UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA MECANICA

ESPECIALIDAD DE INGENIERIA MECATRONICA

"DISEÑO E IMPLEMENTACION DE UN CONTROLADOR PID ADAPTATIVO USANDO ALGORITMOS GENETICOS"

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO MECATRONICO

CRISTHIAN ALBERT VELAZCO SOLANO

Promoción 2001-I

LIMA - PERU

2002

INDICE

Página

DEDICATORIA

Este informe de tesis se lo dedico a mi madre Olinda, que con su apoyo incondicional y permanente me enseño a tener fuerza de voluntad para lograr mis objetivos y jamás dejarme vencer por los avatares de la vida; a mi padre Luis, por sus enseñanzas y consejos que me condujeron por el camino del bien; a mi abuelita Fortunata, a mi hermano Carlos y a mi enamorada Merylyn quienes me brindaron su apoyo moral y me alentaron a seguir adelante. Asimismo, quiero hacer un extensivo agradecimiento a mi asesor, por guiarme en la elaboración de esta tesis y a la Facultad de Ingeniería Mecánica de mi alma mater por entregarme las herramientas del conocimiento para desenvolverme profesionalmente.

PROLOGO

Esta tesis está dividida en seis capítulos. En el capítulo II, revisaremos la teoría básica y algunos métodos de optimización para el diseño de los parámetros PID. Primero hablaremos de la interpretación en el dominio del tiempo continuo del control PID, y que aporte tiene cada uno de los parámetros que lo conforman, así como el efecto de la combinación de los mismos. Se presentaran a continuación los métodos clásicos de sintonización de Ziegler – Nichols, los cuales constituyen un punto de partida para hallar los parámetros de mayor rendimiento. Hablaremos después de la implementación digital del controlador PID, parte fundamental del diseño, sobretodo si se quiere implementar en microprocesadores o microcontroladores, los cuales son dispositivos de rendimiento alto, debido a su velocidad y a la precisión en el procesamiento de los datos. Por último en este capitulo hablaremos de ciertas modificaciones que se hacen a la entrada de datos, dentro del controlador PID, esto para mejorar el rendimiento, y permitir que este actúe en casos extremos o grandes variaciones de la señal de Referencia.

En el capítulo III, hablaremos de los Algoritmos Genéticos, y presentaremos todos los conceptos que ellos involucran. Comenzaremos hablando de la codificación de los AGs como parte fundamental, si se quiere tener precisión en la obtención de resultados, habiendo dos formas clásicas de codificación: la binaria y la de punto flotante, cada uno con sus respectivas ventajas y desventajas. La población constituye la base sobre la cual

se hace la elección de los mejores individuos, por lo cual se hablará de ella y de las distintas formas de conseguir una población inicial. A continuación se hablará de la función que evalúa a los individuos (función idoneidad) y luego de los métodos de selección que clasifica solo a los mejores, los cuales participaran en la creación de la siguiente generación de individuos. Pero dicho proceso va acompañado de los operadores cruzamiento y mutación, cuyo trabajo involucra la búsqueda de nuevas opciones a partir de los mejores individuos seleccionados. Terminaremos este capitulo con un ejemplo simple de optimización de una función, para ver en acción a los AGs, y a la forma como obtienen sus resultados, a través de la búsqueda global y a la evolución de posibles soluciones.

Para el capítulo IV, hemos reservado el desarrollo del Control Adaptativo, el cual está conformado por dos grupos principales: los controladores con auto sintonización y controladores con modelo referencial. Nosotros centraremos nuestro estudio sobre estos últimos, los cuales tratan de ajustar un modelo de Referencia con cierta dinámica que debe de ser acorde con el funcionamiento del proceso. También existe una subdivisión entre los distintos métodos clásicos de diseñar del control Adaptativo por Modelo de Referencia, los cuales serán señalados, incluyéndose un método especial de diseño para sistemas con no linealidades de cierto tipo, que se verá en la sección 4.3. Luego hablaremos de la fusión de ambas teorías de optimización, es decir el control adaptativo y los algoritmos genéticos, el cual se presentará a través de un seudo código en el que se explica las características y los pasos que se tienen que seguir para dicho diseño.

Para finalizar nuestro trabajo hemos reservado el capítulo V para la aplicación de la teoría presentada, por lo cual se eligió el diseño de un controlador adaptativo Genético con Modelo de Referencia para el control de la velocidad de un motor DC, cuyo modelo será obtenido por técnicas de ARX especificadas en el apéndice A, y al cual primero se harán simulaciones usando Matlab y luego se implementará físicamente y se compararán los resultados obtenidos con las simulaciones anteriores para de esta manera verificar el rendimiento de nuestro algoritmo de Control en el seguimiento de la salida de un modelo de referencia.

CAPITULO I

INTRODUCCION

1.1 OBJETIVO

El objetivo de esta tesis es presentar una técnica de sintonización adaptiva para un controlador Proporcional - Integral – Derivativo (PID), basado en las técnicas de optimización de los Algoritmos Genéticos, y aplicarlo en el control en tiempo real de la velocidad de un motor DC, es decir partiendo de valores aleatorios, encontrar los valores de los parámetros PID que cumplan con las expectativas de diseño, en nuestro caso que se acerque lo máximo posible a la dinámica de un modelo de referencia.

1.2 ANTECEDENTES Y MOTIVACIÓN

El diseño basado en controladores PID, es el método predominante en la industria de control automático. Los controladores PID son usados extensamente en industrias con procesos químicos, sistemas eléctricos y electrónicos, pilotos automáticos de aviones, mísiles, barcos, la industria de los robots, entre otros. Esta popularidad es debido a su robustez en un amplio rango de condiciones de operación, la simplicidad de su estructura, así como la familiaridad que encuentran los diseñadores y los operadores en los algoritmos PID. Además no son muy costosos para implementarlos y son suficientes para las necesidades de muchos sistemas de control industrial.

Sin embargo, los controladores PID eran regularmente sintonizados en la práctica de una manera muy pobre, con la mayoría de las sintonizaciones hechas manualmente a través del famoso.procedimiento de *prueba y error*, o por cierto conocimiento previo, dado por la experiencia que tenían del proceso el personal a cargo.

Esto hacía que el proceso de sintonización sea lento y dificultoso, además en muchas situaciones los controladores PID eran reemplazados por un controlador PI, deshabilitando la acción derivativa. Esto sacrificaba rendimiento y eficiencia de la operación para hacer más fácil y rápido el procedimiento de sintonización. Es por ello que para manejar este problema, varias técnicas de sintonización de controladores PID para modelos linealizados fueron introducidas, siendo suficientes para la mayoría de sistemas de control.

Pero actualmente, debido a la demanda de alto rendimiento de los controladores y a que cada día aparecen sistemas más complejos, los cuales necesitan ser controlados, se necesita entonces métodos más eficientes de sintonización. Los avances en la industria electrónica y en la tecnología de microprocesadores han hecho posible el desarrollo de una amplia gama de métodos alternos de "*sintonización*" PID. Pero no solo el avance en estas ramas ha hecho posible el desarrollo de nuevos métodos, sino que también se han ido desarrollando varias teorías y algoritmos, muchos de los cuales en los últimos años se han volcado a emular a la naturaleza y a sus mecanismos aproximados de aprendizaje

y razonamiento, en este caso nos referimos a las ramas de la Inteligencia Artificial, como son: la lógica difusa, las redes neuronales, y los algoritmos genéticos (AGs.) Estos últimos, son programas de computadora que fueron desarrollados para emular crudamente la evolución de poblaciones biológicas de acuerdo a la teoría de la selección natural, de Darwin y el concepto de herencia de los genes propuesto por Mendel.[KOZA_5], y cuya aplicación se da en el campo biológico, en ciencias de la computadora, investigación de operaciones, técnicas híbridas, ciencias físicas, ciencias sociales, optimización de estructuras mecánicas[GOLDBERG_2], y por supuesto en el campo de la ingeniería de control, entre otros..

Así, basándose en la teoría de los AGs, se comenzaron a hacer algunas investigaciones para ayudar en el diseño de sistemas de control y en la identificación de sistemas, pero regularmente estos se hacían fuera de línea (off – line), no permitiendo interactuar a los AGs con el proceso. Sin embargo, actualmente son muchas las investigaciones las que se realizan acerca del uso de los AGs en el diseño de sistemas de control en tiempo real (on – line), así como también en el desarrollo del hardware (procesadores genéticos) que facilite la aplicación e implementación de esta teoría[SALAMI_14].

Para terminar, como se dijo anteriormente, varios algoritmos de sintonización PID en tiempo continuo y discreto para sistemas lineales, han sido introducidos exitosamente en años recientes. Sin embargo, debido a su estructura lineal, ellos operan exitosamente

solo dentro de una región limitada de control, donde las no linealidades del proceso pueden ser efectivamente ignorados. Debido a que las no linealidades son partes intrínsecas de todo proceso real, existe una necesidad de algoritmos PID adaptativos que puedan manejar sistemas con no linealidades, completa o parcialmente desconocidas, sin perder la estructura de control simple. Esta es la motivación detrás de este trabajo de tesis, que presenta una técnica distinta de los métodos clásicos y que se basa en los AGs, los cuales representan la forma en que la naturaleza ha conseguido salir adelante.

CAPITULO II

CONTROLADORES PID

2.1 INTRODUCCION

Un controlador de retroalimentación o en lazo cerrado (feedback) esta diseñado para generar una salida que cause algún esfuerzo correctivo para que sea aplicado a un proceso, de tal manera que se lleva la salida de dicho proceso hacia un valor deseado conocido como Punto de Referencia (Setpoint.) El controlador usa un actuador que afecta al proceso y sensor para medir el resultado. Virtualmente todos los controladores de retroalimentación determinan su salida mediante la observación del error producido entre el Punto de Referencia y la medición de la variable del proceso. El error ocurre cuando el operador humano cambia el punto de referencia intencionalmente o cuando una perturbación o carga sobre el proceso, cambia la variable de proceso accidentalmente. Entonces la misión del controlador es eliminar el error automáticamente. Pero para que esto suceda, la elección del controlador a usar debe de darse siguiendo ciertas especificaciones de diseño, los cuales describen que debe hacer el sistema y como debe hacerlo. Estas especificaciones con únicas para cada aplicación individual y con frecuencia incluyen especificaciones como *estabilidad relativa*, *precisión en estado estable (error)*, *respuesta transitoria*, y *características de respuesta en frecuencia*. Esta ultima consideración a sido históricamente desarrollada con una gran cantidad de herramientas gráficas tales como las trazas de Bode, la traza de Nyquist, la traza de ganancia – fase, y la carta de Nichols[KUO_6], pero debido al desarrollo y la

disponibilidad de software de computadora amigable, rápida y poderosa, la practica de diseño ha cambiado rápidamente, tal es así que el diseñador puede correr, en unos cuantos minutos, un gran numero de diseños empleando especificaciones en el dominio del tiempo. Esto a disminuido considerablemente la ventaja histórica del diseño en el dominio de la frecuencia, el cual está basado en la conveniencia de realizar el diseño gráfico en forma manual. Además es difícil, excepto para el diseñador experimentado, seleccionar un conjunto coherente de especificaciones en el dominio de la frecuencia que correspondan a requisitos de desempeño en el dominio del tiempo.

Algo que si no ha cambiado, es el amplio uso dentro del diseño de sistemas de control, de los controladores Proporcional, Integral, Derivativo (PID), los cuales se utilizan debido a su robustez y a que se aplican en forma casi general a la mayoría de los sistemas de control. En el campo de los sistemas para control de procesos, es un hecho bien conocido que los esquemas de control PID básicos y modificados han demostrado su utilidad para aportar un control satisfactorio, aunque tal vez no aportan un control óptimo en muchas situaciones especificas.

2.2 INTERPRETACION EN EL DOMINIO DEL TIEMPO

Para entender que significa el control PID en el dominio del tiempo, es necesario definir ciertos conceptos que constituyen el cuerpo del controlador.

2.2.1 Control Proporcional

Es una técnica de control la cual multiplica la señal de error(la diferencia entre el punto de Referencia y la variable de proceso) por una ganancia especificada K_p y lo usa como una señal correctiva que ingresa al proceso. El resultado efectivo radica en exagerar el error y reaccionar inmediatamente para corregirlo. K_p no puede eliminar completamente los errores (a menos que el proceso tenga propiedades integrativas inherentes), ya que como el error siguiente se acerca a cero, la corrección proporcional desaparece. Esto produce cierta cantidad de error en estado estable, lo cual se puede apreciar en la figura (2.1), donde existe un desplazamiento entre el valor deseado (punto de referencia) y el valor de la salida del sistema.

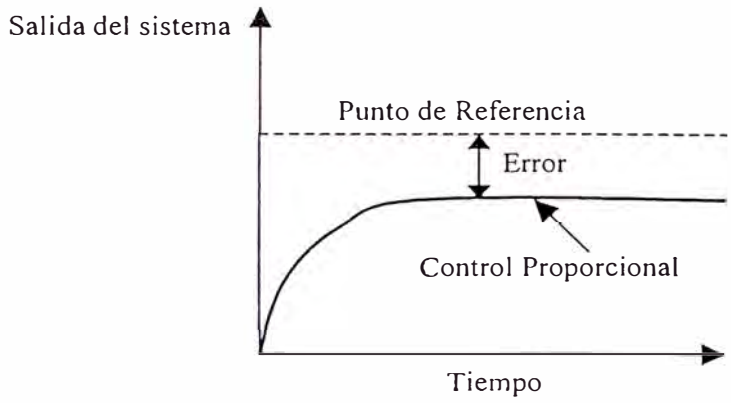

Figura (2.1) Efecto de la acción del Control Proporcional

2.2.2 Control Integral

Es una técnica la cual acumula la señal de error sobre el tiempo, y multiplica dicha suma por una ganancia especificada K_i y usa este resultado como una señal correctiva sobre el

proceso. Debido a que esta técnica actúa sobre errores pasados, el factor de corrección no se va a cero con el error siguiente, eliminando de esta manera el error en estado estable. La ganancia integral tiene un efecto negativo importante, el cual consiste en un factor desestabilizante para el lazo de control para valores grandes de K_i, o para valores usados sin una apropiada amortiguación, los cuales pueden causar severas oscilaciones. Como se aprecia en la figura (2.2), la parte sombreada representa el error acumulado entre los tiempos t1 y t2, que multiplicados con la ganancia Ki, generan el efecto mostrado sobre la salida del sistema, tal y como se muestra en la figura (2.3)

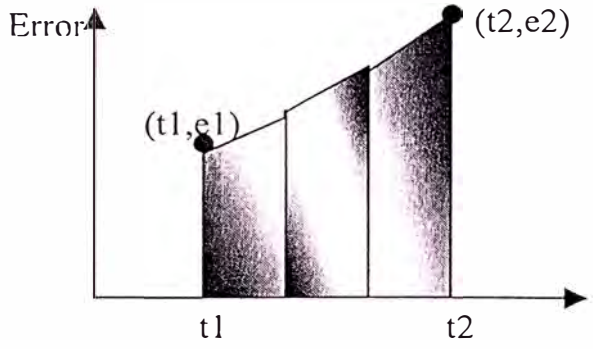

Figura (2.2) Integrando el error en el tiempo

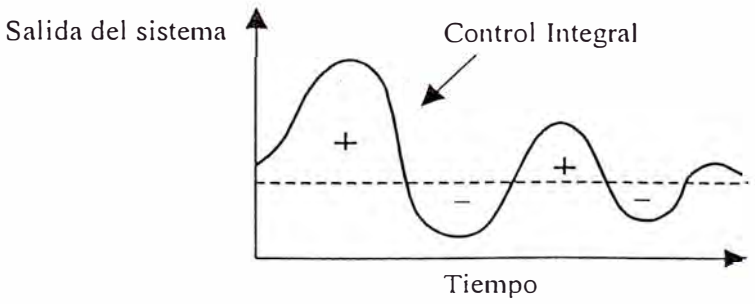

Figura (2.3) Efectos de la acción del Control Integral

2.2.3 Control Derivativo

Se basa en la multiplicación de la razón de cambio del error siguiente por una ganancia especificada K_d y usa este resultado como una señal correctiva sobre el proceso. Otra forma de ver el control derivativo es como un control anticipativo[OGATA_9]. Esto es, al conocer la pendiente el controlador puede anticipar la dirección del error y emplearla para controlar mejor el proceso.

En forma intuitiva, el control derivativo afecta el error en estado estable de un sistema solo sí el error en estado estable varia con el tiempo. Si el error en estado estable de un sistema es constante con el tiempo, la derivada con respecto al tiempo de este error es cero, y la porción derivativa del controlador no provee ninguna entrada al proceso, lo cual se muestra en la figura (2.4) que nos indica que mientras el error existe y varié con el tiempo, se podrá calcular la derivada de dicho error y de esta manera nos permite conocer hacía donde se dirige dicho error.

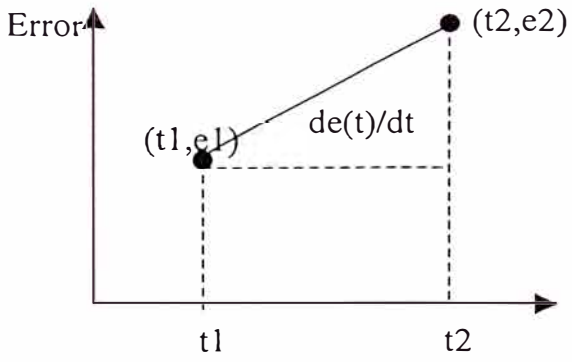

Figura (2.4) Gráfica que muestra la derivada del error

2.2.4 Control PID

De acuerdo a Astrom [GHANADAN_12], la versión del algoritmo de control PID tiene la siguiente forma:

$$u(t) = K[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}] \tag{2.1}$$

$$e(t) = r(t) - y(t) \tag{2.2}$$

Donde u es la variable de control, e representa el error entre la señal de referencia r y el valor medido de la salida del proceso y. La ganancia proporcional K, la constante de tiempo integral T_i, y la constante de tiempo derivativo T_d son los parámetros del controlador. Esta versión del algoritmo PID es también llamada el *"controlador PID no interactivo ideal"* o "ISA" de sus siglas en Inglés (Ideal noninteracting algorithm) [GHANADAN_12]

Otras formas de algoritmos PID usados en la industria son los siguientes [GHANADAN_12, WILLIS_15]:

- Controlador PID interactivo:

$$u(t) = K'[e(t) + \frac{1}{T_i'}\int_0^t e(\tau)d\tau][1 + T_d'\frac{de(t)}{dt}] \tag{2.3}$$

- Controlador ideal PID paralelo:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.4}$$

Donde K_p es la ganancia proporcional, K_i es la ganancia Integral y K_d es la ganancia derivativa.

La simplificación de la ecuación (2.4) nos da la siguiente notación para los algoritmos PID:

$$u(t)=P+I+D \tag{2.5}$$

Donde:

P = Término Proporcional

I = Término Integral

D = Término Derivativo

Se puede deducir una equivalencia entre los valores del algoritmo no interactivo y el algoritmo ideal paralelo de los controladores PID como se muestra a continuación:

$$K_p = K \qquad K_i = \frac{K}{T_i} \qquad K_d = KT_d \tag{2.6}$$

Todas estas versiones son validas y usadas por distintos fabricantes de controladores, por ejemplo, Foxboro y Fisher usan las versiones no interactivos, mientras que Honeywell y Texas Instruments usan el algoritmo interactivo [GHANADAN_12].

Como vemos de las ecuaciones anteriores, los controladores PID operan sobre una señal de error *e*, la cual es generada en línea (on-line) debido a la sustracción de la salida *y* del punto de referencia *r*. Entonces el sistema de control resultante es mostrada en la figura (2.5) donde *z* representan la salida del proceso, en tanto que *l* y *n*, la carga de perturbación y el ruido en la medición respectivamente, siendo estos últimos los que modifican tanto a la salida del controlador, como a la salida del proceso, generándose las señales $u'=u+l$, e $y=z+n$ respectivamente. A este sistema que se le conoce como "sistema con error retroalimentado"[GHANADAN_12].

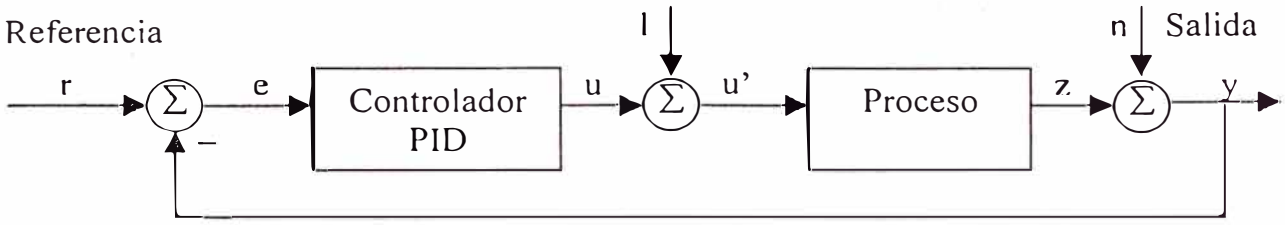

Figura (2.5) Sistema de control PID con error retroalimentado

En el resto de la tesis, entenderemos por *diseño* a la *estructura* usada en los términos P, I y D y por *sintonización* entenderemos a la *selección* de los valores numéricos de los parámetros en los términos P, I y D.

Además, nosotros trabajaremos con la configuración del *"controlador PID no interactivo ideal"*, a la cual modificaremos más adelante para dar mayor estabilidad y robustez al control. La elección de esta configuración es debido a los requerimientos de nuestro algoritmo, que como se verá en el capítulo IV, hará una selección de los mejores controladores, pudiéndose generar valores indeseados, ya que como se sabe por ejemplo si se requiere utilizar Ti, y solo tenemos disponibles los valores de Kp, Ki y Kd, la equivalencia Ti = Kp/Ki se puede dar, pero si ambos valores Kp y Ki fueron escogidos ceros, se produce un valor indefinido.

2.3 REGLAS DE SINTONIZACIÓN DE LOS CONTROLADORES PID

El proceso de seleccionar los parámetros del controlador que cumplan con las especificaciones de desempeño se conoce como sintonización del controlador. Hay sin embargo algunos métodos de sintonización en los cuales primero, un modelo simple del proceso es determinado por el mismo proceso a ser controlado, para luego escogerse los parámetros PID.

Debido a su uso muy difundido en la práctica, existen varios métodos para sintonización de controladores PID [OGATA_9, WILLIS_15], muchos de los cuales datan de varias décadas atrás. Entre estos métodos "clásicos" podemos encontrar a los siguientes:

- Método de Reacción de la Curva de Ziegler-Nichols.
- Método de Oscilaciones Sostenidas de Ziegler-Nichols.
- Método de Cohen - Coon

Las dos primeras reglas fueron sugeridas por Ziegler y Nichols para obtener un modelo aproximado de primer orden del sistema y luego sintonizar los controladores PID basándose tanto en las respuestas experimentales al escalón unitario, como en la estabilidad marginal cuando sólo se usa la acción de control proporcional, respectivamente. En ambos métodos denominados reglas de sintonización de Z – N se pretende obtener un 25% de sobrepaso máximo en la respuesta Y(t) al escalón unitario, tal y como se observa en la figura (2.6.) Por otro lado, Cohen – Coon hizo una modificación a la segunda regla de Z-N, para que la respuesta tenga un mínimo desplazamiento y otras propiedades favorables.

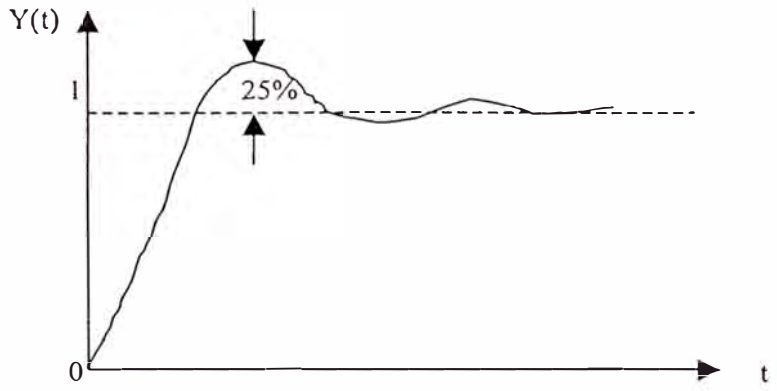

Figura (2.6) Curva de respuesta escalón unitario con 25% de sobrepaso máximo

Las reglas de Ziegler – Nichols, que se presentan a continuación, son muy convenientes cuando no se conocen los modelos matemáticos de las plantas. (Por supuesto, estas reglas se aplican también al diseño de sistemas de control con modelos conocidos.)

2.3.1 Método de Reacción de la Curva de Ziegler-Nichols.

En este primer método la respuesta de la planta a una entrada escalón unitario se obtiene de manera experimental, como se observa en la figura (2.7)

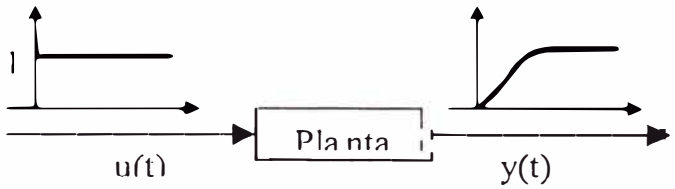

Figura (2.7) Respuesta escalón unitario de una planta

Si la planta no contiene integradores ni polos complejos conjugados, la curva de respuesta escalón unitario puede tener la forma de S, como se observa en la figura (2.8) (sí la respuesta no exhibe una curva en forma de S, este método no es adecuado.) A esta curva también se le conoce como *curva de reacción del proceso*. Tales curvas de respuesta escalón se generan experimentalmente o a partir de una simulación dinámica del proceso.

También en la figura (2.8) se puede apreciar que la curva en forma de S se caracteriza por dos parámetros: el tiempo de retardo L y la constante de tiempo T. El tiempo de retardo la constante de tiempo se determinan dibujando una recta tangente en el punto de inflexión de la curva con forma de S y determinando las intersecciones de esta tangente con el eje del tiempo y la línea y(t) = K. En este caso, la función de transferencia de la planta Y(s)/U(s) se aproxima mediante un sistema de primer orden con un retardo de transporte del modo siguiente:

$$\frac{Y(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1} \tag{2.7}$$

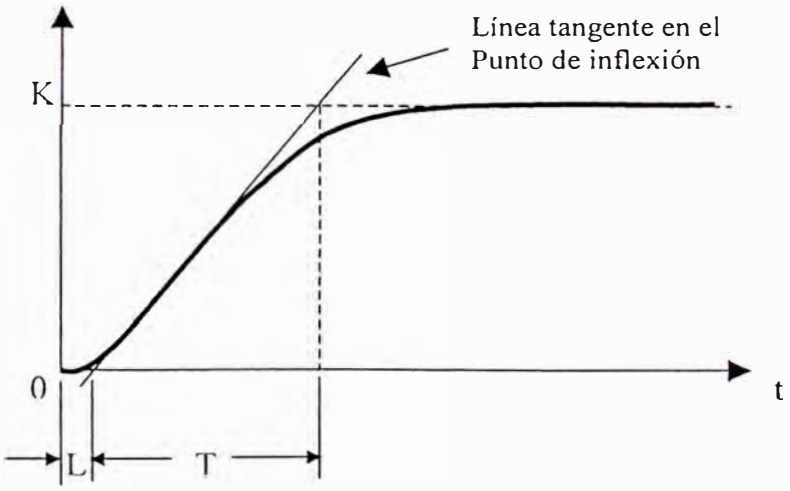

Figura (2.8) Curva de respuesta con forma de S

Ziegler y Nichols sugirieron establecer los valores de Kp, Ti y Td de acuerdo con la fórmula que aparece en la tabla (2.1)[OGATA_9]

Tipo de Controlador	Kp	Ti	Td
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	2L	0.5L

Tabla 2.1 Regla de sintonización de Z –N basada en la respuesta escalón de la planta

Transformando la ecuación (2.1) al plano S y reemplazando los valores dados en la tabla de arriba se obtiene el controlador sintonizado mediante el primer método de las reglas de Ziegler – Nichols:

$$Gc(s) = 0.6T\frac{(s+\frac{1}{L})^2}{s}$$

Por tanto, el controlador PID tiene un polo en el origen y un cero doble en s = -1/L.

