 4.1 de Ziegler-Nichols de la Curva de Reacción se muestran los parámetros del controlador PID.

$$K_p = \frac{1.2 g_0}{K_0 t_0} = \frac{1.2 * 1.3150}{1.0471 * 2.7781} \qquad t_i = 2t_0 = 2 * 2.7781 \qquad t_d = 0.5 t_0 = 0.5 * 2.7781$$

$K_p = 0.5590$ $\qquad$ $t_i = 5.5562$ $\qquad$ $t_d = 1.3891$

4.8 Control de la planta a lazo cerrado. [7]

4.8.1 Discretización de reguladores continuos.

En la figura 4.5 se muestra el proceso de discretización de un regulador continuo. Nótese en la figura 4.5ª, tanto planta G(s) como controlador F(s) son de tipo continuo, en la fig 4.5b se toman muestras de la salida de la planta a un periodo T, el controlador es de tipo discreto F(z) y existe el bloque B(s) que actúa como filtro ideal de forma que la señal discreta de control se convierte a una señal de tipo continua compatible con la entrada de la planta, la fig 4.5c. Es un sistema equivalente al 4.5b solo que mas simplificado, ya que las muestras que llegan al PID discreto están en función del error.

La equivalencia de estos sistemas dependen de T y de B(s), se considera una mejor aproximación si se aumenta el orden de B(s), aproximación al filtro ideal y si se disminuye el período de muestreo, T.

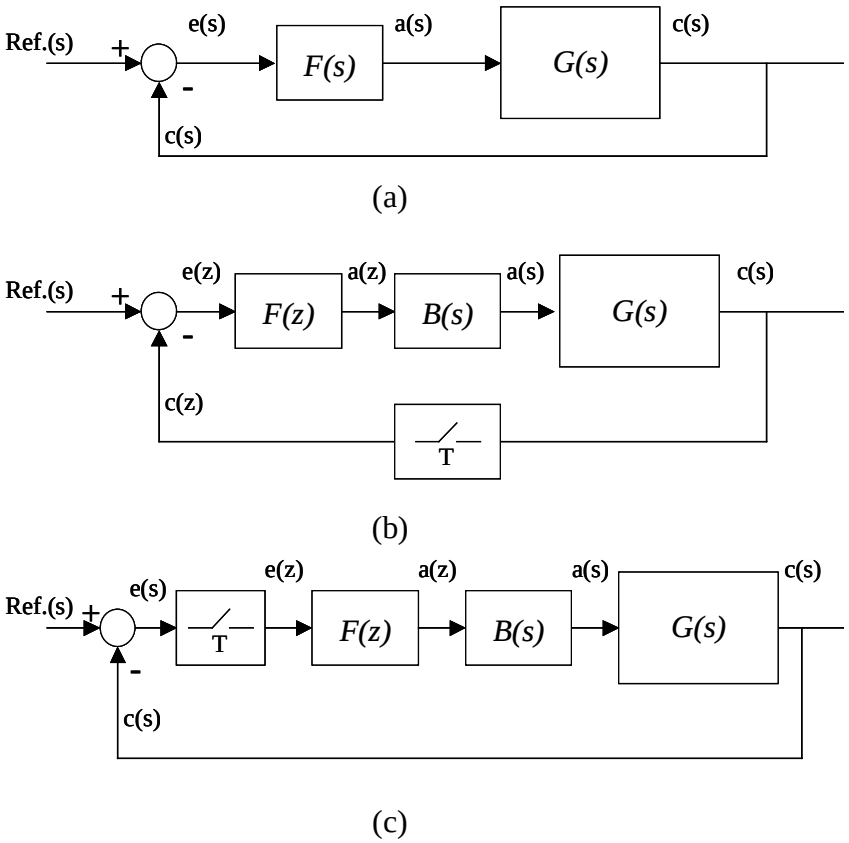


Figura 4.11. (a) muestra la función de transferencia de la Entrada, Error, Controlador, Planta y Salida en función del tiempo. (b) Muestra las funciones de "a" e incluye un controlador, ambos discretizados. (c) Sistema de un controlador discretizado.

4.8.2 Métodos de discretización.

Una forma de obtener un equivalente discreto para la expresión continua de un PID, es partir de su expresión matemática que modela el fenómeno físico como son las ecuaciones diferenciales y llevarlas a su equivalente discreto denominado ecuaciones en diferencia.

En síntesis se tiene:

- Aproximación de derivadas por restas.
- Aproximación de integrales por sumas

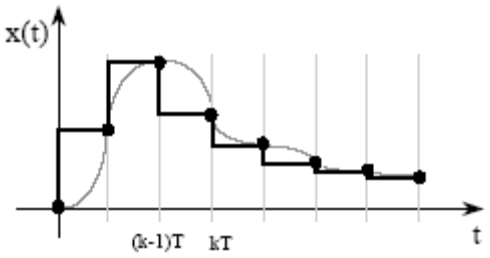


Figura 4.12. Señal muestreada.

Otra alternativa es utilizar la transformada de Laplace al modelo físico del PID, utilizando siempre la ecuaciones diferenciales y llevarlo al dominio de la frecuencia compleja.

Por ejemplo hacer la siguiente sustitución:

s – derivada

1/s – integral

Una vez obtenido el modelo en función de f(s), se puede hacer una transformación al dominio de la frecuencia discreta f(z).

Para esta aplicación se recurrirá al primer método ya que lo que nos interesa es tener una expresión en función de las muestras, es decir, en el dominio del tiempo discreto, solo basta con despejar la variable que interesa de la ecuación en diferencias.

La segunda opción es útil si se quiere utilizar el modelo en función de f(z) para propósitos de simulación, observar la estabilidad y otras parámetros de desempeño. De cualquier forma, vale aclarar que partiendo de f(z) también es posible llevar al dominio del tiempo discreto.

4.9 Regulador “PID“ discreto.

No existe límite en los Sistemas de Control Digital para programar cualquier algoritmo de control. En la figura siguiente muestra la ubicación del modulo PID dentro de

un sistema de control, teniendo como entrada la señal proveniente del error provocado por la diferencia entre la referencia y la señal proveniente del sensor (variable Controlada).

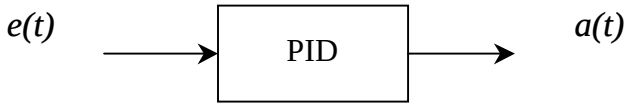


Figura 4.13. Bloque del control PID.

Este módulo genera la salida *a(t)* Variable de Control que muestra la acción de los tres controles: Proporcional, Integral y Derivativo, dicha señal es aplicada a la planta y está definida por la siguiente ecuación:

$$a(t) = K\left[\underbrace{e(t)}_{\text{PROPORC.}} + \underbrace{\frac{1}{T_i}\int_0^t e(t)\,dt}_{\text{INTEGRAL}} + \underbrace{T_d\,\frac{de(t)}{dt}}_{\text{DERIVATIVO}}\right] \qquad \text{ec. 4.81}$$

Para determinar el PID discreto, se sustituye en la ecuación 4.81 los términos diferenciales hasta llegar a una **ecuación en diferencias.**

Tomando en cuenta que el integral = sumatorio y la derivada = diferencia. La ec.4.81 queda expresada así:

$$a_k = K\left[e_k + \frac{T}{T_i}\sum_{i=0}^{k} e_k + \frac{T_d}{T}\left(e_k - e_{k-1}\right)\right] \qquad \text{ec.4.82}$$

Para una muestra anterior:

$$a_{k-1} = K\left[e_{k-1} + \frac{T}{T_i}\sum_{i=0}^{k-1} e_k + \frac{T_d}{T}\left(e_{k-1} - e_{k-2}\right)\right] \qquad \text{ec.4.83}$$

La evaluación de esta expresión se complica por el término de la sumatoria, realizando un arreglo matemático de diferencias entre el valor de la variable de control

presente y una anterior, se elimina dicha sumatoria y se procede a su desarrollo en la siguiente expresión:

$$\Delta a_k = a_k - a_{k-1} = K\left[\underbrace{\left(1+\frac{T}{T_i}+\frac{T_d}{T}\right)}_{q_0} e_k + \underbrace{\left(-1-2\frac{T_d}{T}\right)}_{q_1} e_{k-1} + \underbrace{\frac{T_d}{T}}_{q_2} e_{k-2}\right] \quad \text{ec.4.84}$$

Donde:

$$q_0 = K\left(1+\frac{T}{T_i}+\frac{T_d}{T}\right) \qquad q_1 = K\left(-1-2\frac{T_d}{T}\right) \qquad q_2 = K\left(\frac{T_d}{T}\right)$$

Reduciéndola en función de los factores "q_0", "q_1" y "q_2", resulta:

$$a_k = a_{k-1} + \left[q_0 e_k + q_1 e_{k-1} + q_2 e_{k-2}\right] \quad \text{ec.4.85}$$

5 Diseño de la Aplicación.

Los aspectos a tomarse en cuenta en la realización de la aplicación son:

- Operación de la PCI.
- Cálculo del tiempo de muestreo.
- Utilización del puerto serie.
- Protocolos de comunicación.
- Conversión de datos de entrada.
- Visualización de los datos.
- Detener manualmente la adquisición.
- Captura y presentación de datos adquiridos.
- Establecimiento del nivel del escalón.
- Conversión de datos de salida.
- Almacenamiento y carga de datos en archivo.
- Análisis de los datos.
- Cálculo de los parámetros de un controlador PID
- Control en lazo cerrado de la planta mediante un PID discreto
- Implementación de filtro en la captura de los datos.
- Diagrama de bloques.

5.1 Operación de la PCI:

Para fines de esta aplicación se puede resumir su funcionamiento[1] como sigue:

5.1.1 Adquirir y transferir datos:

La operación de la PCI se basa en programar un área de memoria llamada *Xra*m, las instrucciones que se guarden en ella serán ejecutadas una a continuación de otra cuando se le envie el comando ***RUN***.

Las instrucciones en la ***Xram*** a programar son:

- ***Inbuffer***: Leer canal y almacenarlo en el buffer del canal, el canal escogido es el canal B ya que este posee una mejor resolución (12 bits)
- ***Xramdelay***: Establece un retardo entre las mediciones del canal

Una vez programada dichas instrucciones se esta "listo" para enviar el comando ***RUN*** para que puedan ser ejecutadas.

Existen varios modos para el comando ***RUN***, de acuerdo a la singularidad de esta aplicación se escogió el modo continuo con transferencia de datos a la PC, este modo consiste en que el conjunto de instrucciones en la ***Xram*** son repetidas continuamente y los datos que se hayan almacenado en el buffer del canal son transferidos inmediatamente a la PC, para que el proceso se detenga es necesario enviarle a la PCI el comando ***STOP***.

5.1.2 Otros comandos empleados

Existe un parámetro que ajusta como se responderá en cada ciclo de ejecución de la ***Xram***, este es el comando ***SETCYCLE***, de manera que las instrucciones se puedan ejecutar mas o menos rápido y si se desean que haya repeticiones o condiciones de disparo para iniciar las mediciones. En esta aplicación este parámetro se configura de manera que las

[1] Detalles de la explicación de todos los comandos, incluyendo traducción al español, y modos de operación de la PCI en anexo A.

instrucciones se ejecuten lo mas rápido posible, que no existan repeticiones dentro del ciclo y no se establezcan condiciones de disparo para el inicio de las mediciones.

El comando *SETGAIN* es utilizado para ajustar los rangos de medición de entrada de los canales de modo que estos puedan responder a entradas de voltaje o de corriente.

Con el comando *ClrinBuffer* se borra la *Xram* como también el buffer del canal B y de esta forma garantizar que no existan datos en el buffer de mediciones anteriores(datos "basura") como también evitar que se dupliquen instrucciones en la *Xram*

El comando *Reset*, lleva las condiciones de la PCI al inicio(después del encendido), La primera vez que se carga la interfaz gráfica se procede a reiniciar la PCI, el usuario también podrá realizarlo según su criterio.

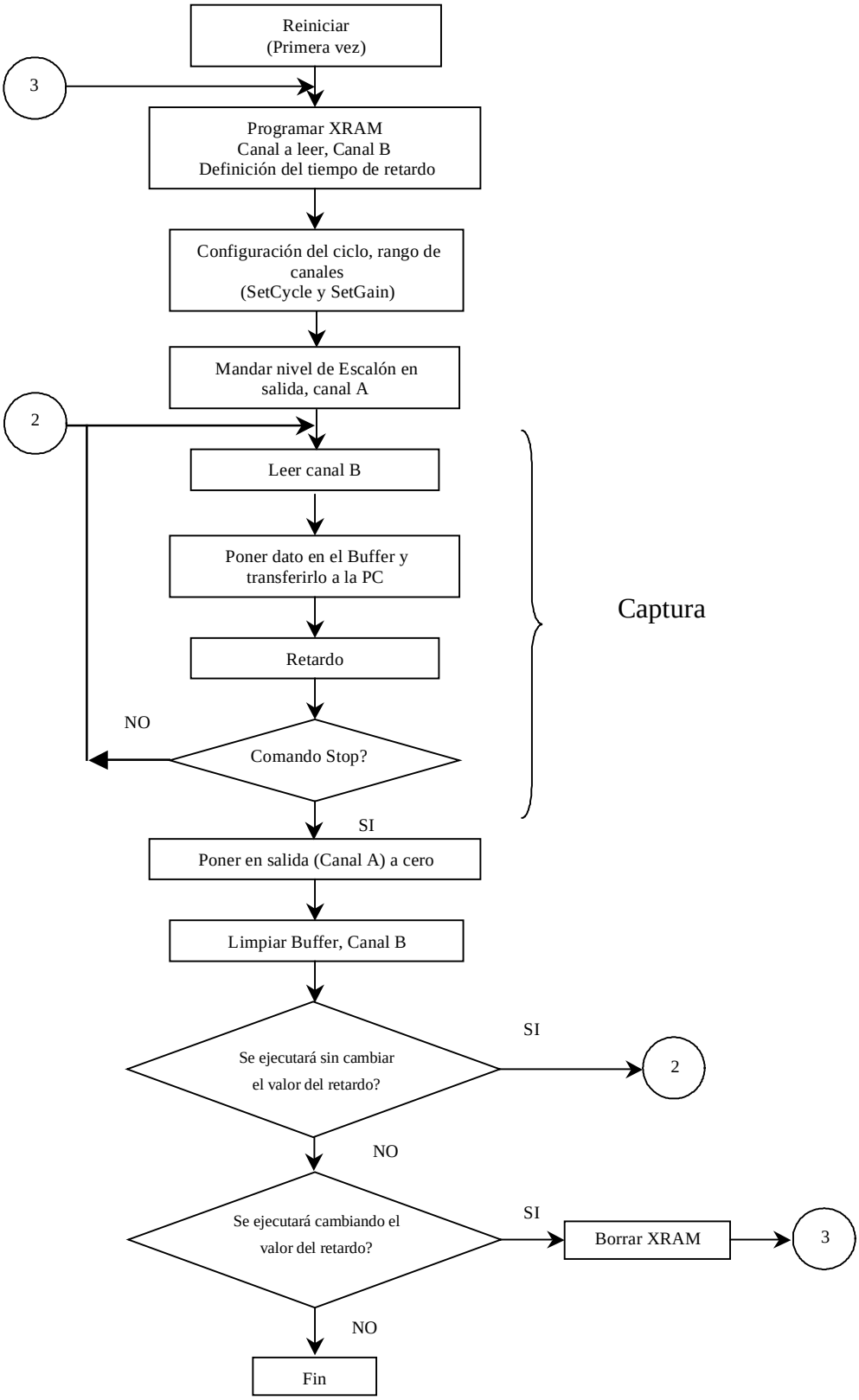
El comando *WriteChn*: Escribir en la salida del canal, se emplea para entregar el nivel de referencia del escalon hacia la entrada de la planta, el canal que se eligio es el A con una resolución de 8 bits.

5.1.3 Tiempo de muestreo y retardo en la *Xram*.

Debido a que el tiempo en que se ejecutan las instrucciones ha sido configurado para que se ejecuten en el menor tiempo posible, el valor del retardo(*Xramdelay*) será el que defina el tiempo de muestreo, con un tiempo máximo de 0.32767 s, de manera que al necesitarse tiempos de muestreos por arriba de este valor se tendrán que incluir tantas instrucciones *Xramdelay* como sean necesarias hasta cumplir con el tiempo deseado, lo que da origen a la siguiente expresión:

Tiempo de muestreo = (Tiempo por cada retardo) * (Cantidad de retardos) ec.5.1

5.1.4 Secuencia en el uso de la PCI



5.2 Cálculo del tiempo de muestreo

Se desarrolló la función ts = cdelay2(tfinal,muestras) esta calcula y programa la cantidad de retardos(*XramDelay*) que deben agregarse a la *Xram* para poder obtener el tiempo de muestro necesario. Tfinal es el tiempo total del experimento, muestras es el número de muestras que satisfacen al tiempo total del experimento, ts es el tiempo de muestreo

Hay que hacer mención que el parámetro con que se configura el comando *XramDelay* debe ser idéntico para todos los retardos programados en la *Xram.*

El tfinal se divide entre el numero de muestras y da el paso deseado Tx, de manera que:

Ts = (tiempo de retardo) * (cantidad de retardos) ~ = Tx

Nótese que el tiempo de muestreo se aproxima al paso deseado ya que estos valores no son continuos sino múltiplos de 10 us .

Lo importante es que los cálculos en el análisis se efectúan con Ts que es el tiempo exacto con que se programa los retardos en la *Xram*

5.3 Utilización del puerto serie

La comunicación entre la PCI y la PC es a través del puerto serie, la PCI demanda las siguientes configuraciones:

- Velocidad de transferencia: 9600 bps
- BIT de inicio: 1
- BIT de datos: 8
- BIT de parada: 2
- Paridad: Ninguna

Matlab permite la utilización del puerto serie[2].

La operación del puerto serie con lleva el siguiente procedimiento:

- Crear Un objeto de tipo puerto serie y asignarlo a una variable: en esta aplicación se crea una variable de tipo global llamada “serie” la cual es utilizada en varias funciones dentro de la interfaz grafica principal y en otras funciones externas.

- Configurar el objeto tipo serie con los parámetros compatibles a la PCI

- Abrir el puerto: Hasta que se a efectuado esta operación es posible que se establezca la transmisión de datos entre la PCI y la PC, es necesario conocer la condición del puerto ya que si se intenta abrir el puerto cuando este ya esta abierto se producira un error

- Cerrar el puerto: Si ya no va existir comunicación por parte del puerto hay que cerrarlo, este paso es necesario ya que otras aplicaciones que utilizan el puerto serie quedarían inhabilitadas sino se realiza este paso. Además es necesario conocer la condición del puerto ya que si se intenta cerrar el puerto cuando este ya esta cerrado se producirá un error

- Borrar la variable, una vez se cierra el puerto y ya no se va realizar la transmisión de datos, deberá eliminarse la variable de la memoria, si no se realiza y se vuelve a crear la variable esta nueva no remplaza a la anterior, la antigua sigue existiendo consumiendo recursos del sistema.

[2] Vease comandos del puerto serie anexo B

5.4 Protocolos de comunicación

5.4.1 Protocolo para los comandos.

Los comandos que operan la PCI[1] : ***XramDelay, ClrinBuffer, InBuffer, SetCycle, SetGain, Stop, Reset, Run,*** consta de dos partes: el comando y el/los parámetro/s cuando estos valores han sido ingresados correctamente la PCI devuelve un carácter de reconocimiento[1]. De manera que la programación y configuración de la PCI se simplifica a enviar por parte de la PC primero el comando y luego el/los parámetro/s y esperar el carácter de reconocimiento, para continuar con el siguiente comando.

5.4.2 Protocolo en la transmisión de los datos adquiridos

Una vez configurada y programada la PCI, esta empezara con la captura y transmisión de los datos al recibir el comando ***RUN*** por parte de la PC y no se detendrá hasta que reciba por esta misma el comando ***STOP***, de lo anterior se hace necesario llevar la cuenta de la cantidad de información ingresada para tomar decisión cuando detenerla.

Como el formato de datos entregados por el canal B es de 12 bits ocupará 2 bytes para transmitir una muestra valida, lo que significa que la cantidad de bytes totales que lleguen a la PC será el doble de la cantidad de muestras esperadas.

La velocidad con que llegan los datos a la PC depende de la velocidad de transmisión y del retardo entre mediciones del canal (tiempo de muestreo), la PC se coloca en un ciclo de espera hasta que el buffer de Matlab detecte que hay datos nuevos, luego se descarga el buffer de Matlab en una variable temporal que luego se suma al vector que va acumulando todos los datos.

5.5 Conversión de datos de entrada

Los datos que la PCI transmite tienen una codificación binaria, por lo que es necesario llevarlo a su equivalente decimal, de manera que el valor que se almacena y se visualiza

[1] Vease comandos del la PCI y caracter de reconocimiento en anexo A.

corresponda al valor de voltaje o corriente medido por el canal B. La función desarrollada que realiza esta tarea es externa a la interfaz gráfica de la forma: y = conventb(x,modo). Donde x el vector binario a ser convertido, y es el equivalente decimal y el "modo" especifica si es de voltaje(modo = 1) o de corriente (modo = 2)

5.6 Visualización de los datos

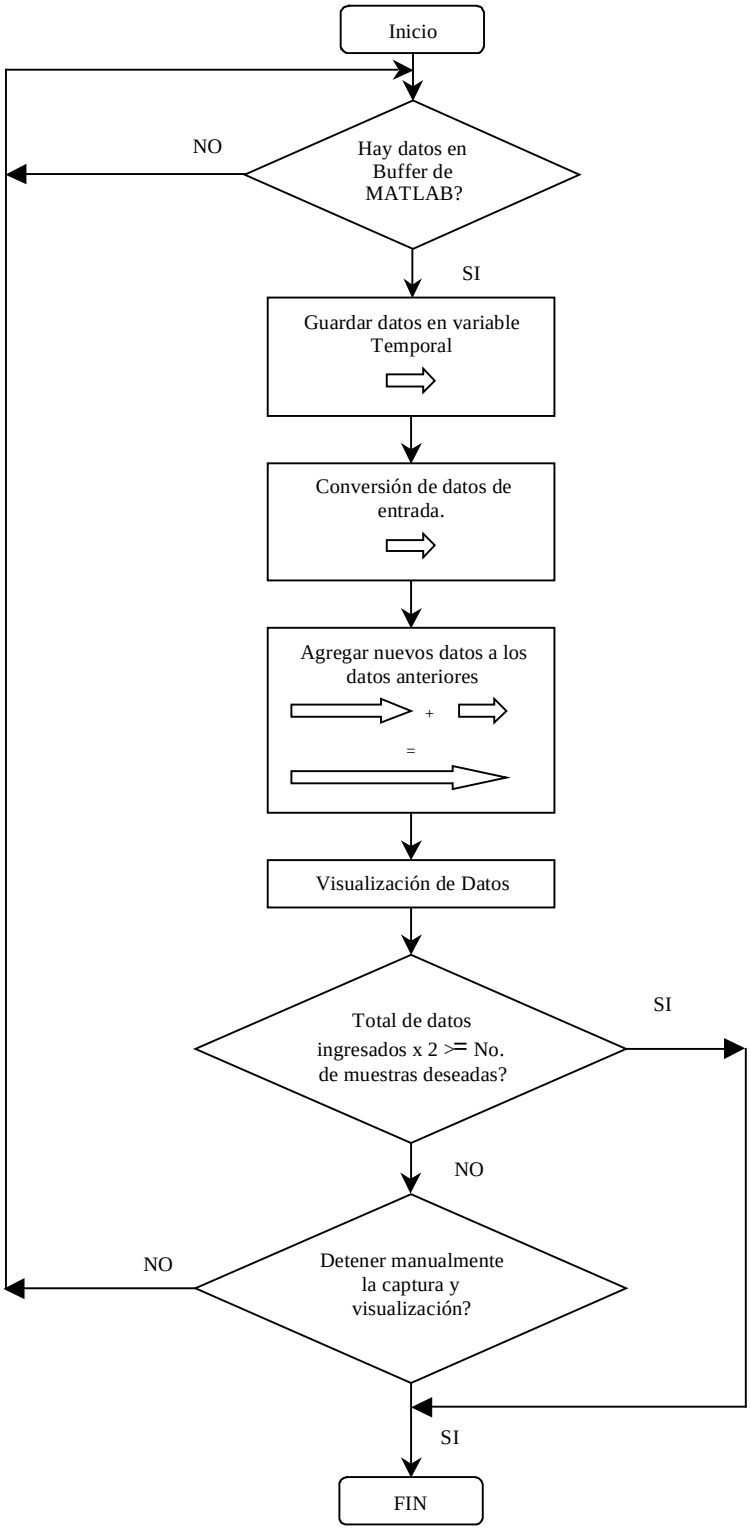
La exhibición de los datos entrantes se desea lo mas cerca posible al tiempo en que estos son adquiridos, para tiempos de experimentación relativamente largos respecto a la percepción del usuario, el retardo en transmisión y "buffereo" puede ser despreciable, considerándose así una representación en tiempo real.

Los datos que se grafican es la entrada multiplicada por un factor de escala que el usuario ingresa contra el vector del tiempo, este vector es creado en base a los pasos del tiempo de muestreo, el titulo del eje Y se muestran las unidades que el usuario ingreso y el titulo del eje X tendrá unidades de segundos.

5.7 Detener manualmente la adquisición.

Esta acción del usuario permite salirse del lazo de Captura/Presentación, la cantidad de muestras se acorta hasta donde se había adquirido, el análisis puede llevarse a cabo siempre y cuando la cantidad de muestras no sea demasiado corta y produzca error.