2.3.2 Método de Oscilaciones Sostenidas de Ziegler-Nichols

En el segundo método, primero establecemos Ti = ∞ y Td =0. Usando sólo la acción de control proporcional (véase la figura (2.9)), se incrementa Kp de 0 a un valor critico Kcr en donde la salida exhiba primero oscilaciones sostenidas. (Si la planta no aplica oscilaciones sostenidas para cualquier valor que pueda tomar Kp no se aplica este

método.) Por tanto, la ganancia crítica Kcr y el periodo Pcr correspondiente se determinan experimentalmente (véase la figura 2.10) Ziegler – Nichols sugirieron que se establecieran los valores de los parámetros Kp,Ti y Td de acuerdo con la fórmula que aparece en la tabla (2.2)[OGATA_9]

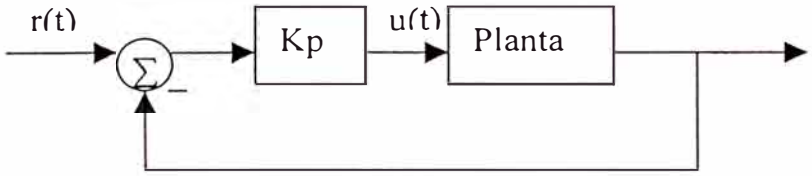

Fig. 2.9 Sistema en lazo cerrado con controlador proporcional

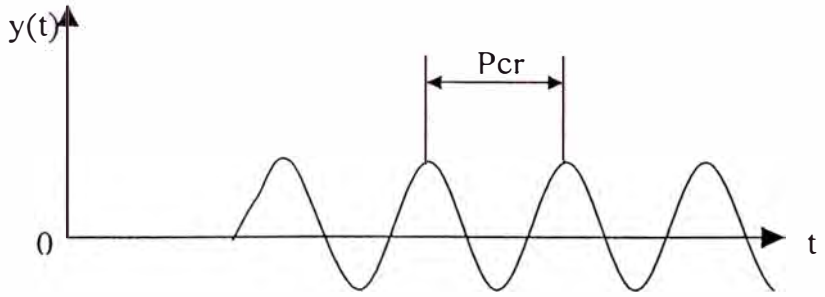

Fig. 2.10 Oscilación sostenida con un periodo Pcr

Tipo de Controlador	Kp	Ti	Td
P	0.5Kcr	∞	0
PI	0.45Kcr	0.83Pcr	0
PID	0.6Kcr	0.5Pcr	0.125Pcr

Tabla 2.2 Regla de sintonización de Z –N basada en el método de oscilaciones sostenidas (Kcr y Pcr)

Reemplazando los valores de la tabla (2.2) en la versión de la ecuación (2.1) en el plano S, el controlador PID sintonizado mediante el segundo método de las reglas de Ziegler - Nichols produce:

$$Gc(s) = 0.075K_{cr}P_{cr}\frac{(s+\frac{4}{P_{cr}})^2}{s}$$

Por tanto, el controlador PID tiene un polo en el origen y cero doble en s = -4/Pcr

2.4 IMPLEMENTANCIÓN DIGITAL

Una vez obtenido los valores de Kp, Ki y Kd directa o indirectamente a través de sus equivalentes obtenidos del algoritmo PID no entrelazado, el siguiente paso es discretizar el controlador PID continuo, para poder implementarlo en circuitos digitales, los cuales por su gran versatilidad y velocidad son los que hoy en día se usan mayormente

Para periodos de muestreo pequeños la ecuación (2.3) puede ser reemplazada por discretización a través de ecuaciones en diferencias. El término derivativo es simplemente reemplazado por una diferencia de primer orden hacia atrás y el término integral puede ser aproximada por una integración rectangular (suma) o una integración trapezoidal. En nuestro caso aplicando una integración rectangular nos da lo siguiente:

$$u(k) = K_p e(k) + K_i T_m \sum_{i=0}^{k-1} e(i) + \frac{K_d}{T_m}(e(k) - e(k-1)) \tag{2.8}$$

Donde T_m es el periodo de muestreo, el cual debe de ser lo más pequeño posible. A esta ecuación también se le conoce como un algoritmo de control no recursivo. Para la formación de la suma todos los errores pasados e(k) tienen que guardarse. Debido al valor producido de la variable manipulada u(k), a este algoritmo también se le conoce como un "algoritmo de posición".

Sin embargo, los algoritmos recursivos son más convenientes para programar. Estos algoritmos se caracterizan por el cálculo de la variable manipulada actual u(k) a través del valor de la variable manipulada previa u(k-1) y de los términos correctivos.

$$u(k-1) = K_p e(k-1) + K_i T_m \sum_{i=0}^{k-2} e(i) + \frac{K_d}{T_m}(e(k-1) - e(k-2)) \tag{2.9}$$

Para esto es necesario sustraer u(k-1) del valor u(k), con lo cual se obtiene el algoritmo recursivo PID en tiempo discreto.

$$u(k) - u(k-1) = q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \tag{2.10}$$

Con los parámetros siguientes:

$$q_0 = K_p + \frac{K_d}{T_m}$$
$$q_1 = -K_p - 2\frac{K_d}{T_m} + K_i T_m \qquad (2.11)$$
$$q_2 = \frac{K_d}{T_m}$$

Al cambio actual en la variable manipulada, $\Delta u(k) = u(k) - u(k-1)$, se le conoce como el "algoritmo de velocidad".

Para el caso en que la integración continua sea reemplazada por una integración aproximada trapezoidal entonces se obtiene la siguiente ecuación para la variable manipulada:

$$u(k) = K_p e(k) + K_i T_m \left(\frac{e(0) + e(k)}{2} + \sum_{i=1}^{k-1} e(i)\right) + \frac{K_d}{T_m}(e(k) - e(k-1)) \qquad (2.12)$$

Y luego de la correspondiente sustracción de u(k-1), se obtiene otra relación recursiva para el algoritmo de control PID.

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \qquad (2.13)$$

Con los siguientes parámetros:

$$q_0 = K_p + \frac{K_d}{T_m} + \frac{K_i T_m}{2}$$
$$q_1 = -K_p - 2\frac{K_d}{T_m} + \frac{K_i T_m}{2} \qquad (2.14)$$
$$q_2 = \frac{K_d}{T_m}$$

2.5 OPTIMIZACION DE LOS PARÁMETROS PID

Cuando se implementa los controladores PID, existen consideraciones adicionales que hay que tener en cuenta, como es la saturación del actuador, los cambios drásticos en la señal de referencia, entre otros. Es por eso que existen técnicas de optimización de los parámetros PID [GHANADAN_12, RODRÍGUEZ_10, WILLIS_15] y que serán presentados a continuación:

2.5.1 Anti -Windup

Cuando se implementa los controladores PID, generalmente se trabaja con actuadores que tienen un rango de trabajo establecido, razón por la cual no se puede aumentar ni disminuir indefinidamente la señal de control. Es por eso que se incluye el término *saturador*, el cual limita la señal de control de acuerdo a los valores máximos y mínimos que pueden admitir el actuador. Esto se puede observar en la figura (2.11), donde *u(t)* es la salida del controlador, *v(t)* es la salida del correspondiente actuador con saturación, *r(t)* es la señal de referencia, *y(t)* es la salida del sistema y *e(t)* es el error de proceso.

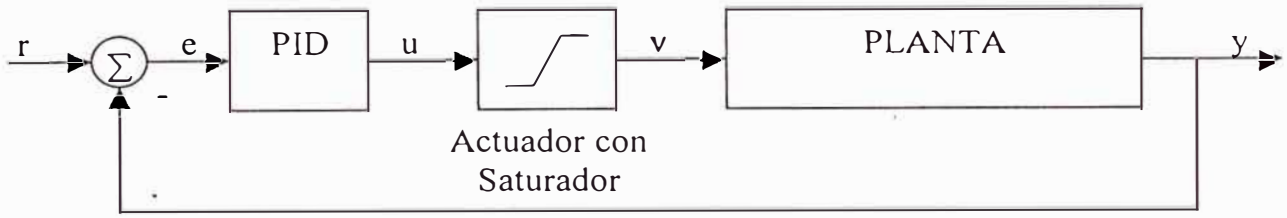

Figura (2.11) Control PID con Actuador que incluye Saturador

El problema con el saturador es que el lazo de retroalimentación se rompe cuando el actuador se satura, lo cual nos lleva a tener un controlador inestable. Es por ello que una modificación común en muchos controladores PID es el "anti – windup", que es una característica adicional del termino integral y cuyo propósito es ejercer una acción compensatoria y prevenir la integración del error cuando la señal de control se satura.

Esto se puede hacer agregando al termino integral I = Ki.e(t) un compensador de la saturación tal y como se muestra en la ecuación (2.15):

$$I = Ki.e(t) + \frac{1}{T_t}(v(t) - u(t)) \qquad (2.15)$$

Donde T_t es la "constante de seguimiento" y cuyas elecciones comunes son:

- $T_t = T_i$, donde T_i es la constante de tiempo Integral definida en la ecuación (2.1)

- $T_t = \sqrt{T_i T_d}$, donde T_d es la constante de tiempo Derivativa definida en la ecuación (2.1) y Ti es igual que en el caso anterior.

Cabe resaltar que sí $0 < T_t \leq T_i$, el estado integrativo se convierte en sensitivo para todos los casos en que $e_s \neq 0$, entonces:

$$I(t) = \int_0 \left[\frac{K.e(\tau)}{T_i} + \frac{e_s(\tau)}{T_t} \right] .d\tau \approx \frac{1}{T_t} \int_0 e_s(\tau).d\tau \qquad (2.16)$$

Donde e_s representa la diferencia entre la señal saturada y la señal de control $e_s = v - u$

Los esquemas comunes del "anti – windup", utilizando el algoritmo de control PID ideal paralelo, se muestran en la figura (2.12a) y (2.12b) y en los cuales se puede observar que cuando no existe una salida medible del actuador se puede usar un modelo del actuador con saturación dentro del algoritmo y de esta manera poder aplicar la compensación cuando la señal de control se satura. También hay que notar que en el primer caso la señal de control corregida v ingresa directamente al proceso, mientras que en el segundo caso, ingresa primero al actuador, y luego este envía la señal de comando al proceso.

a)

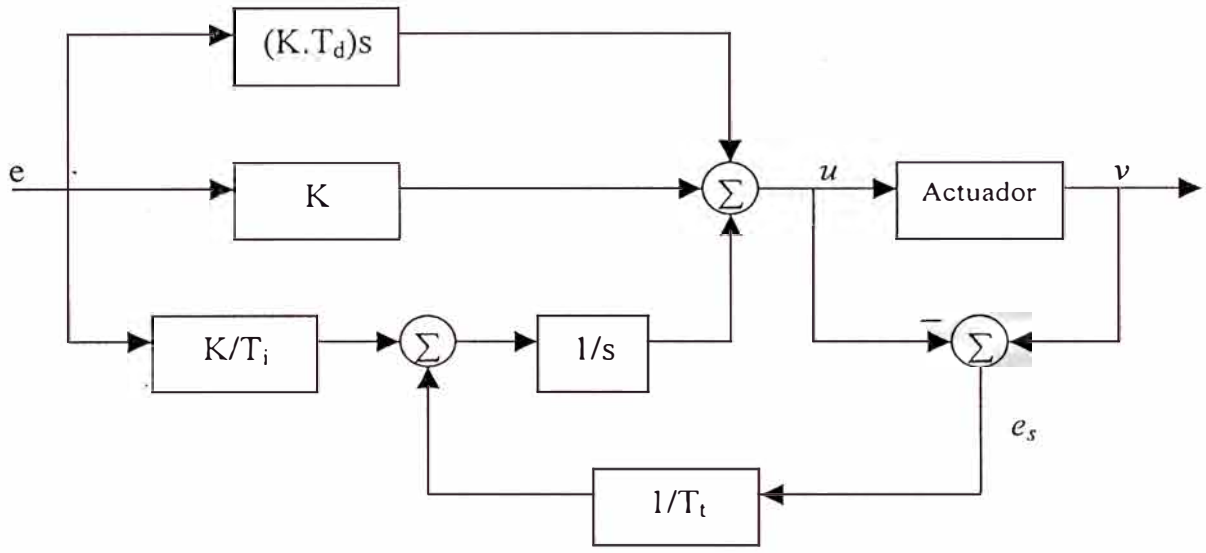

Figura (2.12a) Anti – Windup con salida disponible del actuador

b)

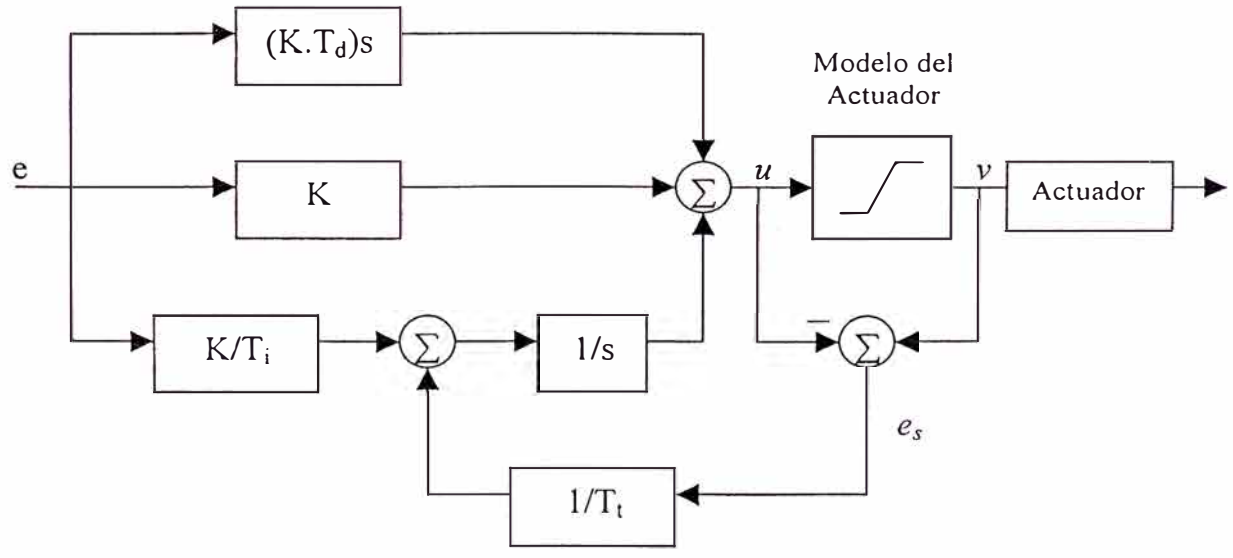

Figura (2.12b) Anti – Windup sin salida disponible del actuador

2.5.2 Otras Aproximaciones del Término Derivativo y uso de Filtros

En la mayoría de los controladores PID propuestos en muchos libros básicos de Control, la acción derivativa opera en la señal de error producida por la diferencia entre la Referencia y la salida del proceso. Sin embargo por cambios abruptos de la señal de referencia, el término derivativo también sufre cambios drásticos, lo cual desestabiliza la acción de control. Es por eso que se introduce una nueva técnica, en la cual, la acción derivativa opera solo en la señal de salida del proceso y no en la señal de referencia para prevenir lo dicho anteriormente [GHANADAN_12].

Por otro lado, una modificación importante es filtrar la acción derivativa con un filtro de primer o segundo orden, para eliminar o disminuir lo máximo posible el ruido derivativo. Esta característica *limita* la medición de alta frecuencia de la amplificación del ruido en la salida del controlador[GHANADAN_12]. Entonces la señal de control tendrá *menos ruido* y la ganancia de alta frecuencia permanecerá dentro de los límites apropiados. A continuación se muestra un término derivativo con un filtro de primer orden:

$$(T_d / N)\frac{dD}{dt} + D = -K.Td\frac{dy_p}{dt} \qquad (2.17)$$

Donde N es el límite de la ganancia derivativa K_d y T_d/N es la constante del filtro, al cual llamaremos τ. Entonces la modificación de D en la ecuación (2.17) es:

$$\tau \frac{dD_s}{dt} + D_s = Kd_s \frac{dS}{dt} \tag{2.18}$$

Para aproximar el término derivativo en la ecuación (2.18) se tiene las siguientes opciones:

- Método de la Diferencia hacia adelante:

$$\tau \frac{D(n+1) - D(n)}{T_0} + D(n) = Kd_s \frac{y(n+1) - y(n)}{T_0}$$

$$D(n+1) = (1 - h/\tau).D(n) + \frac{Kd_s}{\tau}[y(n+1) - y(n)] \tag{2.19}$$

- Método de la Diferencia hacia atrás:

$$\tau \frac{D(n) - D(n-1)}{T_0} + D(n) = Kd_s \frac{y(n) - y(n-1)}{T_0}$$

$$D(n) = \frac{\tau}{\tau + T_0}.D(n-1) + Kd_s.T_0[y(n) - y(n-1)] \tag{2.20}$$

Este último método es siempre estable, motivo por el cual será usado por nosotros para el desarrollo del esquema final del controlador que usaremos, mientras que la ecuación (2.19) es inestable para pequeños periodos de muestreo T_0.

2.5.3 Modificación de la Acción Proporcional K_p

Un concepto adicional y que solo será mencionado es la modificación de la acción proporcional. Como se sabe, el sistema con lazo de retroalimentación propuesto en la ecuación (2.1) trabaja solamente sobre el error de retroalimentación (un grado de libertad), sin embargo, sí separa la salida del proceso de la señal de referencia en el término proporcional de la ley de control (dos grados de libertad), la flexibilidad de diseño se incrementa y por lo tanto se mejora el rendimiento del controlador [GHANADAN_12]. La modificación deseada se consigue parcialmente si modificamos el error *e* en la parte proporcional de la ecuación (2.1)

$$e_p = \beta.r - y \tag{2.21}$$

Donde β es un parámetro constante de diseño. La respuesta transitoria del lazo cerrado se puede mejorar manipulando el valor de β.

Mas grados de libertad para el sistema de control, significa que se necesitan más parámetros a ser sintonizados. En muchos controladores PID convencionales β no es sujeto del proceso de sintonización. Si queremos hacerlo como los otros parámetros K_p, K_d y K_i, nosotros deberíamos de ser capaces de sintonizar cuatro parámetros en vez de tres, lo cual no es muy fácil con métodos convencionales.

Para concluir esta sección, es necesario resaltar que en nuestra investigación se utilizará la configuración del *controlador PID no entrelazado ideal* en su forma modificada discreta tal y como se muestra en la ecuación (2.22), con lo cual se asegura una buena estabilidad del sistema, así como robustez para afrontar los grandes cambios en la señal de referencia, debido a la inclusión de nuevos parámetros que optimizan dicha configuración.

$$\begin{cases} P(k+1) = Kp.e(k) \\ I(k+1) = I(k) + \dfrac{K_p.T}{T_i} e(k) + \dfrac{1}{T_i}(v(k) - u(k)) \\ D(k+1) = \dfrac{\tau}{\tau + T} D(k) + K_p.T_d.T.(y(k) - y(k-1)) \end{cases} \qquad (2.22)$$

CAPITULO III

LOS ALGORITMOS GENETICOS

3.1 INTRODUCCION

Aunque ya se conocía esta aproximación desde hace 30 años, recientemente se ha despertado un gran interés en la búsqueda de algoritmos que presenten analogías con los procesos naturales. Esta tendencia se basa fundamentalmente en la observación de la destreza consumada que poseen los organismos vivos en la resolución de problemas, los cuales manifiestan una versatilidad muy por encima del programa más refinado.

Así, los investigadores más pragmáticos consideran que, "en lugar de envidiar la eficacia de la evolución natural, debemos emularla" [HOLLAND_3] Podemos por tanto tratar de copiar a la naturaleza como una forma de resolución de aquellos problemas en los que no se puedan encontrar soluciones o que éstas no sean lo suficientemente satisfactorias.

Los algoritmos más conocidos que se basan en procesos naturales incluyen, entre otros, los denominados: Programación Evolutiva, Estrategias de Evolución, Algoritmos Genéticos, Enfriamiento Simulado, Sistemas Clasificadores y Redes Neuronales. En este estudio nos centraremos en una subclase de estos algoritmos, aquellos que se basan en los principios de la evolución natural. Tales algoritmos mantienen una población de soluciones potenciales, y poseen algún proceso de selección basado en la adecuación de

las soluciones a su entorno (supervivencia de los más aptos) y algunos operadores de recombinación.

Particularmente hablaremos de los Algoritmos Genéticos (AGs), los cuales representan una técnica paralela de búsqueda que emula las leyes de la evolución, para tratar de encontrar soluciones óptimas a problemas complejos de optimización [GOLDBERG_2], en nuestro caso hallar los controladores óptimos, y cuyo proceso de evolución de la búsqueda genética está basado en: la *diversidad de la población* y en la *presión selectiva* (obligar a que sólo los mejores individuos sobrevivan.) [MICHALEWICZ_8]

La estructura del algoritmo genético puede verse la figura (3.1):

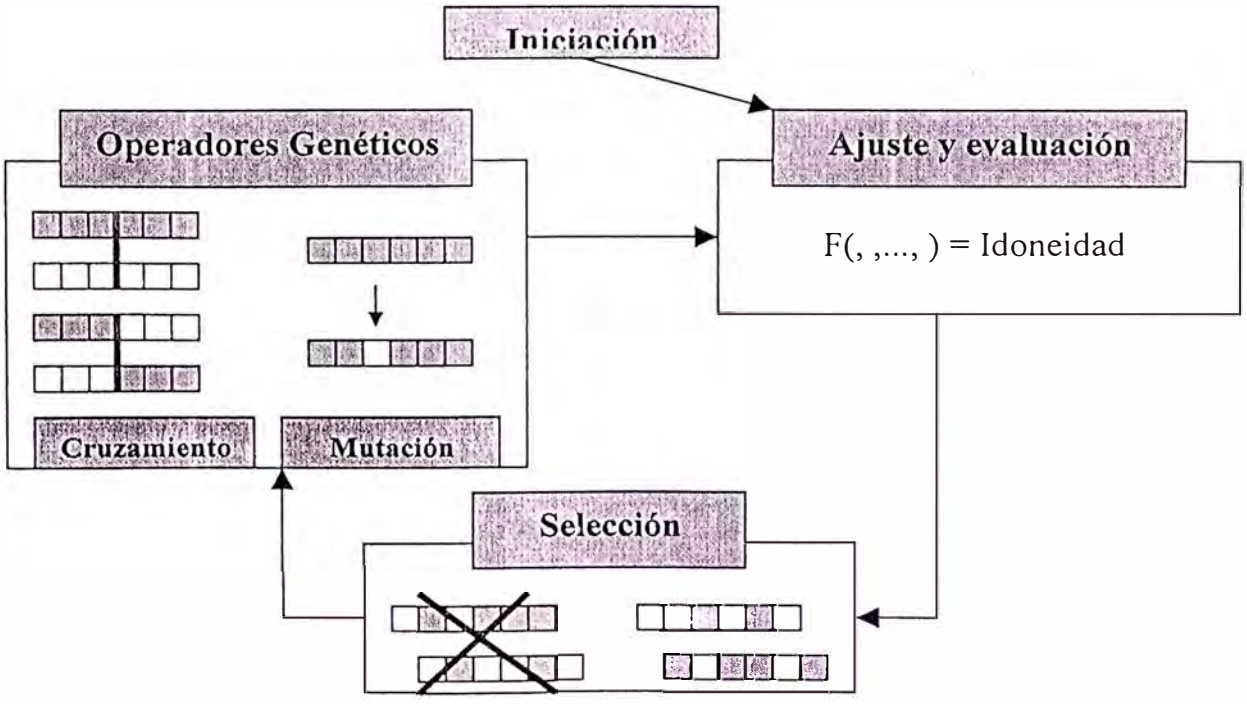

Figura (3.1) Estructura genérica y funcionamiento de un algoritmo genético

Allí observamos que el proceso de optimización comienza con la *iniciación*, consistente en la generación, regularmente aleatoria, de una población de individuos. Luego se hace un *ajuste y evaluación* de dicha población de acuerdo a funciones de eficiencia o idoneidad. Una vez realizado este proceso, se *seleccionan* los mejores individuos, para lo cual existen diversas técnicas, como son: la selección por *Rueda de Ruleta* y la selección por *Torneo*[GOLDBERG_2], centrándonos principalmente en la primera. Los individuos seleccionados entraran en la formación de la próxima generación, no sin antes de ser afectados por *operadores genéticos*: cruzamiento y mutación [MICHALEWICZ_8], los cuales simulan el proceso de reproducción de los seres vivos.

Para terminar esta introducción y antes de pasar a detallar cada uno de los puntos que conforman los AGs, es necesario definir algunos conceptos inherentes de la parte genética, los cuales son necesarios para entender el comportamiento de los AGs, y que serán presentados a continuación:

- **Genoma:** Todos los parámetros que definen a todos y cada uno de los individuos de la población.
- **Genotipo:** La parte del genoma que describe a un individuo concreto. En esta interpretación son los genes que describen un controlador concreto.
- **Gen:** Cada uno de los parámetros que describen a un individuo. En esta interpretación los genes codifican parámetros de un controlador concreto, que en

nuestro caso, puede ser representado por las ganancias Kp, Ki y Kd que constituyen los parámetros del controlador PID. Al conjunto de genes se le denomina también *cromosoma*.

3.2 LA CODIFICACION

El método más utilizado para la *codificación de los genes* es el uso de una cadena de longitud y secuencias fijas. Los parámetros del controlador que se ha decidido ajustar, según los criterios que se han expuesto antes, se almacenan en esta cadena como secuencias fijas de números. De esta manera, el número de genes (en nuestro caso parámetros) resulta fijo, y su valor se guarda siempre en una misma posición de la cadena.

Por otro lado, también se ha de decidir cómo codificar estos números; son habituales codificaciones binarias de 8 bits y de 16 bits, aunque también puede hacerse uso de una codificación en punto fijo o punto flotante. Asimismo, puede emplearse una codificación basada en reservar cierto número de bits para cada gen, y codificar su valor mediante el número de unos presentes, con independencia de su posición. El tipo de codificación seleccionado para los genes influirá en cómo implementar operadores (como el de mutación), así como en la cantidad de memoria necesaria para almacenar el genoma.

Por ejemplo, como se ve en la figura (3.2) los valores de los parámetros PID pueden codificarse de dos maneras distintas:

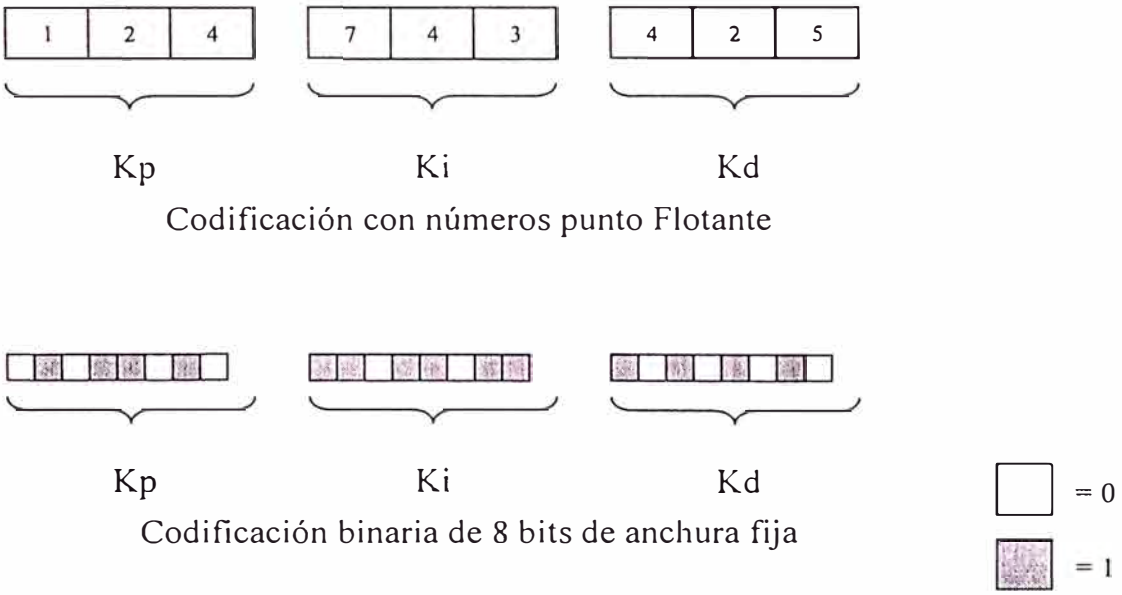

Figura (3.2) Distintos métodos de codificación del cromosoma

En este punto puede surgir la pregunta de ¿usar números binarios ó números con punto flotante?. Para responder esto es necesario conocer, que la representación binaria tradicionalmente usada en los AGs tiene algunas desventajas cuando se aplica a problemas de control multidimensionales y de gran precisión numérica. Por ejemplo, para 100 variables con dominios en los rangos [-500, 500] donde la precisión requerida es de 6 dígitos después del punto decimal, la longitud de cada variable es 30, es decir se requieren 29 dígitos binarios que representan el numero 500.000000 y un dígito que representa el signo, por lo tanto la longitud total del vector solución binario es de 3000,

lo cual a su vez genera un espacio de búsqueda de cerca 2^{3000}. Para este tipo de problemas los AGs evolucionan muy pobremente.

Experimentos realizados por investigadores [KOZA_5, MICHALEWICZ_8], indican que la representación en punto flotante es más rápido, más consistente de corrida a corrida, y provee una gran precisión (especialmente con grandes dominios, donde la codificación binaria requeriría, prohibitivamente, grandes representaciones.) Al mismo tiempo su desempeño puede aumentar por operadores especiales para alcanzar gran exactitud (aún más grande de lo que ofrece la representación binaria.) Es por ello que solo utilizaremos la codificación binaria para mostrar ejemplos sencillos, como en el caso de la sección 3.7, y ya para la implementación de control en tiempo real usaremos la codificación de punto flotante.

3.3 LA POBLACIÓN

Hay dos interpretaciones para el concepto de *individuo* y *población* en la optimización de controladores. La primera, habitualmente llamada *Pitts* [GOLDBERG_2], que es la más utilizada, y que nosotros seguiremos de aquí en adelante, entiende por *población* a un conjunto de individuos, en nuestro caso, cada individuo será un controlador PID completo.

La segunda interpretación, habitualmente llamada *Michigan* [GOLDBERG_2], consiste en considerar como individuo a las habilidades de un controlador, de forma que la población representa el conjunto de todas sus habilidades. Este tipo de interpretación se ha utilizado en el control de sistemas en tiempo real y mayormente en robótica industrial. Por ejemplo, en el caso de robótica, un individuo está formado por el conjunto de reglas que permiten al robot agarrar un objeto; la población está formada por individuos especializados en ciertas tareas, y durante la evolución se seleccionan los mejores para cada proceso.