5.8 Captura y presentación de datos adquiridos.



5.9 Establecimiento del nivel del escalón:

El canal A provee el nivel de referencia del escalón para la entrada de la planta, dicho nivel debe establecerse al inicio de la captura de datos (después de ***RUN***) y se quita el nivel cuando la captura a finalizado.

La acción es solicitada a la PCI como un comando directo, es decir el comando escribir en el canal A y a continuación el dato que representa el nivel de la salida.

5.10 Conversión de datos de salida.

El dato que se envía a la PCI para que ponga un nivel de voltaje o de corriente determinado, debe estar en formato binario. Por lo que se ha creado la función:

y = consala(x,modo)

x: es el dato decimal que se desea su representación a la salida del canal A, y es el dato binario que deberá enviarse a la PCI, modo establece si es de voltaje o de corriente.

5.11 Almacenamiento y carga de datos en archivo.

La información importante para proceder al análisis de datos puede ser guardada y recuperada desde un archivo, estas funciones son controladas por el usuario desde la interfaz gráfica, mostrando menús de diálogo y búsqueda.

La información necesaria es: vector de datos de la salida de la planta, vector de datos de la entrada de la planta, vector de tiempo .

5.12 Análisis de los datos.

Con la información capturada (salida de la planta), el nivel del escalón(entrada de la planta) y el tiempo de muestreo. Es posible crear el modelo matemático.

Los pasos son:

- Encapsular toda la información en un formato *IDDATA*[1]
- Transformar variable *IDDATA* al modelo, en formato *IDSS o IDPOLY*[1]
- Convertir de discreto a continuo el *IDSS/IDPOLY*[1]
- Transformarlo a función de transferencia de modelo *LTI*[2]
- Reducir el modelo eliminando los polos y ceros no dominantes
- Transformar la función de transferencia a su representación en cadena de caracteres(String)
- Seleccionar el tipo de diagrama a mostrarse.

5.12.1 Encapsular la información.

En una sola variable se almacena toda la información respecto a la entrada, salida y tiempo de muestreo.

5.12.2 Transformar a formato IDSS o IDPOLY:

La información encapsulada le sirve de entrada a esta funcion donde es evaluada el modelo de identificación de sistemas, la configuración efectuada es para que el modelo se cree con el menor orden posible, el termino IDSS se refiere a que el procedimiento sera evaluado como espacio de estado y *IDPOLY* como polinomio, los algoritmos utilizados son el del cálculo de error por minimo cuadrado y la estructura es basada en ARMAX con modificaciones al cálculo de prediccion de error y gradiente. Había que decidir el método a emplear debido a que probar todos los métodos; ARX, Output Error y Box Jenkins, sería ineficiente. Se eligió la estructura ARMAX, ya que esta es la mas completa, porque aún podría agregar cada variable actuante en el sistema (e(t), u(t), y(t)), una función de transferencia. Ver tabla 4.1 y figura 4.4.

[1] Vease comandos en caja de identificación de sistemas Anexo B.

[2] Vease comandos en caja de control Anexo B.

5.12.3 Convertir de discreto a continuo el IDSS/IDPOLY.

El modelo como espacio de estado, se encuentra en un formato discreto, sin embargo el resultado final esperado debe trasladarse a continuo.

5.12.4 Transformarlo a función de transferencia de modelo LTI.

Ya que de aquí en adelante se utilizaran las cajas de herramiental de control, se traslada el modelo a una funcion de transferencia en función de "s"

5.12.5 Reducir el modelo eliminando los polos y ceros no dominantes(Opción).

El modelo puede presentar algunos polos y ceros que pueden ser eliminados sin que este resultado distorsione significativamente la respuesta real del sistema, de hecho los algoritmos de identificación de sistema pueden agregar estos ceros y polos proveniente del ruido o de discontinuidad en los datos, por lo que al reducir la expresión se mejora el modelo acercándose mas al valor esperado[1].

El método de reducción desarrollado en la aplicación es "La Aplicación de la Teoría de Polos Dominantes".

Para la reducción del sistema se desarrolló la función: y = reducir(hs) Donde el argumento de entrada es la funcion de transferencia que se desea reducir y el de salida es la funcion de referencia reducida. El criterio para llevar a cabo la reducción es: Eliminar aquellos polos y ceros que se encuentren a una distancia cinco veces mayor al polo mas significante, al modelo reducido se le calcula una ganancia equivalente a la del modelo sin reducir.

5.12.6 Transformar la función de transferencia a su representación decadenas de caracteres (String).

La interfaz grafica no permite la representación de variables con estructura como son la funcion de transferencia, para poder exhibir la funcion en una caja de texto es necesario

[1] Se ilustra en: evaluación del modelo matemático

trasladar la funcion de transferencia a un equivalente de cadena de caracteres. Para lo cual se desarrolló la función :

y = tf2string(num,den)

Los argumentos de entrada son los coeficientes del numerador y denominador de la función de transferencia, y el argumento de salida es una cadena de caracteres representando a dicha función de transferencia.

5.12.7 Seleccionar el tipo de diagrama a mostrarse

Con la función de transferencia encontrada se puede hacer uso de cualquier método de análisis gráfico como son:

- Diagrama de lugar de las raíces.
- Escalón Unitario.
- Diagrama de Bode.
- Diagrama de Nyquist.

El usuario elige cual/es diagramas se presentaran en ventanas individuales.

5.13 Cálculo de los parámetros de un controlador PID.

Para llevarlo a cabo se desarrolló la función:

[kp,ti,td] = param_pid(h)

la cual calcula los parámetros para un controlador tipo PID, los argumentos de salida son:
kp = ganancia proporcional, ti = tiempo integral, td = tiempo derivativo.

El argumento de entrada ‘h’ es la función de transferencia de la planta.

El método utilizado se basa en una modificación al método de la curva de reacción de Ziegler-Nichols a lazo abierto, donde los parámetros se determinan a través de un modelo equivalente de primer orden con tiempo muerto.

Tomando en cuenta lo expuesto en el marco teórico, la secuencia lógica para la realización de dicha función es la siguiente:

- Teniendo la función de transferencia de la planta(*h*) se le añade un retardo con el fin que las constantes temporales no sean demasiado pequeñas para procesos rápidos y de esta forma se garantiza la buena operación de un controlador de tipo discreto, evitando tiempos de muestreo críticos.
- Se obtiene mediante una simulación rápida la respuesta ante una entrada escalón unitario, se registra el tiempo final de estabilización, esto se puede obtener por ejemplo con 100 muestras.
- Debido a que el cálculo de áreas se efectúa integrando de forma discreta:

$$A = \sum_{k=1}^{k=n} y(k) * T_s \qquad \text{ec. 5.2}$$

Se obtendrá una mejor exactitud, si se disponen de mas elementos del vector "*y*", y los espaciamientos son mas cortos, es decir, un tiempo de muestreo mas pequeño y mas muestras, por lo que la simulación se repite hasta que se llegue al mismo tiempo final de estabilización pero con un tiempo de muestreo 10 veces menor al anterior. Por consiguiente, se obtendrían 1000 muestras las cuales representan una mejora significativa en el cálculo.

- Con la finalidad que la ganancia proporcional no resulte muy alta, se recorta el vector de amplitud "*y*" en sus últimos datos hasta lograr que la amplitud se reduzca un 70% del valor de estabilización.

- Se determina la ganancia μ dividiendo la amplitud de la salida en estadi estacionaria entre el nivel del escalón.

- Se calcula A_0 que es el área sobre la curva de la respuesta al escalón mediante la ecuación 4.62.
- Se encuentra t_p mediante la ecuación 4.63.

- Se obtiene el valor de K_p que corresponde al indice de la muestra que coincide con tp empleando:

$$K_p = \frac{t_p}{T_s} \qquad \text{ec.5.3}$$

- Se efectúa redondeo a cifra entera aplicando la regla de redondeo donde explica que si el valor seguido del punto decimal es igual o mayor a 5, se toma el entero superior, de lo contrario se toma el entero inferior.

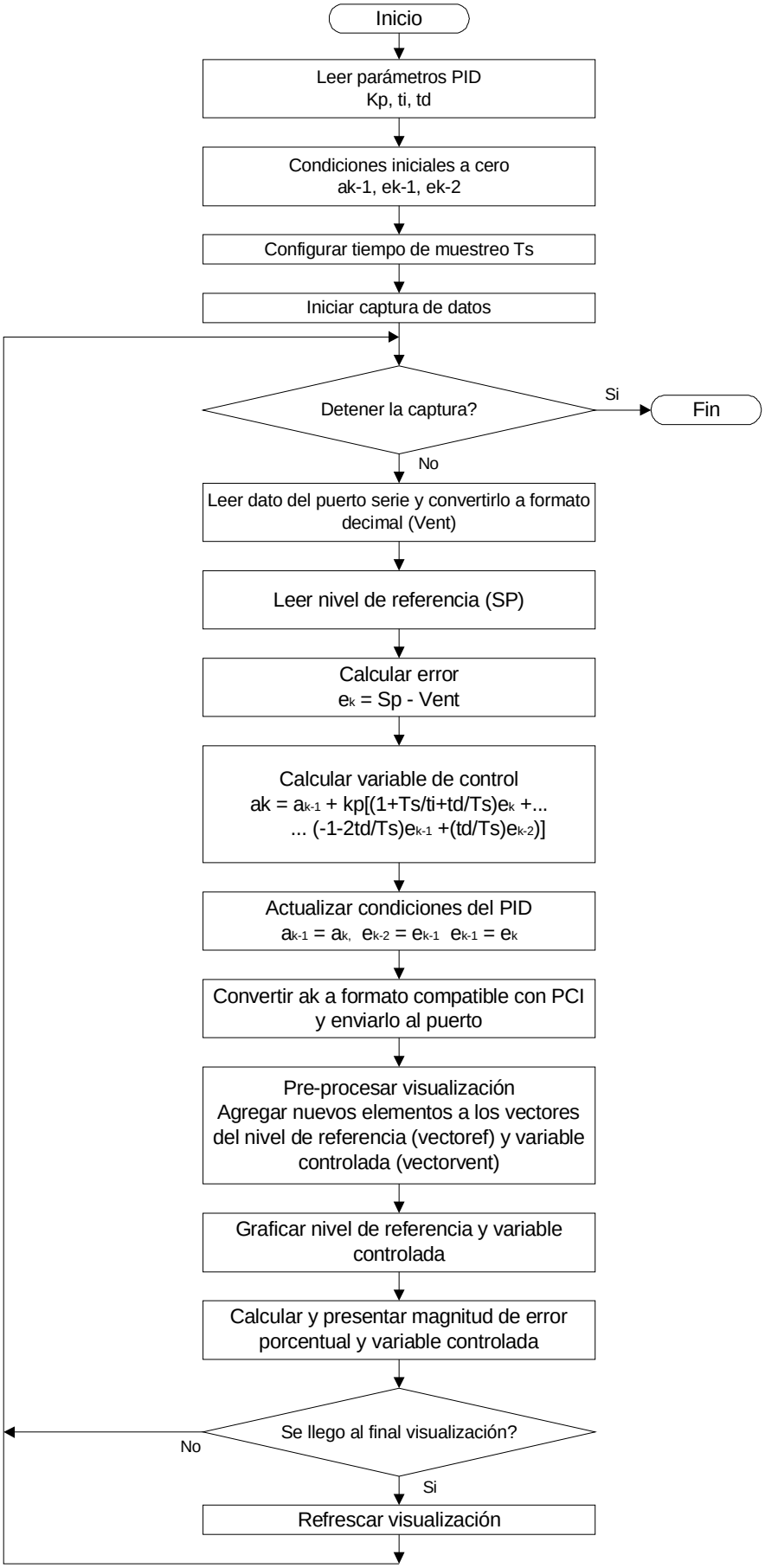
- Se calcula A_1 por medio de la ec.4.64.

- Se obtienen los valores de T y L utilizando 4.67 y 4.68.

- Los valores de K_p, t_i , t_d se determina sustituyendo los valores de T y L en la tabla 4.2 de Ziegler Nichols

5.14 Control en lazo cerrado de la planta mediante un PID discreto.

A continuación se presenta el flujograma que resume la operación de control:


Inicio
Leer parámetros PID
Kp, ti, td
Condiciones iniciales a cero
ak-1, ek-1, ek-2
Configurar tiempo de muestreo Ts
Iniciar captura de datos
Detener la captura?
Si
Fin
No
Leer dato del puerto serie y convertirlo a formato decimal (Vent)
Leer nivel de referencia (SP)
Calcular error
ek = Sp - Vent
Calcular variable de control
ak = ak-1 + kp[(1+Ts/ti+td/Ts)ek +...
... (-1-2td/Ts)ek-1 +(td/Ts)ek-2)]
Actualizar condiciones del PID
ak-1 = ak, ek-2 = ek-1 ek-1 = ek
Convertir ak a formato compatible con PCI y enviarlo al puerto
Pre-procesar visualización
Agregar nuevos elementos a los vectores del nivel de referencia (vectoref) y variable controlada (vectorvent)
Graficar nivel de referencia y variable controlada
Calcular y presentar magnitud de error porcentual y variable controlada
Se llego al final visualización?
No
Si
Refrescar visualización

5.14.1 Leer parámetros PID.

Los parámetros que aparecen en caja de texto que fueron determinados a partir del modelo o ingresados por el usuario representan:

- kp = ganancia proporcional
- ti = tiempo integral
- td = tiempo derivativo

5.14.2 Condiciones iniciales a cero.

El PID discreto depende no solamente de los parámetros como son kp, ti, td y el tiempo de muestreo Ts, sino que también de los valores previos o muestras anteriores, para el procesamiento de la variable de control, por lo que al inicio de la captura se asume que antes de la primer muestra las condiciones sean cero. De forma que:

a_{k-1} = variable de control anterior a la actual = 0

e_{k-1} = error anterior al actual = 0

e_{k-2} = error anterior a e_{k-1} (hace dos muestras) = 0

5.14.3 Configurar el tiempo de muestreo T_s.

A diferencia del control de lazo abierto para la determinación del modelo, en el caso del control PID a lazo cerrado, el tiempo de muestreo será el mismo para cada planta a controlar, salvo que el usuario cambie esta configuración en la ventana “Avanzado”.

El valor de tiempo de muestreo de configuración original es de 0.5 segundos, este tiempo relativamente grande se ha escogido para que en máquinas lentas no exista conflicto por el tiempo de procesamiento de código antes que se actualice una nueva muestra, por las condiciones mismas del PID, muestra que ingresa es procesada, convertida en variable de control y llevada a la entrada de la planta, por lo que no se puede permitir la acumulación de datos en el buffer de entrada de matlab; de manera que se recomienda manipular con cuidado este parámetro.[1]

[1] Véase recomendaciones en Manual de Usuario, ventana “Avanzado” del menú Herramientas, anexo C.

5.14.4 Iniciar captura de datos.

Al igual que en la captura a lazo abierto, la orden que se le da a la PCI es ejecutar la lectura y transmisión de los datos del transductor de la planta de forma continua, hasta que se le envié la orden de detención.

5.14.5 Detener la captura.

A diferencia de captura a lazo abierto donde el proceso se interrumpía, ya sea porque había llegado a la cantidad de datos esperados o porque el usuario dio la orden de detenerse, en el control PID a lazo cerrado solamente se detiene el proceso si el usuario lo solicita.

5.14.6 Leer dato del puerto serie y convertirlo a formato decimal.

Los datos enviados por la PCI llegan al buffer de matlab, si existe mas de un dato en el buffer se toma únicamente el último dato por ser el mas actual, este dato posee un formato binario compatible con la PCI, por lo tanto para ser procesado es necesario convertirlo previamente a su equivalente decimal, como antes fue expuesto en la captura a lazo abierto.

5.14.7 Leer nivel de referencia (Sp).

Es el nivel que el usuario desea que la planta alcance con la ayuda del control PID. Si al ser leído este dato se esta utilizando un factor de escala distinto a 1, el dato es convertido de tal forma que su rango decimal sea de -10 a 10 ya que estos son los rangos compatibles con la PCI.

5.14.8 Calcular error.

Una de las ventajas de implementar un control discreto, es que se puede ahorrar la etapa de un sumador analógico de un control continuo ya que mediante programa se pude efectuar el cálculo del error, es así que en esta aplicación la entrada a la PCI es directamente la variable controlada y con el nivel de referencia se calcula el error mediante:

$$e_k = Sp - Vent \qquad \text{ec. 5.2}$$

5.14.9 Calcular variable de control.

La variable de control es la salida del control PID y que sirve de entrada al actuador de la planta, para lograr alcanzar el nivel de referencia deseada.

La expresión:

$$a_k = a_{k-1} + K\left[\left(1 + \frac{T_s}{t_i} + \frac{t_d}{T_s}\right)e_k + \left(-1 - 2\frac{t_d}{T_s}\right)e_{k-1} + \frac{t_d}{T_s}e_{k-2}\right] \quad \text{ec. 5.3}$$

Desarrollada en el marco teórico es la utilizada para efectuar dicha tarea.

5.14.10 Actualizar condiciones del PID.

Como se mencionó anteriormente el cálculo del PID requiere de las condiciones previas a las del instante por determinar, por lo que es necesario tener preparado para la próxima muestra, que la muestra actual sea la muestra previa en el futuro y similar en todas las condiciones. La asignación es la siguiente:

$$a_{k-1} = a_k \,, \quad e_{k-2} = e_{k-1,} \quad e_{k-1} = e_k \qquad \text{ec.5.4}$$

5.14.11 Convertir variable de control a formato compatible con PCI y enviarlo al puerto.

La variable de control determinada debe ser convertida a formato binario compatible con la PCI, luego de esto enviarlo a la PCI por medio del puerto serie.

5.14.12 Pre-procesar visualización de vectores del nivel de referencia y variable controlada.

Con un proceso similar al realizado en la captura a lazo abierto, en lazo cerrado es necesario crear vectores que van creciendo a medida que los datos actuales van siendo agregados con el fin de enviarse a visualizar.

5.14.13 Graficar nivel de referencia y variable controlada.

Estas dos señales se grafican en una pantalla de visualización, en base al vector de datos y al factor de escala ingresado, se ajustan adecuadamente los ejes del tiempo(x) y las magnitudes(y).

5.14.14 Calcular y presentar magnitud de error porcentual y variable controlada.

Los valores instantáneos de estas magnitudes son presentados con el fin de tener una mejor cuantificación de lo que ocurre en el instante actual. El error porcentual es determinado mediante:

$$\text{Error}(\%) = \text{Error} * 100 / (\text{nivel de referencia}) \qquad \text{ec.5.5}$$

5.14.15 Refrescar visualización.

El área de gráfica se actualiza cada vez que las señales llegan al final de la misma, de forma que visualizar el control de una planta tiene un tiempo indefinido, y solo depende que el usuario sea quien detenga el proceso.

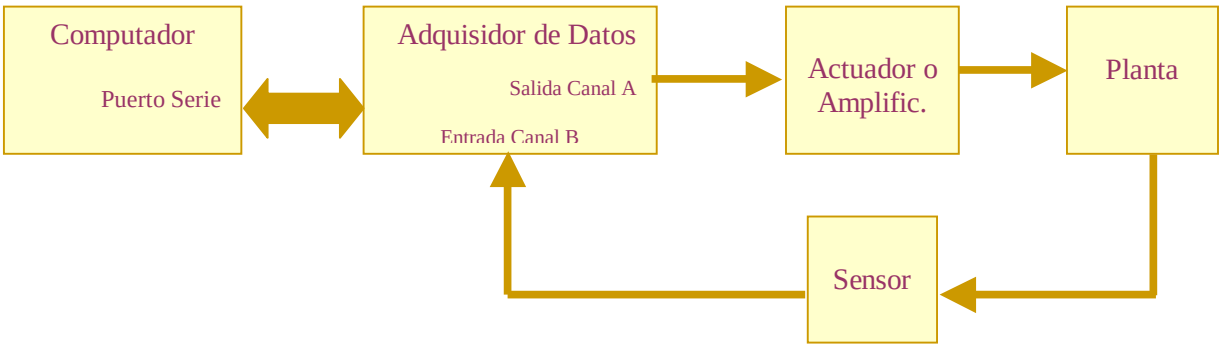
5.15 Implementación de filtro en la captura de los datos.

Se ha implementado un filtro digital de tipo estadístico, con la finalidad de eliminar las componentes de ruido y suavizar los cambios abruptos de la señal de salida de la planta y de esta forma obtener un modelo mas representativo de esta.

A diferencia de los filtros digitales de tipo: respuesta al impulso finito(FIR) o respuesta al impulso infinito(IIR), un filtro estadístico no está diseñado en base a la frecuencia de corte, ganancia, banda de paso, orden del filtro, etc. Su forma de operación es de alinear los datos que se encuentran dispersos de la tendencia de la curva, por lo que no se requieren los parámetros antes mencionados, esto evita la estimación de estos para cada caso a solucionar y el diseño de los coeficientes del filtro.

Para lograr un mayor filtrado, los datos procesados se reingresan al filtro y se repite tantas veces como sea necesario, esto equivale a elevar el orden del filtro; es necesario tener cuidado y no sobrepasar el procedimiento ya que la tendencia de la curva podría cambiar y no representar al modelo correctamente.[1]

5.16 Diagrama de bloques.



[1] Véase recomendaciones en “Filtrar” del Manual de Usuario, Anexo C.

6 Evaluaciones.

6.1 Evaluación del modelado matemático.

6.1.1 Comparación entre modelos.

Con el objetivo de comprobar que tanto el modelo adquirido utilizando los métodos de identificación de sistema se acerca a un modelo matematico deseado, se diseñó una planta electrónica[1], a la que se le efectuaron varias pruebas de las cuales se mostrarán aquellos casos extremos donde el modelo obtenido difiere y se acerca en mayor medida al modelo esperado.
La ecuación teórica deseada es:

$$\frac{V_{out}}{V_{in}} = Gteo = \frac{3.95}{s^2 + .987s + 3.95} \qquad \text{ec.6.1.} \qquad \text{Ecuación Calculada.}$$

6.1.1.1 Experimento 1.

La función de transferencia que se obtiene del adquisidor aplicando un nivel del escalón de 1 Volt. y un tiempo de observación de 10 Seg. es:

$$Gadqui = \frac{-0.8024\ s + 3.915}{s^2 + 1.063\ s + 3.789} \qquad \text{ec. 6.2}$$

Aplicando el escalón unitario a estas dos funciones y comparando tenemos:

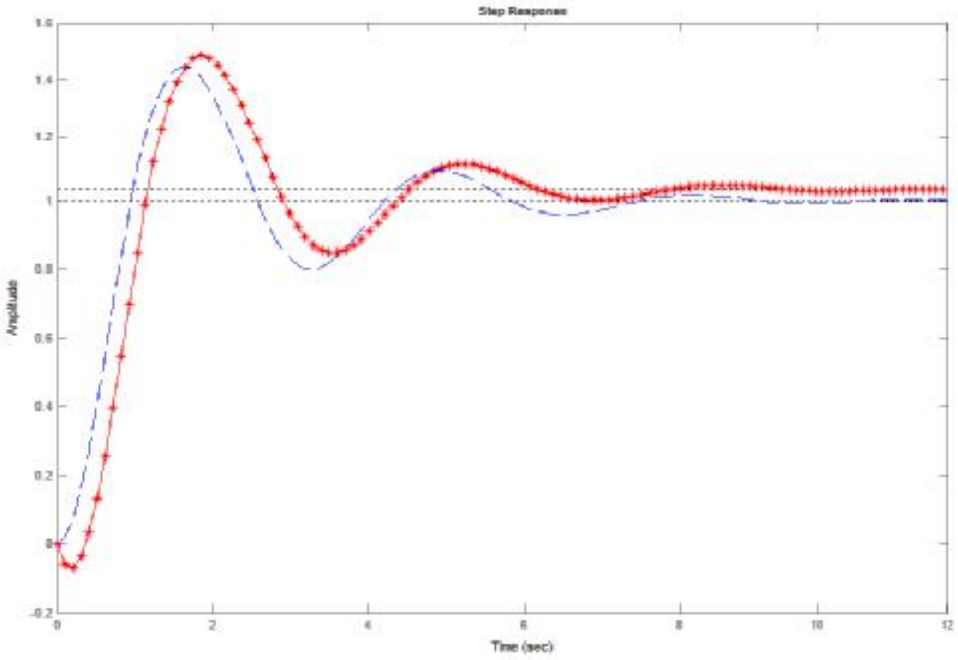


Figura 6.1. Respuesta al escalón de ec.5.1 vrs. Ec. 5.2.

[1] Vease Anexo D.

Donde: -- Modelo Esperado. -* Modelo Adquirido.

Y aplicando Bode a estas dos funciones tenemos:

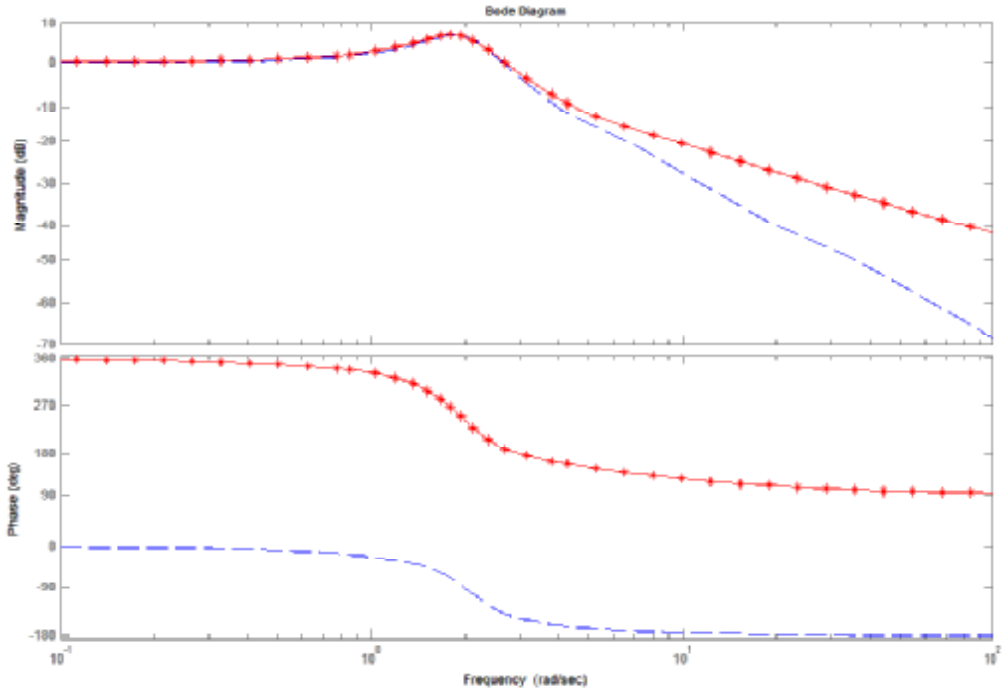


Figura 6.2. Bode de ec.6.1 vrs. Ec. 6.2.

6.1.1.2 Experimento 2.

Si a la función de transferencia dada por el adquisidor se eliminan los polos y ceros "No Dominantes" tenemos la ecuación:

$$Geq = \frac{3.915}{s^2 + 1.063\,s + 3.789} \qquad \text{ec. 6.3}$$

Aplicando el escalón unitario a las funciones Gteo y Geq y comparando tenemos:

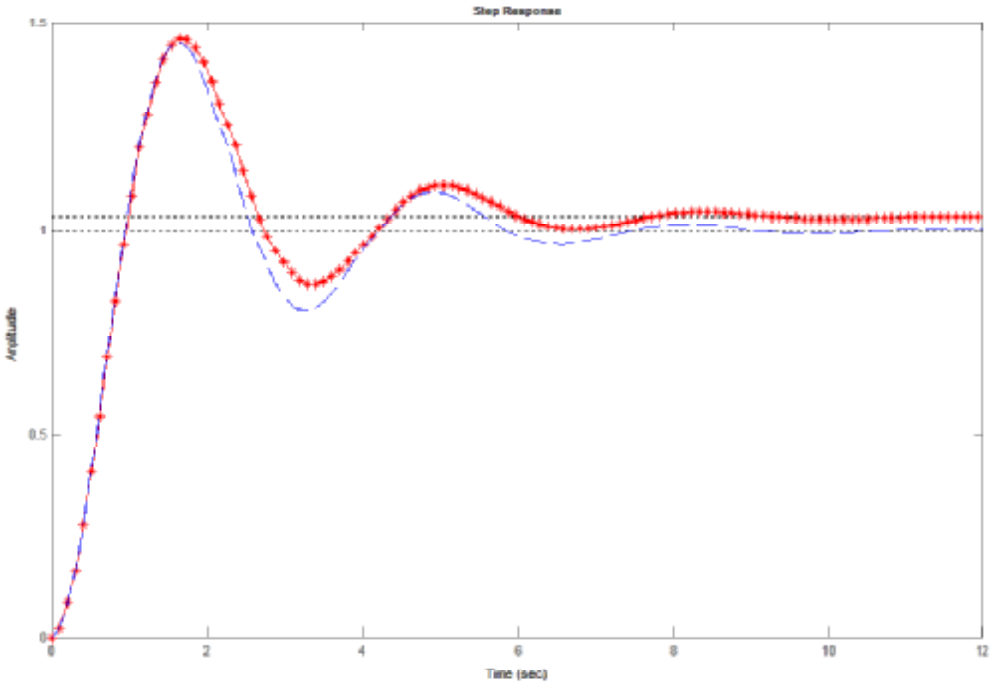


Figura 6.3. Respuesta al escalón de ec. 6.1 vrs. Ec. 6.3.

También podemos comparar con Bode las funciones Gteo y Geq para ver el comportamiento en función de la frecuencia.

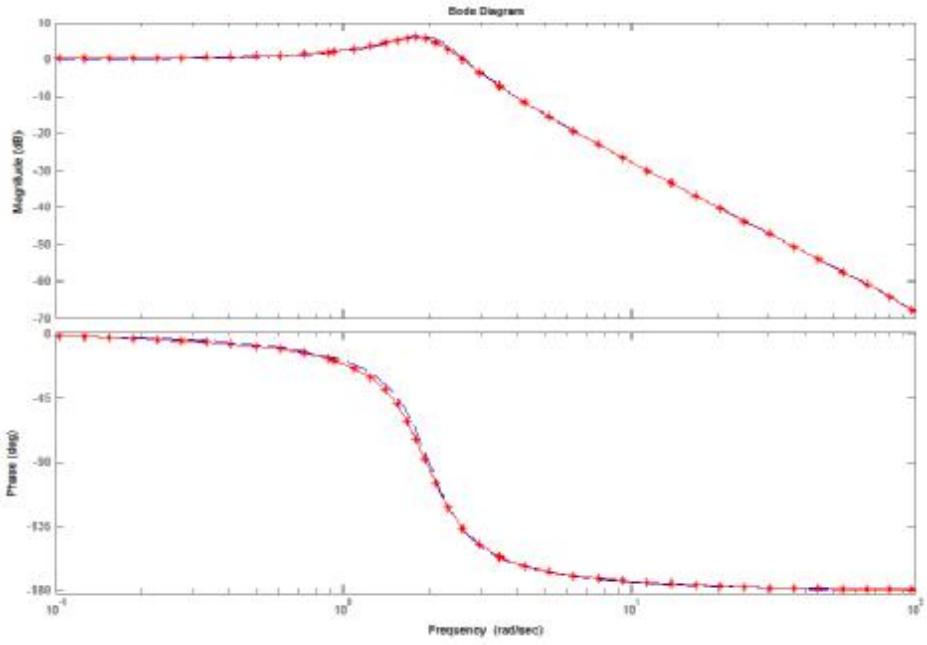


Figura 6.4. Bode de ec.6.1 vrs. Ec. 6.3.

6.1.1.3 Experimento 3.

Ahora una función de transferencia que se obtiene del adquisidor aplicando un nivel del escalón de 7 Volt. y un tiempo de observación de 10 Seg. es:

$$Gadqui = \frac{-0.1451\ s + 3.854}{s^2 + 1.1\ s + 3.884} \qquad \text{ec. 6.4}$$

Aplicando el escalón unitario a estas dos funciones y comparando tenemos:

-- Modelo Esperado.

-* Modelo Adquirido.

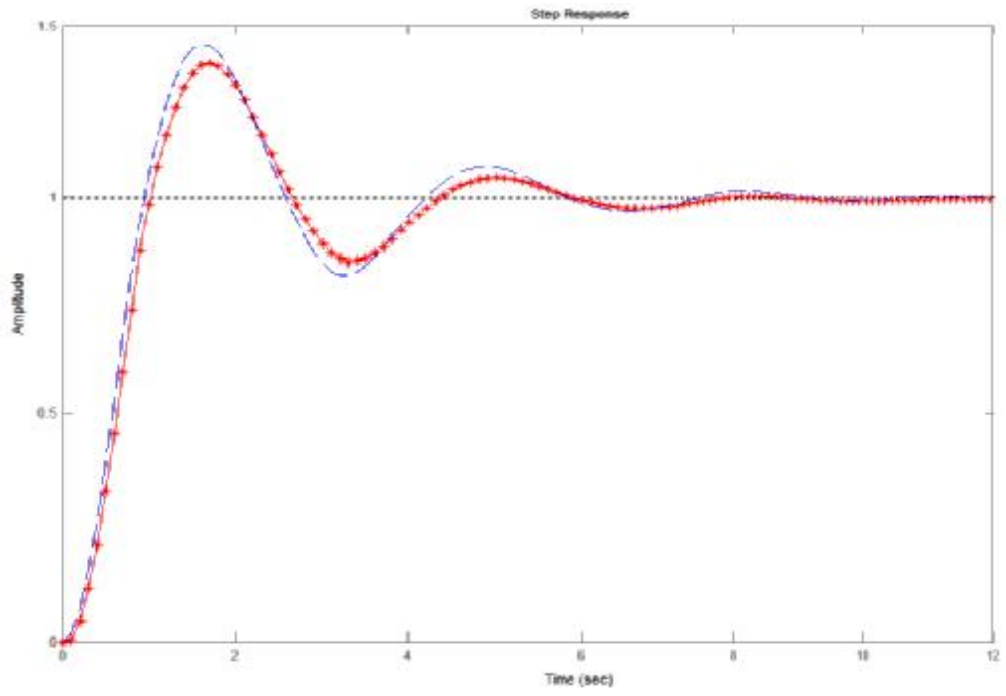


Figura 6.5. Respuesta al escalón de ec.6.1 vrs. Ec. 6.4.

Y aplicando Bode a estas dos funciones tenemos:

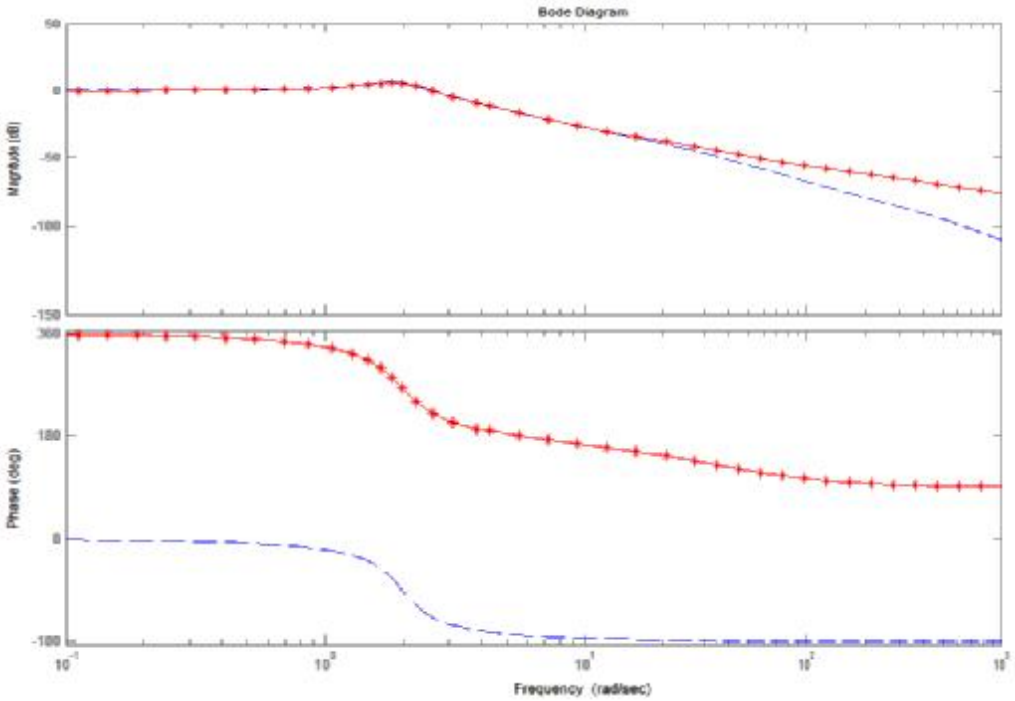


Figura 6.6. Bode de ec.6.1 vrs. Ec. 6.4.

6.1.1.4 Experimento 4.

Si a la función de transferencia dada por el adquisidor se eliminan los polos y ceros "No Dominantes" tenemos la ecuación:

$$Geq = \frac{3.854}{s^2 + 1.1\,s + 3.884} \qquad \text{ec. 6.5}$$

Aplicando el escalón unitario a las funciones Gteo y Geq y comparando tenemos:

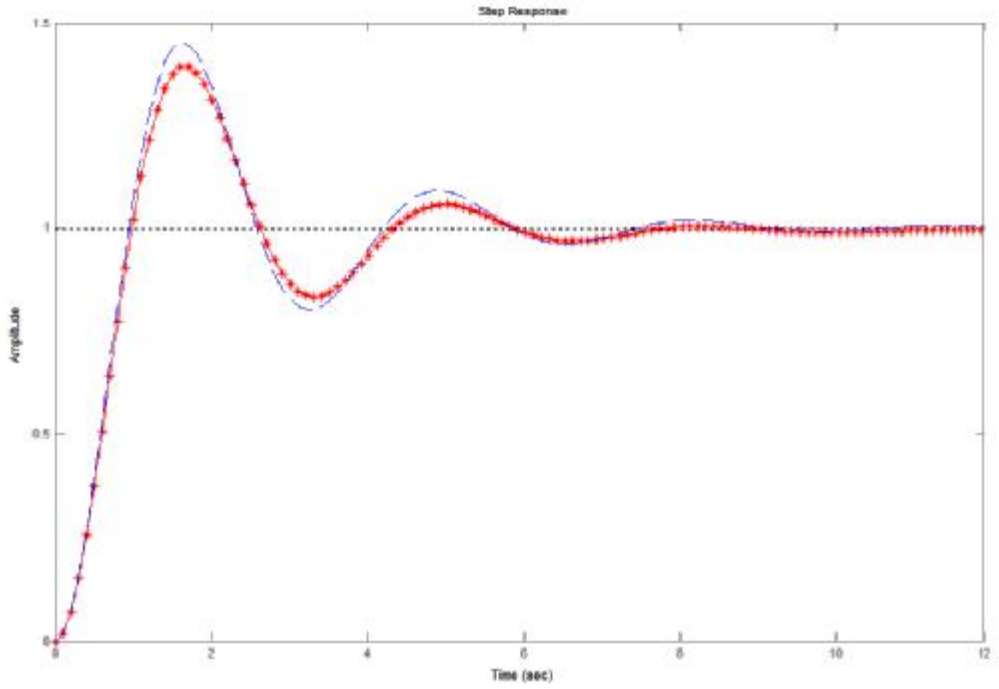


Figura 6.7. Respuesta al escalón de ec.6.1 vrs. Ec. 6.5.

Y aplicando Bode a estas dos funciones tenemos:

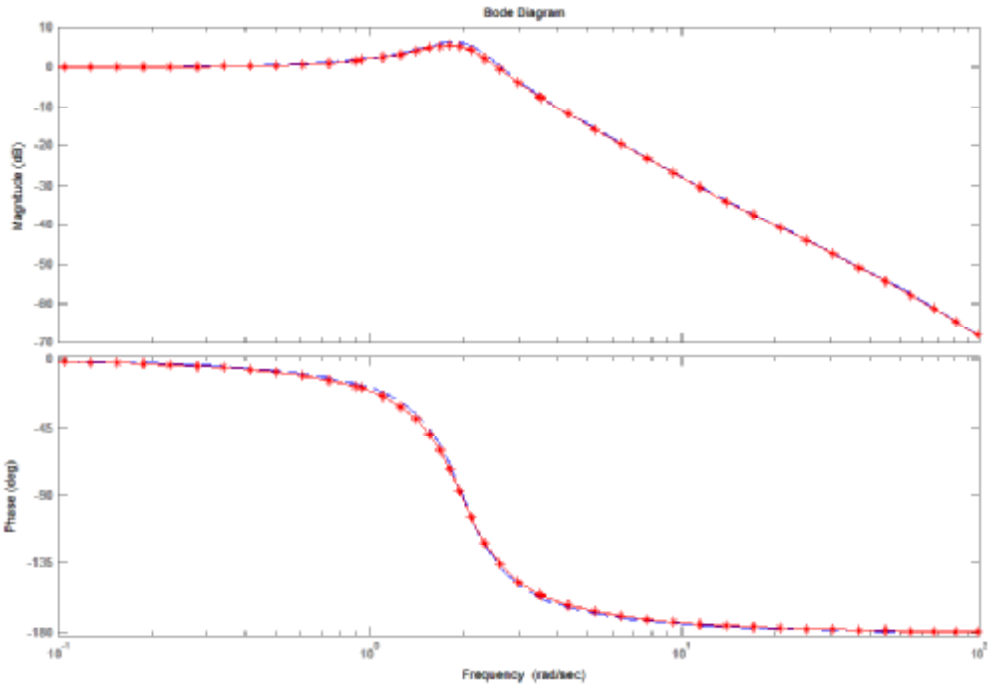


Figura 6.8. Bode de ec.6.1 vrs. Ec. 6.5.

6.1.2 Conclusiones y recomendaciones de los experimentos realizados:

En las pruebas realizadas anteriormente se muestran los 2 casos extremos, el que mejor se acerca al modelo calculado y el que menos se acerca, donde se concluye que:

1. El modelo adquirido mejora incrementando el Nivel del Escalón, debido a que la relación señal a ruido disminuye, esto significa que el ruido no es comparado con los niveles de voltajes adquiridos.

2. La cantidad de combinaciones posibles en la resolución de cero a uno es inferior que de cero a siete, esto significa un modelo menos preciso en la prueba de 1 Volt. Con 10 Seg.

3. El tiempo de observación también es importante, ya que este tiene que ser el tiempo que tarda la etapa del transitorio, si es mayor, el modelo no es preciso porque existen muchos datos repetidos, y si es menor el modelo no es completado por los comandos de identificación de MATLAB.

6.2 Evaluación de parámetros PID y del control en lazo cerrado.

6.2.1 Comparación entre parámetros PID y del control en lazo cerrado discreto y continuo.

A continuación se mostrará que tan efectivos son los cálculos de los parámetros PID (K_p, T_i y T_d) y el manejo del control en lazo cerrado efectuados por el sistema. Para desarrollar esta evaluación se tomará como ejemplo una "planta de temperatura" del Laboratorio de Control Automático del CITT.

EXPERIMENTO:

Se procedió a la captura de los datos de la planta, resultando un modelo de primer orden dado por la siguiente función de transferencia:

$$H(s) = \frac{0.03706}{s + 0.04077}$$

ec.6.6.

A partir de esta función, se calcularon los parámetros K_p, T_i y T_d :

$K_p = 6.11367$; $T_i = 5.75882$; $T_d = 1.4397$

Estos datos fueron introducidos, tanto al controlador PID continuo como al discreto. Además se determinó que el factor de escala del sensor de la planta es de 5.6, esto indica que por cada voltio entregado por el transductor de la planta es equivalente a 5.6 ºC. Como *"Referencia"* para el lazo cerrado se eligió 7 Voltios que en temperatura equivalen a 39.2ºC.

De la figura :

--- Referencia, --- PID continuo, --- PID discreto.

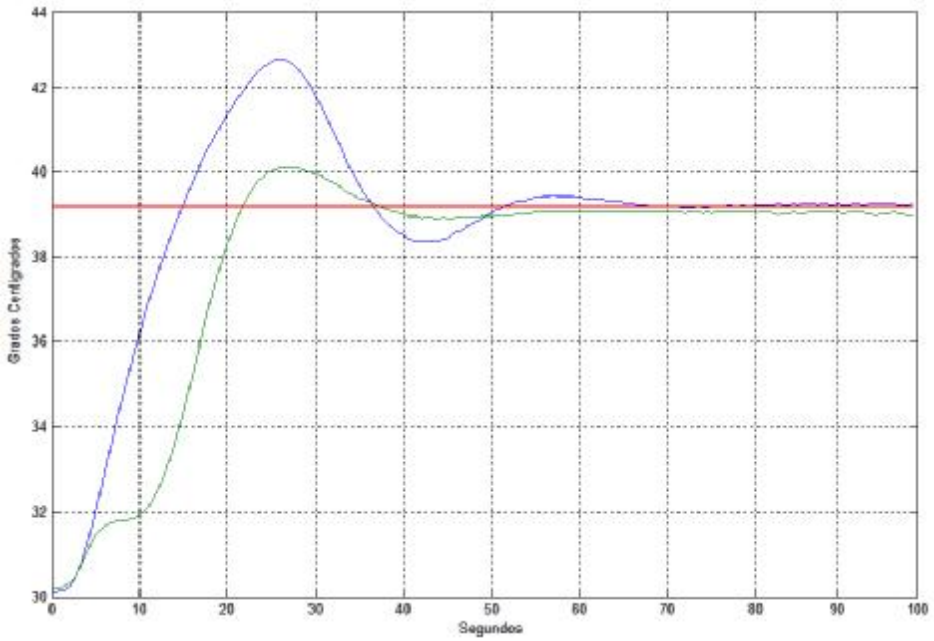


Figura 6.9. Gráfico que muestra la comparación entre el PID discreto y el continuo.

6.2.2 Conclusiones del experimento:

1. Ambas curvas siguen la misma tendencia.

2. Tienen la misma fase.

3. El tiempo de estabilización es similar.

4. El PID discreto corrige el error en estado estacionario, el continuo presenta un leve error, esto podría ser causado por una salida distinta de cero aunque no existe señal a la entrada (offset), ocasionada por los elementos electrónicos analógicos descalibrados que lo componen.

5. Los sobreimpulsos para cada controlador son:

   30 % Discreto.

   12 % Continuo.

6. La diferencia de amplitudes se debe a que en el PID continuo no existen pérdidas de datos, mientras que en el discreto se pierde información entre la toma de una muestra y la otra. Otra de las causas de las diferencias de amplitudes es por la imprecisión en el ajuste de los parámetros PID en el control continuo, ya que son potenciómetros analógicos y podría ocasionar errores de percepción humana.

7. Hablando de la eficiencia en la obtención de los parámetros PID por el sistema, las reglas de Ziegler-Nichols expresan que, "un controlador afinado por dichas reglas presentará un sobreimpulso de aproximadamente de 10% - 60% en la respuesta al escalón"[1]. Se puede concluir que los parámetros están siendo correctamente determinados y solo necesitarán un ajuste fino por parte del usuario.

[1] Pag. 641, Katsuhiko Ogata, *"Ingeniería de Control Moderna"*

7 Conclusiones y Recomendaciones.

7.1 Conclusiones.

- La técnicas de identificación de sistema para la obtención de un modelo matemático dan un resultado estimado, no se puede definir cual es la mejor estructura a elegir para acercarse al modelo real ya que bajo unas condiciones de un experimento un método puede dar mejores resultados, pero si las condiciones del experimento cambian otro método sería el que mejor se apega al modelo real. Una alternativa es probar todos los métodos para cada experimento y comparar las respuestas entre si, esto consume una cantidad impresionante de recursos de cómputo, por lo que para la aplicación desarrollada se ha optado por seguir un tan solo método que es un algoritmo basado en ARMAX con modificaciones al cálculo de predicción de error y gradiente

- Entre los aspectos que restan exactitud a los métodos de identificación de sistema se encuentran: Resolución de los datos, al hacerse pruebas con un canal de baja resolución se obtuvieron resultados mucho menos precisos que con el canal escogido de mayor resolución; los rangos de medición positivos poseen una resolución mayor que los rangos negativos, el tiempo de muestreo debe ser lo mas preciso posible, según pruebas realizadas una variación pequeña en el tiempo de muestreo dio como resultado un modelo que difirió en gran medida al esperado, el ruido en la señal crea la aparición de ceros indeseables.

- El método de áreas para obtención de parámetros de PID resulta un método confiable, debido a que es basado en la curva de reacción de Ziegler-Nichols que ha probado su robustez por décadas y es de fácil implementación ya que efectuar el cálculo de áreas mediante integración de las muestras no implica un algoritmo complicado ni tampoco la utilización excesiva de recursos de computo.

- Si bien la teoría de un PID continuo es el “ideal” a seguir por parte del PID discreto, en la practica este ultimo contiene las siguientes ventajas:

    - Ahorro en Hardware, ya que se puede evitar un sumador.
    - No sufre de falta de calibración por componentes analógicos.
    - Ingreso de parámetros de forma fácil y precisa, evitándose errores por manipulación de escala y ajuste analógicos.
    - Se pueden ejecutar otras tareas por cada ciclo muestreado, por ejemplo: visualización del proceso, cálculos, correcciones etc.

- A pesar que en este documento solo se ha desarrollado la teoría y las rutinas del programa concerniente a la solución implementada, vale la pena hacer mención que en el proceso de investigación para este desarrollo, se estudiaron otras alternativas en la determinación de los parámetros del PID, una de ellas fue la “*asignación de polos en lazo cerrado de la ecuación característica”*, dicha teoría prometía que el diseñador podía especificar con precisión los valores de sobreimpulso y tiempo de establecimiento deseado. A pesar de la elegante y rigurosa demostración matemática con álgebra lineal matricial, en la práctica solo se obtuvo sistemas inestables, que nunca alcanzaron los valores esperados y a lo mucho se obtuvieron resultados muy por debajo que la solución final. Paradójicamente en modo de simulación por computadora todo parecía funcionar, ¿que ocurría?, después de invertir tiempo y desgaste mental se determinó que el modelo matemático obtenido no se limitaba a un modelo real, aquel cuya entrada y salida tienen rangos de operación, el modelo jamás discriminaría que un controlador se satura, que una planta real colapsaría al aplicarle un nivel de 5000 u 8000 voltios o amperios como el modelo matemático lo exigiese. Aunque el método de Ziegler-Nichols es un método empírico que data desde 1950 superó en sobre medida un método que prometía aventajarlo. Para concluir se dirá que en ingeniería las soluciones dependen del conocimiento científico pero también de la experiencia.

- De lo dicho anteriormente hay que hacer mención y dar crédito que gracias a la disposición de los investigadores en llevar mejoras en base a los métodos tradicionales es que ha sido posible llevar a cabo la solución presentada, ya que el método de áreas, es un método desarrollado por William Spinelli en febrero de 2004. También se encontró otras técnicas para la autosintonía de PIDs basadas en integración que tienen mas complejidad de la expuesta. Se concluye que vale la pena retomar lo conocido e innovarlo.