Usualmente, los individuos de la primera población de un AG son generados por medio de un *muestreo aleatorio uniforme* dentro del espacio de las soluciones como se mostró anteriormente en la introducción. Sin embargo, algunas excepciones están contempladas en la literatura[JIMENEZ_13]:

- *Muestreo aleatorio no uniforme.* Existen algunas codificaciones para las cuales un algoritmo rápido de muestreo aleatorio uniforme puede resultar bastante complicado, usándose entonces algoritmos de muestreo aleatorio no uniforme

- *Muestreo aleatorio con información híbrida.* En ciertas aplicaciones prácticas puede resultar de gran importancia introducir en la población inicial buenas soluciones conocidas, obtenidas de algún otro método de resolución propio del problema, como

pueden ser las reglas de sintonización de controladores PID mencionadas en el capítulo anterior, con lo cual combinado con el operador *elitista*, se garantiza que el AG no obtendrá peores soluciones que el método propio. Sin embargo, esta técnica debe implementarse con cuidado, ya que si estas soluciones introducidas en la población están muy por encima de la adecuación media de la población, es bastante probable que ocurra convergencia prematura. Como solución al problema, estas buenas soluciones pueden introducirse cuando hayan transcurrido algunas generaciones.

- *Iniciación parcialmente enumerativa.* La motivación de este método es asegurar que todos los buenos esquemas necesarios para la solución están presentados en la población inicial, por lo que al menos un representante de éstos es generado.

El caso más frecuente, no obstante, consiste en partir de una población aleatoria, pues como se ha explicado antes, los AGs son especialmente indicados para realizar la búsqueda global de soluciones óptimas en un gran espacio.

El tamaño de la población, es una de las elecciones más importantes que encara cualquier usuario de los AGs, y puede ser crítico en muchas aplicaciones. Si el tamaño de la población es muy pequeño, los AGs pueden converger rápidamente; en cambio, si es demasiado grande, los AGs se desgasta mucho tiempo computacional en procesar a

todos los individuos. Como se dijo en la introducción de este capítulo son dos las consideraciones importantes en el proceso de evolución de la búsqueda genética: la diversidad de población y la presión selectiva. Claramente ambos factores están influenciados por el tamaño de la población. Para nuestra aplicación del motor DC que será tratado en el capítulo V, se ha determinado experimentalmente que un tamaño de población intermedio que va de los 50 a los 100 individuos funciona perfectamente, ya que poblaciones mayores consumen mucho tiempo, y en control lo que se requiere es disminuir dicho tiempo, y por ende disminuir el periodo de muestreo.

3.4 LA FUNCION DE EVALUACIÓN

La fase siguiente del proceso es la *evaluación*, en la cual se deja que cada uno de los controladores que forman la población actúe controlando el sistema, normalmente mediante una simulación en el caso del diseño de sistemas de control fuera de línea, siendo evaluados mediante una función de eficiencia o idoneidad (fitness.) Para este caso, la etapa de la simulación es normalmente la que más tiempo requiere, pues se han de simular cada uno de los controladores de la población durante el tiempo necesario para evaluar su eficiencia, y en un número suficiente de situaciones de control. Para el caso del diseño de sistemas de control en línea, como se verá en los siguientes capítulos, la función de idoneidad se evalúa utilizando un modelo linealizado o aproximado de la planta, en un numero de periodos de muestreo pequeño pero suficiente para poder predecir la dinámica de la planta utilizando cada individuo. La definición de esta función de eficiencia es fundamental en el éxito del uso de los AGs para la resolución de un

problema dado. A partir de ella, el diseñador decide qué comportamientos no se han de potenciar, cuáles sí, y en qué medida. Una descripción más detallada del diseño de esta función se verá también más adelante.

Una práctica ampliamente aceptada para mantener unos niveles de competición apropiados durante el proceso de evolución, consiste en el *escalado* de la adecuación bruta (valor de la función idoneidad, f_i'), con lo cual se previene el dominio de los superindividuos que pueden dar a lugar a la convergencia prematura. Entre los mecanismos de escalado más importantes se incluyen los siguientes[GOLDBERG_2, JIMÉNEZ_13]:

- *Escalado lineal.* Con este método la adecuación bruta f_i' de un cromosoma es escalada usando una ecuación lineal de la forma:

$$f''_i = a.f'_i + b$$

Donde los coeficientes a y b son normalmente seleccionados de forma que:

- El valor de la adecuación bruto será igual al valor de la adecuación escalada.
- La mejor adecuación sea un múltiplo (usualmente 2) de la adecuación media.

El escalado lineal trabaja bastante bien excepto cuando aparecen valores de adecuación negativos, los cuales deben de ser tratados de otra manera.

- *Truncamiento sigma.* Para tratar valores de adecuación negativos e incorporar información dependiente del problema, el truncamiento sigma calcula la nueva adecuación de la siguiente forma:

$$f'_i = f'_i + (\overline{f'} - c.\sigma')$$

Donde σ' es la desviación estándar de la población, y c es elegido como un múltiplo razonable de la desviación estándar de la población (entre 1 y 3.) Los resultados negativos ($f'' < 0$) son fijados a cero.

- *Escalado de potencia.* Con este método, la adecuación escalada se toma como alguna potencia especifica de la adecuación bruta:

$$f''_i = f_i'^K$$

Para algún k cercano a 1.

- *Escalado limitador.* A través de este método se fija un límite a la adecuación bruta que al principio del algoritmo puede iniciarse con valores grandes (debido a la

magnitud del error, en estado tran itorio por ejemplo, el cual se verá más adelante en el capítulo IV), y tiene la siguiente forma[MICHALEWICZ_8]:

$$f'_i = \frac{\alpha}{F(f_i) + \alpha}$$

Donde α determina el límite de una función de la adecuación bruta, generalmente dicha función es un factor cuadrático, siendo esta forma la que se usará en nuestro algoritmo de adaptatividad, y que se explicará detalladamente en el capítulo IV.

3.5 LA FUNCION DE SELECCIÓN

La fase siguiente es la de *selección*, en la cual se simula el proceso de selección natural de los individuos en cada generación. En este sentido, se debe de seleccionar qué individuos han de transmitir su genotipo a la generación siguiente. Dada una población de M individuos, son posibles diversas alternativas[MARTIN_7]:

a) *Sólo el mejor se selecciona.* De toda la población se elige al individuo que tiene una mejor evaluación, y solamente se emplea su genoma para crear la siguiente generación.
b) *Sólo los mejores se seleccionan.* De toda la población se eligen los n individuos (n<<M) que tienen una mejor evaluación, y para crear la siguiente generación solamente se utiliza su genoma.

c) *Todos pueden seleccionarse.* Cada individuo de la población se selecciona con una probabilidad mayor cuanto mejor sea su evaluación.

La primera de las alternativas permite una rápida convergencia a la solución, pero puede estancarse en un mínimo local. Ello resulta especialmente probable en aquellos problemas en los que se debe de alcanzar una solución de compromiso entre varios objetivos contrapuestos, ya que si se selecciona sólo el mejor, el genoma se empobrece (en cuanto a variedad), y los individuos pueden tender a especializarse en conseguir parte de los objetivos, dando malos resultados en el resto. Este problema puede resolverse con el uso de la segunda alternativa y una adecuada elección de *n/M*. La última alternativa es adecuada para buscar soluciones a problemas con un gran espacio de búsqueda y para los que no se conoce una buena aproximación a la solución; pero cuenta con el inconveniente de requerir más tiempo de cálculo y poblaciones mayores, por lo que en algunos casos resulta inaplicable.

Para nuestro caso, elegiremos la segunda alternativa, en la cual elegiremos los individuos que tengan la mejor evaluación, para utilizarlos en la siguiente generación.
Para la adecuada elección de estos individuos existen diversos métodos, siendo los más importantes la *selección por Rueda de Ruleta*, y la *selección por Torneo*, mismos que serán descritos a continuación.

3.5.1 Selección por Rueda de Ruleta

En este método la probabilidad de un individuo para reproducirse en la siguiente generación , es proporcional al valor de la función idoneidad evaluada de dicho individuo[MICHALEWICZ_8]. Esto se puede apreciar mejor en el siguiente algoritmo de la selección por Ruleta:

- Se calcula el valor de la función idoneidad $eval(v_i)$ para cada uno de los cromosomas v_i (i = 1,...,tamaño_pobl)

- Luego se encuentra la suma total de las funciones idoneidad, de toda la población.

$$F = \sum_{i=1}^{tamaño\_pobl} eval(v_i)$$

- Se calcula la probabilidad de selección p_i para cada cromosoma v_i (i = 1,...,tamaño_pobl):

$$p_i = \frac{eval(v_i)}{F}$$

- Se calcula la probabilidad acumulativa q_i para cada cromosoma v_i (i = 1,...,tamaño_pobl):

$$q_i = \sum_{j=1}^{i} p_j$$

El proceso de selección está basado en el giro de la Ruleta tamaño_pobl veces, cada vez se selecciona un solo cromosoma para una nueva población de la manera siguiente:

- Se genera un numero aleatorio (float) r que se encuentre en el rango [0...1]

- Si $r<q_1$ entonces se selecciona el primer cromosoma (v_1); de otra forma se selecciona la i-esima cromosoma v_i ($2 \leq i \leq tamaño\_pobl$), tal que $q_{i-1} < r \leq q_i$.

Obviamente, algunos cromosomas serán seleccionados más que otros, esto está en concordancia con la segunda alternativa de selección, donde los mejores cromosomas conseguirán más copias, el promedio todavía permanecerán, y los peores cromosomas morirán.

3.5.2 Selección por Torneo

Este procedimiento estocástico de selección fue propuesto a Brindle por Wetzel en 1983 [KOZA_5]. En este método (también llamado método de Ranking), se combina la idea de las categorías, en una manera interesante y eficiente. Este método (en una sola iteración) selecciona cierto número k de individuos y busca el mejor de este conjunto de

k elementos para usarlo en la siguiente generación [MICHALEWICZ_8]. Cuando dos toros luchan por el derecho de procrearse con una vaca determinada, la selección por torneo está ocurriendo. Este proceso es repetido tamaño_pobl número de veces. Está claro que grandes valores de k, se incrementa la presión selectiva de este procedimiento; valores típicamente aceptados por muchas aplicaciones son $k = 2$ (también llamado tamaño del torneo)

Goldberg propuso una forma interesante de encontrar el par de individuos seleccionados para el torneo. Esto es, las probabilidades de selección son calculadas normalmente y un sucesivo par de individuos son sustraídos de la población usando la selección de Rueda de Ruleta. Después de haber sustraído el par de individuos, aquel que tenga mayor valor de idoneidad será declarado ganador, siendo insertado en la nueva población, mientras que el otro individuo es sacado definitivamente. Este proceso, continua hasta que la nueva población este completa.

Una consideración adicional para encontrar soluciones óptimas dentro del espacio de búsqueda, es el operador *elitista* [MICHALEWICZ_8], en el cual se fuerza al mejor individuo de la población en cada generación a existir en la población de la siguiente generación. Aunque este modelo pueda incrementar el problema de la convergencia prematura, en la mayoría de los casos mejora el rendimiento del AG.

3.6 OPERADORES CRUZAMIENTO Y MUTACIÓN

Los *operadores* que se utilizan realizan modificaciones sobre el genoma, simulando la evolución del genoma de los seres vivos causada por la reproducción sexual y otros factores, como las mutaciones. Los principales operadores utilizados en los AGs son el cruzamiento y la mutación.

3.6.1 Operador Cruzamiento. El cruzamiento permite simular la combinación del genoma entre los individuos. Este puede realizarse sobre parejas de individuos o sobre toda la población. Cuando se realiza sobre parejas de individuos, mejor conocidos como *padres* y de acuerdo a la probabilidad de cruzamiento, al que llamaremos p_c, se copia el genoma en dos memorias; de forma aleatoria se selecciona un punto en esta memoria y se cruzan las dos mitades de cada copia de sus genotipos, como se puede apreciar en la figura (3.3)

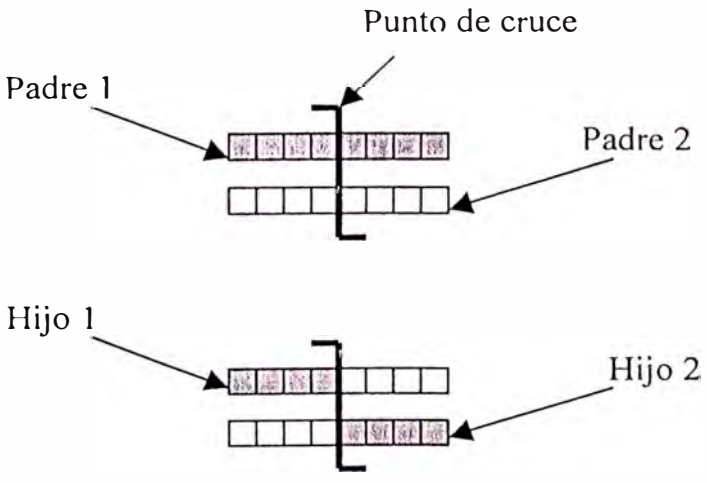

Figura (3.3) Cruzamiento del genoma entre individuos

El cruzamiento permite que las mejores cualidades de los padres se combinen en sus descendientes. En problemas en los que se pretende que el controlador haya de alcanzar simultáneamente varios objetivos, se espera que los descendientes en algún momento de la evolución puedan heredar de cada uno de los padres el conjunto de reglas que permiten lograr cada uno de los objetivos.

Una variante a la forma en que se cruzan los genomas de los individuos es que si se escogen por ejemplo 4 genes a cruzarse, estos se cruzan de forma aleatoria dentro del genoma. Esto se puede visualizar de la siguiente manera: si se tienen 2 cromosomas de longitud 10, [1111111111] y [3333333333], y se elige que se cruzan cuatros genes, está se puede operación se puede hacer de la siguiente manera, [1133313111] y [3311131333], lo cual representa un método de cruzamiento que no es estándar pero que ofrece buenos resultados y que ha demostrado que es más efectivo que el método tradicional de cruzamiento.

Por otro lado, sabemos no obstante, que durante la reproducción de los seres vivos se producen errores en la copia del genoma, a los que asignaremos una cierta probabilidad, que llamaremos p_m. Este efecto introduce diversidad del genoma, y permite explorar nuevas soluciones a los problemas. A este fenómeno se le conoce como mutación y dentro de los AGs se modela a través del operador Mutación que se detalla a continuación.

3.6.2 Operador Mutación. Este operador supone seleccionar un punto de mutación y modificar el parámetro almacenado con una probabilidad p_m. Si la codificación es de tipo binario se modifica uno de los bits, como se puede apreciar en la figura (3.4), mientras que si la codificación es de tipo punto fijo o punto flotante, se suele realizar un cambio aleatorio del gen que ha sido escogido para mutarse dentro de un rango limitado (por ejemplo de 0 a 9.)

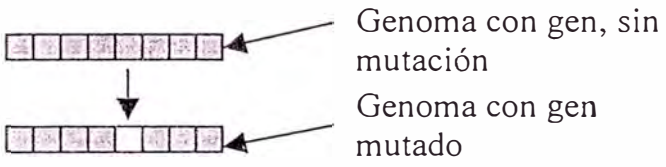

Figura (3.4) Operador mutación actuando sobre un punto del genoma

Según los trabajos realizados por De Jong's [GOLDBERG_2], la probabilidad de cruzamiento p_c se elige de tal manera, que se pueda llevar a cabo el máximo número de cruzamientos, es por eso que se elige un valor alto para p_c. Mientras que el valor de la probabilidad de mutación p_m, se elige en forma inversamente proporcional al tamaño de la población, lo cual nos dice que a mayor numero de individuos, menor es el efecto de las mutaciones.

Finalmente cada una de las funciones que definen a los conceptos presentados en este capítulo serán desarrollados en Matlab y se presentarán en el apéndice B.

3.7 EJEMPLO DE APLICACIÓN: Optimización de una Función Simple

En esta sección veremos en acción las características básicas de los AGs utilizando como ejemplo la optimización de una función simple de una sola variable. Esta función está definida como:

$$f(x) = x.\sin(10\pi.x) + 1$$

Y su gráfica se muestra en la figura (3.5). El problema consiste en encontrar el valor de x dentro del rango [-1..2], que maximize la función f, por ejemplo encontrar un x_0 tal que:

$$f(x_0) \geq f(x) \text{ para todos los } x \in [-1..2]$$

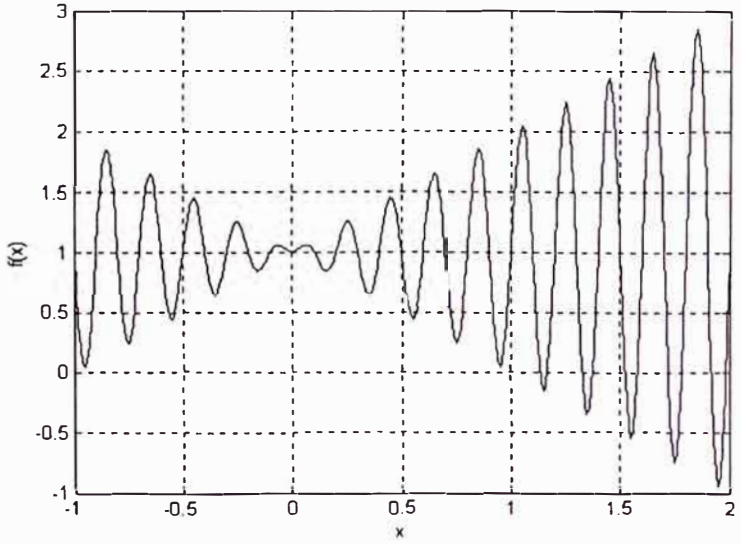

Figura (3.5) Gráfica de la función $f(x) = x.\sin(10\pi.x) + 1$

Podemos observar, que es relativamente fácil analizar la función *f*. Los ceros de la primera derivada *f'* podrían ser determinados:

$$f'(x) = \sin(10\pi.x) + 10\pi.x.\cos(10\pi.x) = 0$$

Dicha fórmula es equivalente a:

$$\tan(10\pi.x) = -10\pi.x$$

Es claro que las ecuaciones de arriba tienen un número infinito de soluciones.

$$x_i = \frac{2i-1}{20} + \varepsilon_i ...para...i = 1,2,...$$
$$x_0 = 0$$
$$x_i = \frac{2i-1}{20} + \varepsilon_i ...para...i = -1,-2,..$$

Donde los términos ε_i, representan secuencias decrecientes de números reales (para i = 1,2,.. y i = -1,-2,...) cercanas a cero.

Hay que notar también que *f* alcanza su máximo local para x_i, sí *i* es un entero impar, y su mínimo local para x_i, sí *i* es un entero par(ver figura (3.5))

Ya que el rango del problema es $x \in [-1..2]$, la función alcanza su valor máximo para dicho rango en $x_{19} = \frac{37}{20} + \varepsilon_{19} = 1.85 + \varepsilon_{19}$, donde $f(x_{19})$ es ligeramente más grande que $f(1.85) = 1.85\sin(18\pi + \frac{\pi}{2}) + 1 = 2.85$. Se asume que se quiere construir un AG para resolver el problema planteado arriba, por ejemplo hallar el máximo de dicha función f.

3.7.1 Representación

Usaremos un vector binario como un cromosoma, para representar valores reales de la variable x. La longitud del vector, depende de la precisión requerida, el cual, en este ejemplo, son seis lugares después del punto decimal.

El dominio de la variable x tiene una longitud de 3; los requerimientos de la precisión implican que el rango [-1..2] será dividido en al menos 3*1000000 partes igualmente espaciados. Esto significa que se requiere de 22 bits como un vector binario (cromosoma):

$$2097152 = 2^{21} < 3000000 \leq 2^{22} = 4194304$$

El mapeo de una palabra binaria $<b_{21}b_{20}...b_0>$ en un número real x del rango [-1..2] es sencillo y es completado en dos pasos:

- Convertir la palabra binaria $<b_{21}b_{20}...b_0>$ de la base 2 a la base 10:

$$(< b_{21}b_{20}...b_0 >)_2 = (\sum_{i=0}^{21} b_i 2^i)_{10} = x'$$

- Encontrar el correspondiente número real x:

$$x = -1 + x' \frac{3}{2^{22} - 1}$$

Donde -1 es el límite izquierdo del dominio y 3 es la longitud de dicho dominio. Por ejemplo, un cromosoma (1000101110101000111) representa el número 0.637197, debido a que:

$$x' = (1000101110101000111)_2 = 2288967$$

y

$$x = -1 + 2288967 \frac{3}{4194303} = 0.637197$$

Está claro que los cromosomas:

(00000000000000000000) y (11111111111111111111)

Representan los límites del dominio, -1 y 2 respectivamente.

3.7.2 Población Inicial

El proceso de iniciación es muy simple: Nosotros creamos una población de cromosomas, donde cada cromosoma es un vector binario de 22 bits. Todos los bits para cada cromosoma son inicializados aleatoriamente.

3.7.3 Función de Evaluación

La función idoneidad de evaluación *eval* para cada vector binario v es equivalente a la función f:

$$eval(v)=f(x)$$

Donde el cromosoma v representa el valor real de x.

Como se vio anteriormente, la función idoneidad de evaluación juega el papel del medio ambiente, evaluando soluciones potenciales en términos de su conveniencia. Por ejemplo, tres cromosomas:

$$v_1 = (1000101110110101000111),$$

$$v_2 = (0000001110000000010000),$$

$$v_3 = (1110000000111111000101),$$

Corresponden a los valores x_1 = 0.637197, x_2 = -0.958973, y x_3 = 1.627888, respectivamente. Consecuentemente, la función idoneidad los evaluará como sigue:

$$eval(v_1) = f(x_1) = 1.586345,$$

$$eval(v_2) = f(x_2) = 0.078878,$$

$$eval(v_3) = f(x_3) = 2.250650,$$

Claramente, el cromosoma v_3, es el mejor de los tres cromosomas, ya que su función idoneidad retorna el valor más alto.

3.7.4 Operadores Genéticos

Durante la fase de alteración de los AGs nosotros podemos usar dos operadores genéticos clásicos: mutación y cruzamiento. Como se mencionó anteriormente, la mutación altera uno o más genes con una probabilidad igual a la razón de mutación. Se asume que el quinto gen del cromosoma v_3, fue seleccionado para mutarse. Desde que el quinto gen en este cromosoma es 0, lo podemos cambiar a 1. Entonces el cromosoma v_3, después de la mutación será:

$$v_3' = (11101000011111111000101),$$

Este cromosoma representa el valor de $x_3' = 1.721638$ y $f(x_3') = -0.082257$. Esto significa que esta mutación en particular resulta en un significativo decrecimiento del valor del cromosoma v_3. Por otra parte, sí el décimo gen fuese seleccionado para mutarse en el cromosoma v_3, entonces:

$$v_3'' = (11100000\mathbf{1}1111111000101),$$

El valor correspondiente de $x_3'' = 1.630818$ y $f(x_3'') = 2.343555$, que constituye un mejoramiento sobre el valor original de $f(x_3) = 2.250650$.

Seguidamente vamos a ilustrar el uso del operador cruzamiento sobre los cromosomas v_2 y v_3. Se asume que el punto de cruzamiento, fue (aleatoriamente) seleccionada después del 5to gen:

$$v_2 = (00000 \mid 01110000000010000),$$
$$v_3 = (11100 \mid 00000111111000101),$$

Los dos descendientes resultantes son:

$$v_2' = (00000 \mid 00000111111000101),$$
$$v_3' = (11100 \mid 01110000000010000),$$

La evaluación de estos descendientes da como resultado:

$$f(v_1') = f(-0.998113) = 0.940865,$$
$$f(v_2') = f(1.666028) = 2.459245,$$

Hay que notar que el segundo descendiente tiene una mejor evaluación que cualquiera de sus padres.

3.7.5 Parámetros

Para este problema en particular y de acuerdo a los criterios de elección de parámetros dado en la sección 3.3 y 3.5, hemos usado los siguientes: tamaño de población *tamaño_pobl* = 50, probabilidad de cruzamiento p_c = 0.9, probabilidad de mutación p_m = 0.02. A continuación se presenta algunos resultados experimentales obtenidos para la resolución del problema planteado, en el software Matlab 6.0.

3.7.6 Resultados Experimentales

En la tabla (3.1) se proporciona el número de generaciones, que conforme van evolucionando, se puede notar el mejoramiento del valor de la función idoneidad de evaluación.

El mejor cromosoma después de 150 generaciones fue:

$$v_{max} = (11110011010001000000101)$$

El cual corresponde al valor $x_{max} = 1.850773$.

Como se esperaba, $x_{max} = 1.85 + \varepsilon$, y $f(x_{max})$ es ligeramente mas grande que 2.85

Número de Generaciones	Función de Evaluación
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

Tabla 3.1 Resultado de 150 generaciones

CAPITULO IV

CONTROL ADAPTATIVO

4.1 ¿QUÉ ES EL CONTROL ADAPTATIVO?

En este capítulo hablaremos de la implementación de un controlador PID adaptativo simple, así como de las metodologías y leyes adaptivas que nos permitan controlar procesos no lineales. Luego desarrollaremos la técnica de diseño adaptativa usando los AGs, y mostraremos el seudo código que rige este algoritmo.

Pero antes necesitamos saber ¿qué es el control Adaptativo.? El término *adaptativo* significa cambiar el comportamiento conforme a nuevas circunstancias. Un regulador adaptativo es un regulador que puede modificar su comportamiento en respuesta a cambios de la dinámica del sistema y a las perturbaciones. Este mismo objetivo es el de la inclusión de la realimentación en el bucle de control, por lo que surge la pregunta de ¿cuál es la diferencia entre control realimentado y control adaptativo?.

Existen muchas definiciones de control adaptativo, siendo una de las más aceptadas, que control adaptativo es un tipo especial de control no lineal en el que el estado del proceso puede ser separado en dos escalas de tiempo que evolucionan a diferente velocidad. La escala lenta corresponde a los cambios de los parámetros y por consiguiente a la

velocidad con lo cual los parámetros del regulador son modificados, y la escala rápida que corresponde a la dinámica del bucle ordinario de realimentación.

El esquema básico de control adaptativo (Landau 1974) [RODRIGUEZ_10], según puede verse en la figura (4.1), está compuesto de un bucle principal de realimentación negativa, en el que actúa al igual que en los sistemas convencionales un regulador y de otro bucle en el que se mide cierto índice de funcionamiento, el cual es comparado con el índice deseado y se procesa el error en un mecanismo de adaptación que ajusta los parámetros del regulador y en algunos casos actúa directamente sobre la señal de control. También puede existir un tercer bucle dedicado a supervisar la marcha de los bucles anteriores (Isermann 1982) [RODRIGUEZ_10], para asegurar la estabilidad del sistema y a mejorar la actuación del conjunto.

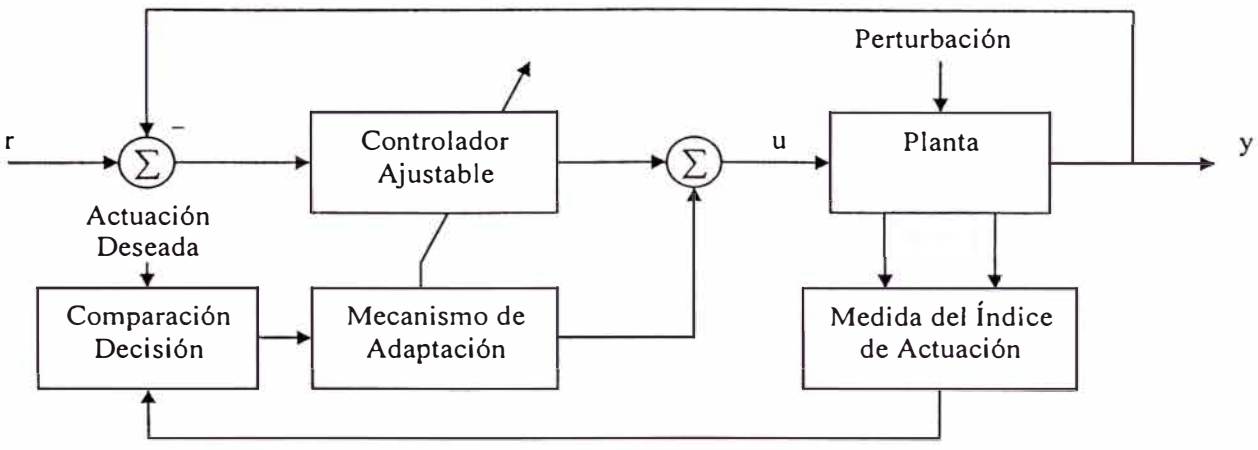

Figura (4.1) Configuración básica del control Adaptativo

El mecanismo de adaptación presenta una solución en tiempo real al problema de diseño para sistemas con parámetros conocidos, o aproximados mediante algún algoritmo de identificación de parámetros.

Los controladores adaptativos se dividen principalmente en dos grupos: Controladores adaptativos con *Modelo de Referencia* (MRAC de sus siglas en inglés, Model Reference Adaptive Control) y *Reguladores Autoajustables* (STR de sus siglas en inglés, Self-Tuning Regulators), aunque también, podemos mencionar el control por Cambio de Tabla (Gain Scheduling en inglés), que es un método también muy utilizado.

MRAC y STR pueden ser considerados como una aproximación a la solución del problema de control adaptativo. La hipótesis que justifica la aproximación es que para cualquier juego de valores posibles de los parámetros de la planta y las perturbaciones, existe un controlador lineal con una complejidad fijada, tal que el conjunto de controlador y planta tienen características pre-especificadas. Para ambos casos se tiene:

- Los controladores adaptativos con modelo de referencia (MRAC), intentan alcanzar, para una señal de entrada definida, un comportamiento en bucle cerrado, dado por un modelo de referencia.

- Los reguladores adaptativos auto ajustables (STR), tratan de alcanzar un control óptimo, sujeto a un tipo de controlador y a obtener información del proceso y sus señales.

Adicionalmente podemos mencionar el control por Cambio de Tabla, el cual consiste en la modificación de los parámetros del controlador a partir de una tabla que ha sido calculada previamente para distintos puntos de funcionamiento, en función de una variable auxiliar. Un caso típico es el control de vuelo de un avión, cuyo regulador puede ser cambiado en función de la altura de éste.

Nosotros nos centraremos en el estudio de los controladores adaptativos con Modelo de Referencia, debido a su rápida adaptación para una entrada definida y a la simplicidad de tratamiento de la estabilidad.

4.2 DISEÑO DE CONTROLADORES ADAPTATIVOS POR MODELO DE REFERENCIA

El control adaptativo por Modelo de Referencia MRAC, es una de las dos principales técnicas usadas en el Control Adaptativo, que está basado en el *seguimiento de modelo*. Esto es, los parámetros del controlador son ajustados de tal manera, que el comportamiento en lazo cerrado se acerque al del modelo de referencia prescrito.

El esquema básico del Control Adaptativo por Modelo de Referencia (que en adelante lo llamaremos por sus siglas en español CAMR), es ilustrado en la figura (4.2.) En esta figura, y_p es la salida del proceso a ser controlado. El rendimiento deseado, se encuentra expresado en términos del *modelo de referencia*, el cual da la respuesta deseada y_m a una señal de Referencia r. El error y_m-y_p es establecido en línea y es monitoreada continuamente a través del bloque llamado "mecanismo de ajuste". Este bloque genera los parámetros del controlador basados en el error que se presenta en el sistema Adaptativo. Por lo tanto he de esperarse que muchas reglas de actualización sean necesarias de especificar en este bloque.

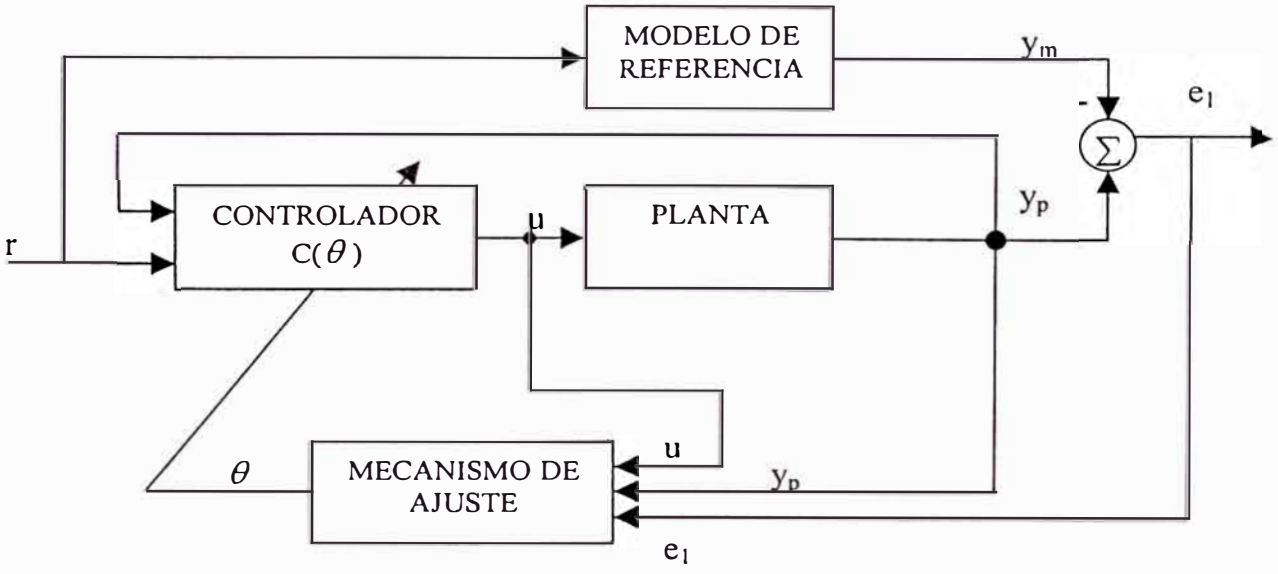

Figura (4.2) Estructura general del esquema CAMR

Los esquemas CAMR pueden ser clasificados como *directos* o *indirectos* y con leyes adaptivas *normalizadas* y *sin normalizar*. En CAMR directos, el vector de parámetros θ del controlador $C(\theta)$ es actualizado directamente por una ley adaptiva, mientras que en CAMR indirectos, θ es calculada en cada tiempo t, resolviendo una cierta ecuación algebraica que relaciona θ con los parámetros del proceso estimados en línea. En ambos esquemas normalizados CAMR, directo e indirecto, la forma de $C(\theta)$ permanece invariable, debido al conocimiento de los parámetros. Dicho controlador $C(\theta)$ es combinado con una ley adaptiva (o una ley adaptiva y una ecuación algebraica en el caso indirecto) que es desarrollado independientemente siguiendo técnicas de estimación de parámetros en línea. Este procedimiento de diseño se realiza mediante una gran cantidad de leyes adaptivas que incluyen el método del gradiente, mínimos cuadrados y aquellos basados en el diseño de estabilidad Real Estrictamente Positiva de Lyapunov. Por otra parte, en el caso de los esquemas CAMR con leyes adaptivas sin normalizar, $C(\theta)$ es modificado para llegar a una ecuación de error cuya forma muestra el uso del diseño de la estabilidad Real Estrictamente Positiva de Lyapunov para generar la ley adaptiva. En este caso, el diseño de $C(\theta)$ y una ley adaptiva es más complicada en ambos casos, directo o indirecto, pero el análisis es más simple. Todos estos métodos han sido ampliamente descritos en la literatura de control adaptativo [ASTROM 1, IOANNOU_4, GHANADAN_12, SLOTINE_11, etc] para el investigador que quiera profundizar en el tema.

Otros autores [RODRÍGUEZ_10] han resumido estas teorías en tres grupos. Él menciona que la primera ley de control adaptativo hacía uso de los modelos de sensibilidad y más tarde de la teoría de estabilidad de Lyapunov y de la teoría de hiperestabilidad de Popov. Estas últimas garantizan la estabilidad del conjunto del sistema, lo que las hace particularmente interesantes, convirtiéndose en métodos de diseño estándar.

Para diseñar un sistema de control adaptativo, nosotros sabemos, por todo lo expuesto anteriormente, que básicamente está formado por tres partes: *un controlador primario*, *un modelo de referencia* y *la ley de adaptación*. Por lo tanto para el diseño de un sistema de control adaptativo, será necesario definir las tres partes. Además dado que la parte que caracteriza al control adaptativo es la ley de adaptación, en lo que sigue nos centraremos fundamentalmente en esa parte.

4.2.1 Controlador Primario

El controlador primario puede tener en principio cualquiera de las configuraciones conocidas para el diseño de controladores lineales. Sin embargo, debe de cumplir la condición de que sea posible que el conjunto del proceso y el controlador puedan reproducir al modelo de referencia. Este requisito, supone restricciones sobre el orden y la estructura del controlador. Por otro lado y esto es muy importante, para que pueda

aplicarse una adaptación directa, la señal de control debe de ser una función lineal de los parámetros.

4.2.3 Modelo de Referencia

El modelo de referencia, que especifica el comportamiento deseado en bucle cerrado, se da usualmente en forma paramétrica. La condición mencionada anteriormente para el seguimiento del modelo de referencia, también condiciona en ciertos aspectos el modelo de referencia posible, en cuanto al orden relativo del proceso (exceso de polos). Por otro lado el modelo elegido debe de ser sensible a la dinámica del proceso, ya que si por ejemplo se elige un modelo con una dinámica muy rápida, la señal de control será muy grande causando saturaciones y no pudiendo el sistema responder bien a dicha dinámica. Por ello la elección del modelo de referencia no es fácil, eligiéndose normalmente un modelo conservador. Nosotros para todos nuestros casos elegiremos modelos de referencia de segundo grado con una dinámica medianamente rápida de acuerdo a la dinámica de los procesos a controlar. A continuación definiremos cada uno de los tres métodos clásicos[RODRÍGUEZ_10], y compararemos la eficiencia de cada uno.

- *Método de la Sensibilidad*

  Este método está basado en el uso de los modelos de sensibilidad para adaptar los parámetros en la dirección correcta. La deducción de este método comienza con el planteamiento de un índice de actuación (función costo), normalmente cuadrático:

$$J(t+T) = \int^{+T} e_1^2(\tau) d\tau \qquad (4.1)$$

En esta ecuación $e_1 = y_p - y_m$, y el índice J se evalúa sobre el periodo T fijo, en el cual los parámetros permanecen constantes. En el instante (t + T) los parámetros son ajustados en la dirección decreciente de J.

$$\theta(t+T) = \theta(t) - \Gamma \frac{\partial J}{\partial \theta} = \theta(t) - \Gamma \int^{+T} 2e_1(\tau) \frac{\partial e_1(\tau)}{\partial \theta} d\tau \qquad (4.2)$$

Donde θ representa los parámetros del controlador y Γ debe de ser una matriz cuadrada definida positiva, que representa la ganancia de adaptación. Usualmente es diagonal, teniendo en cuenta que:

$$\frac{\partial e_1(\tau)}{\partial \theta} = \frac{\partial y_p}{\partial \theta}$$

La ecuación (4.2) queda:

$$\frac{\theta(t+T) - \theta(t)}{T} = -\frac{1}{\Gamma} \int^{+T} 2e_1(\tau) \frac{\partial y_p}{\partial \theta} d\tau \qquad (4.3)$$

Que en el límite para T → 0 da la ley de adaptación:

$$\frac{d\theta}{dt} = -2\Gamma e_1 \frac{\partial y_p}{\partial \theta} \tag{4.4}$$

El factor $\partial y_p / \partial \theta$ representa la sensibilidad de la salida del proceso a las variaciones en θ, y puede ser generado por un modelo de sensibilidad. A esta ley de adaptación también se le conoce como la *regla del MIT*.

El modelo de sensibilidad anterior se sustituye por la sensibilidad del modelo de referencia, dado que normalmente el modelo del proceso se desconoce. Esta suposición se hace basándose en que pasado un tiempo la respuesta del sistema converge a la respuesta del modelo de referencia. La principal desventaja de este método, es la ausencia de un criterio que garantice la estabilidad del sistema de control. El sistema puede hacerse inestable si el modelo de referencia no se escoge adecuadamente o si la ganancia de adaptación se elige demasiado grande.

- *Método de Lyapunov*

  Dado el carácter no lineal y variable en el tiempo de los sistemas adaptativos por modelo de referencia MRAC, no son válidos los criterios de estabilidad de sistemas lineales. Un método bien conocido es el método directo de Lyapunov. Este método establece que un sistema tiene un equilibrio x = 0, asintoticamente estable, si existe una función, llamada de Lyapunov, V(x) que satisface:

$$
\begin{aligned}
&V(x) > 0, \text{ para } x \neq 0 \text{ definida positiva} \\
&V(x) < 0, \text{ para } x \neq 0 \text{ definida negativa} \\
&V(x) \to \infty, \text{ para } \|x\| \to \infty \\
&V(0) = 0
\end{aligned}
\tag{4.5}
$$

Como la función de Lyapunov es similar a una función de energía, ésta debe decrecer con el tiempo. Utilizando este método en el diseño de sistemas adaptativos se trasladan las especificaciones de estabilidad directamente en la ley de adaptación siguiendo los pasos:

- El primer paso es encontrar la *ecuación de error*, bien en la salida $(y_p - y_m)$ o en las variables de estado $(x_p - x_m)$

- Encontrar una función de Lyapunov como una función del error entre las señales y del error en los parámetros $(\phi = \theta - \theta)$. En su forma más simple esta función toma la forma:

$$V = e^T P e + \phi^T \Gamma^{-1} \phi$$

Donde las matrices P y Γ^{-1} deben ser definidas positivas.

- Calcular la derivada de la función de Lyapunov. La derivada debe ser definida negativa. Generalmente toma la forma:

$$V = -e^T Qe + \text{algunos términos incluyendo } \phi$$

El primer término garantiza que la derivada es definida negativa, por lo que, haciendo el resto igual a cero, se tiene una posible solución. La matriz Q es definida positiva. Las matrices P y Q, para un sistema gobernado por una matriz A, están relacionados por la ecuación de Lyapunov:

$$-Q = A^T P + PA \tag{4.6}$$

- Haciendo el término extra igual a cero se obtiene la ley de adaptación. Normalmente tiene la forma:

$$\theta = -\Gamma \varepsilon \xi \tag{4.7}$$

Donde ε está directamente relacionado con el error e y ξ es una versión modificada del vector de señales (referencia, salida, etc.)

- *Método de Hiperestabilidad*

  Con este método también se consigue una ley de adaptación estable. Como en el método de Lyapunov, en primer lugar se formulan las ecuaciones de error. Estas ecuaciones se dividen en una parte lineal invariante con el tiempo y otra no lineal y variable con el tiempo. La primera parte contiene usualmente al modelo de referencia y su salida es la señal de error que es utilizada para la ley de adaptación. La segunda parte contiene la ley de adaptación y su salida negada es la entrada a la parte lineal.

  La teoría de hiperestabilidad garantiza la estabilidad asintótica sí ambas partes (lineal y no lineal), satisfacen ciertas condiciones de pasividad dadas por Landau [RODRÍGUEZ_10, SLOTINE_11] quien en 1979 propuso esta teoría.

  - La parte lineal (llámese *G(s)*), debe ser estrictamente positiva. Ello significa que:
    a) *G(s)* debe ser real si s es real.
    b) Los polos de *G(s)* deben tener parte real negativa.
    c) La parte real de *G(jw)* debe ser mayor que cero para $-\infty < w < \infty$

  - La parte no lineal debe cumplir la desigualdad de Popov [RODRÍGUEZ_10, SLOTINE_11].

$$\int_0^t v^T w.dt \geq -\gamma^2 ; \quad \forall t > 0$$

Siendo v el vector de entrada y w el vector de salida de la parte no lineal, y γ^2 una constante finita positiva que no depende de t.

Todos los conceptos anteriormente tratados se manejan para cualquier tipo de diseño de controladores, compensadores, por lo que ahora nos centraremos en un caso especial, es decir en el uso de los controladores PID en el control adaptativo y su aplicación a cierto tipo de no linealidades.

4.3 CONTROL PID ADAPTATIVO PARA SISTEMAS NO LINEALES

A continuación mostraremos un esquema de diseño que utiliza controladores PID simples y metodologías adaptivas, basadas en la teoría de estabilidad de Lyapunov y en el seguimiento de un Modelo de Referencia, para el control de sistemas no lineales SISO con ciertas características que discutiremos a continuación.

4.3.1 Descripción del Sistema

Aunque los controladores PID tienen una estructura simple, ellos pueden controlar eficientemente la mayoría de los sistemas industriales. En general, y como fue discutido en el capítulo 2, el control PID es suficiente para procesos donde la dinámica dominante

es de *segundo orden*. Debido a que los controladores PID tienen limitada complejidad, los sistemas más complicados con grandes dinámicas de frecuencia deberían ser controlados por controladores con algoritmos más sofisticados. En este caso se analiza una técnica de sintonización PID adaptiva, para una clase de sistemas con dinámica no lineal modelada o aproximada por la siguiente ecuación diferencial no lineal y que varia en el tiempo [GHANADAN_12]:

$$\ddot{y}_p = a_{p1}(y_p, \dot{y}_p; t).\dot{y}_p + a_{p0}(y_p, \dot{y}_p; t).y_p + f(\sigma; t) + b_p(y_p, \dot{y}_p; t).u + v \qquad (4.1)$$

$$\sigma = g(y_p, \dot{y}_p, \ddot{y}_p, \ldots) \ , \ b_p \neq 0$$

$$|v| \leq v_{max} \qquad (4.2)$$

Donde y_p es la salida del proceso, u es la señal de control, a_{p0}, a_{p1}, y b_p son coeficientes no lineales que varían con el tiempo, v es una perturbación, g(...) y f(...) son funciones no lineales desconocidas. Sin embargo se asume que existe una función mensurable $\rho(y_p, \dot{y}_p; t)$ tal que:

$$|f(\sigma; t)| \leq \rho(y_p, \dot{y}_p; t) \qquad \forall \sigma, y_p, \dot{y}_p, t \qquad (4.3)$$

El modelo de referencia, especificado por el diseñador, el cual describe el comportamiento esperado del proceso controlado, es identificado por la siguiente ecuación diferencial de segundo orden invariante en el tiempo y asintoticamente estable:

$$\ddot{y}_m = a_{m1}\dot{y}_m + a_{m0}y_m + b_m r \tag{4.4}$$

Donde r es la señal de referencia que actúa como entrada en el modelo de referencia.

4.3.2 Notaciones y Formulación del Problema

Se define los vectores de estado como:

$$x_p = \begin{bmatrix} x_{p1} \\ x_{p2} \end{bmatrix} \text{ cuya salida es } \begin{bmatrix} y_p \\ \dot{y}_p \end{bmatrix}, \quad x_m = \begin{bmatrix} x_{m1} \\ x_{m2} \end{bmatrix} \text{ cuya salida es } \begin{bmatrix} y_m \\ \dot{y}_m \end{bmatrix}$$

y el vector de errores como:

$$\bar{e} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \text{ que es definido por } x_m - x_p = \begin{bmatrix} y_m - y_p \\ \dot{y}_m - \dot{y}_p \end{bmatrix}$$

rescribiendo las ecuaciones (4.1) y (4.4) tenemos:

$$\dot{x}_p = A_p(x_p;t).x_p + F(\sigma;t) + B_p(x_p;t).u + D \tag{4.5}$$

$$\dot{x}_m = A_m.x_m + B_m.u_c \tag{4.6}$$

Donde:

$$A_p(x_p;t) = \begin{bmatrix} 0 & 1 \\ a_{p0}(x_p;t) & a_{p1}(x_p;t) \end{bmatrix}, \quad F(\sigma;t) = \begin{bmatrix} 0 \\ f(\sigma;t) \end{bmatrix}$$

$$B_p(x_p;t) = \begin{bmatrix} 0 \\ b_p(x_p;t) \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ v \end{bmatrix}$$

$$A_m = \begin{bmatrix} 0 & 1 \\ a_{m0} & a_{m1} \end{bmatrix}, \quad B_m = \begin{bmatrix} 0 \\ b_m \end{bmatrix}$$

En el lazo de control de la figura (4.3), hay varias señales (tales como e_i, y_p, y_m, u_c,...) que se encuentran disponibles para el diseñador, lo cual permite al controlador PID adaptativo tener diferentes combinaciones de los términos definidos en la ecuación (2.5). Además D representa la función derivativa y ε = G(e1,e2) se definirá mas adelante.

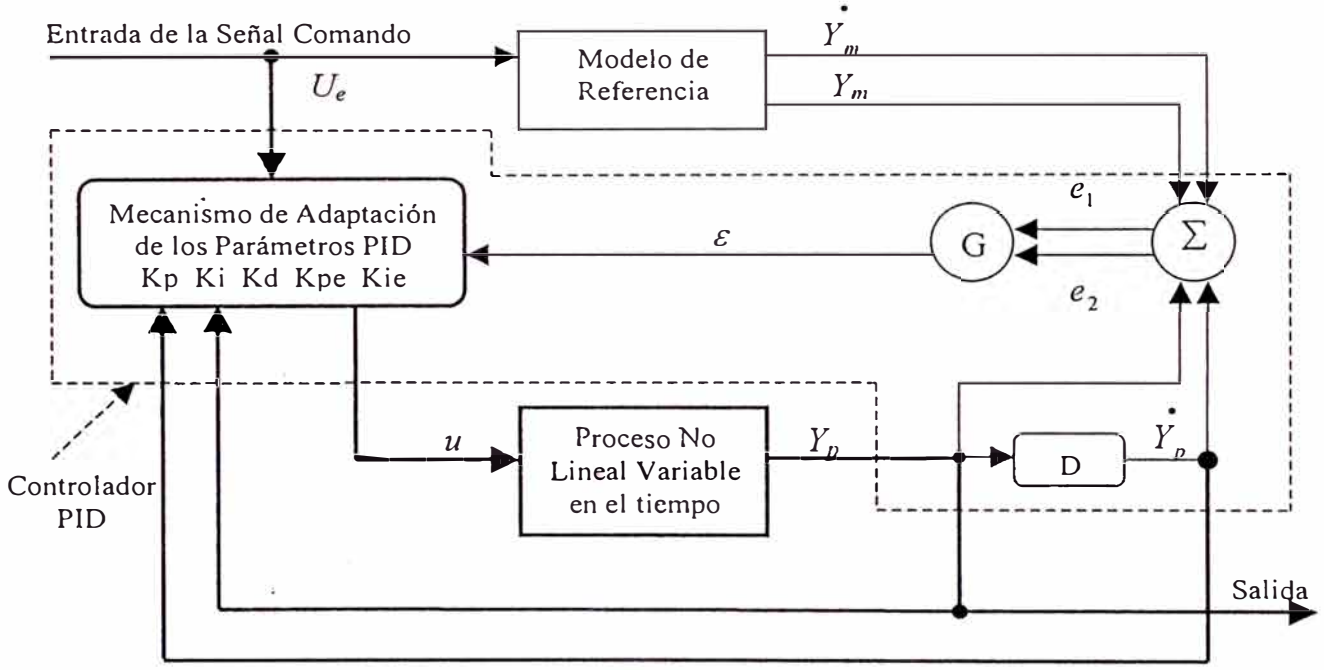

Figura 4.3 Controlador PID Adaptativo con Modelo de Referencia

Por lo tanto, un controlador PID adaptativo tiene más grados de libertad que los reguladores convencionales de error que fueron descritos el capítulo dos. El nuevo regulador PID adaptativo tiene la siguiente forma:

$$u(t) = Py_p + Dy_p + Pu_c + P\varepsilon + I\varepsilon \tag{4.7}$$

Donde ε es una función de e_1 y e_2 llamada el *"error ponderado"*. En esta última ecuación se puede apreciar que la acción proporcional ha sido separada en dos partes, una que afecta a la señal de referencia y otra que afecta a la salida del sistema, por otro lado, en la acción derivativa se considera solo el efecto de la señal de salida del sistema

y no del error producida por la sustracción de la señal de referencia, todos estas variantes han sido adicionadas para un mejor rendimiento, y fueron explicadas en la sección 2.5

El objetivo del controlador PID en la figura (4.3), es garantizar que el error

$$\bar{e}(t) = \begin{bmatrix} y_m - y_p \\ \cdot \quad \cdot \\ y_m - y_p \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ conforme } t \rightarrow \infty$$

Sustrayendo la ecuación (4.5) de la ecuación (4.6), podemos observar que el vector de errores $\bar{e}$ satisface la siguiente ecuación vectorial diferencial:

$$\dot{\bar{e}} = A_m.\bar{e} + (A_m - A_p).x_p - F(\sigma) - B_p.u - D + B_m.u_c \qquad (4.8)$$

La ecuación (4.8) representa un sistema no lineal variable con el tiempo con u definida en la ecuación (2.4), y es conocida como *"el espacio de error"*[GHANADAN_12], la cual describe la evolución del error de proceso con respecto a la trayectoria deseada. El problema consiste en encontrar las respectivas reglas de actualización para los parámetros PID dados en la ecuación (4.7): Kp_{yp}, Kd_{yp}, Kp_{uc}, Kp_{ε} y Ki_{ε}, tal que la solución de la ecuación (4.8) tienda asintoticamente a cero.

4.3.3 Diseño PID Adaptativo

Ghanadan diseñó un esquema PID adaptativo utilizando las consideraciones adicionales dentro del control PID [GHANADAN_12], que se mencionaron en la sección 2.5, y obtuvo la siguiente equivalencia del control PID adaptativo para los sistemas no lineales y variables con el tiempo cuya dinámica se dio en la descripción del sistema, reemplazándose la ecuación (4.7) de la siguiente manera:

$$u(t) = Py_p + Dy_p + Pu_c + P\varepsilon + I$$

$$= Kp_{yp} y_p + Kd_{yp} \dot{y}_p + Kp_{uc} u_c + Kp_{\varepsilon} sat(\varepsilon, \xi_0) + sat(I\varepsilon, c)$$

Donde los términos son definidos como sigue a continuación:

$$\begin{cases} Kp_{yp} = \dfrac{a_{m0} - a_{p0}}{b_p} \\ Kd_{yp} = \dfrac{a_{m1} - a_{p1}}{b_p} \\ Kp_{uc} = \dfrac{b_m}{b_p} \\ Kp_{\varepsilon} = \dfrac{\xi(x_p;t)}{\xi_0} \\ Ki_{\varepsilon} = 2\gamma \end{cases} \tag{4.9}$$

y

$$\xi(x_p;t) = \frac{\rho(x_p;t) + v_{\max}}{|b_p(x_p;t)|} + c \tag{4.10}$$

Además la función $sat(x, y)$ es definida como:

$$sat(x,y) = \begin{cases} y \rightarrow si.....x > y \\ x \rightarrow si.....|x| \leq y \\ -y \rightarrow si.....x < -y \end{cases}$$

Los parámetros c y ζ_0, son los límites de saturación y un parámetro de diseño respectivamente. Con este último se pueden probar distintas alternativas para que el error $\bar{e}$ cambie, de acuerdo a las expectativas de diseño. Los demás parámetros a_{mi}, a_{pi}, b_m y b_p son dados en las ecuaciones (4.1) y (4.4)

Este esquema fue probado exitosamente, aunque como se mencionó anteriormente solo funciona con sistemas donde se conoce el valor límite de sus no linealidades, mientras que para otro tipo de linealidades no opera de manera satisfactoria. Es por eso que necesitamos explorar otro tipo de técnicas o una combinación de las mismas como en este caso, la teoría de control adaptativo con los Algoritmos Genéticos, para poder cubrir más posibilidades de control de distintos sistemas lineales y no lineales y con un gran rendimiento y robustez.

4.4 CONTROL ADAPTATIVO GENÉTICO

Los conceptos de control adaptativo, anteriormente mencionados, se pueden combinar con los AGs para dar forma a una técnica de control que se basa en la adaptación del controlador a los cambios en el proceso, mediante la selección, la evolución y la colocación de los mejores controladores, los cuales son evaluados por cierta función idoneidad, que será desarrollada en este capítulo, de acuerdo a las características de nuestro proceso. Pero para poder saber que controladores son los más efectivos, podemos predecir el uso de estos en un modelo lineal aproximado del sistema y compararlos con el modelo de referencia, para de esta manera saber cuales serán seleccionados para evolucionar a la siguiente generación. Durante cada periodo de muestreo, cada miembro de la población es evaluado sobre la base de cuan bien minimiza el error entre la respuesta en lazo cerrado del sistema predicho y la salida predicha del modelo de referencia.

Se podría decir que esta técnica se acerca también a las características básicas del Control Predictivo, ya que es necesario adelantarnos y ver más allá para poder realizar la mejor elección. El esquema que desarrollaremos, es un tipo de controlador PID genético adaptativo con Modelo de Referencia (CGAMR) como se muestra en la figura (4.4) y donde podemos apreciar que dentro del AG se usa el modelo lineal aproximado del sistema, que nos sirve para predecir la salida del sistema usando cada uno de los controladores que conforman la población y de esta manera mediante una función de evaluación, elegir el mejor controlador, el cual se usará para controlar el proceso.

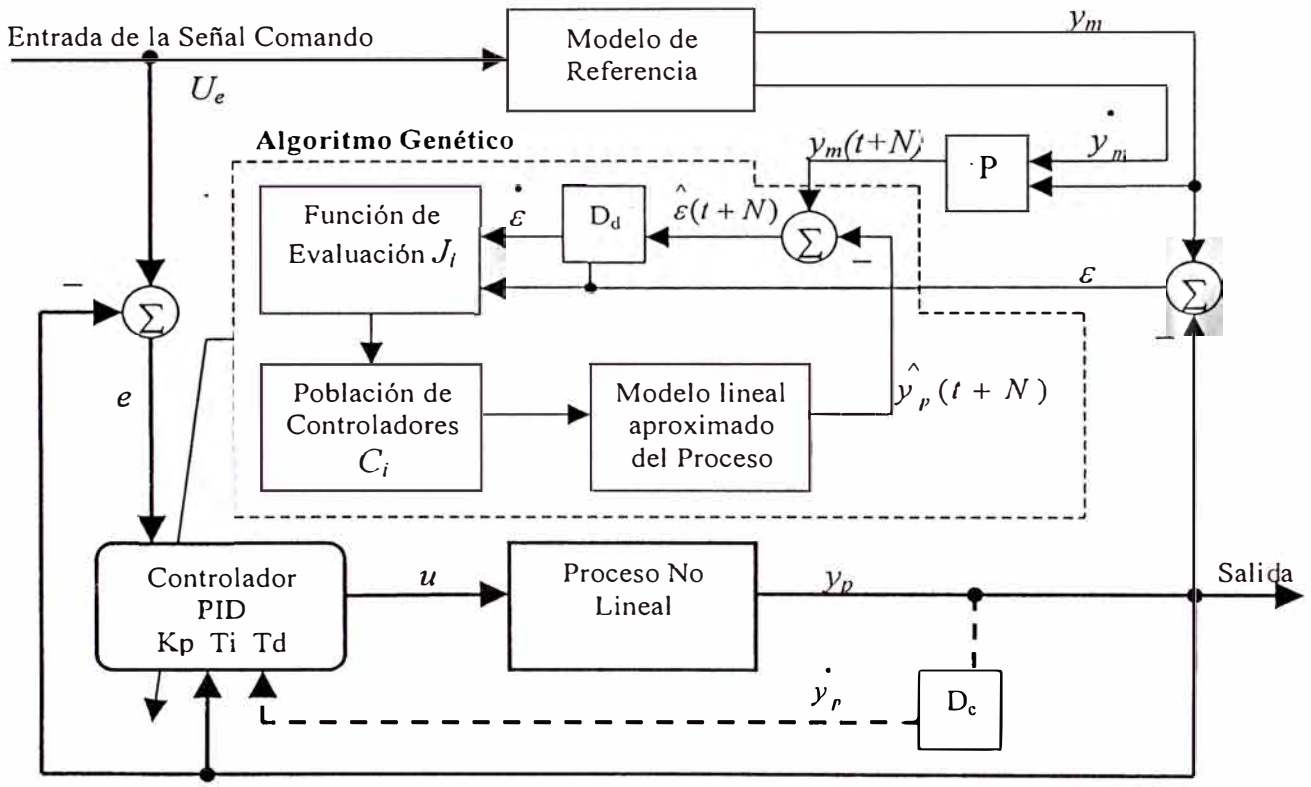

Figura (4.4) Esquema del controlador PID Genético Adaptativo con Modelo de Referencia

En la figura de arriba U_e es la señal de Referencia de Entrada, y_p es la salida del proceso, D_c es la derivada continua que nos proporciona el estado de proceso $\dot{y}_p$[1], u es la señal de control, e es el error de proceso y ε es el error de modelo. Además y_m e $\dot{y}_m$ son los estados del modelo de referencia, los cuales ingresan al predictor de primer orden P, y de esta manera obtener la salida de referencia $y_m(t+N)$, el cual junto con la salida predicha del modelo aproximado del proceso $\hat{y}_p(t+N)$, nos da el error $\hat{\varepsilon}(t+N)$ que

[1] Sólo sí se dispone de la salida $\dot{y}_p$ ya que generalmente se usa la aproximación de la derivada mediante el método de diferencias hacia atrás, como se explico en la sección 2.5.2

se produciría dentro de una ventana de tiempo *N*. Con este valor $\hat{\varepsilon}$ y utilizando una derivada aproximada discreta D_d, se obtiene $\dot{\varepsilon}$, el cual juntamente con ε ingresan a la función de evaluación J_i, para poder seleccionar el mejor controlador que actuará con el proceso.