7.2 Recomendaciones.

- El modelo matemático obtenido por la aplicación se aproximan en mayor medida al modelo real si en el experimento se escogen niveles de entrada mas altos para mejorar la relación señal / ruido y un tiempo de observación que mejor describa al transitorio. De lo anterior se recomienda al usuario del programa:

    1. Se recomienda realizar una captura de información inicial con un tiempo largo y nivel de escalón bajo, observar el tiempo que dure el transitorio y el nivel máximo de amplitud, luego repetir el experimento ajustando el tiempo de observación que llegue hasta un poco mas del inicio de la estabilidad, todo esto para determinar un valor apropiado de el *"Tiempo Total del Evento"*.

    2. Para el escalón elegir un voltaje de entrada de forma que la salida no se sature (10 V).

3. Utilizar la opción de eliminación de los polos y ceros no dominantes.
4. Comparar el gráfico del escalón con el gráfico de los datos capturados si los gráficos concuerdan, este será el mejor modelo, si no, aplicar el filtrado.

- El método de áreas puede emplearse en aplicaciones para determinar los parámetro de un PID de autosintonía, utilizando los datos adquiridos en lugar de un modelo matemático, de forma que para aplicaciones con menores recursos informáticos puedan estimarse en tiempo de ejecución, por ejemplo PLCs, microcontroladores, sistemas mínimos, etc.

- En el control en lazo cerrado se pueden obtener tiempos de muestreo significativamente inferiores si el sistema queda dedicado únicamente al control y la visualización sea soportada por otro sistema. Por ejemplo un sistema con microcontrolador dedicado al control de la planta, dicho sistema estaría comunicado a una PC quien se encargue de la visualización.

- Lamentablemente hasta la fecha no se ha encontrado compilador capaz de transformar los modelos LTI de Matlab y de esta forma crear un archivo ejecutable con miras a comercializar la aplicación, sumado a lo anterior existe la limitante que la aplicación esta orientada al uso de la PCI LM 8912.
  Versiones posteriores de Matlab podrían superar el problema de la compilación.

La aplicación desarrollada cuenta con el recurso de importar archivo de texto de manera que se pueden cargar los datos de la adquisición usando equipo diferente y se logre sin dificultad, la determinación del modelo matemático, análisis y determinación de los parámetros del PID

Para lograr compatibilidad con otros adquisidores, se tendría que contar con la información de comandos y métodos de configuración para cada equipo en particular, dicha información almacenada en un archivo de manera que, al leer el archivo se actualicen los procedimientos para operar el equipo, dicho diferente: la creación de controladores(drivers).

Por otra parte si únicamente interesa la determinación de parámetros del controlador estos pueden ser calculados con los datos adquiridos en lugar de utilizar el modelo matemático, de esta forma no existiría problema con la creación del ejecutable; con esta misma restricción también es posible rediseñar la aplicación en otro lenguaje de programación.

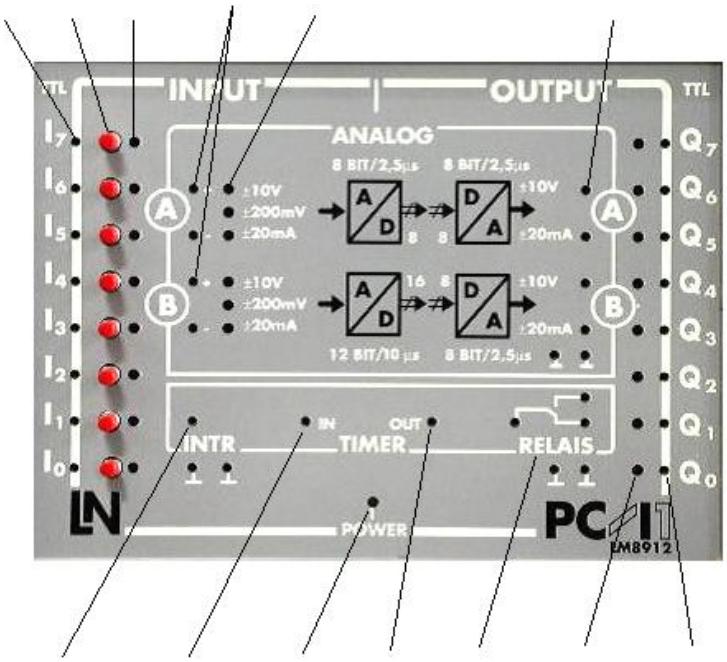
8 Anexos.

Anexo A

Interface PC / I [8]

Parte frontal de la PCI.

[4] [5] [6] [1] [2] [3]



[7] [8] [11] [9] [10] [12] [13]

1. Canales de entrada analógica-diferencial A y B.
2. Indicador del rango de medición.
3. Canales de salida analógica A y B.
4. Entradas digitales.
5. Conmutadores de simulación para las entradas digitales.
6. Indicadores LED para las entradas digitales.
7. Entrada-interruptor para las señales especiales.
8. Entrada del timer para la medición del tiempo y la frecuencia.
9. Salida del timer para la salida de frecuencia.
10. Contacto del relé (contacto conmutador).
11. Indicador LED para la tensión de operación.
12. Indicador LED para las salidas digitales.
13. Salidas digitales.

Estructura de la PC/I-1

La PC/I-1 (PC-Interface 1) es una interface compacta para un ordenador personal con 2 entradas analógicas diferenciales, conmutable (± 10V, ±20mA; cana A y B), para la conexión de sistemas controlados analógicamente, de componentes activos y pasivos, para el registro de las curvas características de máquinas eléctricas en la simulación de diferentes cargas técnicas, etc. En la parte posterior del dispositivo se dispone de 4 canales de entrada analógicos adicionales para conectar a un amplificador aislador mediante un clavijero diodo (±2,4V). Los canales de entrada y de salida poseen una resolución de 8 bits a excepción del canal de entrada B, el cual posee una resolución de 12 bits.

Una interface de 8-Bit-E/A (entrada/salida) sirve para la conexión de conmutadores, activadores, de elementos indicadores digitales y de modelos controlados digitalmente, como por ej. Semáforos, instalación de ascensores y motores de paso. Señales digitales TTL pueden ser leídas o ser predeterminados mediante conmutadores (simuladas). La salida se efectúa a través de una memoria tampón (12mA). Los estados de todas las señales son indicados mediante LEDs.

Adicionalmente se dispone de señales especiales:

- 1 Frecuencia de salida, programable de 10Hz hasta 100KHz.
- 1 Entrada para la medición de frecuencia (10Hz hasta 100KHz).
- 1 Relé (contacto conmutador; carga admisible: 220V/4A.).

La conexión para todas las entradas y salidas se realiza a través de clavijeros de 2mm.

La PC/I-1 está construida como una interface inteligente. Está provista de un procesador (Base: 68000). La razón de ello es la siguiente:

- La interface es independiente del ordenador PC. Se puede conectar a otra PC a través de la interface en serie (RS-232).

- Puede compartir tareas. La PC/I asume la entrada y salida directa de datos del objeto a medir o controlar, mientras que la PC puede asumir las funciones superiores, tales como el propio procesamiento o la representación de los datos medidos.
- La PC/I ofrece un método de conexión del ordenador PC con la periferia a través de cables conectores.

Las funciones de la PC/I-1

La PC/I es controlada mediante comandos a través de la interface serial del ordenador los que activan funciones específicas. Para que la duración de la transmisión de los comandos y de los datos sea la mas corta posible, se transmite inmediatamente un byte (8 bits) como carácter cuando se enciende el aparato o inmediatamente después de un reset; sin que se haga la previa transformación de un byte en 2 caracteres ASCII y se efectúe la transmisión de estos (transmisión de valores hexadecimales en lugar de sus equivalentes ASCII). De esta manera se reduce el número de caracteres transmitidos a la mitad. Por la misma razón se desiste del estandarizado modo de operación con Echo, es decir que los caracteres enviados a la PC/I son "tragados" por esta y sin comentarios. Sin embargo en casos especiales, es posible que se desee que la comunicación con la PC/I sea ejecutada sobre un nivel puramente basado en caracteres. Esto por ejem. Tiene sentido cuando se opera el ordenador con un programa terminal. Para tal fin se dispone de dos comandos con los cuales se efectúa la conmutación de una transferencia binaria a una transferencia en ASCII y a la operación con Echo. Por ej., para poner todas las salidas digitales a uno se debe enviar la siguiente serie de caracteres en modo ASCII:

56 FF ↵

El comando y el parámetro deben separarse con un carácter en blanco; el carácter ↵ designa al retorno del carro ($0D_h$).

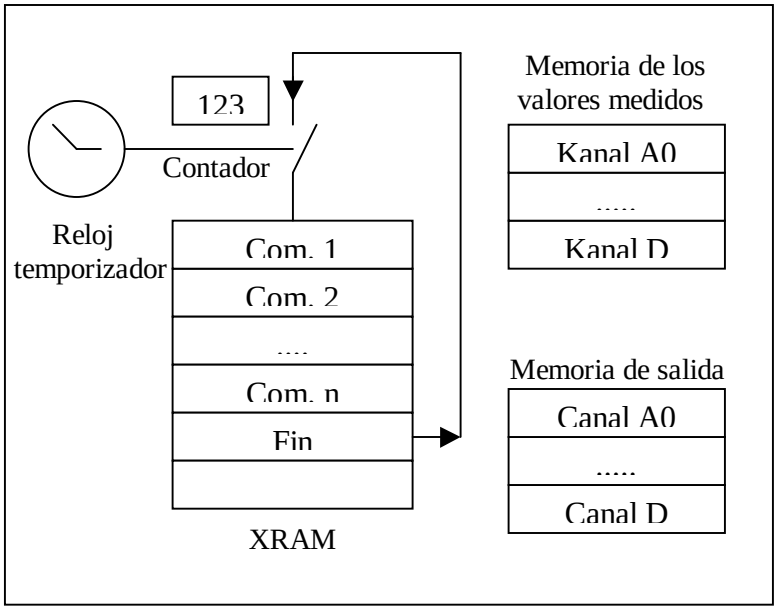
Cuando el modo binario está activo, el comando y el parámetro se transfieren cada uno como un byte, es decir en total como dos bytes.

Los comandos son de dos tipos:

- Comandos directos.
- Comandos basados en la memoria.
- Comandos de control.

Los comandos **directos** producen una acción directa de la PC/I. Este tipo de comandos son por ej. Los que introducen por lectura de los canales analógicos y los que ejecutan la salida por lectura de los mismos. La velocidad de ejecución de estos comandos es limitada por la velocidad de transferencia de la interface serial. La velocidad de transferencia es de 9600 baudios, fija. Para transferir un carácter se requiere aprox. 1 ms. Como para muchas aplicaciones esto es demasiado lento, se dispone de comandos alternativos: los comandos basados en la memoria.

Mediante los comandos **basados en la memoria** la medición se realiza independientemente, si se desea, con una alta velocidad de la PC/I. Para ello se transmiten órdenes especiales a la PC/I, las que por ej. Fijan el número de mediciones y el tiempo entre dos mediciones. Como trabaja este modelo, se muestra en la figura siguiente:



La parte esencial del modelo es la memoria de ejecución conocida como XRAM. En ella se almacenan los propios comandos, como por ej. Los de la lectura de datos. Si se inicializa una medición mediante el comando **Run**, todas las órdenes introducidas en la XRAM son procesadas unas tras de la otra. El reloj temporizador garantiza que todos los comandos sean procesados en una determinada secuencia de tiempo. El mas corto intervalo de tiempo es de 100 µs. La función del contador es la de garantizar que solo el número deseado de mediciones sean llevadas a cabo.

Cada canal analógico de entrada y de salida así como las salidas digitales de entrada y de salida poseen cada una un área de memoria para 1024 valores de medición asignados en la secuencia siguiente:

Entrada analógica A0
Entrada analógica A1
Entrada analógica A2
Entrada analógica A3
Entrada analógica A4
Entrada analógica B
Entrada Digital D0
Salida Analógica A0
Salida Analógica B
Salida Digital D0

Durante una medición los valores medidos asignados a los canales pueden ser almacenados directamente para luego ser solicitados. Con ello se evita de solicitar cada valor de medición individual a través de la interface serial. De esta forma se obtiene un registro de los valores esencialmente mucho mas rápido que al emplear los comandos directos (>400 kHz en comparación de un max. De 1 kHz para 9600 baudios).

Durante el proceso de medición la memoria tampón asignada a los canales de entrada y salida puede llenarse con los valores medidos o vaciarlos. Esto se realiza con los comandos

basados en la memoria que son transferidos a la PC/I antes de una medición. En contraposición a los comandos directos estos no dependen del tiempo ya que no son transferidos durante la medición. Por esta razón se les puede acompañar con parámetros, como por ej., el número de canal, da tal manera que sólo son necesarios unos cuantos comandos.

La propia medición se inicializa con un comando de control determinado (Run, ver mas abajo).

Los parámetros de un comando basado en la memoria se transfieren, como los propios comandos, en código hexadecimal. Existen parámetros de 8 bits (nn) así como de 16 bits (nnnn), los cuales pueden ser separados con un espacio en el modo ASCII.

Dentro de un comando los canales se diferencian mediante los siguientes números de canales:

Canal	No.
Entrada/salida analógica A0	00
Entrada analógica A1	02
Entrada analógica A2	04
Entrada analógica A3	0
Entrada analógica A4	08
Entrada/salida analógica B	0C
Entrada/salida digital	20

Como en el caso de los comandos de control, el ordenador recibe un "!" como **caracter de reconocimiento** o de otro modo un "?", después de que se haya ejecutado un comando y haya sido correcto. Si el modo ASCII está activo se indicará el siguiente mensaje de error:

- Error en el parámetro.
- Error en el comando.
- Memoria llena.

ReadChn(Lectura de canal) [40]...[46],[48]

- **Descripción.**

  Estos comandos retornan el valor leído del canal correspondiente.

- **Valor de Retorno.**

  El valor de retorno es un Byte o una palabra (Word) dependiendo del comando.

  En la tabla siguiente se muestra la relación entre los comando y los valores de retorno:

No.	Entrada	Retorno en	Rango del valor
40	Canal analógico A0	Byte	80..0..7F
41	Canal analógico A1	Byte	80..0..7F
42	Canal analógico A2	Byte	80..0..7F
43	Canal analógico A3	Byte	80..0..7F
44	Canal analógico A4	Byte	80..0..7F
45	Canal analógico B	Word	8000..0..7FFF
46	Canal digital D	Byte	0..FF
48	TIMER IN	Word	1..2710

Para las entradas analógicas el rango de valores en Bytes se encuentra entre -127 y + 128 (entre 80h y 7Fn) y para los valores en Words, entre -32.767 y +32.768 (entre 8000n y 7FFFn).

En las entradas digitales cada bit del byte retomado corresponde a una de las entradas hastaa 17, en donde el estado de 17 se almacena como el bit más significante.

En el caso del temporizador (timer) la frecuencia medida es retornada como un múltiplo de 10Hz. El rango de medición se encuentra entre 10Hz y 100KHz. lo que resulta en un rango de valores desde 1 hasta 10.000 (entre 1 y 27 10n).

- **Nota**

  No se transferiría un carácter de reconocimiento.

Cuando se transfieren palabras (dos Bytes) en el modo binario (ASCII en OFF), ambos bytes se transfieren en una frecuencia inversa, debido a que los lenguajes BASIC, Pascal y C simplifican la lectura de los valores de 16 bits. Primeramente se transfiere el byte menos significante y luego el más significante.

ClrinBuffer(Limpia Memoria de Entrada) [30][nn]

- **Descripción**

Este comando borra la memoria de los valores medidos del canal de entrada. Todos los valores son puestos a cero.

- **Parámetro**

nn Número de canal {00/ 02/04/06/08/0C/20/40}
Si se introduce el parámetro **40** la memoria de ejecución **XRAM** será borrada.

- **Valor de retorno**

Aparece un carácter de reconocimiento.

ReadinBuffer(Leer Memoria de Entrada) [33][nn] [nnnn]

- **Descripción**

Con este comando se puede leer de una memoria de un canal de entrada un número determinado de valores medidos.

- **Parámetro**

nn Número de canal {00 | 02 | 04 | 06 | 08 | 0C | 20}

nnnn Número de bytes a ser leídos. Este puede elegirse hasta un valor máximo de 32767. el valor específica básicamente el número de bytes. Si Ud. Quiere p. ej., leer 100 valores de la memoria de entrada del canal B, deberá entonces transferir 200, ya que estos valores están almacenados como palabras (16 bit).

- **Valor de retorno**

Es un carácter de reconocimiento seguido de una serie de nnnn bytes. Estos bytes son transferidos en la misma secuencia en la cual han sido almacenados. Esto significa que los valores de 16 bits se transferirán primero el más significativo. Por el contrario si se trata de una variable entera primero se tendrá el byte menos significativo.

- **Nota**

El número de valores no debe ser mayor que el tamaño de la memoria de los valores medidos (p. ej. 1024). Sin embargo es posible llenar varias memorias de valores medidos con un solo comando **ReadinBffer.** En este caso si se permite que el numero sea mayor. Observe la secuencia de memorias previamente definidas.

InBuffer(Poner en memoria) [34][nn]

- **Descripción**

El comando **InBuffer** escribe la orden leer un valor de un canal específico y escribirlo en el próximo lugar libre en la memoria de valores medidos en la memoria ejecución (XRAM). El número de valores medidos es dependiente del número de ciclos de medición definidos por el comando **setCycle** (ver más adelante). De esta

manera se pueden utilizar subsiguientes áreas de memoria de canales **que no son requeridos** (ver anteriormente: secuencia de las áreas de memoria).

- **Parámetro**

nn Número de canal {00 | 02 | 04 | 06 | 08 | 0C | 20}

- **Valor de retorno**

Aparece un carácter de reconocimiento.

- **Nota**

Si la orden se llama varias veces sin borrar el contenido de la memoria XRAM, entonces se escribirá en la memoria XRAM un número correspondiente de entradas.
Así p. eje. el Comando leer canal A0 puede ser escrito 100 veces en la memoria de ejecución. Con un solo procesamiento de la memoria de ejecución se leería el canal A0 100 veces con la velocidad más alta posible. El máximo número de entradas depende del tamaño de la memoria de ejecución. Un comando XRAM ocupa 4 bytes.

LOS COMANDOS DE CONTROL.

AsciiMode(Modo Ascii) [01]

- **Descripción**

El modo de transmisión binario se deberá ajustar después de encender el dispositivo o después de una inicialización para mantener la duración de la transferencia de comandos y de datos lo mas corto posible. Esto significa que con un caracteres transfiere inmediatamente un byte (8bit) si que anteriormente se haya convertido el byte en 2 caracteres ASCII. Y después se transfiera (transmisión de valores binarios

en lugar de sus equivalentes ASCII). Con ello se reúce el número de caracteres o transferencias a la mitad.

Este modo de operación es inapropiado si se opera la PC/1 con un programa terminal. Por esta razón el comando **AsciiMode** puede conmutar el modo de operaciones a un modo de trabajo estrictamente con caracteres. Así se transferirán todos los comandos y parámetros como caracteres ASCII. El comando para leer las entradas digitales no se transfiere como un byte (46n), sino como la cadena de caracteres "46", es decir como dos bytes.

Este comando tiene un efecto conmutativo, es decir después de que el modo ASCII haya sido activado, un nuevo envío del comando resulta en una desactivación del modo.

- **Valor de retorno**

Aparece un carácter de reconocimiento y finalmente una cadena de caracteres con el modo presente, esto es: ASCII - Off ó ASCII – On.

- **Nota**

Cuando se opera con el programa terminal el modo ASCII se puede conmutar mediante la combinación de teclas **Ctrl+A.**

Reset(Reinicio) [03]

- **Descripción**

El hardware y el software de la PC/1 son llevados a sus posiciones básicas. Esto corresponde al estado después del encendido del dispositivo. Todas las salidas son puestas a cero, los rangos de medición a $\pm$10V, el relé de contacto se abre, el timer es detenido, la memoria de ejecución y la de valores medidos son borrados y los modos ASCII y ECHO son desactivados.

Esta función debe ser llamada al iniciar cada programa.

- **Valor de retorno**

Aparece un carácter de reconocimiento.

- **Nota**

Observe que la función requiere de un determinado tiempo para la reinicialización, durante el cual la PC/1 no recibe ningún comando. El carácter de reconocimiento se envía inmediatamente después de recibir el comando, es decir no se tiene la información si el comando se procesó o no. Por ello se debe prever un tiempo de retardo de aprox. 2 segundos antes de enviar el próximo comando.

EchoMode(Modo con Eco) [05]

- **Descripción**

Como para el modo ASCII, al encender el dispositivo o al ejecutar una reinicialización, la operación con eco no se emplea para poder reducir la duración de la transferencia, esto significa que los caracteres enviados a la PC/I son “tragados” por ella y sin comentario. El comando **EchoMode** permite que los caracteres enviados a la PC/I vuelvan al ordenador PC con el fin de controlar mejor el intercambio de datos cuando se opera con un programa terminal.

El comando tiene efecto comunicativo, es decir después de que el modo Echo haya sido activado, en nuevo envío del comando resulta en una desactivación del modo.

- **Valor de retorno**

Aparece un carácter de reconocimiento y finalmente una cadena de caracteres con el modo presente, esto es: **Echo – Off ó Echo – On**.

- **Nota**

Cuando se opera con el programa terminal el modo Echo se puede conmutar mediante la combinación de teclas **Ctrl.+E.**

Help(Ayuda) [08]

- **Descripción**

Cuando se envía este comando a la PC/I, ésta envía a la PC una lista de todos los comandos disponibles. Este comando es muy útil cuando se opera la PC/I con un programa terminal para fines de prueba.

- **Valor de retorno**

Aparecen varias "Páginas" con el listado de comandos. Antes de cada "Página" se transfiere el carácter de control ^L ($0C_n$) el cual permite borrar la pantalla cuando se opera con dispositivos terminales.

Run(Ejecutar) [25] [nn]

- **Descripción**

El subprograma **Run** arranca el procesamiento de los comandos presentes en la memoria de ejecución (XRAM).

- **Parámetro**

nn Determina el modo de procesamiento.

0 La memoria de ejecución es procesada una sola vez.

1 La memoria de ejecución es procesada continuamente. Este modo solo se puede interrumpir mediante el comando **Stop.**

2 La memoria de ejecución es procesada una sola vez con transferencia de datos a la PC.

3 La memoria de ejecución es procesada continuamente con transferencia de datos a la PC. Este modo solo se puede interrumpir mediante el comando **Stop.**

Los **modos 0 y 1** no transfieren datos a la PC. Los datos deberán ser leídos después de la medición mediante el comando **ReadlnBuffer.**

Sólo los **modos 2 y 3** transfieren datos simultáneamente a la PC. La transferencia empieza tan pronto como se haya escrito un valor en la memoria tampón (buffer). Aquí se tiene que sólo el contenido de la memoria del canal con el número de canal más bajo se transfiere automáticamente a la PC.

Los **modos 2 y 3** son apropiados para los procesos largos. El acceso a los valores ya medidos se efectúa sin tener que esperar a que se complete el ciclo de medición.

- **Valor de retorno**

  Aparece un carácter de reconocimiento.

- **Nota**

  Cuando el programa estan contenidos todos los comandos son procesados correctamente.

SetCycle(Configuración del Ciclo) [27] [nnnn$_1$] [nnnn$_2$] [nnnn$_3$] [nn$_1$]

- **Descripción**

  Este comando fija el tiempo del ciclo de procesamiento de los comandos presentes en la memoria de ejecución (XRAM).

- **Parámetro**

$nnnn_1$ Especifica el número de ciclos de medición, es decir, cuantas veces van a ser procesadas las instrucciones del XRAM, por ejemplo: Al emplear el comando **InBuffer**, cuántas veces va a ser requerido y almacenado un valor de medición de un canal. Por ello, el numero de ciclos de medición no deberá ser mayor que el tamaño de la memoria de un canal (400_h); sin embargo el número máximo no deberá ser mayor que la longitud de la memoria del canal seleccionado hasta el termino del ultimo canal (rango de memoria de la salida digital).

$nnnn_2$ Es el número de valores de medición después de que se haya cumplido la condición del disparador (trigger).

$nnnn_3$ Especifica el período del ciclo. Este tiempo esta prefijado como un múltiplo de 10µs. El tiempo mínimo a introducirse es de 100µs y el tiempo máximo es de 284440µs. Este parámetro solo se emplea cuando **nn_1** = 1.

nn_1 Con esta variable se define el modo de procesamiento. En total se dispone de cuatro modos diferentes.

a. El procesamiento de los comandos se lleva a cabo lo más rápidamente posible como consecuencia se obtienen períodos de ciclos menores a 100µs. El periodo de ciclo real es independiente del número de comandos presentes en la memoria de ejecución.
   Si se lee, por ejemplo, el canal A0, el período de ciclo será de 4µs+2.2µs. El primer valor es el retardo dependiente del programa y el segundo valor es el tiempo requerido para leer el canal analógico.

b. El procesamiento de os comandos se lleva a cabo en el marco de tiempo especificado.

c. Tiene la misma función que **0** con la diferencia de que aquí se tiene que considerar la condición del disparador.

d. Tiene la misma función que **1** con la diferencia de que aquí se tiene que considerar la condición del disparador.

- **Valor de retorno**

Aparece un carácter de reconocimiento.

Stop(Detener) [1B]

- **Descripción**

El comando **Stop** interrumpe el procesamiento de los comandos, en la memoria de ejecución, previamente iniciado por el comando **Run.** Este se hace efectivo solo cuando la PC/I se encuentra en el modo Run. Este comando se ha implementado para interrumpir el procesamiento de la memoria de ejecución. Por ejemplo en caso de aplicaciones que requieran tiempos muy largos.

- **Valor de retorno**

Aparece un carácter de reconocimiento.

SetGain(Configuración de Ganancia) [26] [nn_1] [nn_2]

- **Descripción**

Este comando sirve para ajusta los rangos de medición para los canales de entrada analógica A0 y B.

- **Parámetro**

nn_1 Número de canal {00 | 0C}. Solo se permiten transferir los valores para los canales A0 y B. Los canales A1-A4 tienen un rango de medición fijo de +/- 2.4V.

nn_2 Especifica el rango de medición deseado.

0 +/- 10V

1 +/- 200 mV

2 +/- 20 mA

Todos los otros valores producen mensajes de error.

- **Valor de retorno**

Aparece un carácter de reconocimiento.

Anexo B

Cajas de Herramientas de MATLAB.

A continuación se detallarán solamente los comandos y la sintaxis a utilizar dentro del proyecto.

IDENTIFICACION DE SISTEMAS

AR

Estima los parámetros del modelo AR para series de tiempo escalar.

Los parámetros de la estructura en el modelo AR son estimados usando el método de variación de mínimo cuadrado.

Model = AR(Y,N) o TH = AR(Y,N,Approach) o TH = AR(Y,N,Approach,Win)
donde:

Model: Devuelve el modelo de tipo IDPOLY con los parámetros estimados del modelo AR.

Y: Serie de tiempo a ser modelado, debe ser ingresado como un objeto de tipo IDDATA.

N: Orden del modelo.

APPROACH: Método utilizado, cualquiera de los siguientes:

'fb': Aproximación adelante – atrás (por defecto).

'ls': Método de mínimo cuadrado.

'yw': Método Yule-Walker.

'burg': Método Burg.

'gl': Método del Arreglo Geométrico.

Pueden obtenerse coeficiente de reflexión y pérdidas utilizando la forma:

[Model,REFL] = AR(y,n,approach)

Si alguno de estos argumentos termina con cero ('burg0') el cálculo de la covarianza es suprimida.

Win: Ventana Empleada, cualquiera de los siguientes:

‘now’: No ventaneado (por defecto, excepto cuando approach= 'yw').

‘prw’: Pre ventaneado.

‘pow’: Post ventaneado.

‘ppw’: Pre y post ventaneado.

ARMAX

Calcula la predicción de error estimada de un modelo ARMAX

M = ARMAX(Z,[na nb nc nk]) or M = ARMAX(Z,'na',na,'nb',nb,'nc',nc,'nk',nk)

Donde:

M: Devuelve el modelo estimado en un objeto con formato IDPOLY.

Z: Datos estimados en un objeto con formato IDDATA.

[na nb nc nk]: Ordenes y retardos del modelo.

Si los datos tienen varias entradas, nb y nk son vectores fila cuya longitud es igual al número de canales de entrada.

ARX

Los parámetros de la estructura en el modelo ARX son estimados usando el método de variación de mínimo cuadrado.

M = ARX(DATA,ORDERS)

Donde:

M: Devuelve los parámetros estimados del modelo ARX junto con la covarianza y la información de la estructura. Para sistemas de salida única M es un objeto de tipo IDPOLY, mientras que en sistemas de múltiples salidas M es un objeto de tipo IDARX.

DATA: Datos estimados como un objeto de tipo IDDATA.

ORDERS = [na nb nk], ordenes y retardos del modelo.

BJ

Calcula la predicción de error estimada del modelo Box-Jenkins.

M = BJ(Z,[nb nc nd nf nk]) o M = BJ(Z,'nb',nb,'nc',nc,'nd',nd,'nf',nf,'nk',nk)

(Al omitir el orden es tomado como cero y el argumento del orden es arbitrario).

Donde:

M: Devuelve los parámetros estimados del modelo BJ junto con la covarianza y la información de la estructura.

Z: Datos estimados como un objeto de tipo IDDATA.

[nb nc nd nf nk]: Ordenes y retardos del modelo.

COMPARE

Compara la salida simulada/predecida con la salida medida.

COMPARE(DATA,SYS,M)

Donde:

DATA: Datos salida-entrada con lo cual se hace la comparación (datos de validación), es un objeto de tipo IDDATA.

SYS: Modelo en un objeto de tipo IDMODEL.

M: Horizonte Predictivo, si M = Inf, se obtiene la simulación del sistema.

YH: El resultado de la Salida al ser simulada/predecida.

COMPARE grafica YH junto con la salida medida en Z y muestra la variación respecto al modelo.

C2D

Convierte un modelo de tiempo continuo a tiempo discreto.

MD = C2D(MC,T,METHOD)

Donde:

MC: Modelo de tiempo continuo como un objeto de tipo IDMODEL.

T: Intervalo de muestreo.

MD: Modelo en tiempo discreto, objeto de tipo IDMODEL.

METHOD: 'Zoh' (por defecto) o 'Foh', correspondiente a la asunción que la entrada se mantenga en orden cero o en primer orden.

DTREND

Remueve valores medios o medias lineales, offset, ya sea en forma total o parcial (por tramos).

Y = DETREND(X)

Donde:

Y: Vector del cual se le han extraído valores fuera de la tendencia.

X: Valores originales de entrada.

D2C

Convierte un modelo de tiempo discreto a tiempo continuo.

MC = D2C(MD) o MC = D2C(MD, METHOD)

Donde:

MC: Modelo de tiempo continuo como un objeto de tipo IDMODEL.

MD: Modelo en tiempo discreto, objeto de tipo IDMODEL.

METHOD: Alguno de los siguientes:

‘tustin’, 'prewarp', 'matched'

IDDATA

Empaca los datos de entrada-salida en un objeto que se denomina IDDATA.

data = iddata(y,u,Ts)

Donde:

Data: Objeto iddata creado

Y: Es el vector columna que contiene los datos de los canales de salida.

U: Es el vector columna que contiene los datos de los canales de entrada.

Ts: Intervalo de muestreo

IDFILT

Filtra los datos utilizando filtros de tipo: General y Butterworth.

ZF = IDFILT(Z,N,Wn) or ZF = IDFILT(Z,FILTER)

Donde:

Z: Datos de entrada-salida como una matriz o un objeto IDDATA.

ZF: Datos filtrados. Iran acorde a Z, es decir de tipo IDDATA o como vector columna.

FILTER: Un sistema lineal SISO, especificado como: LTI, IDMODEL, Arreglo de matriz en el espacio de estado o Numerador, Denominador.

N: Orden del filtro Butterworth.

WN: Frecuencia de corte, en fracciones de la frecuencia de Nyquist.

OE

Calcula la predicción de error estimada del modelo “output error”.

M = OE(Z,[nb nf nk])

Donde:

M: Devuelve los parámetros estimados del modelo. En un objeto de tipo IDPOLY junto con la covarianza y la información de la estructura.

Z: Datos estimados como un objeto de tipo IDDATA.

[nb nf nk]: Ordenes y retardos del modelo.

PE

Calculo de predicción de error

E = PE(Model,DATA)

Donde:

E: Es el error de predicción. E.OutputData contiene los errores

DATA: Los datos de salida-entrada como objeto IDDATA.

Model: El modelo como tipo IDMODEL.

PEM

Calcula la predicción de error estimado de un modelo lineal general

MODEL = PEM(DATA,Mi)

Donde:

MODEL: Devuelve el modelo estimado. En un objeto de tipo IDPOLY o IDSS junto con la covarianza y la información de la estructura.

DATA: Los datos de salida-entrada como objeto IDDATA.

Mi: Una matriz o un objeto que define el modelo de la estructura.

Mi = nx (como un entero), da el orden del modelo lineal en espacio de estado.

SS, TF, ZPK, FRD

Transforma el objeto del modelo del Toolbox de Identificación de Sistema a un modelo LTI del Toolbox de Sistema de Control.

sys = ss(mod)

sys = tf(mod)

sys = zpk(mod)

sys = frd(mod)

Donde:

Mod: es cualquier objeto IDMODEL.

SYS: devuelve el modelo LTI.

SISTEMAS DE CONTROL

BODE

Calcula la respuesta en frecuencia de Bode en los modelos LTI.

bode(sys)

Bode calcula la magnitud y fase de la respuesta en frecuencia de los modelos LTI. La magnitud se grafica en decibeles (dB) y la fase en grados.

NYQUIST

Calcula la respuesta en frecuencia de Nyquist en los modelos LTI.

nyquist(sys)

El grafico de Nyquist es usado para analizar propiedades del sistema que incluyen margen de ganancia, margen de fase y estabilidad.

RLOCUS

Grafica el lugar de las raíces.

rlocus(sys)

Calcula y grafica el lugar de las raices de un modelo de sistema SISO en lazo abierto. La función puede ser aplicada a un lazo de retroalimentación negativa.

SS

Define o convierte un modelo LTI a uno de espacio de estado.

sys = ss(a,b,c,d)

De acuerdo al siguiente modelo:

$$x = Ax + Bu$$

$$y = Cx + Du$$

STEP

Calcula y grafica la respuesta al escalón Unitario.

step(sys)

El modelo puede ser continuo o discreto, SISO o MIMO.

TF

Define o convierte un modelo LTI a la forma de función de transferencia.

sys = tf(num,den)

sys = tf(num,den,Ts)

Donde:

Sys: función de transferencia del sistema

Num: son los coeficientes del polinomio del numerador.

Den: son los coeficientes del polinomio del denominador.

Ts: tiempo de muestreo para el caso de sistemas discretos.

CONTROL DE INSTRUMENTOS.

PUERTO SERIE

Obj: Objeto de tipo *instrument* (para nuestro caso, puerto serie).

FCLOSE

Desconecta el puerto serie con el objeto de tipo *instrument.*

fclose(obj)

FOPEN

Conecta el puerto serie con el objeto de tipo *instrument.*

fclose(obj)

FREAD

Lee datos binarios desde el puerto serie.

A = fread(obj,size)

Donde:

Size: Número de datos a ser leídos.

A: Datos binarios devueltos por el puerto.

FWRITE

Escribe datos binarios al puerto serie.

fwrite(obj,A)

Donde:

A: Datos binarios a ser transmitidos.

SERIAL

Crea el objeto tipo puerto serie.

obj = serial('port','PropertyName',PropertyValue,...)

Donde:

‘**port’:** Nombre del Puerto serie.

'PropertyName': Nombre de la propiedad del Puerto.

PropertyValue: Valor tolerado por el **PropertyName.**

SET

Configura o muestra las propiedades del puerto.

set(obj,'PropertyName',PropertyValue,...)

Donde:

Vease la función anterior.

PROPIEDADES A CONFIGURARSE DEL PUERTO SERIE

Estas se ajustan con PropertyName y PropertyValue.

BaudRate: Velocidad de transmisión = 9600 bps

DataBits: Número de bits de datos a transmitir = 8

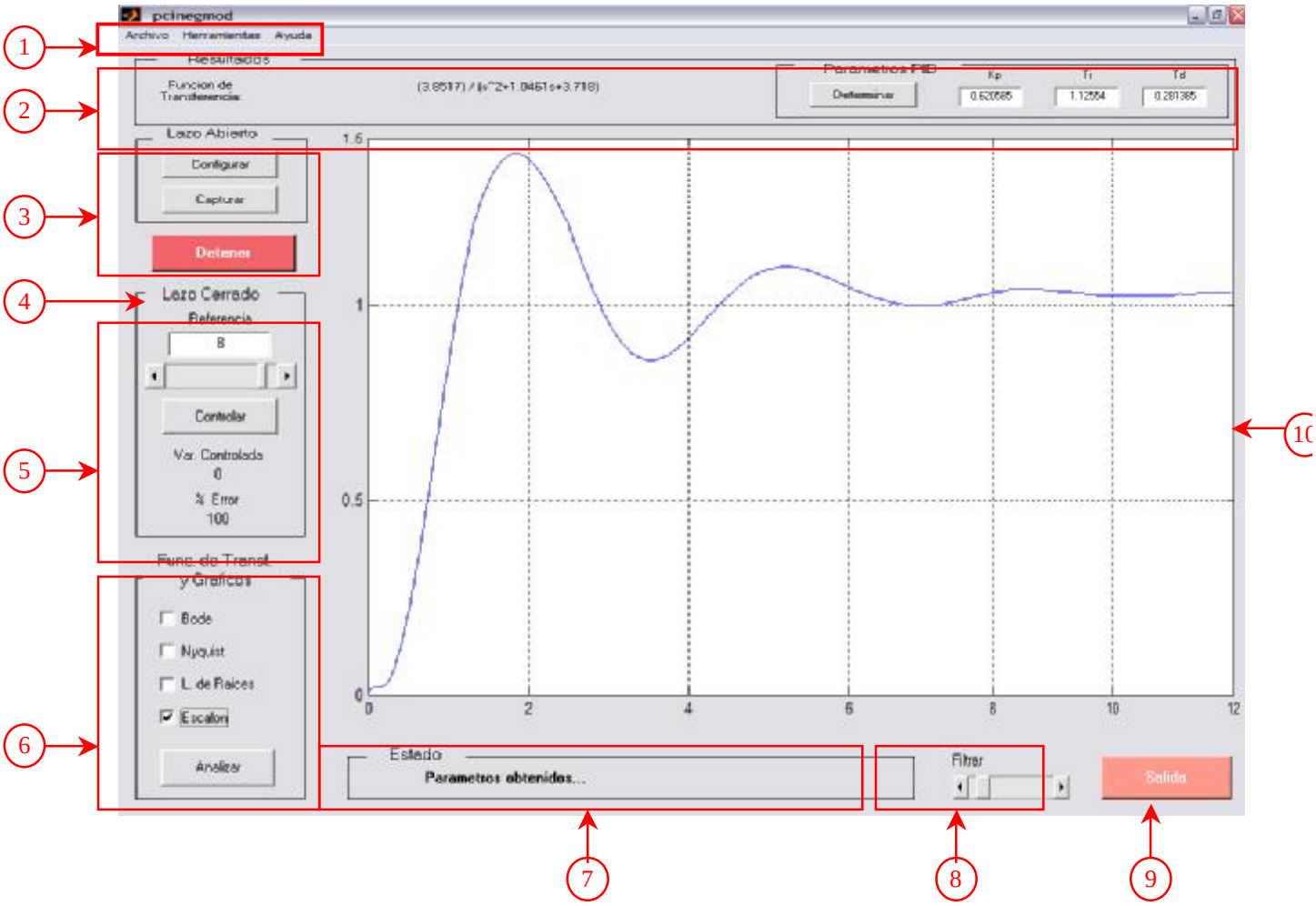
FlowControl: Control de Flujo = Ninguno.

StopBits: Bits de paro = 2

Parity: Paridad = Ninguna.

Anexo C

Manual del Usuario.



1. **MENÚS PRINCIPALES.** Menús desplegables para realizar tareas comunes de Windows y propios de la interfaz, estos menús son: ARCHIVO, HERRAMIENTAS Y AYUDA.

   ARCHIVO:

   Abrir: Recupera información de una determinada planta previamente capturada y almacenada.

   Guardar: Almacena los datos: Entrada de la planta, Salida de la planta y el vector tiempo capturados por el sistema.

   Importar Archivo de Texto: Permite abrir un archivo en formato código ascii que haya sido escrito como tres vectores columnas en el siguiete orden: tiempo, entrada, salida. La columna tiempo lleva implicito el tiempo de mustreo de la captura de lazo abierto, el vector entrada es la señal tipo escalón que se le aplica a la planta y la salida es la señal proveniente del transductor de la misma.

   Salir de Interfaz: Opción que ejecuta el abandono del sistema.

   HERRAMIENTAS:

   Reducir Modelo: Opción que el usuario puede elegir para eliminar polos y/o ceros no dominantes, que se generan cuando existen factores externos como ruido y perturbaciones instantáneas que afectan la captura original de la planta. La reducción del sistema debe ser seleccionada antes de oprimir el botón “Analizar”

   Escala: Como se muestra en la figura 4.1, es una ventana donde se especifica el valor que multiplicará a cada unidad de voltaje representado en la captura, además de las unidades que representan, esto le permite al usuario una mejor visualización y comprensión de lo que se está analizando y/o controlando. Para introducir estos datos, el usuario deberá verificar la información que le proporcione el transductor de la planta. Por ejemplo si un sensor de un motor especifica que por cada voltio proporcionado representa “1000 Revoluciones por Minuto”, se especifica en el Factor de Escala “1000” y en Unidades “RPM”. Estos datos son mostrados en el eje “Y” del gráfico de la interfaz.

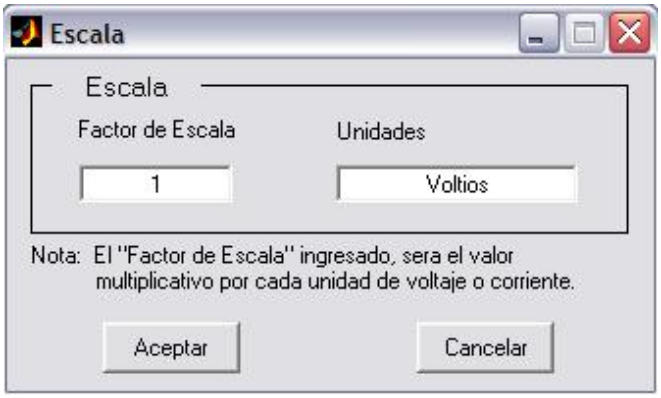


Figura 8.1. Ventana de captura de datos del factor de escala y las unidades.

Reiniciar: El hardware y el software de la PC/I son llevados a sus posiciones básicas. Esto corresponde al estado después del encendido del dispositivo. Todas las salidas son puestas a cero, los rangos de medición a $\pm$10V.

Avanzado: Si el número de muestras son muy pocas, el modelo probablemente no sea calculado o lo hará con poca precisión. Caso contrario, si las muestras son muchas el sistema puede capturar hasta el mas mínimo ruido indeseable de la planta y llevará una cantidad considerable de tiempo de procesamiento. Su valor por defecto recomendable son 500 muestras. Con respecto al tiempo de muestreo de lazo cerrado, si se disminuye mucho, puede que el tiempo de procesamiento de la PC tarde mas que esto, generando problemas, como que el sistema se bloquee y hasta cerrar la aplicación (MATLAB). Si el tiempo de muestreo es muy grande, esto puede hacer al sistema demasiado lento e impreciso, ya que entre muestras pueden suceder perturbaciones que el sistema no pueda detectar o responder eficazmente.

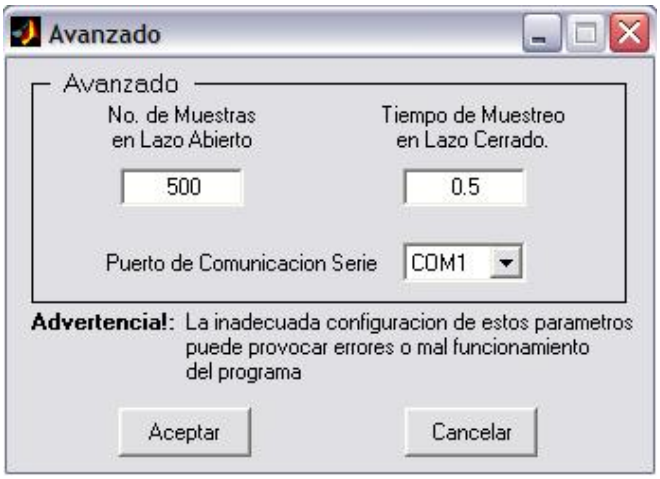


Figura 8.2. Ventana de opciones avanzadas.

Puede elegirse el Puerto de Comunicación Serie para la conexión entre la PC y la PCI por medio del menú desplegable.

AYUDA: Provee información acerca del funcionamiento y recomendaciones del manejo de la Interfaz Gráfica.

2. **RESULTADOS:**

Presenta información del análisis y determinación de parámetros PID.

¿Como obtener la función de Transferencia?: Presionar el botón "Analizar" y el proceso de identificación de la planta, capturada o guardada previamente, se ejecutará dando como resultado la ecuación que modela matemáticamente al sistema.

¿Como obtener lo Parámetros PID?: Estos parámetros del controlador PID (K_p = Ganancia Proporcional, T_i = Tiempo Integral y T_d = Tiempo Derivativo) pueden ser calculados con el botón "Determinar", siempre y cuando se haya creado previamente una función de transferencia, estos parámetros podrán ser utilizados en el control de Lazo Cerrado; o si ya se conocen, introducirlos directamente para proceder a dicho control.

3. **LAZO ABIERTO:** Después de haber conectado PC/I – PC y PCI – Sensores y Actuadores, energizado los sistemas se procede a establecer los parámetros que van regir la captura de información de la planta, el botón configurar muestra la siguiente ventana.

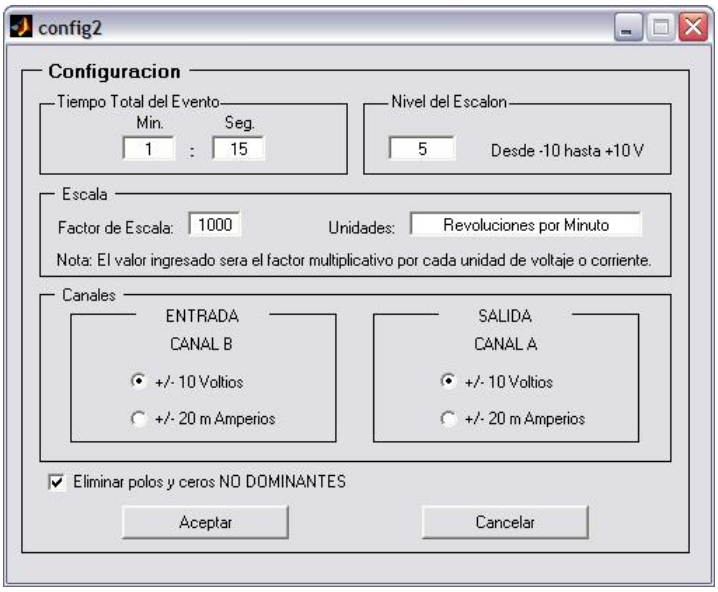


Figura 8.3. Ventana de configuración de parámetros que definen la captura.

Se recomienda realizar una captura de información inicial con un tiempo largo y nivel de escalón bajo, observar el tiempo que dure el transitorio y el nivel máximo de amplitud, luego repetir el experimento ajustando el tiempo de observación que llegue hasta un poco mas del inicio de la estabilidad, todo esto para determinar un valor apropiado de el *"Tiempo Total del Evento"*. Para el *"Nivel del Escalón"* elegir un voltaje de entrada de forma que la salida no se sature (10 V). Se definen también el *"Factor de Escala"*, las "*Unidades"* y el canal de captura y de salida que es el que proporciona el nivel del escalón. La opción *"Reducir Polos y Ceros No Dominantes"* es la misma función de reducir el modelo del menú Herramientas. Una vez definidos todos los parámetros, se procede a la captura de los datos presionando el botón *"Capturar"*.

4. **BOTÓN DETENER:** Ya sea en la captura en lazo abierto o en el control en lazo cerrado, se dispone de la opción de finalizar el proceso en curso.

5. **LAZO CERRADO:** Teniendo los parámetros PID, se procede al control de la planta, especificando el valor de *"Referencia"*, que es el valor a alcanzar por la planta, éste tiene la opción de ser negativa o positiva; por ejemplo, en un motor, un cambio de signo en este valor representa un cambio de giro. El botón *"Controlar"* inicia el proceso del control en lazo cerrado, durante este proceso se observa información importante como es la *"Variable Controlada"* y el "*Porcentaje de Error"* que es el valor actual de la planta y la diferencia de éste al valor de referencia, respectivamente.

6. **FUNCIÓN DE TRANSFERENCIA Y GRÁFICOS:** Si es necesario analizar, en esta área se encuentran las herramientas matemáticas indispensables en Control Automático como: gráficos y la función de transferencia, aplicables a los datos de la planta que han sido capturados por la PCI y procesados. El botón *"Analizar"* ejecuta los gráficos seleccionados y la función de transferencia. No se tendrá acceso si antes no se ha efectuado la captura de los datos o abierto un archivo guardado.

7. **ESTADO:** Para conocer las condiciones del sistema y dar indicaciones importantes al usuario de la ejecución del programa, se ha reservado un área denominada *"Estado"*.

8. **FILTRAR:** En los datos capturados, pueden existir señales indeseables como ruido; éste puede ser eliminado utilizando este filtro. El uso indebido de esta herramienta puede ocasionar cálculos erróneos. A continuación se detallan las recomendaciones para su buen uso:
    a. Realizar la captura de la planta.
    b. Ejecutar *"Analizar"* incluyendo el gráfico del Escalón.
    c. Comparar el gráfico del Escalón con el gráfico de los datos capturados.
    d. Si los gráficos concuerdan, este será el mejor modelo, si no,
    e. Aplicar el filtrado una vez y repetir paso b.

9. **BOTON SALIDA:** Si se desea abandonar la Interfaz Gráfica, presione este botón.

10. **VENTANA DE VISUALIZACIÓN.** El área donde se grafican los datos que se van adquiriendo tanto para lazo abierto como en lazo cerrado, con sus debidas unidades. En lazo abierto el eje del tiempo "X" es una ventana definida por el *"Tiempo Total del Evento"* mientras que en lazo cerrado es una ventana donde el eje "X" se refresca cada 250 segundos[1]; el eje "Y" es autoajustable para ambos casos.

SOLUCION A PROBLEMAS.