La obtención del modelo aproximado del sistema (o los modelos lineales aproximados del sistema si se requiere más precisión, ya que están particionados en rangos de operación del proceso) se hará(n) utilizando la técnica denominada ARX, que nos proporciona la versión del programa Matlab 6.0, y que será tratada en el apéndice A

4.5 ALGORITMO DE ADAPTATIVIDAD

A continuación se presentará los requerimientos mínimos para poder implementar el algoritmo de adaptatividad usando los AGs y basándose en los parámetros PID optimizados.

4.5.1 Elección de la Codificación

En primer lugar es necesario conocer en que rango se pueden encontrar cada uno de los parámetros PID y que precisión se desea alcanzar utilizando los métodos explicados en la sección 3.2, es decir la codificación binaria o la de punto flotante. El rango de soluciones de cada uno de los parámetros PID se pueden obtener de un conocimiento a

priori, dado por ejemplo por las Reglas de Sintonización de Ziegler – Nichols, o una auto sintonización no muy refinada, para el punto de arranque del proceso.

Una vez conocido el rango se decide cuantos dígitos de precisión se desea tener para cada parámetro, y de esta forma diseñar la forma del cromosoma; por ejemplo, se ha encontrado que para cierto proceso se han encontrado que el rango de soluciones de un controlador simple PD es el siguiente:

$$-10 < p \leq 0, \qquad -200 < d \leq 0$$

Se ha elegido utilizar la codificación de punto flotante debido al rango muy amplio de la ganancia derivativa *d*, siendo un posible cromosoma *[1234123456]*. Esto se puede traducir en una ganancia proporcional de $p = -1.234$, y en una ganancia derivativa de $d = -123.456$. Hay que notar que el signo negativo es implícito, así como el punto decimal después de un dígito para la ganancia *p*, y después de tres dígitos para la ganancia *d*. También es necesario tener otras consideraciones como la restricción que debe de tener el quinto digito del cromosoma, debido a que solo puede variar entre 1 y 0 para no salirse del rango establecido de soluciones.

4.5.2 Seudo Código del Algoritmo CGAMR

Esta sección nos muestra los pasos que debemos de seguir en el diseño del CGAMR. Para evaluar a la población de controladores, los AGs necesitan encontrar: el error e entre la salida del proceso y_p y la entrada de referencia U_e, necesarios para la acción proporcional e integrativa, y el estado $\dot{y}_p$ o su aproximación el cual es necesario en la acción derivativa. Luego para cada miembro de la población, el algoritmo de adaptatividad calcula la salida de la señal de control u_i usando las ganancias PID de los cromosomas. Con esta señal u_i se estiman tanto la salida del proceso (sobre un modelo aproximado del mismo), como la señal de salida del modelo de referencia, predicción que se hace en ambos durante un tiempo NT_m, donde N es el número de periodos adelante en los que se hace la predicción y T_m es el periodo de muestreo. Usando esta información, los AGs puede determinar que controladores en la población mantienen la salida del proceso tan cercano como es posible a la salida del modelo de referencia.

El siguiente seudo – código define más precisamente la función de evaluación del AG y por lo tanto la operación del CGAMR.

a) Se calcula el error entre la salida del proceso y la entrada de referencia como sigue a continuación:

$$e(t) = U_e(t) - y_p(t) \tag{4.11}$$

b) Se predice la salida del modelo de referencia NT_m segundos en el futuro, utilizando su propia ecuación en diferencias, o a través de una aproximación de primer orden:

$$y_m(t + NT_m) = y_m + NT_m \dot{y}_m(t)$$

En esta ecuación $\dot{y}_m(t)$ no es la derivada continua, sino es la aproximación de primer orden discreta de la derivada.

$$\dot{y}_m(t) = \frac{y_m(t) - y_m(t - T_m)}{T_m}$$

c) Se calcula el error actual entre la salida del modelo de referencia y la salida del proceso.

$$\varepsilon(t) = y_m(t) - y_p(t) \qquad (4.12)$$

d) Supongamos que el *i-esimo* candidato del controlador $C_i = (Kp_i, Ti_i, Td_i)$. Para cada uno de los candidatos del controlador, C_i, se hace lo siguiente:

- Se determina la señal de control u_i usando las ganancias Kp_i, Ti_i y Td_i que se obtienen del cromosoma candidato, y que se reemplazan conforme a la ecuación (2.22):

$$u_i = P_i + I_i + D_i$$

Donde:

$$\begin{cases} P_i(k+1) = Kp_i.e(k) \\ I_i(k+1) = I(k) + \dfrac{Kp_i.T_m}{Ti_i}e(k) + \dfrac{1}{Tt_i}(v(k) - u(k)) \\ D_i(k+1) = \dfrac{\tau}{\tau + T_m}D(k) + Kp_i.Td_i.T_m.(y(k) - y(k-1)) \end{cases} \quad (4.13)$$

Además se escoge $Tt_i = Ti_i$, y $\tau = Td/10$ de acuerdo a las opciones que se dieron en las secciones 2.5.1 y 2.5.2 respectivamente.

- Se inicializa el modelo del proceso con muestras pasadas del proceso. En caso de no disponer de dichas muestras, se inicializa dichas muestras con valores igual a cero, de igual manera sucede con las muestras pasadas de la señal de control, con lo cual se consigue que el controlador se ajuste más rápido, debido a que no se usa un controlador inicial que genere dichas muestras pasadas, y que demore el proceso de adaptatividad. Entonces se tiene:

$$\hat{Yp}_i(k-j) = Yp_i(t - jT_m), \quad \text{j = 0,1,2,... grado del modelo del proceso}$$

$$u_i(k-j) = u(t - jT_m), \quad \text{j = 0,1,2,... grado del modelo del proceso}$$

- Asumiendo que la señal de control permanece constante[2] para los siguientes N muestras ($u_i(k+1) = ... = u_i(k+N) = u_i$), se predice la salida del modelo del proceso, $\hat{Yp}_i$ (k+N), N muestras en el futuro, usando la ecuación del modelo:

  Para j = 1 hasta N

$$\hat{Yp}_i(k+j) = a_1 u_i(k+j-1) + a_2 u_i(k+j-2) + ...$$
$$..... - b_1 \hat{Yp}_i(k+j-1) - b_2 \hat{Yp}_i(k+j-2) - ...$$

  Siguiente j

  Donde los coeficientes a_i y b_i dependen del grado del modelo aproximado del proceso.

- Usando la salida del modelo aproximado del proceso, se estima el error entre el proceso y la salida del modelo de referencia NT_m segundos en el futuro.

---

[2] Como una alternativa uno podría permitir al controlador C_i variar la señal de control sobre este intervalo de tiempo. Una señal de control constante u es asumida porque es más precisa que estimar $u_i(k+j)$ basado en las ganancias del candidato de controlador y una estimación de la señal de error.

$$\hat{\varepsilon}_i(t + NT_m) = Y_m(t + NT_m) - \hat{Y}\, p_i(k + N) \tag{4.14}$$

- Se estima la derivada del error entre la salida del modelo del proceso y la salida del modelo de referencia, usando una aproximación de primer orden.

$$\dot{\varepsilon}_i(t) = \frac{\hat{\varepsilon}_i(t + NT_m) - \varepsilon(t)}{NT_m} \tag{4.15}$$

- Se asigna el rendimiento, J_i, de cada candidato de controlador, C_i:

$$\bar{J}_i = \varepsilon(t) + \beta.\dot{\varepsilon}_i(t) \tag{4.16}$$

Luego

$$J_i = \frac{\alpha}{\bar{J}_i^{\,2} + \alpha} \tag{4.17}$$

Donde la elección de los valores de α y β serán explicados más adelante

e) Se repite el paso 4 para cada miembro de la población.

f) El controlador con mayor rendimiento durante la predicción en el modelo aproximado se convertirá en el controlador usado por el sistema durante el siguiente periodo de muestreo.

El valor elegido de α es para fijar un límite superior sobre el rendimiento J_i como se menciono en la sección 3.4, y su elección debe de ser considerada cuidadosamente. Si α es demasiado pequeño, entonces existirá una gran disparidad en la población de los valores rendimiento J_i (un pequeño cambio en $\bar{J}_i$ resultará en un gran cambio en J_i), y en general solo unos pocos individuos serán seleccionados para reproducirse en la siguiente generación. El resultado podría ser una población de individuos cercanamente parecidos y por lo tanto se eliminaría el mecanismo de búsqueda paralela de los AGs. Sin embargo, si α es elegido demasiado grande, cada uno de los miembros de la población tendrá valores de rendimiento cercanamente igual y por lo tanto tendrían similar oportunidad de reproducirse, comprometiendo de ese modo, la "supervivencia entera del mejor" genero del AG. Generalmente α es seleccionado para estar aproximadamente en un orden de magnitud más pequeño que el valor promedio de $\bar{J}_i$

El valor de β es elegido basado en la eficiencia deseada del sistema y la habilidad para controlar la planta. Heurísticamente hablando β define cuan rápido queremos que el error ε, entre el proceso y el modelo de referencia, llegue a cero. Para ver esto, hay que notar que la función idoneidad es maximizada cuando $\bar{J}_i^2$ es minimizada, lo cual corresponde cuando $\bar{J}_i = 0$. Mirando en la ecuación (4.16), la función de idoneidad es maximizada cuando $\dot{\varepsilon}_i(t) = -\varepsilon(t)/\beta$. Por ejemplo, si $\varepsilon(t) = 1.0$, entonces nos gustaría que $\dot{\varepsilon}_i(t) = -0.2$, de tal manera que el error sea conducido a cero en aproximadamente 5

segundos. En este ejemplo el mejor controlador es uno que produce $\varepsilon_1(t)$ cercano a –0.2. Cuando elegimos β, es deseable que lo hagamos con una valor tan pequeño como sea posible, pero no es deseable tampoco introducir oscilaciones, no pudiendo ser ignorados las no linealidades como la saturación o que la dinámica del sistema sea lenta.

La "mirada adelante en el tiempo", N, se puede elegir por ejemplo con 10 muestras, luego sí el periodo de muestreo $T_m = 0.5s$, entonces $NT_m = 5s$; Es necesario hacer una buena elección de N, porque si se escoge un valor bajo, regularmente los efectos de la entrada actual no son rápidamente aparentes en la salida del sistema. Por otra parte el alargar el tiempo es mejor, sí es que estamos dispuestos a evaluar los efectos que la entrada tiene sobre el sistema. Sin embargo, el error entre lo que el modelo del proceso predice y como el proceso actual se comporta, se incrementa conforme N se incrementa, degradando por lo tanto, la precisión de la función idoneidad. En general hemos encontrado experimentalmente que una ventana de tiempo de 10 a 15 muestras discretas trabaja muy bien.

Una vez que cada controlador ha sido evaluado con la función de idoneidad J_i, el AG usa la selección por Rueda de Ruleta, descrita en el capítulo III, para encontrar los controladores que se reproducirán en la siguiente generación. A los individuos seleccionados para reproducirse se les llaman los *padres* de la siguiente generación. Luego, se continúa el proceso de cruzamiento y mutación de los padres, después de lo

cual se encuentran listos para ser evaluados nuevamente durante el siguiente periodo de muestreo. Opcionalmente se puede usar el operador *elitista* si se quiere mantener al mejor individuo intacto, para que participe en la próxima generación.

CAPITULO V

APLICACIÓN EN EL CONTROL DE LA VELOCIDAD DE UN MOTOR DC

5.1 INTRODUCCION

En este capítulo se aplicará la teoría de CGAMR para la sintonización de los parámetros PID en tiempo real, que permitan controlar la velocidad de un motor DC y cuyo modelo lineal aproximado será obtenido por técnicas de identificación de parámetros ARX, el cual se usará luego dentro del algoritmo de adaptatividad.

Una vez seleccionado todos los parámetros que participan en el CGAMR, haremos una simulación en Matlab utilizando el modelo lineal aproximado y evaluaremos como responde el sistema debido a los cambios en la señal de referencia. Luego pasaremos a hacer la misma prueba pero en tiempo real, utilizando distintas configuraciones de los AGs, es decir variaremos las probabilidades de cruzamiento y mutación, el tamaño de población e intercambiaremos los métodos de selección entre Rueda de Ruleta y Torneo. De esta manera podremos apreciar la respuesta y el rendimiento de nuestro algoritmo en el seguimiento de un modelo de referencia.

5.2 DESCRIPCIÓN DEL HARDWARE

En esta sección se describen todas las partes físicas que componen el sistema de control de Velocidad de nuestro motor DC, el cual está representado en la figura (5.1), donde se puede apreciar las siguientes partes:

- Motor DC
- Encoder Incremental
- Amplificador de Potencia
- Microcontrolador PIC para adquisición y envío de la señal de Control
- Una PC para procesar los datos y realizar el control.

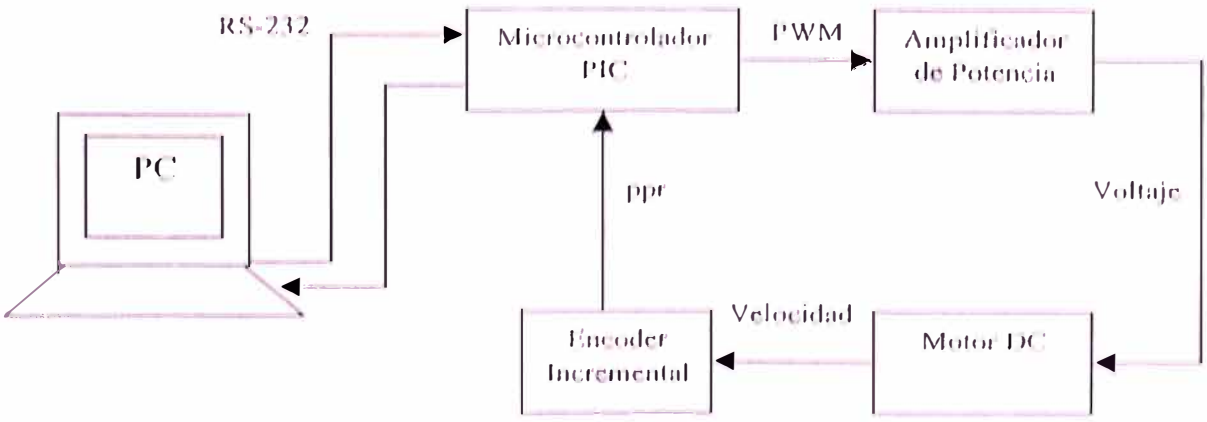

Figura (5.1) Implementación del sistema de control

A continuación pasaremos a explicar cada una de las partes que componen nuestro sistema de control.

a) *Motor DC.*- El motor que usaremos, como se muestra en la figura (5.2), tiene las siguientes características básicas:

- Fabricante: PITTMAN
- Modelo: 9413D319
- Voltaje Máximo: 19.1 VDC
- Año de Fabricación: 1983

Los demás parámetros de funcionamiento, se desconocen, por ser un modelo obsoleto y que el fabricante ya no proporciona. Es por eso que no se puede hacer un modelamiento aproximado basándose en sus datos técnicos, por lo que se buscará otro método para encontrar un modelo aproximado que se asemeje lo máximo posible a la dinámica de nuestro motor. Por otro lado no usaremos el máximo voltaje permisible para no dañar nuestro motor, sino que estableceremos como voltaje máximo un valor menor, el cual escogimos en 15.8V

b) *Encoder Incremental.*- El tipo de encoder que usaremos es de tipo incremental, como se muestra en la figura (5.2), es decir que nos sirve para medir velocidad, pero no para medir posición. Internamente tiene un codificador óptico rotatorio compuesto de un disco de plástico que en el perímetro posee 500 líneas oscuras a manera de ranuras y un emisor – sensor óptico que genera una fuente de luz y

sensa luego la presencia de está de acuerdo a su interrupción o no-interrupción producida por las líneas oscuras del disco. Este encoder tiene las siguientes características:

- Fabricante: Hewlett Packard
- Modelo: A-2807-56
- Resolución: 500 ppr
- Canales: 2, Canal A y Canal B

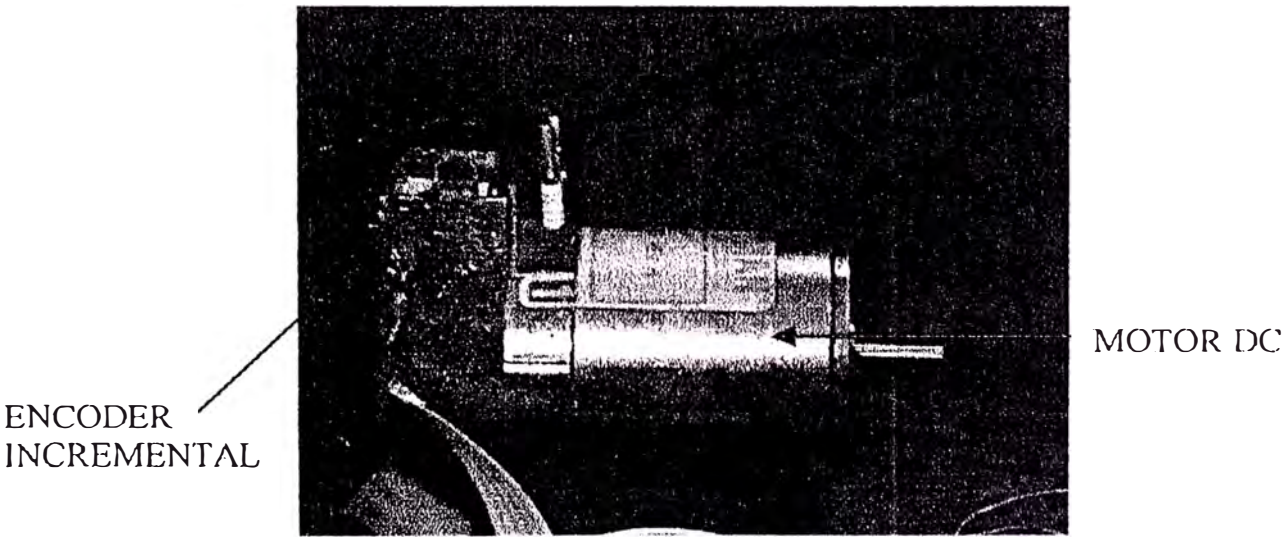

Figura (5.2) Encoder Incremental y Motor DC

El codificador óptico da como muestra un tren de pulsos con una frecuencia proporcional a la velocidad angular del disco y un tren de pulsos desfasado en +90 ó –90 grados respecto del primer tren, según sea el sentido del giro del disco. La correlación de estos pulsos nos permite saber el sentido de giro del motor,

aunque si solo se quiere hacer girar al motor en un sentido solo es necesario el uso de cualquiera de los dos.

c) *Amplificador de Potencia*.- Usamos como amplificador de potencia al puente H LMD18200T, el cual se muestra en la figura (5.3). Este amplificador está basado en la modulación de ancho de pulso PWM como entrada y en un voltaje proporcional a dicha señal PWM como salida, el cual alimenta al motor.

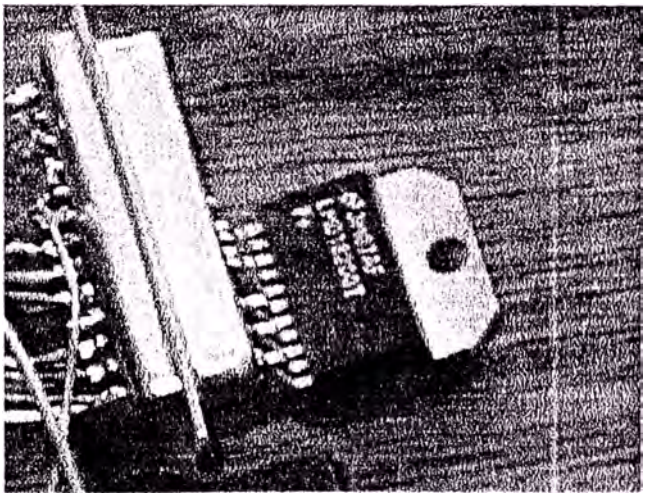

Figura (5.3) Amplificador de Potencia H-Bridge LM18200T

d) Microcontrolador PIC para adquisición y envío de la señal de Control.- El microcontrolador PIC 16F873 como se muestra en la figura (5.4) nos servirá como interfaz, entre la computadora y el amplificador de Potencia. Este dispositivo cuenta con 3 temporizadores, los cuales serán usados de la siguiente manera:

- Timer0, el cual se encargará de contar los pulsos que envía el encoder, permitiéndonos de esta manera conocer la velocidad angular con que se mueve el motor DC.

- Timer1, cuyo conteo generará el periodo de muestreo, es decir llamará a una interrupción que obtendrá y enviará a la Pc el total de ppr que se contó del encoder, cuando el tiempo transcurrido sea igual a un periodo de muestreo.

- Timer2, el cual enviará el tren de pulsos PWM al amplificador de potencia como una señal traducida de la señal de Referencia que le envía la Pc a través del puerto serial RS-232

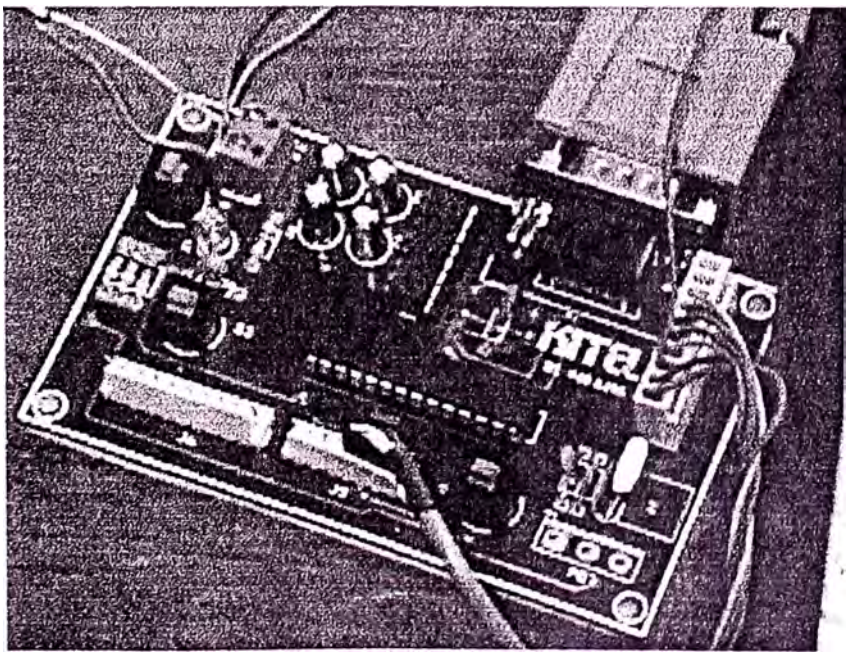

Figura (5.4) Modulo de control para el PIC 16F873

e) *La Pc.*- El cual hará el procesamiento de los datos adquiridos por el PIC, en este caso convertirá la velocidad en ppr (pulsos por revolución) a rad/s, aplicará el algoritmo de adaptatividad y luego enviará la señal de control resultante c durante cada periodo de muestreo al dispositivo de interfase, en este caso el mismo microcontrolador PIC.

5.3 MODELAMIENTO DEL SISTEMA

El motor DC es un dispositivo físico que debido a diversos factores, como la fricción, la inductancia, etc., representa un sistema no lineal, sin embargo debido a la característica de su respuesta proporcional al voltaje de entrada, se le puede obtener un modelo lineal cuando no está sometido a cargas externas. Esta obtención del modelo lineal se hará por técnicas de identificación de sistemas usando ARX, ya que se desconoce la mayoría de sus parámetros, dado la caducidad del modelo, y a que el fabricante no provee información adicional del mismo.

Para esto, se sometió al motor DC a una entrada variable (voltaje variable), tal y como se muestra en la figura (5.5), ya que es fundamental que nuestro modelo aproximado cumpla para todas o la mayoría de las condiciones de funcionamiento.

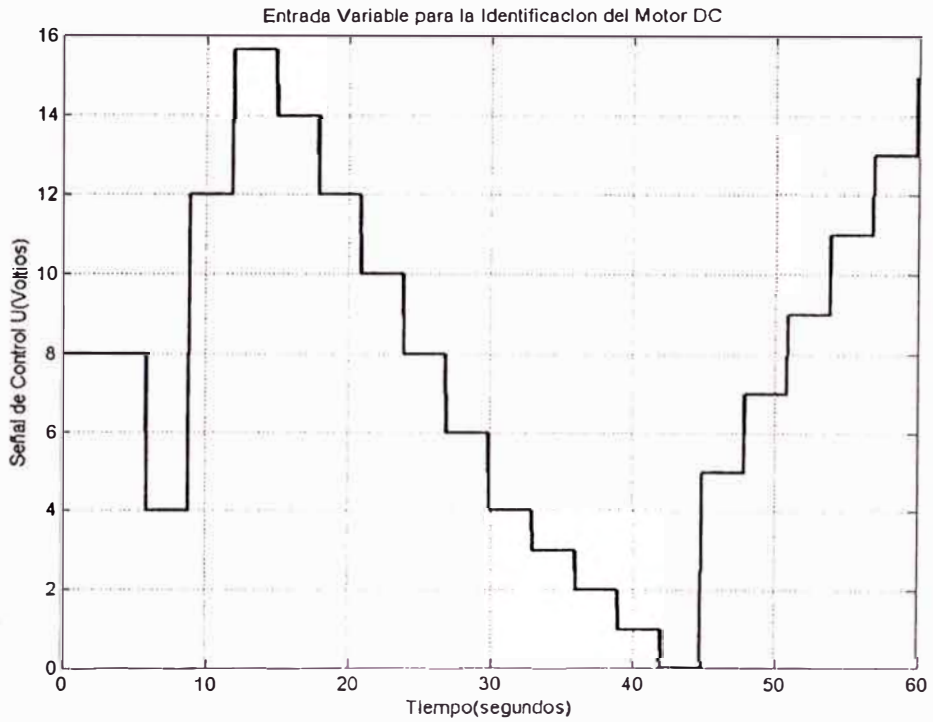

Figura (5.5) Entrada Variable para la Identificación del modelo aproximado del Motor DC usando las técnicas de identificación del ARX

Como se puede apreciar en el gráfico de arriba, se sometió al motor DC a una prueba que duró aproximadamente 60 segundos, durante el cual se varió constantemente el voltaje de entrada. El periodo de muestreo que se escogió para la recolección de los datos fue $T_m=0.03s$. Esta elección se hizo basándose en que debe de ser un valor pequeño y que nos permita capturar el mayor número de datos posibles.

Los valores obtenidos como resultado de esta prueba, se procesaron luego usando el ToolBox de Identificación de Sistemas que provee el software Matlab 6.0, específicamente usando la técnica ARX.

Luego fue necesario configurar el orden de nuestro sistema, por lo cual elegimos un modelo simple sin retardo con 3 polos y dos ceros, el cual nos ofrecía un 94.6% de semejanza con el motor, pero a su vez no representaba un sistema sobrecargado de polos ni de ceros, dando como resultado el siguiente modelo:

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

>>A(q) = 1 - 0.988 q^-1 + 0.3454 q^-2 - 0.05533 q^-3

>>B(q) = -0.1216 + 6.144 q^-1 + 1.872 q^-2

>>Estimated using ARX from data set cavs

>>Sampling interval: 0.03

Donde *A(q)* representa el denominador que contiene los polos en el plano *Z*, *B(q)* representa el numerador que contiene los ceros en el plano *Z*, *u* es la señal de control e *y* es la velocidad de salida del motor.

Luego, como se puede apreciar en la figura (5.6) dentro del rango de 250 a 350 rad/s existe un error mínimo entre el modelo identificado y el modelo real, lo cual nos lleva a concluir, que se pueden hacer distintos modelos en rangos determinados donde la semejanza sea mayor que 94.6%.