- El programa no responde cuando se ha oprimido el botón configurar o se trata de reiniciar:

  La PCI envía un mensaje de presentación en código ASCII cada vez que se enciende, este código puede crear conflictos con la interfaz. Si se tiene este problema, la forma de solucionarlo es que antes de correr la interfaz se descargue el código en la hiperterminal de windows, y no se vuelva a apagar la PCI, solo se desconecta de la hiperterminal y se corre la aplicación.

[1] Utilizando el tiempo de muestreo prefijado de 0.5 seg.

- El programa no responde o deja de responder cuando esta en lazo cerrado:
  Incrementar el tiempo de muestreo en la ventana de "Avanzado" del menú herramientas.

- Se produce un error al analizar :
  Incrementar el número de muestras antes de efectuar la captura, en la ventana de "Avanzado" del menú herramientas.

- Valores infinitos(inf) o no numéricos (Nan) o errores al determinar parámetros PID:
  Verificar que el escalón producido por el modelo vaya acorde con los datos de captura si no lo es varíe el filtro hasta encontrar uno que corresponda.

- Los parámetros determinados no corresponden a valores numericos(Nan) o aparecen valores infinitos(Inf):
  Estos valores corresponden a modelos no determinables, con alto componente de ruido o distorsión en la señal, para solucionarlo se manipulan los datos adquiridos con el filtro digital, y se grafica la respuesta escalon, si dicha gráfica es acorde con los datos adquiridos, se ha encontrado el modelo esperado y se podría determinar los parámetros siempre y cuando sea un modelo libre de integrador.

Anexo D

Diseño de la Planta de prueba. [9]

Con el objetivo de comprobar el funcionamiento de la Interfaz Gráfica se implementó electrónicamente un sistema conteniendo una función de transferencia de segundo orden de la forma:

$$\frac{\omega_n^2}{S^2 + 2\xi\omega_n S + \omega_n^2}$$

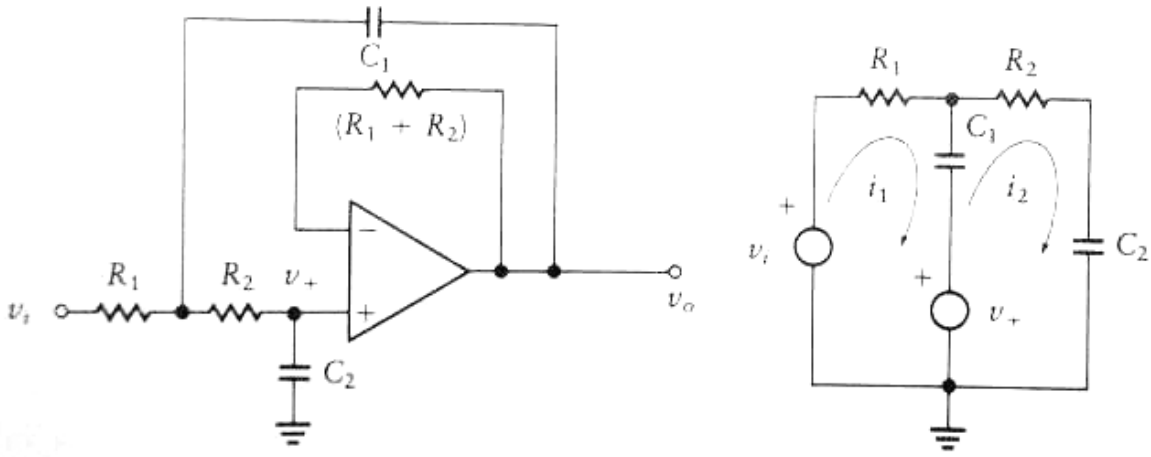
Con las siguientes características: $\xi = 0.25$, $\omega_n = 2.0$

Sustituyendo valores tenemos la función de la planta:

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

Diseño electrónico de la planta:

Es un circuito Pasa – Bajas de Segundo Orden:



Las ecuaciones de lazo están dadas por:

$$(R_1 + 1/sC_1)\, I_1 - (1/sC_1) = Vi - V+$$

$$(-1/sC_1) I_1 + (R_2 + 1/sC_1 + 1/sC_2)\, I_2 = V+$$

$$V+ = Vo = (1 / sC2)\, I2$$

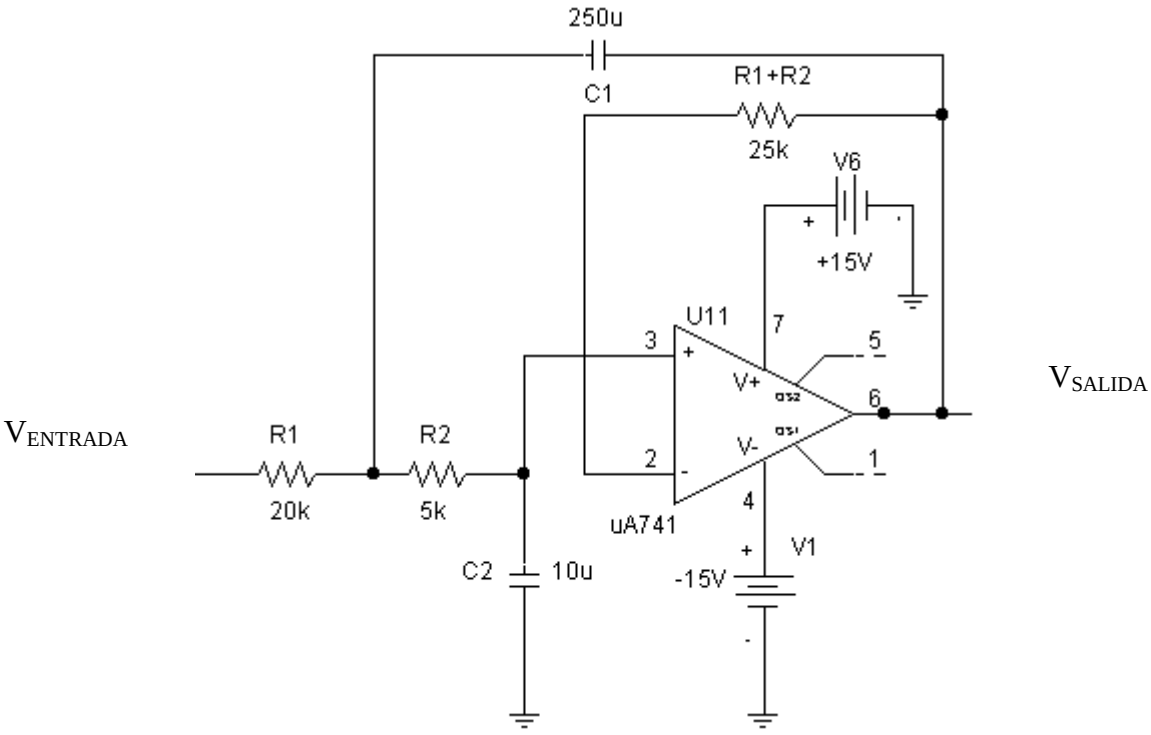
$$(R_1C_1s + 1)\, I_1 - I_2 = sC_1Vo$$

$$-C2\, I_1 + (R_2C_1C_2s + C_2 + C_1) = C_1C_2sVo$$

$$I_2 = sC_2Vo$$

Despejando Vo y eliminando I_1 e I_2 se obtiene la función de transferencia Vo / Vi:

$$\frac{V_{(SALIDA)}}{V_{(ENTRADA)}} = \frac{1}{(R_1 R_2 C_1 C_2) S^2 + (R_1 + R_2) C_2 S + 1}$$



De la función de transferencia del circuito, normalizamos la función de la planta:

$$G(s) = \frac{1}{0.25\ s^2 + 0.25\ s + 1}$$

Eligiendo $C_2 = 10$ uF en el termino: $(R_1 + R_2)C_2 = 0.25$

$$R_1 + R_2 = \frac{0.25}{100} \quad \textbf{à} \quad R_1 + R_2 = 25\ K$$

$R_1 = 20$ K $\qquad$ $R_2 = 5$ K

$$R_1 R_2 C_1 C_2 = (20K)(5K)(10uF) C_1 = 0.25$$

$$C_1 = \frac{0.25}{1000}$$

$C_1 = 250$ u F

Respuesta al escalón unitario de la planta a lazo abierto (teórico):

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

Nota: Debido a que no se encontró capacitor de 250 uF, sino dos, uno de 220 uF y otro de 33.3 uF; haciendo un equivalente de 253.3 uF y la respuesta al escalón unitario de la planta a lazo abierto se ajusta a:

$$\frac{V_{out}}{V_{in}} = Gteo = \frac{3.95}{s^2 + .987s + 3.95}$$ Ecuación Calculada.

Anexo E

Codigo Fuente.

```
function varargout = newpci5(varargin)

% Esta interfaz grafica es una herramienta que permite la obtencion del
% modelo como funcion de transferencia partiendo de datos adquiridos
% experimentalmente de una planta industrial, los datos se optienen con la
% ayuda de una PCI LM389 conectada al puerto serie. Ademas permite realizar
% los graficos: Escalon unitario, Diagrama de Bode, Lugar de las raices y
% Diagrama de Nyquist.
%
% UDB 2004

% Codigo de inicializacion GUI

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @newpci5_OpeningFcn, ...
                   'gui_OutputFcn',  @newpci5_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Final inicializacion GUI


% --- Se ejecuta antes de ser visible la GUI, Configuraciones iniciales
function newpci5_OpeningFcn(hObject, eventdata, handles, varargin)

   warning off
% Configura el puerto

global serie  % Variable con la estructura del puerto valida en varias funciones

serie = serial('COM1');         %Creacion de la estructura para manejo del puerto
set(serie,'StopBits',2,'InputBufferSize',1000);        % Configuracion del puerto

% Inicializacion de variables

handles.flag_ejecutar = 0;  % llave para evitar ejecutar sin configurar
handles.flag_reiniciar = 1; % llave en configuracion para reiniciar la primera vez
handles.flag_graficar = 0;  % llave para evitar graficar sin antes ejecutar
handles.salida = 0;         % Datos adquiridos
handles.entrada = 0;        % Vector conteniendo el nivel del escalon
handles.ts = 1;             % Tiempo de muestreo
handles.vref = 1;           % Voltaje de referencia
handles.pci = 1;            % Muestra el estado de la PCI
handles.esc = 1;            % Valor del escalamiento en la presentacion
handles.modoentb = 1;       % Modo del canal de entrada B +-10V o +-20 mA
handles.modosala = 1;       % Modo del canal de salida A +-10V o +-20 mA
handles.haydatos = 0;       % Actua como llave para poder guardar datos
handles.reduccion = 1;      % Reduccion de los polos y ceros no dominantes
```

```
   % Carga los datos de configuracion original para la interfaz configuracion

   % Guarda los datos de configuracion original en un archivo de uso
   % temporal
   % Informacion numerica: [ts,vref,escala,modoentb,modosala,elimina];

   handles.confinum = [15 1 1 1 1 1];

% Informacion de texto para las unidades
   handles.confiuni = 'Voltios'; % Unidades del eje y
% Mensaje en linea de estado
  set(handles.Texto_Estado,'String','Bienvenido...');

  handles.output = hObject;

% Actualiza la estructura del Handles
  guidata(hObject, handles);

% Funcion de Salida
  function varargout = newpci5_OutputFcn(hObject, eventdata, handles)
  varargout{1} = handles.output;

% Boton detener, suspende la captura actual de datos
  function Boton_Detener_Callback(hObject, eventdata, handles)
  global adquirir    % Variable de el lazo de captura y visualizacion
  adquirir = 0;      % Deshabilita la captura y visualizacion
% Actualiza la estructura del Handles
  guidata(hObject, handles);

% Boton Configuracion, Reinicia, Configura y Programa la PCI
function Boton_Confi_Callback(hObject, eventdata, handles)

% Limpia la Func de Transf y el grafico de captura
set(handles.Texto_TF,'Visible','off');
set(handles.Valor_TF,'String','');
plot(0);
axis([0 1 0 1]);
grid;

handles.flag_graficar = 0;          % Cierra el acceso a la funcion graficar

global serie;                       % Variable con la estructura del puerto serie

% Verifica estado del puerto, de estar cerrado se abrira

estado_puerto = serie.status;
if strcmp(estado_puerto,'closed') == 1
   fopen(serie);
end

% Limpia Presentacion al encender PCI
while (serie.BytesAvailable) ~= 0
     nada = fread(serie,serie.BytesAvailable);
end

% Solo la primera vez se reinicia la PCI
if handles.flag_reiniciar == 1
pci = reiniciar_Callback;
handles.pci = pci;                  % Llave que habilita poder configurar
end
% Estado de la PCI, 1: Hay comunicacion adecuada
if handles.pci == 1
```

```
handles.flag_reiniciar = 0;    % Se niega la llave para no reiniciar en lo sucesivo
set(handles.Texto_Estado,'String','Configure...');
% Recupera la configuracion
confinum = handles.confinum;
confiuni = handles.confiuni;
[valores,unidades] = config2(confinum,confiuni); % Abre la ventana de Configuración

tsi = valores(1);             % Tiempo total del experimento
vref = valores(2);            % Voltaje de referencia del escalon unitario
escala = valores(3);          % Valor de la escala
modoentb = valores(4);        % Modo del canal de entrada B
modosala = valores(5);        % Modo del canal de salida A
reduccion = valores(6);       % Eliminacion de los polos y ceros no dominantes
muestras = 500;               % Cantidad de muestras
handles.vref = vref;          % Voltaje de referencia del escalon unitario(handles)
handles.esc = escala;            % Valor de la escala
handles.unidades = unidades;     % Unidades de escala
handles.modosala = modosala;     % Handle del modo de canal a de salida
handles.modoentb = modoentb;     % Handle del modo de canal b de entrada
handles.reduccion = reduccion;   % Permite reducción del modelo

% Guarda la ultima configuracion
handles.confinum = valores;
handles.confiuni = unidades;

set(handles.Texto_Estado,'String','Espere mientras se configura...');
% Set Gain                         Ajusta rangos de medicion

% Configura el canal b +/- 10V
%if modoentb == 1
% fwrite(serie,38);                % Comando del modo
%fwrite(serie,12);                 % Canal B
%fwrite(serie,0);                  % Configura en voltaje
%reconocer_Callback                % Carácter de reconocimiento
%end

% Configura el canal b +/- 20 mA
% if modoentb == 2
% fwrite(serie,38);                % Comando del modo
% fwrite(serie,12);               % Canal B
% fwrite(serie,2);                 % Configura corriente
% reconocer_Callback               % Carácter de reconocimiento
% end

% Configura el canal a +/- 10V
  %if modoentb == 1
  %fwrite(serie,38);               % Comando del modo
  %fwrite(serie,0);                % Canal A
  %fwrite(serie,0);                % Configura en voltaje
  %reconocer_Callback              % Carácter de reconocimiento
  %end

% Configura el canal a +/- 20 mA
 % if modoentb == 2
 % fwrite(serie,38);               % Comando del modo
 % fwrite(serie,0);                % Canal A
 % fwrite(serie,2);                % Configura corriente
 % reconocer_Callback              % Carácter de reconocimiento
 % end
```

```
% ClrinBuffer                    % Borra el buffer y la memoria Xram de la PCI
  limpiar_Callback;              % Limpia el buffer de la PCI y MATLAB

% Borra Xram                     % La memoria Xram del PCI es borrada
  fwrite(serie,48);
  fwrite(serie,64);
  reconocer_Callback

% In Buffer:                     % Programa en la PCI Leer el Canal B
  fwrite(serie,52);
  fwrite(serie,12);
  reconocer_Callback

% XramDelay,
  ts = cdelay2(tsi,muestras);    % Configura el Delay en la PCI y calcula ts
                                 % ts = tiempo de muestreo

 % SetCycle:  Configura repeticiones periodo y modo entre ejecuciones
  ciclos = 1;                    % Una repeticion por cada ejecucion
  periodo = 0;                   % Maxima velocidad entre repeticiones
  fwrite(serie,39);
  fwrite(serie,ciclos,'uint16');            % No.de ciclos de med.
  fwrite(serie,0,'uint16');                 % Trigger
  if periodo ~= 0
      fwrite(serie,periodo,'uint16');       % Periodo del ciclo
      fwrite(serie,1);                      % Procesamiento
  else
      fwrite(serie,0,'uint16');             % Tiempo max
      fwrite(serie,0);                      % Procesamiento(tiempo max)
  end
  reconocer_Callback

  handles.flag_ejecutar = 1;                % habilita se pueda capturar

  handles.ts = ts;                          % Tiempo de muestreo

  set(handles.Texto_Estado,'String','PCI configurada, puede Capturar');

  end
  guidata(hObject, handles);                % Actualiza la estructura del Handles

% Define creacion del check box Bode
function Check_Bode_Callback(hObject, eventdata, handles)

% Define creacion del check box Nyquist
function Check_Nyquist_Callback(hObject, eventdata, handles)

% Define creacion del check Lugar de las Raices
function Check_Raiz_Callback(hObject, eventdata, handles)

% Define creacion del check box Escalon Unitario
function Check_Escalon_Callback(hObject, eventdata, handles)

% Realiza la identificacion del sistema, modelo en funcion de transferencia
% y analisis grafico: Bode, LGR, Nyquist, Escalon
function Boton_Analizar_Callback(hObject, eventdata, handles)

if handles.flag_graficar == 1;   % Llave ejecutarse primero

% Waitbar controla la presentación de barra de progreso
    h2 = waitbar(0,'Espere mientras se efectuan los calculos...');

set(handles.Texto_Estado,'String','Analizando datos...');
```

```
% Variables que contienen el resultado de la adquisicion
salida2 = handles.salida;       % Datos adquiridos de la planta, representan su
salida
entrada2 = handles.entrada;     % Datos de la senal aplicada al sistema en la
entrada
                                % Definidos por el nivel del escalon
ts2 = handles.ts;               % Tiempo de muestreo

% Convierte los datos de salida/entrada y tiempo de muestreo en una estructura
% denominada: "IDDATA
datos = iddata(salida2,entrada2,ts2);

waitbar(5/100)

% Devuelve el modelo estimado. En un objeto de tipo IDPOLY o IDSS junto con la
% covarianza y la información de la estructura.

set(handles.Texto_Estado,'String','Identificando el sistema...');
md = pem(datos);   % Convierte el modelo estimado.
waitbar(20/100)
set(handles.Texto_Estado,'String','Creando el modelo...');

% Convierte el modelo discreto a un equivalente continuo
set(handles.Texto_Estado,'String','Convirtiendo de discreto a continuo...');
mc = d2c(md);
waitbar(45/100)

% Transforma la estructura IDPOLY o IDSS del modelo a un equivalente en
% funcion de transferencia
set(handles.Texto_Estado,'String','Determinando Funcion de Transferencia...');
hms = tf(mc('m'));

% Permite la reduccion de polos y ceros no dominantes
if handles.reduccion == 1
    hms = reducir(hms);
end
 hms
waitbar(65/100)

% Extrae de la funcion de transferencia los coeficientes del numerador y
% denominador
[num,den] = tfdata(hms,'v');

% Transforma a texto la representacion de la funcion de transferencia
hstr = tf2string(num,den);

% Muestra la funcion de transferencia
set(handles.Texto_TF,'Visible','on');
set(handles.Valor_TF,'String',hstr);
waitbar(85/100)

% Graficas

% Grafica diagrama de Bode
if (get(handles.Check_Bode,'Value') == get(handles.Check_Bode,'Max'))
    set(handles.Texto_Estado,'String','Graficando diagrama de Bode...');
    figure
    bode(hms)
end

% Grafica Escalon unitario
if (get(handles.Check_Escalon,'Value') == get(handles.Check_Escalon,'Max'))
```

```
    set(handles.Texto_Estado,'String','Graficando Escalon unitario...');
    figure
    step(hms)

end

% Grafica Diagrama de Nyquist
if (get(handles.Check_Nyquist,'Value') == get(handles.Check_Nyquist,'Max'))
    set(handles.Texto_Estado,'String','Graficando diagrama de Nyquist...');
    figure
    nyquist(hms)

end

% Grafica Lugar de las raices
if (get(handles.Check_Raiz,'Value') == get(handles.Check_Raiz,'Max'))
    set(handles.Texto_Estado,'String','Graficando diagrama del lugar de las
raices...');
    figure
    rlocus(hms)

end
waitbar(100/100)
set(handles.Texto_Estado,'String','Analisis Terminado...');
delete(h2)

else   % Llave ejecutarse primero
       set(handles.Texto_Estado,'String','Primero debe Capturar');
end    % Llave ejecutarse primero


% Crea la caja de texto donde se mostrara la funcion de transferencia
function Valor_FT_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a la caja de texto para la Funcion de transferencia
function Valor_FT_Callback(hObject, eventdata, handles)

% Crea la caja de texto donde se mostrara Valor de ganancia
function Valor_Ganancia_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a funcion de la caja de texto donde se mostrara Valor de ganancia
function Valor_Ganancia_Callback(hObject, eventdata, handles)

% Crea la caja de texto donde se mostrara el tiempo TD
function Valor_Td_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a funcion de la caja de texto donde se mostrara tiempo TD
```

```
function Valor_Td_Callback(hObject, eventdata, handles)

% Crea la caja de texto donde se mostrara el tiempo Ti
function Valor_Ti_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a funcion de la caja de texto donde se mostrara tiempo TD
function Valor_Ti_Callback(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Funcion reservada a figure1
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)

% Cierra el puerto serie, borra variables, cierra todas las figuras
% incluyendo la Interfaz
function Boton_Salida_Callback(hObject, eventdata, handles)

global serie   % Variable de tipo estructura para la utilizacion del puerto
estado_puerto = serie.status; % Determina si el puerto esta cerrado o abierto
if strcmp(estado_puerto,'open')   == 1
   fclose(serie);             % Cierra el puerto
end

delete(serie);                % Borra la estructura del puerto serie
clear global;                 % Elimina todas las variables globales
close all                     % Cierra la interfaz y sus graficas

% Reservado
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Reservado
function edit5_Callback(hObject, eventdata, handles)
% Crea la caja de texto donde se mostrara el estado de la interfaz
function Texto_Estado_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a funcion de la caja de texto donde se mostrara el estado de la interfaz
function Texto_Estado_Callback(hObject, eventdata, handles)

% Llamada a funcion del menu archivo
function Archivo_Callback(hObject, eventdata, handles)

% Llamada a funcion del menu abrir
function Abrir_Callback(hObject, eventdata, handles)

[archivo,ruta] = uigetfile('*.mat','Escoga un archivo');
```

```
% Convierte a string posibles ceros cuando se cancela la accion
archivo = num2str(archivo);
ruta = num2str(ruta);
if ( strcmp(ruta,'0')~=1) & ( strcmp(archivo,'0')~=1) % Garantiza que exista ruta y
archivo

    buscarchivo = strcat(ruta,archivo);     % Obtiene el nombre completo
    load (buscarchivo)
if (exist('entrada')==1) & (exist('salida')==1) & (exist('tiempo')==1)
     plot(tiempo,salida)             % Grafica valores guardados
     grid
     handles.ts = tiempo(2);         % Guarda el handlet del tiempo de muestreo
     handles.tiempo = tiempo;
     handles.entrada = entrada;      % Guarda el handlet con los valores de entrada
     handles.flag_graficar = 1;      % Valida la entrada al analisis de datos
     handles.salida = salida;        % Guarda el handlet de la salida
     handles.flag_ejecutar = 0;      % evitar ejecutar sin antes configurar
     handles.haydatos = 1;           % Valida que pueda guardarse la informacion
     set(handles.Texto_Estado,'String','Modo Simulacion, Datos Cargados, Puede...
     analizar');

else
    errordlg('Formato de archivo no valido','Error');
end

end                                  % End de: garantiza que exista ruta y archivo
  % Actualiza la estructura del Handles
    guidata(hObject, handles);

% Llamada a funcion del menu guardar
function Guardar_Callback(hObject, eventdata, handles)

if handles.haydatos == 1                    % Garantiza guardar datos validos

     tiempo = handles.tiempo;        % Optiene el vector tiempo
     entrada = handles.entrada;      % Obtiene vector entrada
     salida =  handles.salida;       % Obtiene vector salida
     set(handles.Texto_Estado,'String','Guardando archivo sin nombre');
     save('sinnombre','entrada','salida','tiempo') % Guardando archivos
     pause(0.5)
     set(handles.Texto_Estado,'String','');
else                              % No hay datos validos
    errordlg('No existen datos que guardar','Error');
end                               % End de: garantiza guardar datos validos

% Actualiza la estructura del Handles
    guidata(hObject, handles);

% Llamada a funcion del menu Salir
function Salir_Callback(hObject, eventdata, handles)

global serie   % Variable de tipo estructura para la utilizacion del puerto
estado_puerto = serie.status;    % Determina si el puerto esta cerrado o abierto
if strcmp(estado_puerto,'open')   == 1
   fclose(serie);                % Cierra el puerto
end

delete(serie);                % Borra la estructura del puerto serie
clear global;                 % Elimina todas las variables globales
close all                     % Cierra la interfaz y sus graficas

% Llamada a funcion del menu guardar como
```

```
function Guar_com_Callback(hObject, eventdata, handles)

if handles.haydatos == 1                  % Garantiza guardar datos validos

[archivo,ruta] = uiputfile('*.mat','Escriba un Nombre');

% Convierte a string posibles ceros cuando se cancela la accion
archivo = num2str(archivo);
ruta = num2str(ruta);
if ( strcmp(ruta,'0')~=1) & ( strcmp(archivo,'0')~=1) % Garantiza que exista ruta y
archivo

     buscarchivo = strcat(ruta,archivo);    % Obtiene el nombre completo

     tiempo = handles.tiempo;               % Optiene el vector tiempo
     entrada = handles.entrada;             % Obtiene vector entrada
     salida =  handles.salida;              % Obtiene vector salida
     set(handles.Texto_Estado,'String','Guardando');
     save(buscarchivo,'entrada','salida','tiempo')  % Guardando archivos
     pause(0.5)
     set(handles.Texto_Estado,'String','');

end                             % End de: garantiza que exista ruta y archivo

else                                        % No hay datos validos
    errordlg('No existen datos que guardar','Error');
end                             % End de: garantiza guardar datos validos

% Actualiza la estructura del Handles
    guidata(hObject, handles);


% Ejecuta programa en Xram(realiza la captura), grafica la captura, guarda
% variables
function Boton_Capturar_Callback(hObject, eventdata, handles)

% Entra si esta configurado
if handles.flag_ejecutar ==1    % Llave: Configurar Primero
% Limpia la Func de Transf
    set(handles.Texto_TF,'Visible','off');
    set(handles.Valor_TF,'String','');
    global serie                % Variable con la estructura del puerto serie
    global adquirir             % Variable que inicia y/o detiene la captura

% Verifica estado del puerto, de estar cerrado se abrira
estado_puerto = serie.status;
if strcmp(estado_puerto,'closed') == 1
   fopen(serie);
end
limpiar_Callback; % Limpia el buffer de la PCI y MATLAB
% Da el nivel al escalon
  vref1 = handles.vref;       % Handlet con el valor del escalon
  vref2 = consala(vref1,handles.modosala); % Rutina que convierte a valores
                                           % compatibles con la PCI
  fwrite(serie,80);           % Comando de escritura en el canal A
  fwrite(serie,vref2);        % Parametro con el dato

% Run
  fwrite(serie,37);           % Comando de ejecucion(Run)
  fwrite(serie,3);            % Parametro con el modo 3 de ejecución
  nada = fread(serie,1);      % Caracter de reconocimiento
  muestras = 500;             % No. de muestras
  set(handles.Texto_Estado,'String','Capturando...');
```

```
  contador = 0;              % Lleva la cuenta de la cantidad de datos ingresados
  adquirir = 1;              % Variable que inicia y/o detiene la captura
  temporal = 0;              % Se carga con el contenido del buffer de entrada
  salida = 0;                % Se van almacenando los datos adquiridos
  ts1 = handles.ts;          % Tiempo de muestreo

% Lazo de captura y presentacion
while adquirir == 1
       if (serie.bytesavailable)/2 ~= 0  % Entra si existen datos en el buffer
            if (serie.bytesavailable)/2 >=500 % Se sale del lazo al llegar a 500
                fwrite(serie,27);            % Detiene la PCI
                nada = fread(serie,1);       % Caracter de reconocimiento

                adquirir = 0;                % Niega la bandera para salirse del
                %lazo
                temporal = fread(serie,500,'uint16'); %Trunca el vector a 500

            else        % Captura si hay menos de 500 muestras
            temporal = fread(serie,(serie.bytesavailable)/2,'uint16'); % actuales
            end
            % Procedimiento "Bufferear" los datos
            vector = conventb(temporal,handles.modoentb);  % Rutina, convierte
            % datos de PCI a datos Numericos

            lvector = length(vector); % determina longitud vector de datos actuales

            lsalida = length(salida); % determina longitud vector de datos totales

            salida(lsalida + 1 : lvector + lsalida)=vector; % Agrega nuevos
                                                            % elementos
            contador = length(salida);  % Determina cuantos datos existen en este
                                        % instante

            if contador >=500           % Se sale del lazo al llegar a 500 muestras
                 fwrite(serie,27);      % Detiene la captura de la PCI
                 nada = fread(serie,1); % Reconocimiento
                 adquirir = 0;          % Niega la bandera para salirse del lazo

                 tempo = salida(1:500); % Trunca los datos a 500 muestras
                 salida = tempo;        % Finaliza el truncamiento

            end

            % Grafica los datos adquiridos
            pause(0.001);               % Espera, para una correcta visualizacion
                                        % de los datos
            tiempo = 0:ts1:ts1*(length(salida))-ts1; % Crea el vector del tiempo
                                                     % para el eje x
            escsalida = salida.* handles.esc; % Datos multiplicados por factor de
                                              % escala
            plot(tiempo,escsalida)            % Grafica datos escaleados respecto
                                              % al tiempo
            axis([0 (500*ts1) min(escsalida) (max(escsalida)*1.1)]); % Definine los
                                                                     % de lod ejes
            grid % Muestra rejilla
            %nstr = num2str(contador);     % Opcion si se desea mostrar No.muestras
            %title(nstr);
            %nstr = num2str(salida(length(salida)));
            ylabel(handles.unidades);      % Unidades en el eje y
            xlabel('Segundos');            % Unidades de tiempo para eje x

       end
 end
```

```
   fwrite(serie,27);                        % Detiene la captura de la PCI
   nada = fread(serie,1);                   % Caracter de reconocimiento

  %Quita el nivel del escalon
   fwrite(serie,80);                         % Escribir en canal A
   fwrite(serie,0);                          % Poner cero a la salida

  limpiar_Callback; % Limpia el buffer de la PCI y MATLAB

     fclose(serie);                         % Cierra el puerto serie
     longi = length(salida);                % Determina la longitud del tamano de
                                            % los datos
     entrada = ones(longi,1);               % Crea el vector de entrada con unos
     entrada(1)=0;                          % Se normaliza el primer dato con cero
     entrada = entrada.*vref1;              % Se multiplica con el valor del
escalon
     handles.ts = ts1;                      % Guarda el handlet del tiempo de
muestreo
     handles.entrada = entrada;             % Guarda el handlet con los valores de
entrada
     handles.flag_graficar = 1;             % Valida la entrada al analisis de
datos(llave capturar)

     salida = salida';                      % Coloca en vector columna la salida
     tiempo = tiempo';                      % Coloca en vector columna el tiempo
     handles.tiempo = tiempo;               % Guarda el handlet de el tiempo
     handles.salida = salida;               % Guarda el handlet de la salida
     handles.haydatos = 1;                  % Valida que pueda guardarse la
                                            % informacion
     set(handles.Texto_Estado,'String','Captura finalizada');

else  % Llave: Configurar Primero
     set(handles.Texto_Estado,'String','Debe configurar primero');

end   % Llave: Configurar Primero

% Update handles structure
  guidata(hObject, handles);

% Llamada a la funcion conexion
function Conexion_Callback(hObject, eventdata, handles)

% Llamada a la funcion conectar
function Conectar_Callback(hObject, eventdata, handles)

% Llamada a la funcion desconectar
function Desconec_Callback(hObject, eventdata, handles)

% Llamada a la funcion Controlador
function Controlador_Callback(hObject, eventdata, handles)

% Llamada a la funcion ayuda
function Ayuda_Callback(hObject, eventdata, handles)

% Abre la Pagina de ayuda
function Contenido_Callback(hObject, eventdata, handles)

% Abre la pagina de ayuda
web('C:\MATLAB6p5\work\ayuda.html','browser');

% Llamada a la funcion Pendient
```

```
function Pendient_Callback(hObject, eventdata, handles)

% Llamada a la funcion Acerca de del menu help
function Acerca_Callback(hObject, eventdata, handles)

% Abre la pagina de ayuda
% web('C:\MATLAB6p5\work\acerca.html');

% Revisa la condicion del caracter de reconocimiento
function reconocer_Callback(hObject, eventdata, handles)
 global serie                                % Variable con la estructura del
puerto serie
 handles = guidata(gcbo);                    % Carga los handlets
     data_adqui = fread(serie,1);            % Carga el caracter de
reconocimiento
       if data_adqui ~= 33                   % Compara con el valor del caracter
              set(handles.Texto_Estado,'String','Error de reconocimiento,espere,...
se reiniciara la PCI');
              pause(0.5)
            pci = reiniciar_Callback;        % Reiniciar si hubo error
        end

% Reinicia La PCI
 function y = reiniciar_Callback(hObject, eventdata, handles)

 handles = guidata(gcbo);                    % Carga los handlets
 global serie                                % Variable con la estructura del
                                             % puerto serie
 % Verifica estado del puerto, de estar cerrado se abrira
 estado_puerto = serie.status;
 if strcmp(estado_puerto,'closed') == 1
   fopen(serie);
 end

 fwrite(serie,3);                            % Comando que reinicia la PCI
   set(handles.Texto_Estado,'String','Espere, se reiniciara la PCI...');
 pause(5)
     data_adqui = fread(serie,1);
       if data_adqui == 33                   % Espera caracter de reconocimiento

              set(handles.Texto_Estado,'String','Aceptado...'); % Si no hubo
                                                                % errores
              pause(0.7)
              set(handles.Texto_Estado,'String','');
              y = 1;                         % Habilitar poder configurar
        else                                 % Si hubieron errores
        set(handles.Texto_Estado,'String','Error, No hay comunicacion con PCI');
        y = 0;                               % Deshabilita configurar
        errordlg('Verifique la conexion o si la PCI esta encendida','Error');
        end

 pause(0.01);
 % Limpiar el buffer de Matlab
 while serie.BytesAvailable ~= 0
     data_adqui = fread(serie,serie.BytesAvailable);
 end

 % Limpia el buffer del adquisidor y de Matlab
function limpiar_Callback(hObject, eventdata, handles)
 handles = guidata(gcbo);                    % Carga los handlets
 global serie                                % Variable con la estructura del
                                             % puerto serie
 % ClrinBuffer
```

```
   % Borra Buffer
   fwrite(serie,48);                          % Comando limpiar Buffer PCI
   fwrite(serie,12);                          % Canal B
   nada = fread(serie,1);                     % Caracter de reconocimiento

% Limpiar el buffer de Matlab
   while (serie.BytesAvailable) ~= 0
     nada = fread(serie,serie.BytesAvailable);
   end

% Llamada al menu PCI
function Pci_Callback(hObject, eventdata, handles)

% Reinicia la PCI
function Reini_Callback(hObject, eventdata, handles)

pci = reiniciar_Callback;          % Reiniciar la PCI
guidata(hObject, handles);         % Actualiza la estructura del Handles


% Permite escoger la reduccion o no de los polos y ceros no dominantes
function redusim_Callback(hObject, eventdata, handles)

% Verifica la condicion del cheque en el submenu reducir del menu PCI
if strcmp(get(handles.redusim,'checked'),'on')
   set(handles.redusim,'checked','off')
   handles.reduccion = 0;      % No permite la reduccion de los polos y ceros no
                               % dominantes
else
    set(handles.redusim,'checked','on')
    handles.reduccion = 1;     % Reduccion de los polos y ceros no dominantes
end

guidata(hObject, handles);     % Actualiza la estructura del Handles



function varargout = config2(varargin)
% Esta interfaz se utiliza en conjunto con newpci4, su funcion es la de
% ventana de configuracion.
% Posee igual numero de argumentos de entrada como de salida de la
% siguiente forma: [D,U] = config2(D,U) donde: D es un vector con datos
% numericos con 5 elementos que corresponden en su orden a: tiempo total de evento,
% nivel del escalon, factor de escala, modo del canal de entrada B, modo
% del canal de salida A, eliminacion de polos y ceros no dominantes y U es
% una variable de texto(String) asignada para las unidades
%
% UDB 2004

% Codigo de inicializacion
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @config2_OpeningFcn, ...
                   'gui_OutputFcn',  @config2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de la inicializacion


% Carga los parametro de configuracion y asigna valores almacenados
function config2_OpeningFcn(hObject, eventdata, handles, varargin)
% Carga de parametros
dato = varargin{1};                % Carga el primer argumento de entrada de
                                   % valores numericos

% Extrae del vector datos todos los parametros numericos de configuracion
tsi = dato(1);                     % Tiempo total del evento
vref = dato(2);                    % Nivel del escalon
escala = dato(3);                  % Factor de escala
modoentb = dato(4);                % Modo canal entrada B
modosala = dato(5);                % Modo canal de salida A
elimina = dato(6);                 % Eliminacion de polos y ceros No Domi

% Extrae del segundo argumento de entrada el valor de texto (String)
unidades2 = varargin{2};           % Unidades
% arregla el tiempo total en minuto y segundos
minutos = floor(tsi/60);
segundos = mod(tsi,60);

set(handles.minutos,'String',minutos);         % Valores guardados para minutos
set(handles.segundos,'String',segundos);       % Valores guardados para segundos
set(handles.escalon,'String',vref);            % Valores guardados para nivel del
escalon
set(handles.escala,'String',escala);           % Valores guardados para factor de
escala
set(handles.unidades,'String',unidades2);      % Valores guardados para las
                                               % unidades

% Modo del canal de entrada B, Voltaje/Corriente
if modoentb == 1
    set(handles.in_volt,'value',1);           % Selecciona Voltaje
    set(handles.in_amp,'value',0);
end

if modoentb == 2
    set(handles.in_volt,'value',0);           % Selecciona Corriente
    set(handles.in_amp,'value',1);
end

% Modo del canal de Salida a, Voltaje/Corriente
if modosala == 1
    set(handles.out_volt,'value',1);          % Selecciona Voltaje
    set(handles.out_amp,'value',0);
end

if modosala == 2
    set(handles.out_volt,'value',0);          % Selecciona Corriente
    set(handles.out_amp,'value',1);
end

% Reducccion de polos y ceros no dominantes
if elimina == 1
    set(handles.elimina,'value',1);           % Reducira
else
     set(handles.elimina,'value',0);          % No reducira
end
```

```
handles.dato = dato;                          % Almacena en handle el vector
numerico
handles.unidades2 = unidades2;                % Almacena en handle el vector de
                                              % unidades


% Guarda  la salida de la GUI
handles.output = hObject;

% Actualiza estructura de handle
guidata(hObject, handles);

uiwait; % Detiene el proceso de la GUI que hizo la llamada

% Funcion de salida de config2, valores que retornan
function varargout = config2_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;      % Se almacenan las variables numericas
varargout{2} = handles.salida;      % Se almacena la variable tipo String
close;                              % Cierra la ventada Config

% Crea la funcion de caja de texto segundos
function segundos_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a la funcion de caja de texto segundos
function segundos_Callback(hObject, eventdata, handles)

% Crea la funcion de caja de texto minutos
function minutos_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a la funcion de caja de texto minutos
function minutos_Callback(hObject, eventdata, handles)

% Crea la funcion de caja de texto nivel del escalon
function escalon_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a la funcion de caja de texto nivel del escalon
function escalon_Callback(hObject, eventdata, handles)

% Crea la funcion de caja de texto factor de escala
function escala_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a la funcion de caja de texto factor de escala
function escala_Callback(hObject, eventdata, handles)

% Crea la funcion de caja de texto unidades
function unidades_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% Llamada a la funcion de caja de texto unidades
function unidades_Callback(hObject, eventdata, handles)

% boton de seleccion excluyente, Voltaje de entrada canal B
function in_volt_Callback(hObject, eventdata, handles)

set(handles.in_amp,'value',0);  % Deselecciona (excluye) boton amperios del canal B
set(handles.unidades,'String','Voltios'); % Escribe en unidades "Voltios"
guidata(hObject, handles);  % Guarda el handle

% boton de seleccion excluyente, Corriente de entrada Canal B
function in_amp_Callback(hObject, eventdata, handles)

set(handles.in_volt,'value',0);  % Deselecciona (excluye) boton Voltios del canal B
set(handles.unidades,'String','miliAmperios'); % Escribe en unidades "miliAmperios"
guidata(hObject, handles);  %Guarda el handle

% boton de seleccion excluyente, Voltaje de Salida canal A
function out_volt_Callback(hObject, eventdata, handles)

set(handles.out_amp,'value',0); % Deselecciona (excluye) boton amperios del canal A
guidata(hObject, handles);  % Guarda el handle

% boton de seleccion excluyente, Corriente de salida canal A
function out_amp_Callback(hObject, eventdata, handles)

set(handles.out_volt,'value',0); % Deselecciona (excluye) boton Voltios del canal A
guidata(hObject, handles);  % Guarda el handle

% "Cheque" de eliminar polos y ceros no dominantes
function elimina_Callback(hObject, eventdata, handles)

% Obtiene y guarda los parametros escogidos en la interfaz
function aceptar_Callback(hObject, eventdata, handles)

minutos = get(handles.minutos,'String');         % Obtiene parametro minutos
segundos = get(handles.segundos,'String');       % Obtiene parametro segundos
vref = get(handles.escalon,'String');            % Obtiene parametro nivel del
escalon
escala = get(handles.escala,'String');           % Obtiene parametro factor de
                                                 % escala

% Los datos obtenidos tienen formato de texto, los codigos siguientes
% convierten cada parametro en su equivalente numerico
minutos = str2double(minutos);
segundos = str2double(segundos);
vref = str2double(vref);
escala = str2double(escala);
```

```
% Tiempo total del evento expresado en segundos
tsi = (minutos*60) + segundos;

% Obtiene el modo del canal de entrada B
if (get(handles.in_volt,'Value') == get(handles.in_volt,'Max'))
    modoentb = 1;                % Coloca 1 si se selecciono Voltios
else
    modoentb = 2;                % Coloca 0 si se selecciono miliAmperios

end
% Obtiene el modo del canal de salida A
if (get(handles.out_volt,'Value') == get(handles.out_volt,'Max'))
    modosala = 1;               % Coloca 1 si se selecciono Voltios
else
    modosala = 2;               % Coloca 0 si se selecciono miliAmperios
end

% Obtiene condicion de eliminar polos y ceros No Dominantes
if (get(handles.elimina,'Value') == get(handles.elimina,'Max'))
    elimina = 1;                % Eliminar polos y ceros No Dominantes
else
    elimina = 0;                % No eliminar polos y ceros No Dominantes
end

unidades2 = get(handles.unidades,'String');    % Obtiene el valor de las unidades

% Guarda la configuracion actual de manera que la proxima vez que se abra
% la ventana esta mostrara los parametros anteriores

% Guarda en el vector "dato' todos los parametros numericos
dato(1)=tsi;                        % Tiempo total del evento
dato(2)=vref;                       % Nivel del escalon
dato(3)=escala;                     % Factor de escala
dato(4)=modoentb;                   % Modo canal entrada B
dato(5)=modosala;                   % Modo canal de salida A
dato(6)=elimina;                    % Eliminacion de polos y ceros No Domi
% Transfiere el vector al handle de salida, para retornarse como argumentos
% de salida
handles.output = dato;
% Transfiere la variable de texto unidades al hanle de salida, para retornarse como
% argumentos de salida
handles.salida = unidades2;
% Guarda el handle
guidata(hObject, handles);
% Le devuelve el control a la GUI que esta es espera de los datos
uiresume;

% Boton Cancelar, Carga la ultima configuracion la guarda y pasa los datos
% al vector de salida y a la variable de salida tipo texto
% "Ningun cambio hecho en la GUI sera almacenado y/o transferido
function cancelar_Callback(hObject, eventdata, handles)

% Carga la ultima configuracion
dato = handles.dato;
unidades2 = handles.unidades2;
% Transfiere el vector al handle de salida, para retornarse como argumentos
% de salida
handles.output = dato;
% Transfiere la variable de texto unidades al handle de salida, para retornarse
% como argumentos de salida
handles.salida = unidades2;
% Guarda el handle
```

```
guidata(hObject, handles);
% Le devuelve el control a la GUI que esta es espera de los datos
uiresume;


function ts = cdelay2(tfinal,muestras)

% Esta funcion ts = cdelay2(tfinal,muestras) calcula y programa la cantidad de
% retardos(XramDelay) que deben agregarse a la Xram para poder obter el
% tiempo de muestro necesario.
% "tfinal" es el tiempo total del evento, "muestras" es el numero de muestras
% que satisfacen al tiempo total, ts es el tiempo de muestreo que satisface
% las condiciones del experimento
%
% UDB 2004

global serie                        % Variable con la estructura del puerto serie
paso = tfinal/muestras;             % Resolucion inicial
retardo = paso/10e-6;               % Compatibilidad con XramDelay (multiplo de
10us)
contador = 1;                       % Numero de retardos a programarse
bandera_lazo = 1;                   % Control del lazo

% Calcula la cantidad de veces a repetir los retardos
% El parametro con que se programa el retardo es un valor multiplo de 10us
% y este valor es como maximo 32767 es decir 0.32767 s, tiempos de muestreos
% inferiores a este valor solo se emplea 1 retardo. Cuando este tiempo es
% mayor sera necesario valores tales que: tr*n = ts, donde n: cantidad de
% retardos, tr: tiempo de cada retardo, ts: tiempo de muestreo.
% Para simplificar el calculo no se hara en segundos, sino en
% multiplos de 10us que entrega el valor del parametro.

while bandera_lazo == 1
    pararetardo = retardo/contador;
    % Al dividirse por el Kesimo retardo, su cociente sera mayor que 32767 ?
    % De serlo es que necesita dividirse mas, es decir faltan retardos
    if pararetardo > 32767
        contador = contador + 1;   % debe agregarse otro retardo
        % Cuando paradelay < 32767 inplica que ya no hay que agregar mas
        % retardos, de manera que el valor del contardor contiene la
        % cantidad de retardos y paradelay el valor que lo programa
    else
        bandera_lazo = 0;               % Se sale del lazo
        pararetardo = ceil(pararetardo); % Se aproxima al entero superior
    end
end

% Se procede a programar la Xram con los retardos
for k=1:contador
      fwrite(serie,40);                  % Agregar delay a Xram
      fwrite(serie,pararetardo,'uint16'); % Valor del Delay
      a = fread(serie,1);                % Caracter de reconocimiento
end

% El tiempo de muestreo viene dado por el parametro del retardo por la cantidad
% de ellos por 10 us

ts = (pararetardo*contador)*(10e-6);


function y = tf2string(num,den)

% Esta funcion y = tf2string(num,den) da como resultado una representacion de
```

```
% funcion de transferencia como cadena de caracteres(String), los valores de
% num,den son vectores con los coeficientes del numerador y el denominador de la
% funcion de transferencia que se desea representar.
%
% UDB 2004

numstr = cambiotf_st(num);     % Transforma num en un polinomio en funcion de s
denstr = cambiotf_st(den);     % Transforma den en un polinomio en funcion de s

% Concatena numstr y denstr con el operador de division y parentesis de
% manera que y representa a la funcion de transferencia
y = strcat('(',numstr,')',' / ',' (',denstr,')');

function z = cambiotf_st(x)
% La funcion z = cambiotf_st(x) transforma el vector x como un polinomio en
% terminos de s dicho polinomio se construye como una cadena de
% caracteres(String)

bandera = 0;                        % Llave, control de flujo
longi = length(x);                  % Longitud de x
    z = '';                         % z se inicia con vacio
for k = 1:longi                     % El lazo se repite k veces la longitud del
vector

    temp = x(k);                    % Carga en temp el kesimo valor del
                                    % vector(coeficiente)
    expo = longi - k;               % Calcula el exponente(orden) del polinomio
    expostr = num2str(expo);        % Se convierte a String
    elevar = '^';                   % Define el simbolo de la potencia
    ese = 's';                      % Define la variable s

    % Si el exponente es menor o igual que uno entonces dicho valor no se
    % muestra por lo tanto tampoco el simbolo de la potencia
    if expo <=1
        expostr = '';               % Queda vacio
        elevar = '';                % Queda vacio
end


    if temp >= 0
        signo = '+';          % Si el coeficiente es >=0 el signo sera +
    else
        signo = '-';          % Si el coeficiente es < 0 el signo sera -
    end

    % El primer termino del polinomio no lleva signo si es positivo
    if k == 1 & strcmp(signo,'+')
        signo = '';
    end

    % si el valor del coeficiente es uno o cero no se escribe
    if temp == 1 | temp == 0
        coef = '';
        % si el valor del coeficiente es distinto a cero o a uno se escribe
        % como tipo string
    else
        coef = num2str(temp);       % Se convierte el coeficiente a String
    end

    % Si el coeficiente es cero no se escribe la variable, el simbolo de
    % potencia, el exponente ni el signo
    if temp == 0
        ese = '';                   % Queda vacio
```

```
        elevar = '';                % Queda vacio
        expostr = '';               % Queda vacio
        signo = '';                 % Queda vacio
    end

    % Si el exponente es cero no se escribe la variable pero si el
    % coeficiente
    if expo == 0
        ese = '';
        coef = num2str(temp);
    end


    % Antes de concatenar debe verificarse que no todas esten vacias
    if   (isempty(signo)== 0) | (isempty(coef)== 0) | (isempty(ese)== 0) |...
 (isempty(elevar)== 0) | (isempty(expostr)== 0)
        if bandera == 0      % El primer valor no lleva signo
            signo ='';
        end
        % Se concatenan todas las variables
         datastr=strcat(signo,coef,ese,elevar,expostr);
        % Se concatena con el total(Vector completo)
         z=strcat(z,datastr);
         bandera = 1; % Se deshabilita que borre el signo
    end

end



function y = reducir(hs)
% Esta funcion y = reducir(hs), permite la reduccion de polos y ceros no dominantes
%
% UDB 2004

[num,den] = tfdata(hs,'v');         % Se obtiene el numerador y denominador de la
Func de Transf
longi = length(den);                % Longitud de el denominador

% Se optiene el cociente de los terminos S^0 en la funcion original
r1 = num(longi)/den(longi);
[cero,ganancia] = zero(hs);         % Se optiene informacion de los ceros y la
                                    % ganancia
polo = pole(hs);                    % Se optiene informacion de los polos
longpolo = length(polo);            % Dimension del polo

% Se determinara cual es el polo con el valor absoluto mas pequeno ya que
% este polo sera el que se encuentre mas cercano al origen
polomin = polo(1);                  % Inicia con la primer posicion del vector
polominreal = real(polomin);        % Extrae unicamente la parte real
for k = 1:longpolo
    menor = abs(real(polo(k)));     % Intereza la magnitud y parte real

% Se compara con cada uno de los polos al encontrarse uno menor este remplaza al
% anterior
    if menor < (abs(polominreal))
        polominreal = real(polo(k));% Sustituye al anterior por ser menor
    end
end

poloreduc = compapolo(polo,polominreal);   % Rutina que reduce el polo
ceroreduc = compapolo(cero,polominreal);   % Rutina que reduce el cero
```

```
% Se construye una funcion de tipo zeros/polos/ganancia con los polos y ceros
% reducidos
% Este es un resultado parcial ya que aun falta determinar la ganancia
hredu = zpk(ceroreduc,poloreduc,1);

% Se le extraen el numerador y denominador a la funcion reducida parcial
[numredu,denredu] = tfdata(hredu,'v');
longiredu = length(denredu);        % Longitud del denominador reducido

% Se optiene el cociente de los terminos S^0 en la funcion reducida parcial
r2 = numredu(longiredu)/denredu(longiredu);

% Se obtiene la nueva ganancia para la funcion reducida final
ganancia = r1 / r2;

% Se construye la funcion tipo zeros/polos/ganancia reducida final
hzpk = zpk(ceroreduc,poloreduc,ganancia);

% La expresion se traslado al formato de funcion de transferencia
htf = tf(hzpk);
y = htf;   % La funcion de transferencia es devuelta como argumento de salida

function z = compapolo(x, polominreal)

% Esta funcion elimina los polos o ceros que cumplen con el criterio de
% estar alejados 5 veces el valor del polo mas significante(Aquel mas
% cercano del origen)

nuevovect = [];                     % Inicia con un vector vacio
longi = length(x);                  % Determina la longitud del vector
indice = 1;                         % Controla la adicion de elementos
xreal = real(x);                    % Interesa comparar la parte real
for k = 1 : longi
    % Se determina la distancia que existen entre cada Kesimo polo con el
    % mas significante, si cumple con el criterio se incluye en el nuevo
    % vector
    dist = abs(xreal(k)-polominreal);
    if dist <= (abs(polominreal*5))
      nuevovect(indice)=x(k);       % Nuevo vector reducido
      indice = indice + 1;
    end
end
  z = nuevovect;                    % El vector reducido devuelve su argumento de
salida

function y = conventb(x,modo)
% Esta funcion y = conventb(x,modo), permite trasladar el valor de x
% binario proveniente del canal de entrada B, a su equivalente numerico,
% de manera que el valor de voltaje o corriente medido por el canal B tenga
% un equivalente decimal.
% el "modo" especifica si es de voltaje(modo = 1) o de corriente (modo = 2)
%
% UDB 2004

n = length(x);                      % Se determina la longitud del vector de
entrada
y=x;                                % Se crea el vector de salida

% Operacion en modo1, cuando el canal B es entrada de Voltaje
if modo == 1
% Valores de x entre 0 a +10 V
    for k=1:n
       temp = x(k);
```

```
       if temp <=32767
       y(k)=(10*temp)/32767;              % Ecuacion definida para el rango
       % Valores negativos de x hasta -10 V
       else
       y(k)=((10/32768)*(temp-32768))-10;  % Ecuacion definida para el rango
       end
    end
end

% Operacion en modo2, cuando el canal B es entrada de Corriente
if modo == 2
    % Valores de x entre 0 a +20 mA
    for k=1:n
       temp = x(k);
       if temp <=32767
       y(k)=(20*temp)/32767;              % Ecuacion definida para el rango
       % Valores negativos de x hasta -20 mA
       else
       y(k)=((20/32768)*(temp-32768))-20;  % Ecuacion definida para el rango
       end
    end
end



function y = consala(x,modo)
% Esta funcion y = consala(x,modo), permite trasladar el valor de x a su
% equivalente binario
% compatible con la PCI de manera que a la salida del canal A se entregue
% el valor de voltaje o corriente especificado por x, el "modo" especifica
% si es de voltaje(modo = 1) o de corriente (modo = 2)
%
% UDB 2004

n = length(x);                    % Se determina la longitud del vector de entrada
y=x;                              % Se crea el vector de salida

% Operacion en modo1, cuando el canal A es salida de Voltaje
if modo == 1
% Valores de x entre 0 a +10 V
for k=1:n
    temp = x(k);
    if temp >=0
       y(k)=(127*temp)/10;        % Ecuacion definida para el rango
       y(k) = round(y(k));
            if x>10               % Si se pasa de 10 se coloca el max permitido
              y(k) = 127;
            end
% Valores negativos de x hasta -10 V
    else
    y(k)=((128/10)*(temp+10))+128; % Ecuacion definida para el rango
    y(k) = round(y(k));
           if x<-10                % Si se pasa de -10 se coloca el min permitido
              y(k) = 128;
           end
    end
end

end


% Operacion en modo2, cuando el canal A es salida de Corriente
if modo == 2
```

```
% Valores de x entre 0 a +20 mA
for k=1:n
    temp = x(k);
    if temp >=0
       y(k)=(127*temp)/20;          % Ecuacion definida para el rango
       y(k) = round(y(k));          % Si se pasa de 20 se coloca el max permitido
            if x>20
              y(k) = 127;
            end
    % Valores negativos de x hasta -20 mA
    else
    y(k)=((128/20)*(temp+20))+128;  % Ecuacion definida para el rango
    y(k) = round(y(k));
           if x<-20                 % Si se pasa de -20 se coloca el min permitido
              y(k) = 128;
           end
    end
end

end


% Efectua el control a lazo cerrado mediante un PID discreto
function controlar_Callback(hObject, eventdata, handles)

kp = str2double(get(handles.Kp,'String'));  % Ganancia proporcianal
ti = str2double(get(handles.Ti,'String'));  % Tiempo integral
td = str2double(get(handles.Td,'String'));  % Tiempo derivativo
sp = (get(handles.Spb,'Value'));            % Punto de fijacion
ts1 = handles.tcerrado;                     % Tiempo de muestreo
contador = 0;                   % Lleva la cuenta de la cantidad de datos
                                % ingresados

 temporal = 0;                  % Se carga con el contenido del buffer de
                                % entrada
  salida = [];                  % Se van almacenando los datos adquiridos
  muestras = handles.nuestras;  % Numero de muestras
  ak = 0;                       % Variable de control actual
  a_1 = 0;                      % Variable de control muestra anterior
  ek = 0;                       % Error actual
  e_1 = 0;                      % Error muestra anterior
  e_2 = 0;                      % Error hace 2 muestras anteriores
  vctrl = 0;                    % Variable controlada formato PCI
  to = 0;                       % refresco visualizacion eje x
  flag_min = 1;                 % Control refresco visualizacion
  nivel = [];                   % Vector nivel de referencia

% Llave: Impide entrar si no hay parametros
if (isnan(kp)==0) & (isnan(ti)==0) & (isnan(td)==0) & (isnan(sp)==0)
    global serie                % Variable con la estructura del puerto serie
    global adquirir             % Variable que inicia y/o detiene la captura
    adquirir = 1;               % Variable que inicia y/o detiene la captura
% Verifica estado del puerto, de estar cerrado se abrira
estado_puerto = serie.status;
if strcmp(estado_puerto,'closed') == 1
   fopen(serie);
end
% Configura la PCI con el tiempo de muestreo
if handles.conflazo == 0        % Solo se configura una vez
% ClrinBuffer                   % Borra el buffer y la memoria Xram de la PCI
  limpiar_Callback;             % Limpia el buffer de la PCI y MATLAB
% Borra Xram                    % La memoria Xram del PCI es borrada
  fwrite(serie,48);
```

```
  fwrite(serie,64);
  reconocer_Callback
% In Buffer:                    % Programa en la PCI Leer el Canal B
  fwrite(serie,52);
  fwrite(serie,12);
  reconocer_Callback
% XramDelay:
  tsi = muestras*ts1;
  ts1 = cdelay2(tsi,muestras);    % Configura el Delay en la PCI y calcula ts
  handles.tcerrado = ts1;
% SetCycle:
  fwrite(serie,39);
  fwrite(serie,1,'uint16');           % No.de ciclos de med.
  fwrite(serie,0,'uint16');           % Trigger
  fwrite(serie,0,'uint16');           % Tiempo max
  fwrite(serie,0);                    % Procesamiento(tiempo max)
  reconocer_Callback
  handles.conflazo = 1;               % Solo se configura una vez
end
  % Da el nivel al escalon

  vref2 = consala(sp,handles.modosala); % Rutina que convierte a valores
compatibles con la PCI
  fwrite(serie,80);           % Comando de escritura en el canal A
  fwrite(serie,vref2);        % Parametro con el dato

% Run
  fwrite(serie,37);           % Comando de ejecucion(Run)
  fwrite(serie,3);            % Parametro con el modo 3 de ejecucion(captura
                              % continua}
  nada = fread(serie,1);      % Caracter de reconocimiento
  T = ts1;
  set(handles.Texto_Estado,'String','Control en lazo cerrado...');

% Lazo de captura y presentacion
while adquirir == 1
       if (serie.bytesavailable)/2 ~= 0  % Entra si existen datos en el buffer

              temporal = fread(serie,(serie.bytesavailable)/2,'uint16');
% Datos actuales

            % Procedimiento "Bufferear" los datos
            vector = conventb(temporal,handles.modoentb);  % Rutina, convierte
                                          % datos de PCI a datos Numericos

            sp = (get(handles.Spb,'Value'));           % Punto de
                                                       % fijacion(referencia)
            ek = sp - vector(end);                     % Error
            % Variable de control
     ak = a_1 + kp*( (1+(T/ti)+(td/T))*ek + (-1-2*(td/T))*e_1 + (td/T)*e_2 );
            e_2 = e_1;
            e_1 = ek;
            a_1 = ak;
            % Enviar el dato de control
            vctrl = consala(ak,handles.modosala);     % Rutina que convierte a
                                          % valores compatibles con la PCI
            fwrite(serie,80);           % Comando de escritura en el canal A
            fwrite(serie,vctrl);        % Parametro con el dato

    % Agrega nuevos elementos
    salida(length(salida) + 1 : length(vector) + length(salida))=vector;

            contador = length(salida); % Determina cuantos datos existen en
```

```
                                % este instante
      nivel(1:contador)=sp;     % vector referencia

      % Grafica los datos adquiridos
      pause(0.0001);            % Espera, para una correcta
                                % visualizacion de los datos
      tiempo = to:ts1:(ts1*(length(salida))-ts1)+to; % Crea el vector
                                          % del tiempo para el eje x
      escsalida = salida.*handles.esc; % Datos multiplicados por factor
                                       % de escala
      escnivel = nivel.*handles.esc;
      plot(tiempo,escsalida,tiempo,escnivel,'r')        % Grafica datos
                                        % escaleados respecto al tiempo

      % Determina rango de eje y para la visualizacion
      % referencia positiva
      if nivel(end)>0
         minimo = 0;
         % magnitud maxima eje y
         maximo = max(escsalida);
           if maximo < max(escnivel);
               maximo = max(escnivel);
           end
      % Referencia negativa
      else
         % magnitud minima eje y
         minimo = min(escsalida);
         maximo = 0;
           if minimo > min(escnivel)
               minimo = min(escnivel);
           end
      end
      % Ajuste de ejes
      if minimo < maximo
      axis([to (to+(ts1*500)) (minimo*1.1) (maximo*1.1)]); % Definine
                                                           % los ejes
      end

      grid % Muestra rejilla

      ylabel(handles.unidades);      % Unidades en el eje y
      xlabel('Segundos');            % Unidades de tiempo para eje x

      set(handles.varcontrol,'String',(escsalida(end)));      % Presenta
                                                        % var controlada
      set(handles.error,'String',(ek*100/sp));                % Presenta
                                                % el error porcentual

       % Refresco de visualizacion, al llegar a un conteo de 500 se
       % procede a limpiar todos los vectores y establecer un nuevo
       % inicio en el eje del tiempo x
       if contador >=500
           salida =[];
           vector = [];
           to = tiempo(end)+ts1;
           flag_min = 0;
           escsalida =[];
           tempo = [];
           nivel =[];
           escnivel = [];
        end

end
```

```
 end


   fwrite(serie,27);                        % Detiene la captura de la PCI
   nada = fread(serie,1);                   % Caracter de reconocimiento

  %Quita el nivel del escalon
  fwrite(serie,80);                         % Escribir en canal A
  fwrite(serie,0);                          % Poner cero a la salida

  limpiar_Callback; % Limpia el buffer de la PCI y MATLAB

     fclose(serie);                         % Cierra el puerto serie
  set(handles.Texto_Estado,'String','Control detenido...');

else  % Llave: Evita entrar si no hay parametros
     set(handles.Texto_Estado,'String','Ingrese todos los parametros');

end   % Llave: Evita entrar si no hay parametros

% Actualiza la estructura del Handles
guidata(hObject, handles);
% --------------------------------------------------------------------

% Determina el valor de los parametros del PID
function param_Callback(hObject, eventdata, handles)

determinar = 1;                  % Bandera que habilita proceso de adquisicion de
parametros
hay_modelo = handles.hay_modelo;  % Indica que ya existe modelo o TF

h = handles.hms ;

% Se evita intentar procesar si no existe modelo
if hay_modelo == 0
set(handles.Texto_Estado,'String','No hay modelo para procesar...');
determinar = 0;
end

if determinar == 1                % Llave si no existe las condiciones para
procesar

escalon = handles.entrada(2);     % Datos de la senal aplicada al sistema en
                             % la entrada Definidos por el nivel del escalon

set(handles.Texto_Estado,'String','Determinando Parametros...');

% Funcion que determina los parametros del PID
[kp,ti,td] = param_pid(h);
% Se muestran y almecenan los parametros en cajas de texto
set(handles.Kp,'String',kp);
set(handles.Ti,'String',ti);
set(handles.Td,'String',td);
set(handles.Texto_Estado,'String','Parametros obtenidos...');

end

guidata(hObject, handles);      % Actualiza la estructura del Handles
% --------------------------------------------------------------------

% Codigo que crea la caja de texto para el nivel de referencia
function Sp_CreateFcn(hObject, eventdata, handles)
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --------------------------------------------------------------------
function Sp_Callback(hObject, eventdata, handles)
% Caja de texto donde se almacena el valor de referencia en el lazo cerrado

escsp = str2double(get(handles.Sp,'String'));   % Optiene el valor de nivel de ref
% verifica que la caja no este vacia
if (isnan(escsp)==0)
% El valor de la caja esta multiplicada por la escala de modo que se tiene
% que dividir por este para obtener su valor absoluto

sp = escsp/handles.esc;

% Delimita los valores compatibles con la PCI
if sp>10
    sp = 10;
set(handles.Sp,'String',10*handles.esc);
end
if sp<-10
    sp = -10;
set(handles.Sp,'String',-10*handles.esc);
end

set(handles.Spb,'Value',sp);                % Actualiza la barra de desplazamiento
                                            % correspondiente
end
guidata(hObject, handles);      % Actualiza la estructura del Handles

% --------------------------------------------------------------------
function Spb_CreateFcn(hObject, eventdata, handles)

% Codigo para crear la barra de desplazamiento del nivel de
% referencia del lazo cerrado
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --------------------------------------------------------------------
function Spb_Callback(hObject, eventdata, handles)
% Barra de desplazamiento del nivel de referencia en lazo cerrado
sp = (get(handles.Spb,'Value'));      % Obtiene el valor de la barra
escsp = sp*handles.esc;
set(handles.Sp,'String',escsp);        % El valor de la barra es colocado en caja
de texto

guidata(hObject, handles);          % Actualiza la estructura del Handles

% --------------------------------------------------------------------

function adquirido2_Callback(hObject, eventdata, handles)

% Verifica la condicion del cheque en el submenu reducir del menu PCI

   set(handles.adquirido2,'checked','on')
   handles.pidmetodo = 1;       % Determina parametros segun datos adquiridos
```

```
   set(handles.modelo2,'checked','off')
guidata(hObject, handles);      % Actualiza la estructura del Handles


% --------------------------------------------------------------------
function avanzado_Callback(hObject, eventdata, handles)
% Efectua un llamado a la interfaz avanzado

   dato = Avanzado([handles.nuestras,handles.tcerrado,handles.i_puerto]);
   handles.nuestras = dato(1);     % Numero de muestras
   handles.tcerrado = dato(2);     % tiempo de muestreo en lazo cerrado
   handles.i_puerto = dato(3);     % Guarda el indice del puerto (com1=1,com2=2...)
   handles.conflazo = 0;           % Reconfigura tiempo de muestreo
   guidata(hObject, handles);      % Actualiza la estructura del Handles

% --------------------------------------------------------------------
function mescala_Callback(hObject, eventdata, handles)
% Efectua un llamado a la interfaz avanzado

[es_cala,unidades] = escala(handles.esc,handles.unidades);
handles.esc = es_cala;            % Valor de la escala
handles.unidades = unidades;      % Unidades de escala
guidata(hObject, handles);        % Actualiza la estructura del Handles



function varargout = Avanzado(varargin)
% Esta interfaz se utiliza en conjunto con newpci4, su funcion es la de
% ventana de configuracion para la eleccion del puerto de comunicaciones,
% el numero de muestras en lazo abierto y el tiempo de muestreo en lazo cerrado.
% Posee igual numero de argumentos de entrada como de salida de la
% siguiente forma: [D] = escala(D) donde: D es un vector de 3 elementos
% correspondientes a D(1) = numero de muestras, D(2) = tiempo de muestreo y
% D(3) es el indice del puerto COM1 = 1,...
%
% UDB 2004

% Codigo de inicializacion
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Avanzado_OpeningFcn, ...
                   'gui_OutputFcn',  @Avanzado_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de la inicializacion

% Carga los parametro de configuracion y asigna valores almacenados
function Avanzado_OpeningFcn(hObject, eventdata, handles, varargin)
dato = varargin{1};      % Carga el argumento de entrada
Muestras = dato(1);      % Numero de muestras
Tcerrado = dato(2);      % Tiempo de muestreo
val = dato(3);           % Indice del puerto
```

```
set(handles.muestras,'String',Muestras); % coloca en caja de texto: muestras
set(handles.tclose,'String',Tcerrado);   % coloca en caja de texto: tiempo de
                                         % muestreo
set(handles.puerto,'Value',val);         % coloca en menu desplegable: Indice
                                         % del puerto
handles.dato = dato;                     % Guarda la info en handlet
handles.output = hObject;

% Actualiza el handlet
guidata(hObject, handles);

uiwait; % Detiene el proceso de la GUI que hizo la llamada

% Funcion de salida
function varargout = Avanzado_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;      % Se almacenan las variables numericas
close;

% Creacion de objetos graficos
function muestras_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function muestras_Callback(hObject, eventdata, handles)

function tclose_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function tclose_Callback(hObject, eventdata, handles)

% Obtiene y guarda los parametros escogidos en la interfaz
function aceptar_Callback(hObject, eventdata, handles)

Muestras = str2double(get(handles.muestras,'String'));     % Obtiene parametro
                                                           % muestras
Tcerrado = str2double(get(handles.tclose,'String'));    % Obtiene parametro ts
dato(1)=Muestras;                        % muestras
dato(2)=Tcerrado;                        % tiempo de muestreo
global serie   % Variable de tipo estructura para la utilizacion del puerto

estado_puerto = serie.status; % Determina si el puerto esta cerrado o abierto
if strcmp(estado_puerto,'open')   == 1
   fclose(serie);             % Cierra el puerto
end

% Obtiene el indice del puerto
val = get(handles.puerto,'Value');
switch val
case 1
    puer_to = 'com1';
case 2
    puer_to = 'com2';
case 3
```

```
    puer_to = 'com3';
end
set(serie,'port',puer_to);   % Configura el indice del puerto
dato(3) = val;
handles.output = dato;       % Guarda argumentos de salida

% Guarda el handle
guidata(hObject, handles);

% Le devuelve el control a la GUI que esta es espera de los datos
uiresume;

% Boton Cancelar, Carga la ultima configuracion la guarda y pasa los datos
% al vector de salida y a la variable de salida tipo texto
% "Ningun cambio hecho en la GUI sera almacenado y/o transferido
function cancelar_Callback(hObject, eventdata, handles)
% Carga la ultima configuracion
dato = handles.dato;
handles.output = dato;

% Guarda el handle
guidata(hObject, handles);
% Le devuelve el control a la GUI que esta es espera de los datos
uiresume;

% Crea menu desplegable para indice del puerto
function puerto_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function puerto_Callback(hObject, eventdata, handles)



function varargout = Escala(varargin)
% Esta interfaz se utiliza en conjunto con newpci4, su funcion es la de
% ventana de configuracion para el factor de escala y unidades.
% Posee igual numero de argumentos de entrada como de salida de la
% siguiente forma: [D,U] = escala(D,U) donde: D es una variable que posee
% el valor del factor de escala
% U es una variable de texto(String) asignada para las unidades
%
% UDB 2004

% Codigo de inicializacion
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Escala_OpeningFcn, ...
                   'gui_OutputFcn',  @Escala_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% Fin de la inicializacion

% Carga los parametro de configuracion y asigna valores almacenados
function Escala_OpeningFcn(hObject, eventdata, handles, varargin)
% Carga de parametros
dato = varargin{1};         % Carga el primer argumento de entrada
escala = dato(1);           % Correspondiente al factor de escala
% Extrae del segundo argumento de entrada el valor de texto (String)
unidades2 = varargin{2};            % Unidades

set(handles.es_cala,'String',escala);         % Valores guardadosfactor de
                                              % escala
set(handles.u_nidades,'String',unidades2);    % Valores guardados para
                                              % unidades

handles.dato = dato;                          % Guarda el F. escala
handles.unidades2 = unidades2;                % Almacena en handle el vector
                                              % de unidades
handles.output = hObject;

% Actualiza el handlet
guidata(hObject, handles);

uiwait; % Detiene el proceso de la GUI que hizo la llamada


% Funcion de salida
function varargout = Escala_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;      % Se almacenan las variables numericas
varargout{2} = handles.salida;      % Se almacena la variable tipo String
close;

% Creacion de objetos graficos
function es_cala_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function es_cala_Callback(hObject, eventdata, handles)
function u_nidades_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function u_nidades_Callback(hObject, eventdata, handles)

% Obtiene y guarda los parametros escogidos en la interfaz
function aceptar_Callback(hObject, eventdata, handles)

escala = str2double(get(handles.es_cala,'String'));   % Obtiene parametro
                                                      % F.escala
unidades2 = get(handles.u_nidades,'String');          % Obtiene parametro
                                                      % unidades
dato(1)=escala;
% guarda los valores a para que se pasen como argumentos de salida
```

```
handles.output = dato;
handles.salida = unidades2;
% Guarda el handle
guidata(hObject, handles);
% Le devuelve el control a la GUI que esta es espera de los datos
uiresume;

% Boton Cancelar, Carga la ultima configuracion la guarda y pasa los datos
% al vector de salida y a la variable de salida tipo texto
% "Ningun cambio hecho en la GUI sera almacenado y/o transferido
function cancelar_Callback(hObject, eventdata, handles)
% Carga la ultima configuracion
dato = handles.dato;
unidades2 = handles.unidades2;
handles.output = dato;
handles.salida = unidades2;
% Guarda el handle
guidata(hObject, handles);
% Le devuelve el control a la GUI que esta es espera de los datos
uiresume;


function cerrar_gui
% La funcion cerrar_gui es una modificacion de la funcion closereq.m cuya
% funcion es de cerrar las ventanas de tipo fig. el codigo que se le ha
% agregado es para eliminar el objeto utilizado para usar el puerto serie
% como de eliminar de la memoria las variables globales
%
% UDB 2004

instrreset;     % Desconecta e invalida objeto de tipo serie
clear global;   % elimina las variables globales

% El siguiente codigo es el original para cierre de las ventanas de tipo
% fig
shh=get(0,'ShowHiddenHandles');
set(0,'ShowHiddenHandles','on');
currFig=get(0,'CurrentFigure');
set(0,'ShowHiddenHandles',shh);
delete(currFig);
```