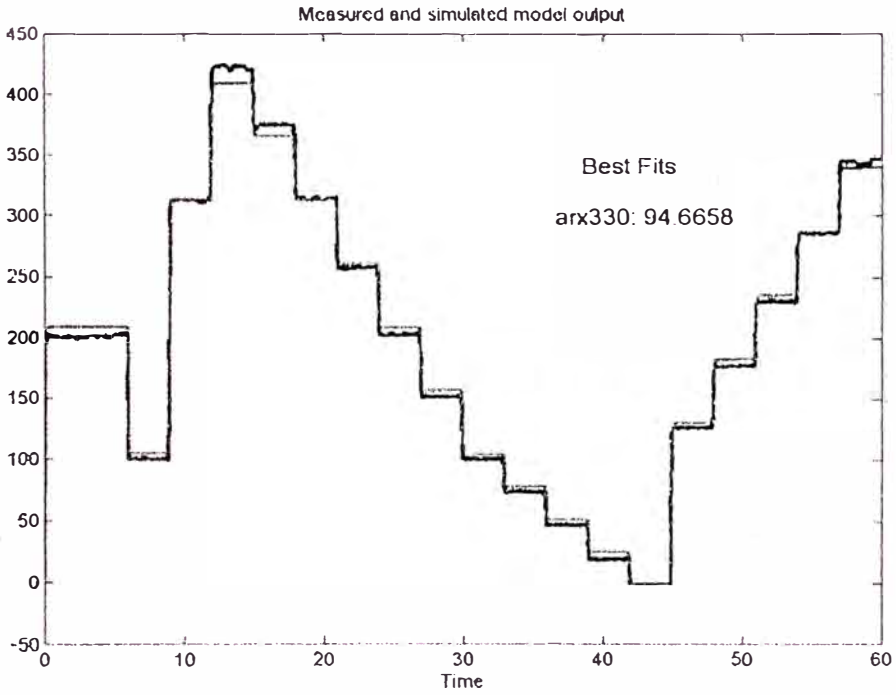

Figura (5.6) Este diagrama muestra el nivel de semejanza que existe entre el modelo aproximado (verde) y el modelo real (negro)

Sin embargo para nuestro caso no es necesario debido a que dicho porcentaje es muy bueno y sirve para nuestro algoritmo de Adaptividad, el cual no solo se basa en dicho modelo aproximado para predecir la respuesta del motor, sino que también se basa en el error real actual que existe entre la señal de referencia y la velocidad del motor.

5.4 IMPLEMENTACION DEL SOFTWARE

La implementación del software se hizo usando el software de programación Visual Basic 6.0 y comprende varias etapas, como la selección de los parámetros adecuados para el AG, así como la adecuada elección del periodo de muestreo T_m, entre otros.

5.4.1 Elección del periodo de muestreo

La elección del periodo de muestreo representa uno de los factores críticos, debido a que se debe de tener en cuenta el tiempo de procesamiento de la señal de control, cuyo efecto real no se aprecia cuando uno realiza la simulación, pero que si puede influenciar enormemente durante la implementación física. Nosotros tomaremos el valor de $T_m=0.09s$, tanto para la simulación, como para la implementación física. Este valor se escogió basándose en pruebas y mediciones que se hicieron en el osciloscopio tanto del tiempo que se demora la Pc en enviar la señal de control como del tiempo que tarda en enviar el PIC los datos a la computadora, elaborándose un gráfico que representa la intersección de dichos tiempos y que se muestra en la figura (5.7)

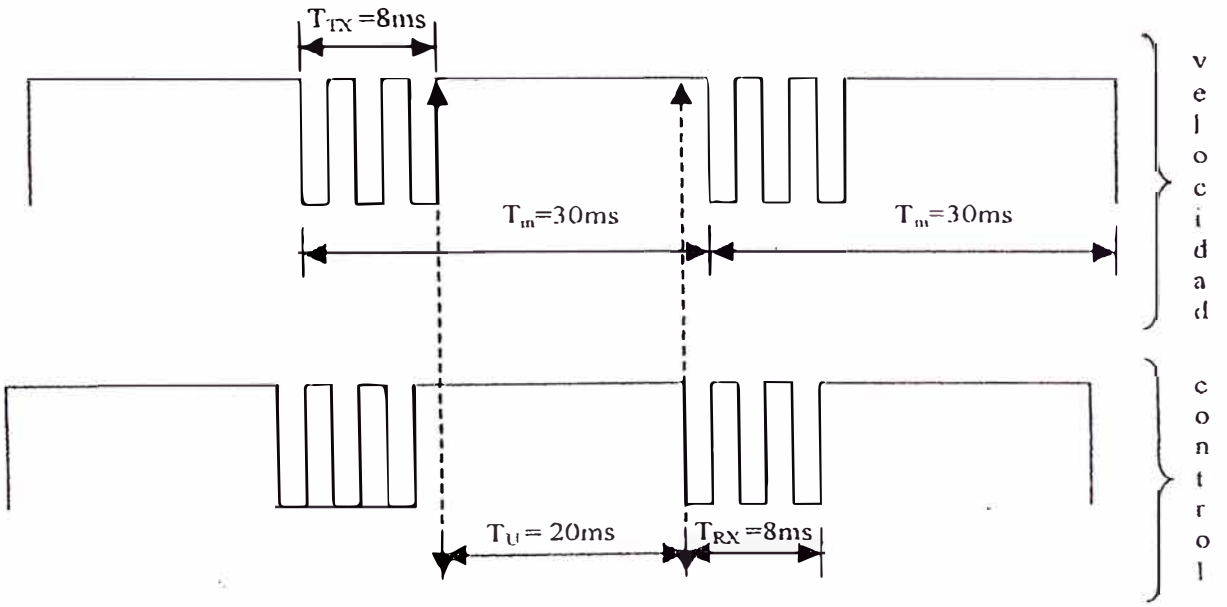

Figura (5.7) Diagrama que muestra la coincidencia de tiempos entre la transmisión de la señal de velocidad y la señal de control

En la figura de arriba se observa que una vez transferidos los datos de la velocidad del motor en ppr desde PIC a la Pc, en un tiempo T_{Tx}, este último se encarga de procesarlos. Dicho procesamiento se demora un tiempo T_u de aproximadamente 20ms (con 50 individuos dentro de la población en los AGs), luego de lo cual tiene que ser enviado de la Pc al PIC, para que este ejerza la acción correctiva. Sin embargo se aprecia que justo coincide este tiempo T_{Rx} con el siguiente periodo de muestreo, generándose paralelamente interrupciones de recepción serial tanto en la Pc como en el PIC. Esto nos da como resultado que no se ejerció dicha acción correctiva antes del siguiente periodo de muestreo y que se coincidan las señales en ambos dispositivos, produciéndose errores de procesamiento de datos.

Esto se puede solucionar si se escoge un periodo de muestreo mayor, que permita que la señal de control sea procesada y que se aplique como acción correctiva en el motor, mucho antes de que se genere el siguiente periodo de muestreo, durante un tiempo T_c. Además fue necesario también rebajar el tiempo de envío y recepción de datos, aumentando la velocidad de 2400 baudios a 9600 baudios. Como resultado el osciloscopio nos dio el siguiente gráfico, que se muestra en la figura (5.8):

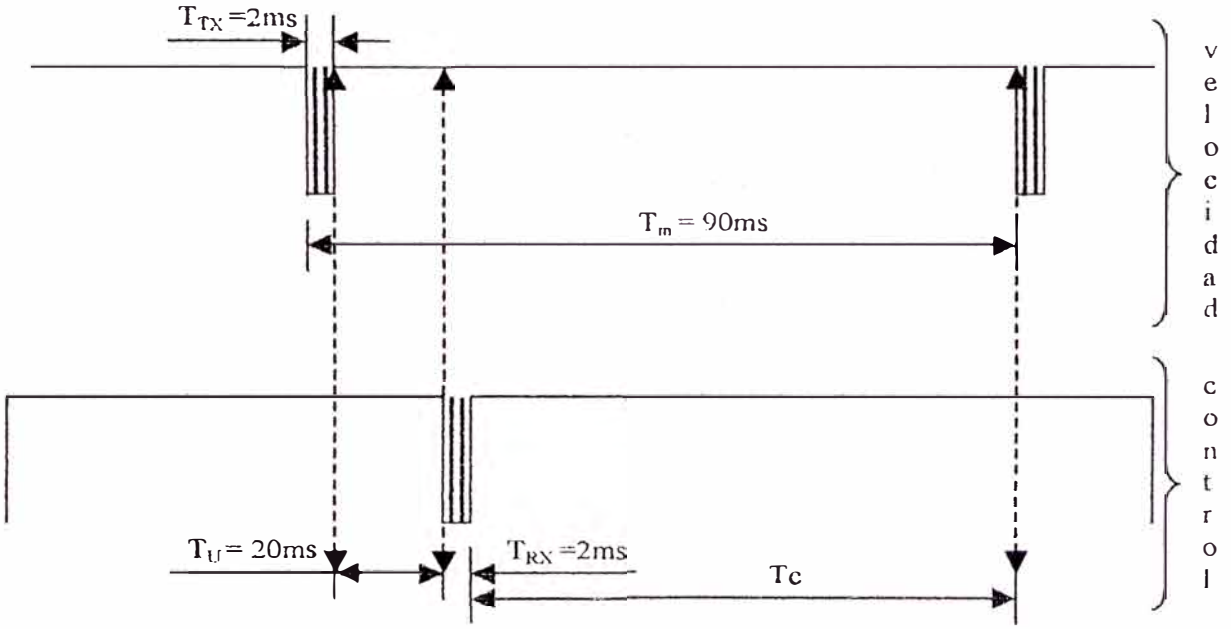

Figura (5.8) Diagrama que muestra los tiempos con el nuevo periodo de muestreo y donde no existe coincidencia entre el envío de la señal de control y la recepción de datos de velocidad en el siguiente periodo de muestreo

5.4.2 Elección del Rango de valores de las ganancias PID

El siguiente paso, consiste en hallar el rango dentro del cual se encuentran las ganancias PID, para lo cual recurrimos al primer método de sintonización de Z-N, ya que como sabemos el modelo linealizado del motor DC no posee integradores y por lo tanto su respuesta genera una curva en forma de S. Nosotros obtuvimos experimentalmente esa curva, y la adquisición de los datos se hizo con un periodo de muestreo pequeño T_m=0.01, esta elección debido a que teníamos la necesidad de obtener la mayor cantidad de datos posible. En la figura (5.9) se muestra la curva mencionada, la cual fue obtenida por la aplicación del escalón unitario (u=1voltio) al motor.

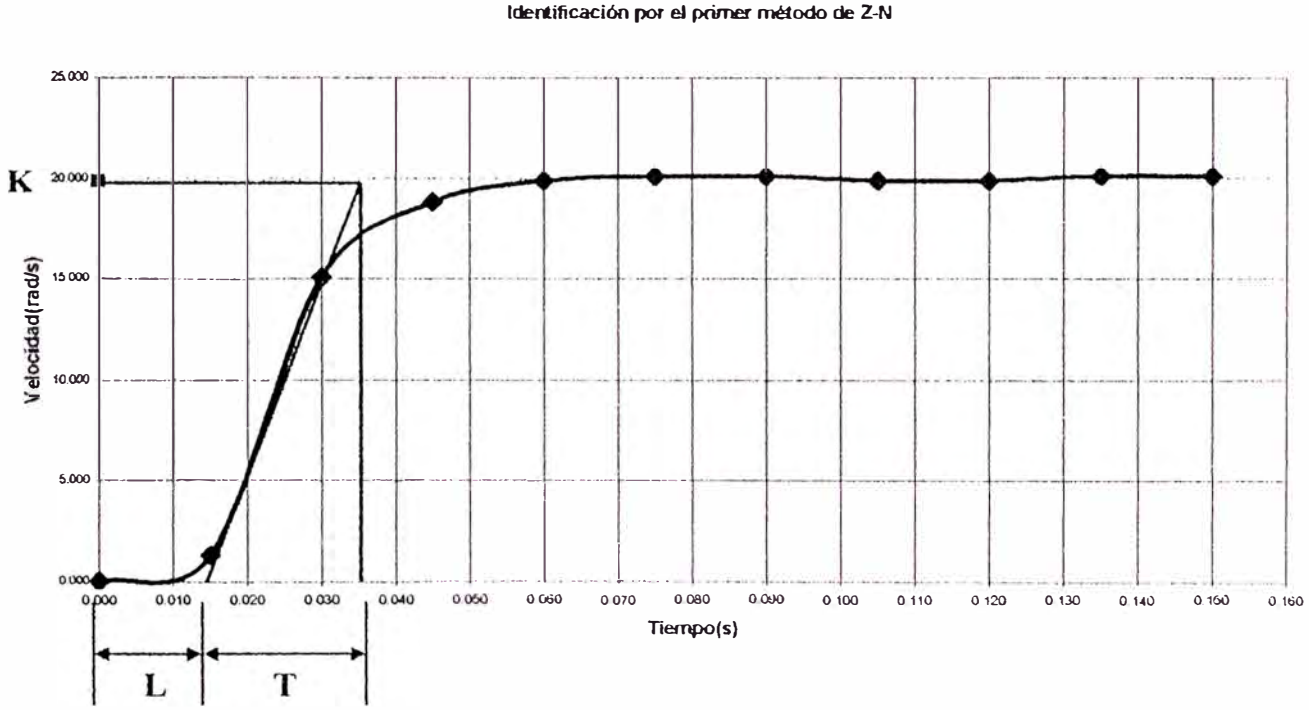

Figura (5.9) Curva en S que muestra la respuesta al escalón unitario del motor DC, para sintonizar los parámetros PID mediante la primera regla de Z-N

La figura de arriba se hizo en Excel, y como resultado del análisis gráfico se obtuvo los siguientes parámetros:

$$L = 0.015, \quad T = 0.02, \quad K = 20$$

Reemplazando estos parámetros en la tabla (2.1), se obtiene los siguientes parámetros PID:

$$Kp = 1.6, \quad Ti = 0.030, \quad Td = 0.0075$$

Estos valores nos dan una idea del rango en que se mueven los valores de las ganancias PID, además que también nos indican el grado de precisión que deben de tener dichas soluciones. Basándonos en esto, la primera elección es usar 4 dígitos para cada una de las ganancias, cuyos rangos para Ti y Td son de $[0 - 9999]*10^{-4}$, y para el valor de Kp es de $[0 - 9999]*10^{-3}$. Además escogimos la codificación de punto flotante, donde cada gen variará en el rango de [0 - 9], con lo cual la longitud final del cromosoma será igual a 12. Luego haciendo simulaciones lograremos establecer bien dentro de que rangos debe de encontrarse dicha solución, para seguidamente poder realizar las pruebas en tiempo real con el motor DC.

5.4.3 Elección de los parámetros para los AGs

Otro punto importante es la adecuada elección de los parámetros que regulan el funcionamiento de los AGs, nos estamos refiriendo específicamente al tamaño de población, a los operadores cruzamiento y mutación, y al método de selección que usaremos.

Como se mencionó en la sección 3.3 el tamaño de población optimo dentro de nuestra aplicación varía en el rango de [50 - 100] individuos, los cuales permiten que la señal de control sea evaluada en un tiempo que aproximadamente haciendo mediciones en el osciloscopio varía de 20 a 45ms, siendo este último la mitad del periodo de muestreo T_m, y lo que nos indica que dicha señal de control se aplicara aproximadamente como

mínimo 45ms, resultando satisfactorio para controlar el proceso, en nuestro caso el motor DC.

Por otra parte el operador cruzamiento se elegirá con una probabilidad de 0.9, mientras que el operador mutación será inversamente proporcional al tamaño de población, es decir variará de 0.01 a 0.02, ambas elecciones debido a las razones dadas en la sección 3.6. En cuanto al método de selección que usaremos, se utilizará ambos métodos descritos y se comparará el rendimiento de cada uno, lo cual se mostrará en la sección de los resultados. Además se utilizará el operador *elitista* para conservar el mejor controlador, y usarlo en la siguiente generación.

Finalmente hay que tener presente que dentro de control adaptativo, el término "en tiempo real", implica que los parámetros del controlador se ajustan en cada periodo de muestreo. Siguiendo este concepto, para nuestro caso cada generación evolucionará durante cada periodo de muestreo, lo cual nos indica que el número de generaciones es indeterminado y durará el tiempo en que se efectúe el control, ó hasta que la acción adaptativa se deshabilite.

5.4.4 Elección del Modelo de Referencia

El modelo de referencia es otra de las partes fundamentales dentro del CGAMR, y su elección se hará, respetando la dinámica del motor DC, es decir teniendo en cuenta que

el tiempo de levantamiento y el tiempo de establecimiento no sean de respuesta demasiada rápida, ni tampoco muy lentos. Cierto conocimiento de esa dinámica la podemos obtener del método de sintonización de Z-N, sobretodo de la curva que obtenemos al aplicar el escalón unitario. Como se observa en la figura (5.9) el tiempo de levantamiento dura aproximadamente 40ms y el tiempo de establecimiento es de 80ms. Con base a esto y haciendo simulaciones, se obtuvo el siguiente modelo en el plano S:

$$\frac{Y_m(s)}{U_c(s)} = \frac{10^2}{(s+10)(s+10)}$$

Donde $Y_m(s)$ es la salida del modelo de Referencia, y la entrada $U_e(s)$ es la señal de Referencia (set point)

La dinámica de este modelo se acerca muy bien a la de nuestro motor tal y como se puede apreciar en la figura (5.10), que representa la respuesta al escalón unitario de nuestro modelo de Referencia, y donde el tiempo de levantamiento dura aproximadamente 45ms, mientras que el tiempo de establecimiento dura aproximadamente 85ms, siendo similar a la respuesta al escalón del proceso.

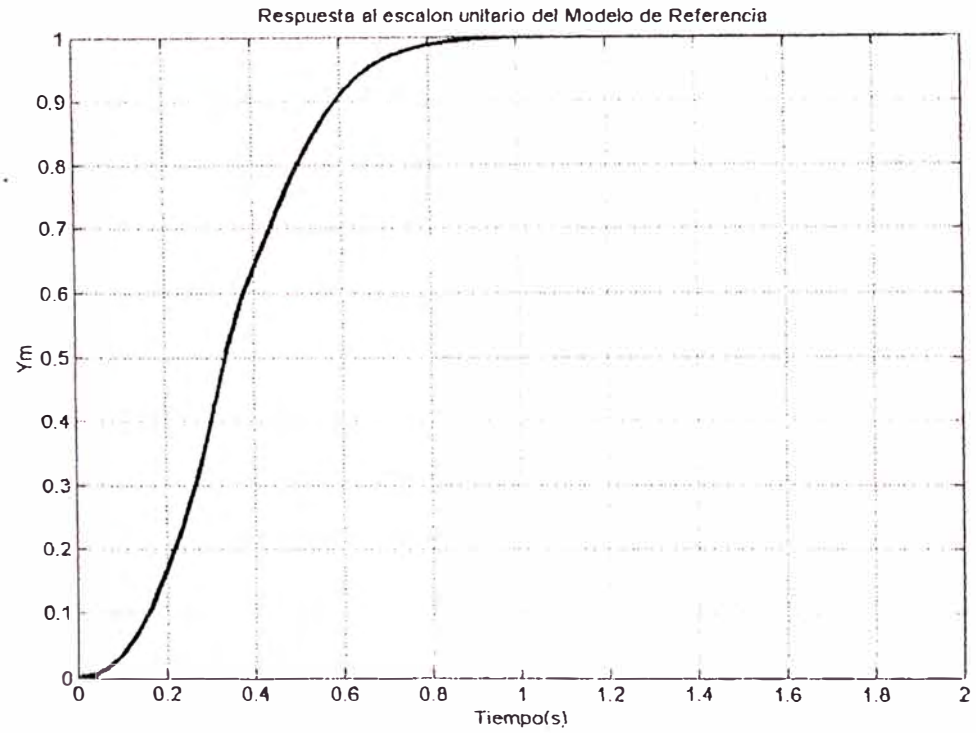

Figura (5.10) Curva que muestra la dinámica que exhibe el Modelo de Referencia elegido, como respuesta al escalón unitario.

Luego una vez elegido el Modelo de referencia, el siguiente paso es la discretización de dicho modelo, lo cual se hace teniendo en cuenta el periodo escogido, en nuestro caso $T_m = 0.09s$, lo cual nos da como resultado el siguiente modelo en el plano Z:

$$\frac{Y_m(z)}{U_e(z)} = \frac{0.2275 + 0.1246z^{-1}}{1 - 0.8131z^{-1} + 0.1653z^{-2}}$$

También es necesario pasar el modelo aproximado del motor DC al plano Z con el periodo de muestreo elegido, lo cual se realiza rápidamente usando el software Matlab, obteniéndose el siguiente modelo:

$$\hat{G}_p(z) = \frac{13.7958 + 9.0707\, z^{-1} + 0.2998\, z^{-2}}{1 - 0.1066\, z^{-1} - 0.0068423\, z^{-2} - 0.0001769\, z^{-3}}$$

Donde $\hat{G}_p(z)$ es el modelo aproximado del motor con periodo de muestreo $T_m = 0.09s$.

5.5 SIMULACIÓN DEL CGAMR

Tomando las consideraciones anteriores se hizo las siguiente simulaciones, donde se puede apreciar tanto la respuesta del Motor DC, así como el error que existe con el Modelo de Referencia

a) Usando los rangos iniciales de las ganancias PID dados en la sección 5.4.3 tenemos:

Parámetros de Diseño:

Set Point Inicial (rad/s)	100
N° de Generaciones	1000
N° de Individuos	80
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.0125
Método de Selección	Rueda de Ruleta
Operador Elitista	Sí
Rango de la ganancia K_p	[0 – 9999]*1e-3
Rango de la ganancia T_i	[0 – 9999]*1e-4
Rango de la ganancia T_d	[0 – 9999]*1e-4
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	10

Resultados:

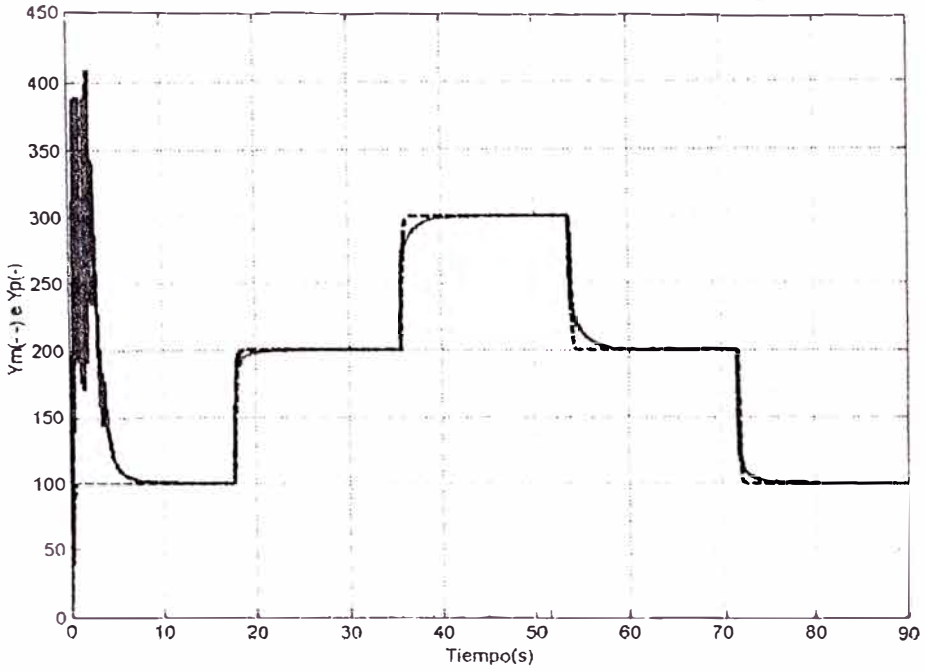

Figura (5.11a) Resultados usando los parámetros iniciales de diseño, donde se muestra tanto el comportamiento del Modelo de Referencia(- - -) y el motor DC(—)

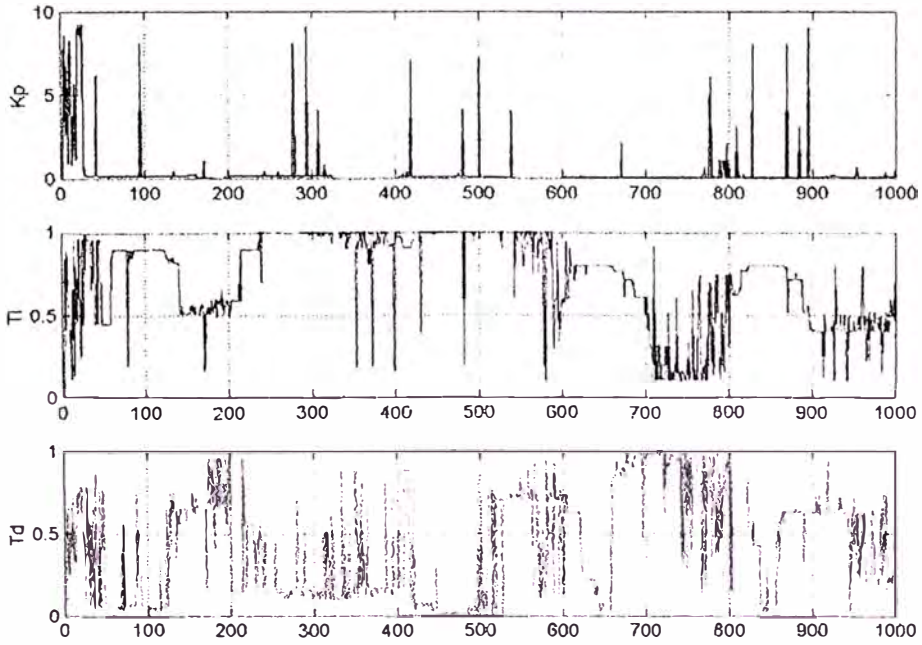

Figura (5.11b) Se muestra como van cambiando los valores de las ganancias PID, para poder adaptar la salida del motor a la salida del Modelo de Referencia

Como se puede apreciar en la figura (5.11a), debido a los valores aleatorios con los que inicia el AG, la señal de control tarda 10 segundos en estabilizar al sistema y adaptarlo para que siga al Modelo de Referencia. Por otro lado, en la figura (5.11b), podemos ver que los valores de la ganancia *Kp* tienden a estabilizarse cerca a 0, razón por la cual es necesario cambiar el rango de esta ganancia. Haciendo distintas pruebas para hallar el nuevo rango, encontramos también que esto afecta al resto de las ganancias, por lo que fue preciso cambiar el rango de las mismas como sigue a continuación:

$$K_p \in [0 - 9999] * 1e\text{-}5 \qquad T_i \in [0 - 9999] * 1e\text{-}5 \qquad T_d \in [0 - 9999] * 1e\text{-}5$$

b) Con los nuevos rangos, se obtuvieron los siguientes resultados de la simulación:

Parámetros de Diseño:

Set Point Inicial (rad/s)	100
N° de Generaciones	1000
N° de Individuos	80
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.0125
Método de Selección	Rueda de Ruleta
Operador Elitista	Sí
Rango de la ganancia K_p	[0 - 9999]*1e-5
Rango de la ganancia T_i	[0 - 9999]*1e-5
Rango de la ganancia T_d	[0 - 9999]*1e-5
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	2

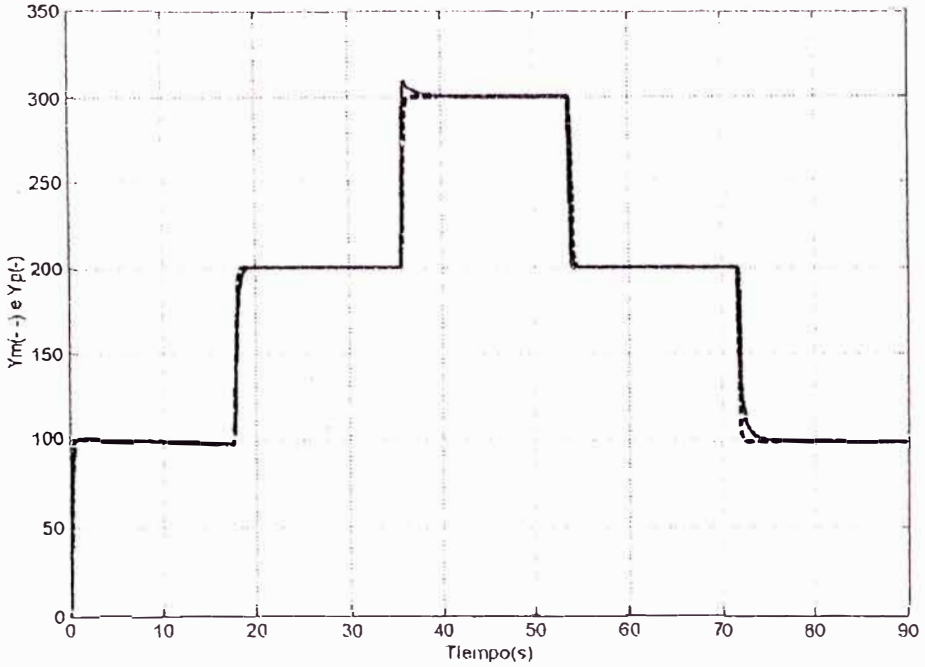

Figura (5.12a) Resultados del Motor DC(—) y el Modelo de Referencia(- - -), usando los nuevos rangos de las ganancias PID y donde también se varió la constante BETA

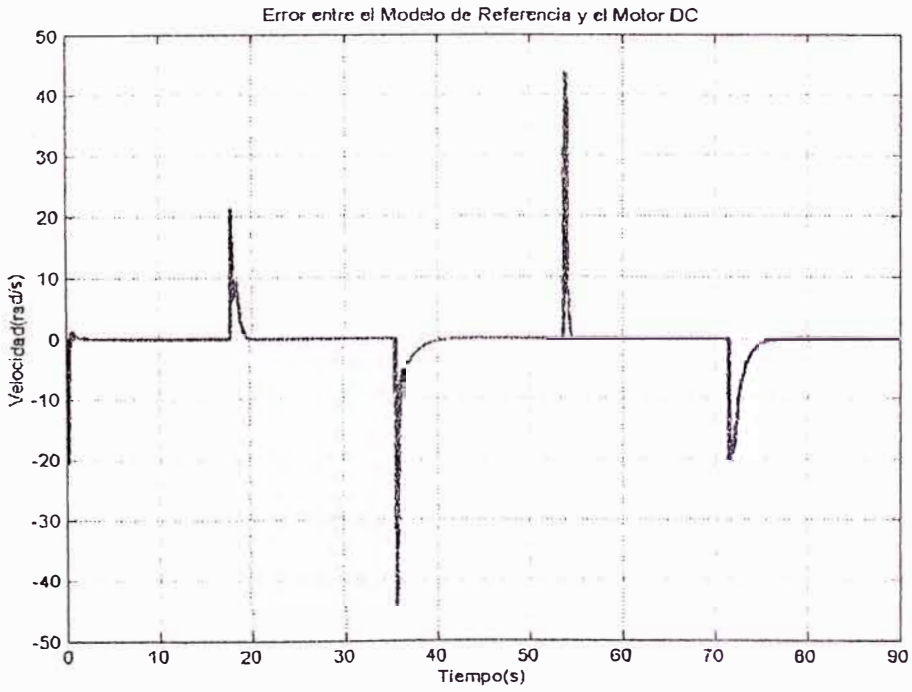

Figura (5.12b) Error entre la Velocidad del Motor DC y la salida del Modelo de Referencia

Como se aprecia en la figura (5.12a), este cambio de rangos mejoró la respuesta de adaptación del motor DC desde el inicio del control. Además el cambio de la constante BETA se hizo debido que el valor de BETA = 10 generaba alta dependencia en la predicción del error con el modelo lineal aproximado, lo cual, en presencia de perturbaciones y cuando se implementa en el modelo físico real, arroja malos resultados; por lo cual en adelante usaremos valores de BETA que oscilen entre 1 y 2. Por otro lado también se aprecia en la figura (5.12b) que la mayor parte del error entre el motor DC y el modelo de referencia ocurre en estado transitorio, es decir cuando hay variaciones en la señal de Referencia, mientras que en estado estable dicho error es 0. Para minimizar ese error en estado transitorio simularemos el CGAMR usando la selección por Torneo.

c) Simulación de la respuesta del Motor DC utilizando la selección por Torneo.