9 Referencias Bibliográficas.

[1] M.K. Masten. *"Modern Control Systems". Study Guide"*. IEEE/EAB Self-Study Course. 1995.

[2] L. Ljung. *"System Identification. Theory for the user"*. Prentice Hall. 1987.

[3] Platero Dueñas, Carlos; Hernando Gutierrez, Miguel. *"Apuntes de Servosistemas del 2º cuatrimestre"*. Servicio Publicaciones-EUITI – UPM. Madrid 2002

[4] Katsuhiko Ogata, *"Ingeniería de Control Moderna"*. Segunda Edición. Prentice Hall. 1993.

[5] Damir Vrančić J. Stefan Institute. *"new pid controller auto-tuning method based on multiple integrations"*, Jamova 39, SI-1001 Ljubljana, Slovenia.
home page: http://www-e2.ijs.si/People/Damir.Vrancic.html
Visitada el 7 de junio de 2004.

[6] MATHWORKS, Inc. (The). (2002). Ayuda en pantalla del programa de computadora: MATLAB versión 6.5.0.180913a release 13, June 18, 2002. MATHWORKS, Inc. (The). Página web oficial. Disponible: http://www.mathworks.com
Visitada en mayo de 2004.

[7] Kuo, Benjamín C., *"Sistemas de Control Digital"*. Continental, 1984.

[8] Lucas Nülle, Manual del PC/I-1.

[9] Savant, Roden y Carpenter, *"Diseño Electrónico. Circuitos y Sistemas"*. Segunda Edición. Addison-Wesley Iberoamericana. 1992.