Parámetros de Diseño:

Set Point Inicial (rad/s)	100
N° de Generaciones	1000
N° de Individuos	80
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.0125
Método de Selección	Torneo
Operador Elitista	Sí
Rango de la ganancia K_p	[0 - 9999]*1e-5
Rango de la ganancia T_i	[0 - 9999]*1e-5
Rango de la ganancia T_d	[0 - 9999]*1e-5
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	2

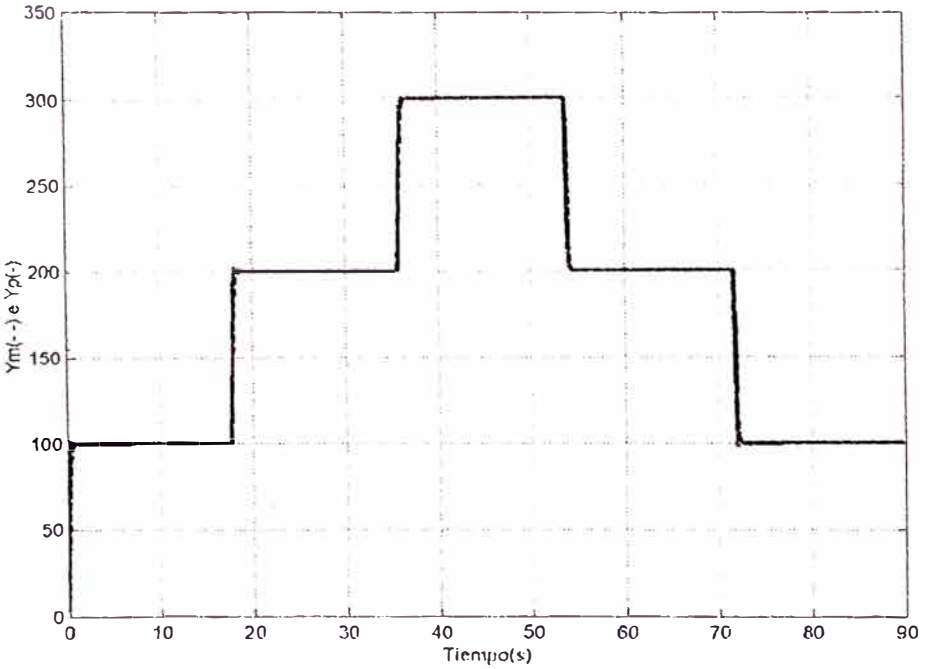

Figura (5.13a) Resultados del Motor DC(—) y el Modelo de Referencia(- - -), usando los nuevos rangos de las ganancias PID y la selección por Torneo

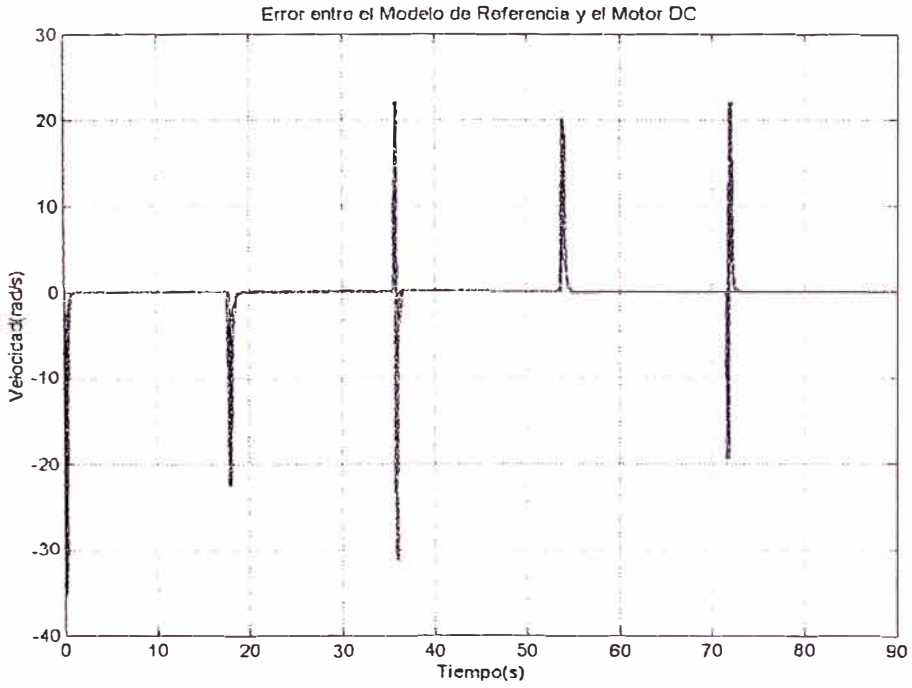

Figura (5.13b) Error entre la Velocidad del Motor DC y la salida del Modelo de Referencia usando la selección por Torneo

En las figuras (5.13a) y (5.13b) se puede observar que la respuesta ha mejorado, aunque con el inconveniente de que la selección por Torneo, demora más tiempo computacional, lo cual al momento de implementar nuestro modelo, puede ser perjudicial debido a que el tiempo de procesamiento de la señal de control puede ser próxima o mayor al periodo de muestreo.

Ahora también es necesario conocer la respuesta de nuestro motor frente a una entrada que incluya señales rampa, los cuales se pueden usar en control de posición a través de la generación de perfiles de movimiento.

d) A continuación se muestran los resultados al aplicar señales rampa al motor DC

Parámetros de Diseño:

Set Point Inicial (rad/s)	40
N° de Generaciones	360
N° de Individuos	100
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.01
Método de Selección	Rueda de Ruleta
Operador Elitista	Sí
Rango de la ganancia K_p	[0 - 9999]*1e-5
Rango de la ganancia T_i	[0 - 9999]*1e-5
Rango de la ganancia T_d	[0 - 9999]*1e-5
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	3

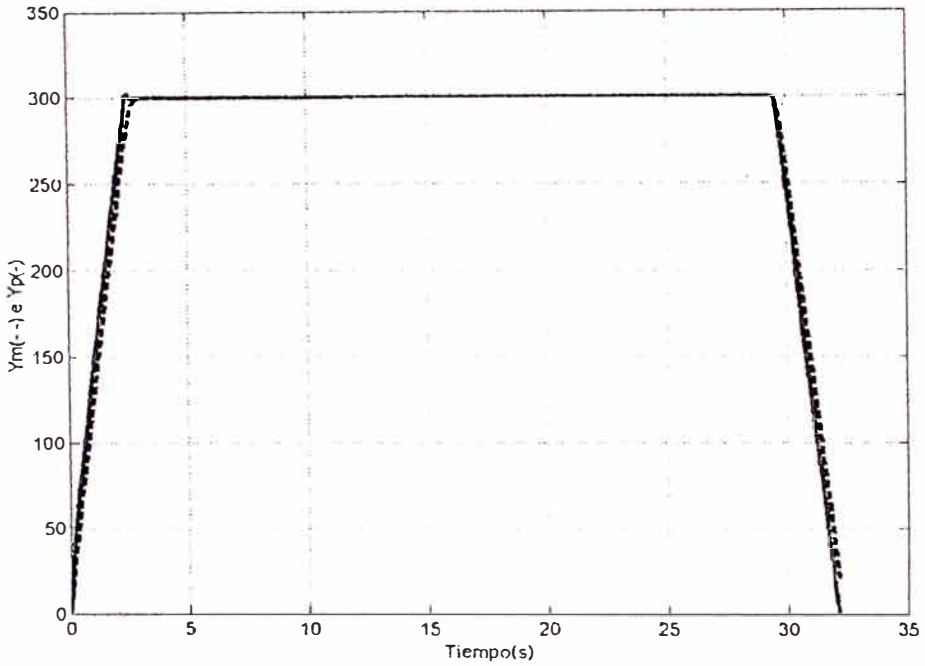

Figura (5.14a) Resultados del Motor DC(—) y el Modelo de Referencia(- - -), al aplicar señales de referencia rampa, aplicados en la generación de un perfil de movimiento

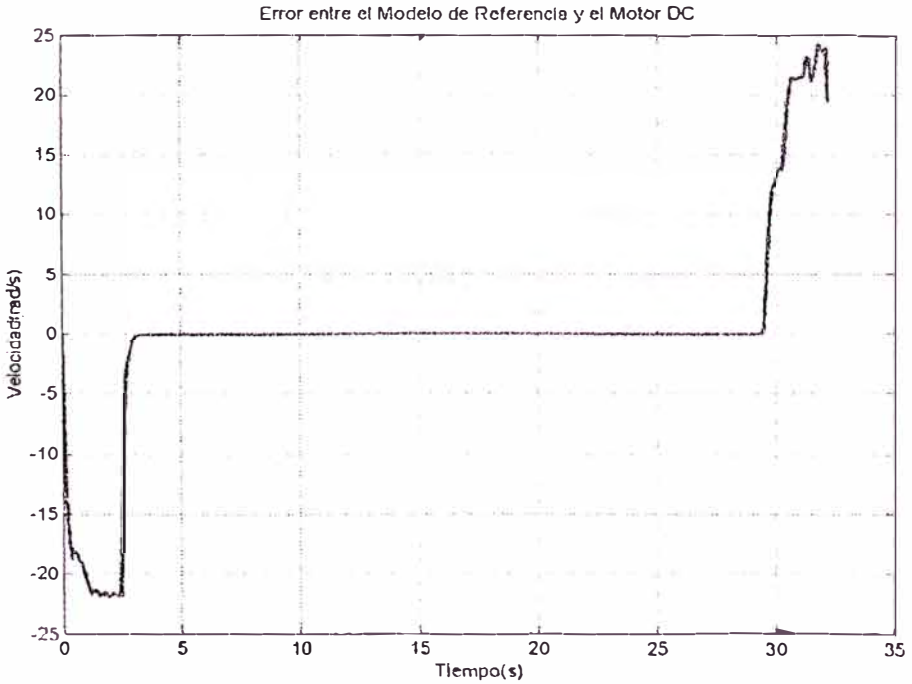

Figura (5.14b) Error entre la Velocidad del Motor DC y la salida del Modelo de Referencia durante la aplicación de señales rampa

Como se aprecia en las figuras (5.14a) y (5.14b), el sistema se adapta bien frente a las entradas tipo rampa, aunque el error se mantuvo de +20 a –20. Este tipo de entradas se utiliza mucho en la generación de perfiles de movimiento para control de posición, y por su poca duración, fue que vimos por conveniente cambiar ciertos parámetros de diseño para concentrarnos más en el rendimiento de nuestro algoritmo CGAMR. Uno de los principales cambios que vimos por conveniente fue aumentar el número de individuos de la población de 80 a 100, para obtener mayor precisión. El otro cambio consistió en aumentar el factor BETA, para influenciarnos más de la predicción del modelo durante el tiempo transitorio y de esta manera no alejarnos demasiado de la salida del modelo de referencia.

5.6 RESULTADOS EN TIEMPO REAL

Luego de realizar las simulaciones correspondientes a nuestro modelo para poder establecer ciertos parámetros de diseño, el siguiente paso consiste en probar nuestro algoritmo CGAMR en tiempo real usando nuestro motor DC. Los resultados de las pruebas en tiempo real que a continuación se muestran, serán adquiridos y almacenados en una base de datos, y la gráfica de dichos datos se hará en el software Excel

e) Prueba en tiempo real usando el método de selección de Rueda de Ruleta, utilizando los mismos parámetros y la misma entrada variable de la señal de referencia que en el caso de la simulación (b)

Parámetros de Diseño:

Set Point Inicial (rad/s)	100
N° de Generaciones	1000
N° de Individuos	80
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.0125
Método de Selección	Rueda de Ruleta
Operador Elitista	Sí
Rango de la ganancia K_p	[0 - 9999]*1e-5
Rango de la ganancia T_i	[0 - 9999]*1e-5
Rango de la ganancia T_d	[0 - 9999]*1e-5
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	2

Resultados:

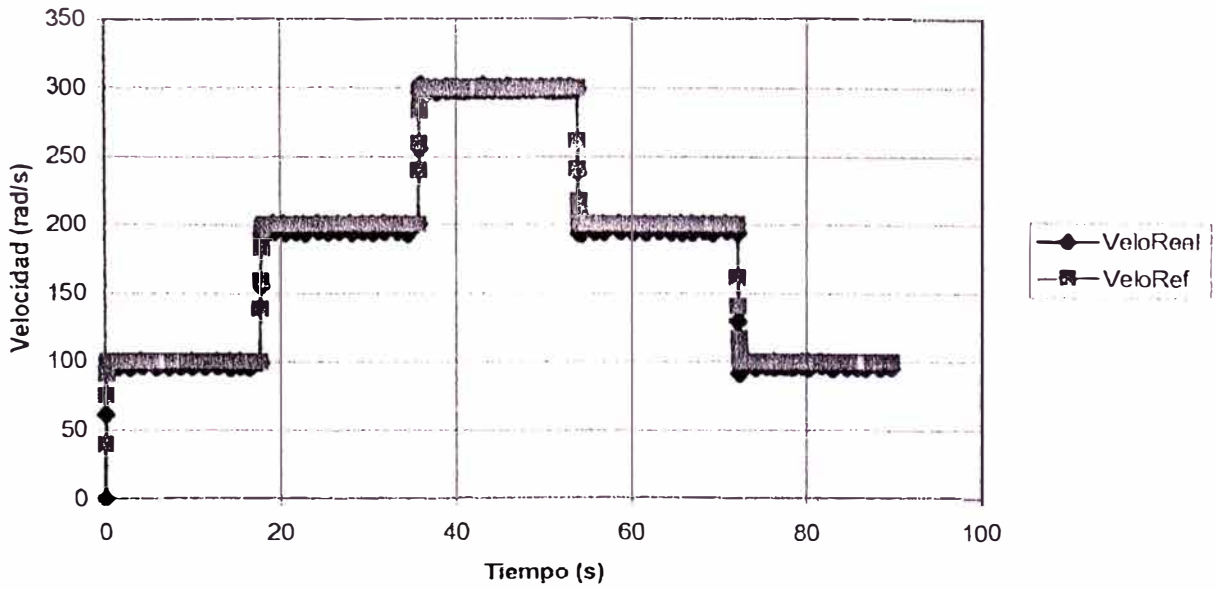

Figura (5.15a) Resultados que muestran el control en tiempo real del Motor DC (VeloReal) y la comparación con la salida del Modelo de Referencia(VeloRef)

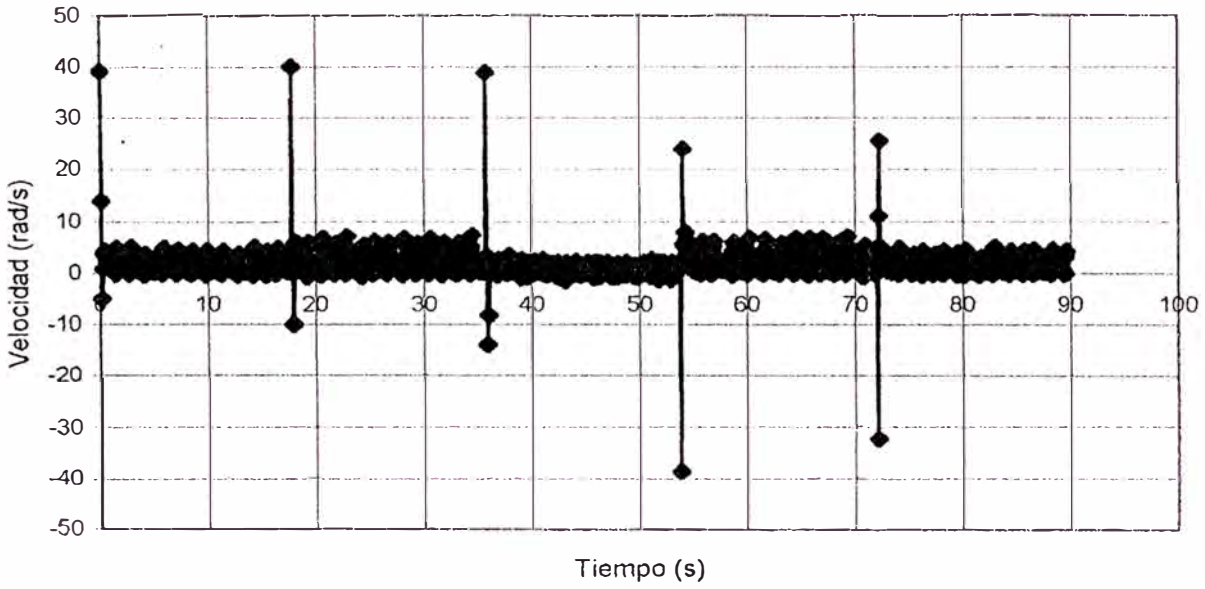

Figura (5.15b) Error entre la Velocidad del Motor DC y la salida del Modelo de Referencia, durante el control en tiempo Real

Como se puede apreciar en la figura (5.15a) el CGAMR hace que la salida de velocidad del motor se adapte muy bien a la salida del Modelo de Referencia tal y como se había simulado en el caso (b). El error que se puede apreciar en la figura (5.15b) es debido a que a pequeños ruidos que afectan la lectura de nuestro encoder, pero en general respetan el criterio de error en estado estable del 2% [OGATA_9]

f) La siguiente prueba se realizará, para probar el rendimiento de nuestro algoritmo CGAMR, cuando se aplica entradas tipo rampa, para la generación de un perfil de movimiento, y con los mismos parámetros de diseño que se presentaron en la simulación (c)

Parámetros de Diseño:

Set Point Inicial (rad/s)	40
N° de Generaciones	360
N° de Individuos	100
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.01
Método de Selección	Rueda de Ruleta
Operador Elitista	Sí
Rango de la ganancia K_p	[0 - 9999]*1e-5
Rango de la ganancia T_i	[0 - 9999]*1e-5
Rango de la ganancia T_d	[0 - 9999]*1e-5
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	3

Resultados:

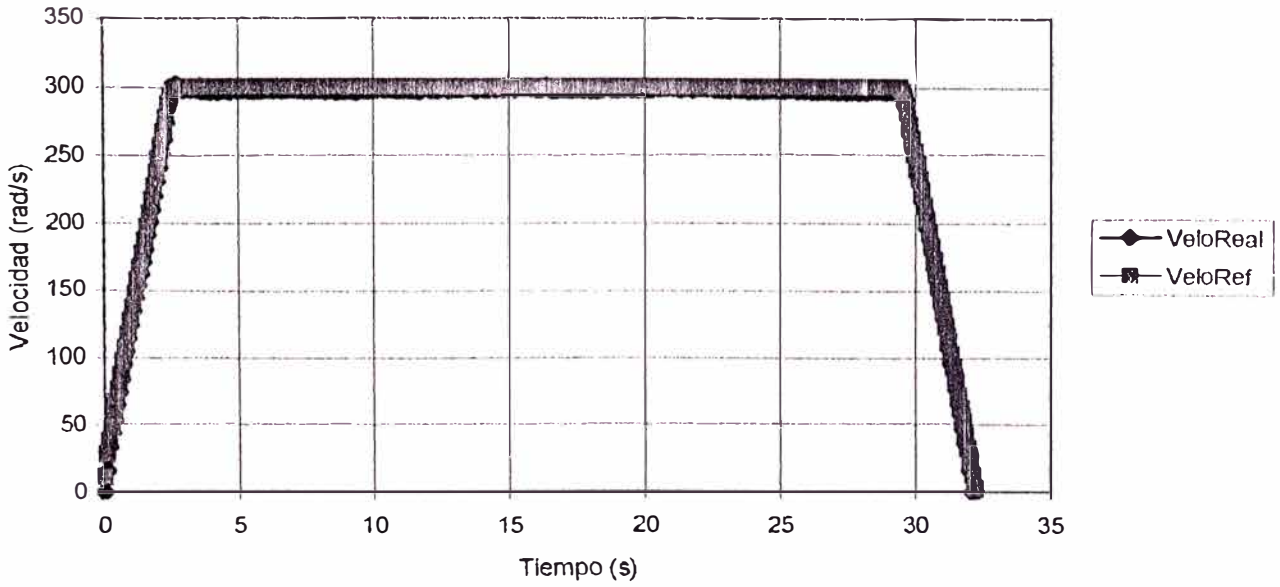

Figura (5.16a) Resultados del Motor DC (VeloReal) y el Modelo de Referencia (VeloRef), al aplicar señales de entrada tipo rampa durante el control en tiempo real y para la generación de un perfil de movimiento

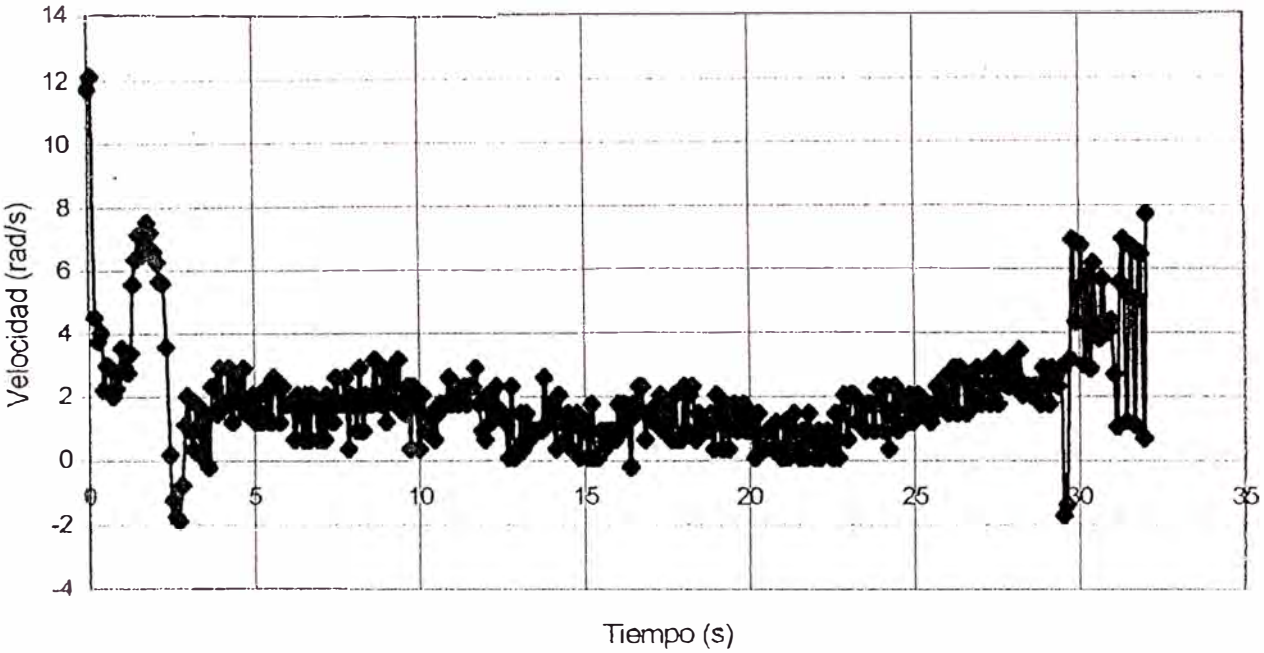

Figura (5.16b) Error entre la Velocidad del Motor DC y la salida del Modelo de Referencia durante la aplicación de señales rampa, en el control en tiempo real

También en la figura (5.16a) se aprecia que el motor en tiempo real se adapta muy bien frente a entradas tipo rampa, tal y como ocurre en la simulación (c), mientras que en la figura (5.16b) también podemos ver que el margen de error cumple con el criterio del 2% cuando se encuentra en velocidad constante, teniendo un error más alto durante la adaptación de la rampa, el cual representa el criterio del error en estado estable del 5%.

g) En esta última prueba se mostrará la versatilidad que tiene nuestro algoritmo para otro tipo de entradas, en este caso una entrada de tipo sinusoidal, la cual es usada en múltiples aplicaciones tales como el control de robots que realizan tareas tales como los remachados en serie, entre otros.

Parámetros de Diseño:

Set Point Inicial (rad/s)	200
N° de Generaciones	1400
N° de Individuos	100
Prob. de Cruzamiento (p_c)	0.9
Prob. de Mutación (p_m)	0.01
Método de Selección	Rueda de Ruleta
Operador Elitista	Sí
Rango de la ganancia K_p	[0 - 9999]*1e-5
Rango de la ganancia T_i	[0 - 9999]*1e-5
Rango de la ganancia T_d	[0 - 9999]*1e-5
Voltaje Máximo (Voltios)	15.8
ALFA	0.0001
BETA	2

Resultados:

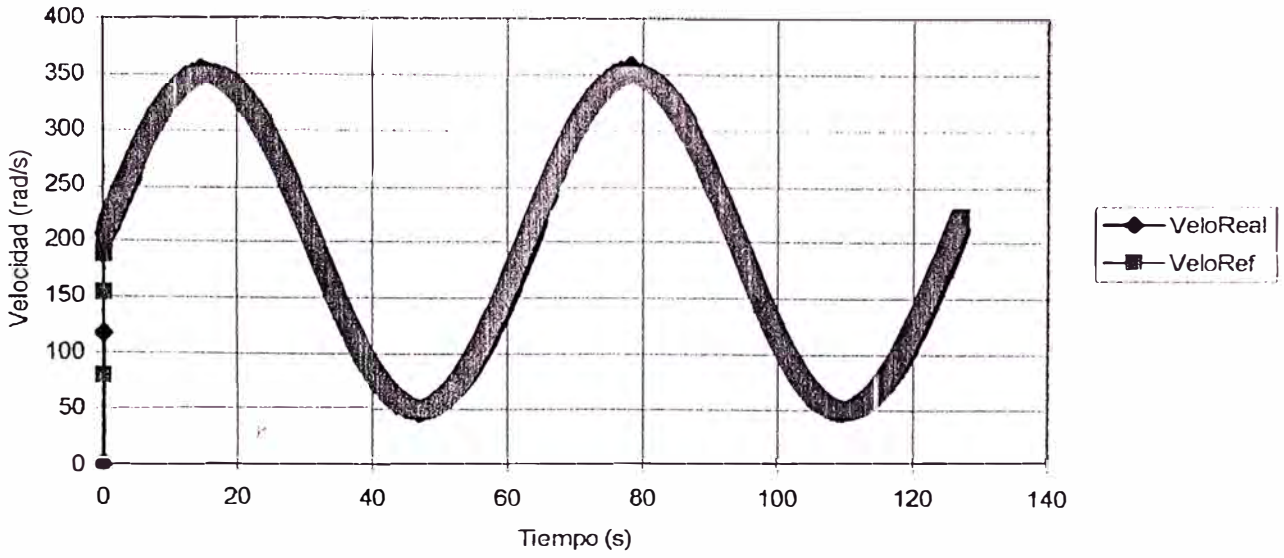

Figura (5.17a) Resultados del Motor DC (VeloReal) y el Modelo de Referencia (VeloRef), al aplicar señales de entrada tipo sinusoidal durante el control en tiempo real.

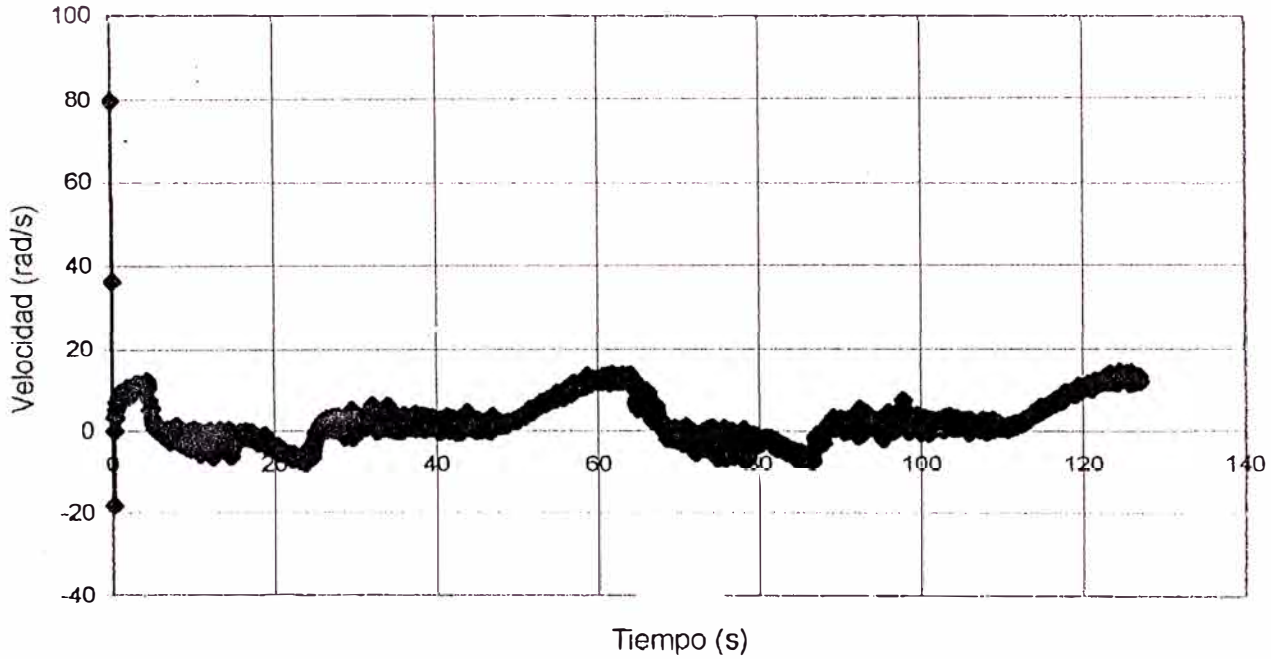

Figura (5.17b) Error entre la Velocidad del Motor DC y la salida del Modelo de Referencia durante la aplicación de señales sinusoidales, en el control en tiempo real

Como se puede apreciar en la figura (5.17a), el seguimiento de la salida sinusoidal del modelo de Referencia es muy bueno, aunque como se aprecia en la figura siguiente (5.17b) el error durante el flanco de subida de la señal sinusoidal es ligeramente más alto que en el flanco de bajada. En este tipo de seguimiento error es mayor que en los otros casos manteniéndose dentro del margen de −10 a +10 aproximadamente, esto debido a que el modelo de referencia está en constante cambio por lo que la adaptación se realiza constantemente.

OBSERVACIONES Y CONCLUSIONES

El buen diseño y la implementación de un algoritmo de control, implica conocer ciertas características del modelo del proceso que vamos a controlar. Pero muchas veces debido a la falta de información del mismo es imposible modelarlo con sus parámetros inherentes, por lo cual, como se pudo observar existen otras técnicas como la que escogimos, es decir el método de identificación de parámetros ARX, el cual nos permitió generar un modelo aproximado del mismo para usarlo en el diseño. También se pudo observar que el uso de las ganancias PID involucra tener presente el aporte que genera cada una de estas, así como el efecto de la variación de las mismas. Además fue necesario conocer dentro de que rangos se encontraban dichas ganancias, lo cual no hubiese sido posible sin valernos de un método complementario que nos diera el punto de arranque, en este caso nos referimos a los resultados obtenidos usando el primer método de sintonización de Z-N el cual se eligió por las características de nuestro modelo.

Luego, la combinación de una técnica de control avanzado, es decir el control adaptativo por modelo de referencia, con los métodos de optimización de los AGs, nos proporcionaron un enfoque distinto en el diseño de esquemas de control (CGAMR) para sistemas lineales, y sistemas no lineales (sistemas cuya dinámica puede linealizarse, en un punto de operación, para obtener un modelo aproximado) y en el cual solo es necesario conocer el rango aproximado en el que se encuentran las ganancias PID, ya que estas partiendo de valores aleatorios evolucionaran para encontrar los mejores

parámetros que cumplan en un momento dado, durante un tiempo determinado y de acuerdo a las variaciones de la dinámica del sistema a controlar. Los buenos resultados obtenidos se pudieron apreciar primero mediante simulaciones y luego en tiempo real con la aplicación en el control de velocidad del Motor DC.

Durante las simulaciones, el análisis de los primeros gráficos, es decir la figura (5.11a) y (5.11b) nos permitieron conocer el rango óptimo de cada una de las ganancias PID, mientras que a partir de las figuras (5.12a), (5.12b) ya se observaba buenos resultados aplicando la selección por Rueda de Ruleta, el cual muestra menos rendimiento que los resultados que se muestran en las figuras(5.13a) y (5.13b), donde se uso la selección por Torneo, pero el cual debido a su complejidad, tarda más tiempo computacional, tanto en la simulación como en otras pruebas en tiempo real, lo que al final resulta perjudicial al control, ya que es necesario aumentar el periodo de muestreo, contradiciendo la dinámica mucho más rápida que muestra el Motor DC.

Por otra parte, los AGs dieron resultados bastante aceptables durante la simulación de perfiles de movimiento, como se mostró en las figuras (5.14a) y (5.14b), los cuales se usan frecuentemente en el control de posición de motores DC, lo que nos lleva a concluir que el algoritmo de adaptatividad es muy eficiente para otro tipo de entradas que no sean escalón.

Ya durante las pruebas en tiempo real, el algoritmo trasladado de Matlab a Visual Basic, demostró alto rendimiento, para las entradas variables tipo escalón, tal y como se mostró en las figuras (5.15a) y (5.15b) ya que el error en estado estable se mantuvo dentro del criterio del 2%. También para las entradas tipo rampa y sinusoidales cuyos resultados se mostraron en las figuras (5.16a), (5.16b), (5.17a) y (5.17b), nuestro algoritmo CGAMR, mostró un ligero incremento del error, todo lo cual se había previsto en las simulaciones, pero que en la totalidad de la prueba demostró gran eficiencia, lo que nos lleva concluir que la elección de los rangos había sido correcta y que la búsqueda de soluciones que se efectuó usando los AGs había dado buenos resultados adaptando la salida de nuestro motor DC a la del Modelo de Referencia elegido.

BIBLIOGRAFIA

LIBROS:

[1] K.J. ASTROM y B. WITTENMARK – "Adaptive Control" - Addison Wesley

[2] D.E. GOLDBERG - "Genetic Algorithms in Search, Optimization, and Machine Learning" – Addison Wesley Publishing Company – 1989

[3] J.H. HOLLAND - "Adaptation in Natural and Artificial Systems: An Introductory Análisis with Applications to Biology, Control, and Artificial Intelligence" - MIT Press Edition - 1975

[4] P.A. IOANNOU y J. SUN – "Robust Adaptive Control" - Prentice Hall – 1996

[5] J.R. KOZA - "Genetic Programming: On the Programming of Computers by Means of Natural Selection" - MIT Press Edition – 1992

[6] B.C. KUO – "Sistemas de Control Automático" - Prentice Hall – 1996

[7] B. MARTIN DEL BRIO y A. SANZ - "Redes Neuronales y Sistemas Borrosos: Introducción teórica y práctica" - RAMA

[8] Z. MICHALEWICZ - "Genetic Algorithms + Data Structures = Evolutions Programs" – Springer – 1999

[9] K. OGATA - "Ingeniería de Control Moderna" - Prentice Hall – 1998

[10] F. RODRÍGUEZ y M.J. LOPEZ - "Control Adaptativo y Robusto" - Universidad de Sevilla – 1996

[11] J.J. SLOTINE y W. LI - "Applied Nonlinear Control" - Prentice Hall – 1991

PUBLICACIONES:

[12] R. GHANADAM – "Adaptive PID Control of Nonlinear Systems" – 1990

[13] F. JIMENEZ – "Computación Evolutiva" – Universidad de Murcia - 2001

[14] M. SALAMI y G. CAIN – "An Adaptive PID Controller Based on Genetic Algorithm Processor" – IEEE – 1995

[15] M.J. WILLIS – "Proportional, Integral y Derivative Control" – University of Newcastle - 1998

APENDICE "A"

TÉCNICA ARX DEL TOOLBOX DE IDENTIFICACIÓN DE SISTEMAS DEL MATLAB 6.0

arx

Estimate the parameters of an ARX or AR model.

Syntax

```
m = arx(data,orders)
m = arx(data,'na',na,'nb',nb,'nk',nk)
m= arx(data,orders,'Property1',Value1,...,'PropertyN',ValueN)
```

Description

The parameters of the ARX model structure

$$A(q)y(t) = B(q)u(t - nk) + e(t)$$

are estimated using the least-squares method.

`data` is an `iddata` object that contains the output-input data. `orders` is given as

```
orders = [na nb nk]
```

defining the orders and delay of the ARX model. Specifically,

$$na: \_\_\_ A(q) = 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na}$$
$$nb: \_\_\_ B(q) = b_1 + b_2 q^{-1} + \ldots + b_{nb} q^{-nb+1}$$

See Polynomial Representation of Transfer Functions in the "Tutorial" chapter for more information. The model orders can also be defined by explicit pairs `(...,'na',na,'nb',nb,'nk',nk,...)`.

`m` is returned as the least-squares estimates of the parameters. For single-output data this is an `idpoly` object, otherwise an `idarx` object.

For a time series, `data` contains no input channels and `orders = na`. Then an AR model of order `na` for `y` is computed.

$$A(q)y(t) = e(t)$$

Models with several inputs

$$A(q)y(t) = B_1(q)u_1(t - nk_1) + \ldots + B_{nu}u_{nu}(t - nk_{nu}) + e(t)$$

are handled by allowing `nb` and `nk` to be row vectors defining the orders and delays associated with each input.

Models with several inputs and several outputs are handled by allowing `na`, `nb`, and `nk` to contain one row for each output number. See Multivariable ARX Models: The idarx Model in the "Tutorial" chapter for exact definitions.

The algorithm and model structure are affected by the property name/property value list in the input argument.

Useful options are reached by the properties `'Focus'`, `'InputDelay'`, and `'MaxSize'`.

See `Algorithm Properties` for details of these properties and possible values

When the true noise term **e(t)** in the ARX model structure is not white noise and `na` is nonzero, the estimate does not give a correct model. It is then better to use `armax`, `bj`, `iv4`, or `oe`.

Examples

Here is an example that generates data and estimates an ARX model.

```
A = [1  -1.5  0.7]; B = [0 1 0.5];
m0 = idpoly(A,B);
u = iddata([],idinput(300,'rbs'));
e = iddata([],randn(300,1));
y = sim(m0, [u e]);
z = [y,u];
m = arx(z,[2 2 1]);
```

Algorithm

The least-squares estimation problem is an overdetermined set of linear equations that is solved using QR-factorization.

The regression matrix is formed so that only measured quantities are used (no fill-out with zeros). When the regression matrix is larger than `MaxSize`, the QR-factorization is performed in a `for`-loop.

arxdata

Extract the ARX parameters from `idmodel` models.

Syntax

```
[A,B] = arxdata(m)
[A,B,dA,dB] = arxdata(m)
```

Description

`m` is the model as an `idarx` or `idpoly` model object. `arxdata` will work on any `idarx` model. For `idpoly` it will give an error unless the underlying model is an ARX model, i.e., the orders `nc=nd=nf=0`. (See the reference page for `idpoly`.)

`A` and `B` are returned in the standard multivariable ARX format (see `idarx`), describing the model.

$$y(t) + A_1 y(t-1) + A_2 y(t-2) + \ldots + A_{na} y(t-na) =$$
$$B_0 u(t) + B_1 u(t-1) + \ldots + B_{nb} u(t-nb) + e(t)$$

Here A_k and B_k are matrices of dimensions *ny*-by-*ny* and *ny*-by-*nu*, respectively (*ny* is the number of outputs, i.e., the dimension of the vector $y(t)$ and *nu* is the number of inputs). See Multivariable ARX Models: The idarx Model in the "Tutorial" chapter.

The arguments `A` and `B` are 3-D arrays that contain the A matrices and the B matrices of the model in the following way:

A is an `ny`-by-`ny`-by-`(na+1)` array such that

```
A(:,:,k+1) = Ak
A(:,:,1) = eye(ny)
```

Similarly B is an `ny`-by-`nu`-by-`(nb+1)` array with

```
B(:,:,k+1) = Bk
```

Note that `A` always starts with the identity matrix, and that leading entries in `B` equal to zero means delays in the model. For a time series `B = []`.

`dA` and `dB` are the estimated standard deviations of `A` and `B`.

arxstruc

Compute loss functions for a set of different model structures of single-output ARX type.

Syntax

```
V   arxstruc(ze,zv,NN)
V   arxstruc(ze,zv,NN,maxsize)
```

Description

NN is a matrix that defines a number of different structures of the ARX type. Each row of NN is of the form

```
nn = [na nb nk]
```

with the same interpretation as described for arx. See struc for easy generation of typical NN matrices for single-input systems.

Each of ze and zv are iddata objects containing output-input data. Models for each of the model structures defined by NN are estimated using the data set ze. The loss functions (normalized sum of squared prediction errors) are then computed for these models when applied to the validation data set zv. The data sets, ze and zv, need not be of equal size. They could, however, be the same sets, in which case the computation is faster.

Note that arxstruc is intended for single-output systems only.

The output argument V is best analyzed using selstruc. It contains the loss functions in its first row. The remaining rows of V contain the transpose of NN, so that the orders and delays are given just below the corresponding loss functions. The last column of V contains the number of data points in ze. The selection of a suitable model structure based on the information in V is normally done using selstruc. See Model Structure Selection and Validation in the "Tutorial" chapter for advice on model structure selection and cross-validation.

See Algorithm Properties for an explanation of maxsize.

Examples

Compare first to fifth order models with one delay using cross-validation on the second half of the data set. Then select the order that gives the best fit to the validation data set.

```
NN = struc(1:5,1:5,1);
V = arxstruc(z(1:200),z(201:400),NN);
nn = selstruc(V,0);
```

m = arx(z,nn);

B.3 PROGRAMAS EN VISUAL BASIC PARA LA IMPLEMENTACION DEL CONTROL DEL MOTOR DC USANDO CGAMR

A continuación mostraremos el programa hecho en Visual Basic 6.0 el cual nos servirá para controlar la velocidad del Motor DC usando CGAMR. En la figura (B.1) se puede apreciar la interfaz gráfica del programa, el cual nos permite ingresar los parámetros del AG, así como la velocidad inicial deseada. Una vez corriendo el programa en la sección de DATOS RECIBIDOS, podremos apreciar tanto la velocidad real a la que se mueve el motor como el número de generaciones(que es igual al número de periodos de muestreo)

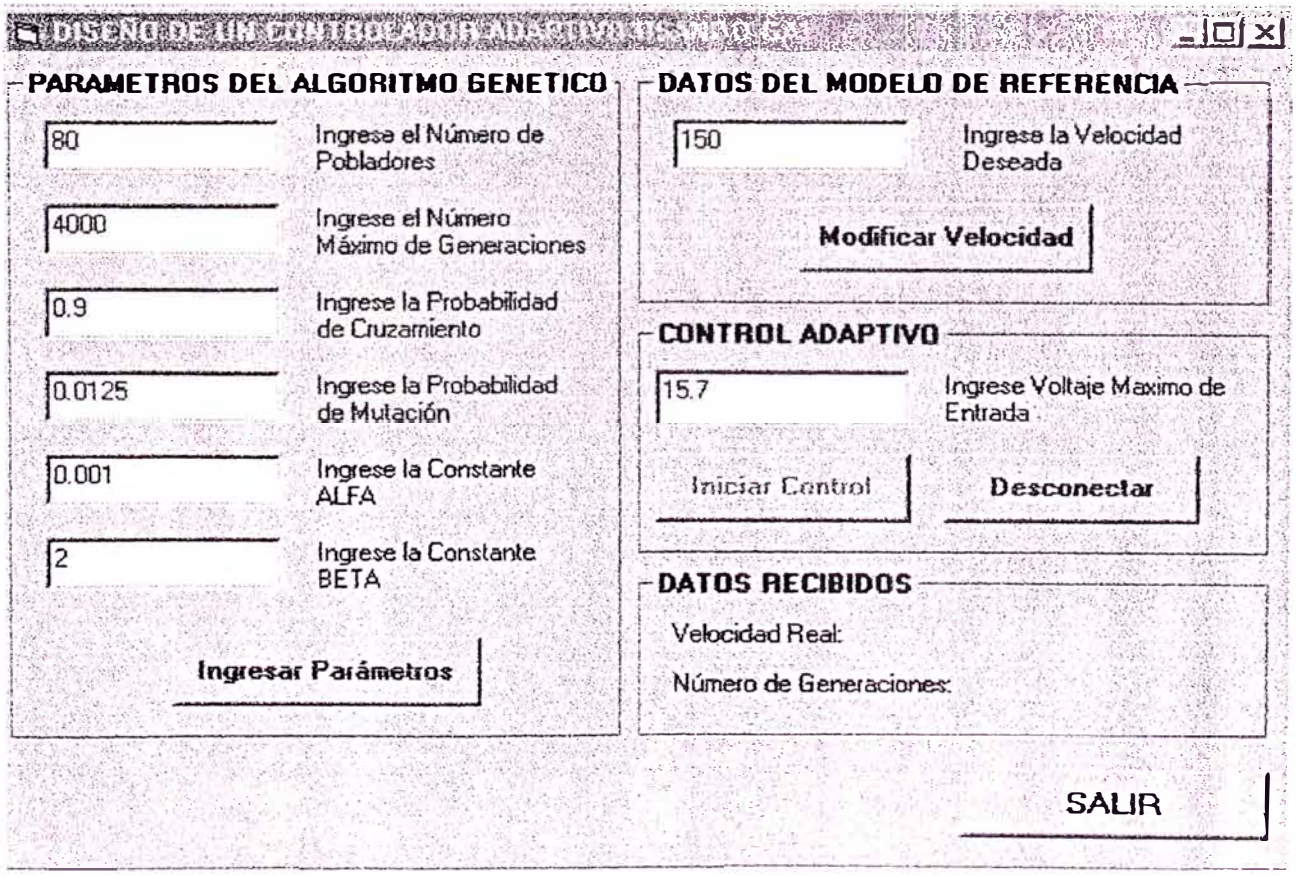

Figura (B.1) En esta figura se puede apreciar la interfaz del programa de control de velocidad de un motor DC Pittman, usando CGAMR

APENDICE "B"

PROGRAMAS DE SIMULACIÓN E IMPLEMENTACION DEL CGAMR

B.1 PROGRAMAS EN MATLAB PARA LA SIMULACIÓN DEL CGAMR

Dentro de las funciones que se necesita dentro del programa de adaptatividad encontramos, los siguientes archivos: *decimalPF.m*, *segmentación.m*, *seleccionR.m*, *seleccionT.m*, *flip.m*, *mutacion.m*, *cruzamiento.m*, *generacionR.m*, *generacionT.m*. Y como programa principal que hace uso de las funciones anteriores encontramos a: *adaptivo.m*. A continuación se mostrarán los códigos de las funciones y el programa principal, hechos en Matlab 6.0, y que nos permitieron hacer las simulaciones de nuestro CGAMR.

1. **Función "decimalPF.m".-** Es una función que convierte un vector v compuesto por números que van del 0 al 9, en un numero entero.

Código:

```
function s =decimalPF(v)
lo=length(v);
s=0;
for i=1:lo
   s=s+v(i)*10^(lo-i);
end
```

Ejemplo:

```
v = [7 5 8 9];
b = decimalPF(v)
>> b = 7589            %Respuesta
```

2. **Función "segmentación.m".-** Es una función que convierte un número entero en un vector de longitud lcromo.

Código:

```
function L=segmentacion(a,lcromo)
t=a/10;
v=1;
while t>=0.1
  k(v)=round((t-floor(t))*10);
  t=floor(t);
  t=t/10;
  v=v+1;
end
j=0;
for m=v-1:-1:1;
   j=j+1;
   L(m)=k(j);
end
if a==0
   L=[zeros(1,lcromo)];
elseif length(L)<lcromo
  L=[zeros(1,lcromo-length(L)) L];
end
```

Ejemplo:

```
k = 7589;
lcromo = 8;
b = segmentacion(k,lcromo)
>> b =  [0 0 0 0 7 5 8 9]              %Respuesta
```

3. **Función "seleccionR.m".-** Es la función que realiza la selección entre los individuos de la población usada en los AGs por el método de Rueda de Ruleta.

Código:

```
function y=seleccionR(numpobl,sumFitness,Fitness)
sumParcial=0;
j=0;
aleatorio=rand(1)*sumFitness;
while (sumParcial<aleatorio) & (j~=numpobl)
  j=j+1;
  sumParcial=sumParcial+Fitness(j);
end
y=j;
```

4. **Función "seleccionT.m".-** Es la función que realiza la selección entre los individuos de la población usada en los AGs por el método de Torneo.

Código:

```
function y=seleccionT(numpobl,sumFitness,Fitness)
par1=seleccionR(numpobl,sumFitness,Fitness);
par2=seleccionR(numpobl,sumFitness,Fitness);
if Fitness(par1)>=Fitness(par2)
  y=par1;
else
  y=par2;
end
```

5. **Función "flip.m".-** Es una función que tiene como salida el valor lógico de la comparación dc un valor aleatorio con una probabilidad dada como entrada.

Código:

```
function y=flip(probabilidad)
if probabilidad==1
  y=1;
```

```
elseif rand(1)<=probabilidad
  y=1;
else
  y=0;
end
```

6. **Función "mutación.m".-** Es la función que realiza la mutación de cualquier gen perteneciente al cromosoma, lo cual se realiza dependiendo de la probabilidad *pmuta* que tenga para mutarse. Además devuelve el número de mutaciones que se viene realizando.

Código:

```
function [y,nmuta]=mutacion(gen,pmuta,nmuta)
if flip(pmuta)
  nmuta=nmuta+1;
  y=round(rand(1)*9);
else
  y=gen;
end
```

7. **Función "cruzamiento.m".-** Es la función encargada de realizar el cruzamiento entre dos cromosomas los cuales pertenecen a la población cuyos miembros fueron seleccionados para evolucionar.

Código:

```
function [hijo1,hijo2,ncruza,nmuta,jcruza]=cruzamiento(padre1,
padre2,lcromo,ncruza,nmuta,pcruza,pmuta,desactivar)
if flip(pcruza)
  jcruza=round((lcromo-2)*rand(1))+1;
```

```
  ncruza=ncruza+1;
else
  jcruza=lcromo;
end
if desactivar.
   jcruza=lcromo;
end
for j=1:jcruza
        [hijo1(j),nmuta]=mutacion(padre1(j),pmuta,nmuta);
  [hijo2(j),nmuta]=mutacion(padre2(j),pmuta,nmuta);
end
if jcruza~=lcromo
        for j=(jcruza+1):lcromo
        [hijo1(j),nmuta]=mutacion(padre2(j),pmuta,nmuta);
    [hijo2(j),nmuta]=mutacion(padre1(j),pmuta,nmuta);
  end
end
```

8. **Función "generaciónR.m".-** Es la función que se encarga de evolucionar a los individuos de la población, basándose en ciertos parámetros, como la probabilidad de mutación, cruzamiento, el número de pobladores, y el rango y la precisión de los mismos.

Código:

```
function [ncruza,nmuta,Kp,Ti,Td,pobla]=generacionR(numpobl,sumFitness,Fitness,
lcromo,ncruza,nmuta,pcruza,pmuta,pobla_ant,nmax,elitismo)
j=1;
while (j<=numpobl)
  desactivar=0;
  S1=seleccionR(numpobl,sumFitness,Fitness);
  S2=seleccionR(numpobl,sumFitness,Fitness);
  if elitismo
     if S1==nmax | S2==nmax
        desactivar=1;
        elitismo=0;
     end
```

```
  end
[pobla(j,:),pobla(j+1,:),ncruza,nmuta,jcruza]=cruzamiento(pobla_ant(S1,:),pobla_ant(
S2,:),lcromo,ncruza,nmuta,pcruza,pmuta,desactivar);
  j=j+2;
end
for i=1:numpobl
  Kp(i)=decimalPF(pobla(i,1:lcromo/3))*0.00001;
  Ti(i)=decimalPF(pobla(i,(lcromo/3)+1:2*(lcromo/3)))*0.00001;
  if Ti(i)==0
     Ti(i)=0.1;
  end
  Td(i)=decimalPF(pobla(i,2*(lcromo/3)+1:lcromo))*0.00001;
end
```

9. **Función "generaciónT.m".-** Es lo mismo que la función anterior, pero usando el método de selección por Torneo.

Código:

```
function [ncruza,nmuta,Kp,Ti,Td,pobla]=generacionR(numpobl,sumFitness,Fitness,
lcromo,ncruza,nmuta,pcruza,pmuta,pobla_ant,nmax,elitismo)
j=1;
while (j<=numpobl)
  desactivar=0;
  S1=seleccionT(numpobl,sumFitness,Fitness);
  S2=seleccionT(numpobl,sumFitness,Fitness);
  if elitismo
     if S1==nmax | S2==nmax
        desactivar=1;
        elitismo=0;
     end
  end
[pobla(j,:),pobla(j+1,:),ncruza,nmuta,jcruza]=cruzamiento(pobla_ant(S1,:),pobla_ant(
S2,:),lcromo,ncruza,nmuta,pcruza,pmuta,desactivar);
  j=j+2;
end
for i=1:numpobl
  Kp(i)=decimalPF(pobla(i,1:lcromo/3))*0.00001;
  Ti(i)=decimalPF(pobla(i,(lcromo/3)+1:2*(lcromo/3)))*0.00001;
```

```
  if Ti(i)==0
     Ti(i)=0.1;
  end
  Td(i)=decimalPF(pobla(i,2*(lcromo/3)+1:lcromo))*0.00001;
end
```

10. **Programa "adaptivo.m".-** En este programa se definen los parámetros del motor, del modelo de referencia, los parámetros de los AGs, para luego ingresar en el bucle que permite simular el proceso de adaptación de nuestro sistema.

Código:

```
clear all
close all
clc

%Parámetros estimados del motor DC usando ARX
a1 = 13.7958;
a2 = 9.0707;
a3 = 0.2998;
b1 = -0.1066;
b2 = -0.0068423;
b3 = -0.0001769;

%Parámetros del Modelo de Referencia
a1r = 0.2275;
a2r = 0.1246;
b1r = -0.8131;
b2r = 0.1653;

velocidad=40;
%Otros Parámetros
velocidadD(1) = 0;
velocidadD(2) = 0;
velocidadD(3) = velocidad;
Yref(1) = 0;
Yref(2) = 0;
```

```
Yref(3) = 0;
Yref(4) = a1r * velocidadD(3) + a2r * velocidadD(2) - b1r * Yref(3) - b2r * Yref(2);

Ucontrol(1) = 0;
Ucontrol(2) = 0;
Ucontrol(3) = 0;
YiDis(1) = 0;
YiDis(2) = 0;
YiDis(3) = 0;
veloReal(1) = 0;
veloReal(2) = 0;
veloReal(3) = 0;
veloReal(4) = 0;
u=[0 0 0];

%Parámetros del Algoritmo Genético
lcromo=12;                  %Longitud de Cromosoma: [----Kp(4)----!----Ki(4)----!----
Kd(4)----]
numpobl=100;                %Número de Pobladores
numgen=360;                 %Numero de Generaciones
ncruza=0;                   %Inicialización de Número de cruzamientos
nmuta=0;                    %Inicialización de Número de mutaciones
pcruza=0.9;                 %Probabilidad de Cruzamiento
pmuta=1/numpobl;            %Probabilidad de Mutación

%generando la población inicial;
for i=1:numpobl
  Kp(i)=decimalPF(round(rand(1,lcromo/3)*9))*0.00001;
  Ti(i)=decimalPF(round(rand(1,lcromo/3)*9))*0.00001;
  Td(i)=decimalPF(round(rand(1,lcromo/3)*9))*0.00001;
end
%Concatenando los valores de Kp, Ki y Kd en un solo cromosoma
for i=1:numpobl
   aux=[segmentacion(Kp(i)*100000,lcromo/3) segmentacion(Ti(i)*100000,lcromo/3)
segmentacion(Td(i)*100000,lcromo/3)];
   pobla(i,:)=aux;
end
%Generación de la población Inicial, al igual que inicialización de la generación
antigua
pobla_ant=pobla;

%Parámetros para el tiempo Discreto
tf=1;
Tmuest=0.09;
```

```
NumPred=10;
ALFA = 0.0001;
BETA = 2;
VoltMax=15.8;
%Inicio del Algoritmo Genético para el número dado de Generaciones
ErrorV=0;
TDerivadAnt=0;
TIntegraAnt=0;
velocidadR=0;
for n=4:numgen
   n
   %Entrada tipo rampa para perfil de movimiento
   if n<=30
      velocidad=10*n;
   elseif n>30 & n<=330
      velocidad=300;
   else
     velocidad=-10*n+3600;
   end
   velocidadD(n)=velocidad;
   ErrorV = velocidadD(n) - veloReal(n);
   delta(n - 2) = velocidadD(n - 2);
   delta(n - 3) = velocidadD(n - 3);
   for j=n-1:NumPred + n-1
      delta(j)= velocidadD(n);
      Yref(j + 1) = (a1r * delta(j) + a2r * delta(j - 1) - b1r * Yref(j) - b2r * Yref(j - 1));
   end
   ErrorModel = Yref(n) - veloReal(n);
   suma = 0;
   minimo = 0;
   maximo = 0;
   nmax = 0;
   for i = 1:numpobl
      TProporc = Kp(i) * ErrorV;
      Tau=Td(i)/10;
      Tt=Ti(i);
      TDerivad = (Tau/(Tau+Tmuest))*TDerivadAnt +
Kp(i)*Td(i)*Tmuest*(veloReal(n) - veloReal(n-1));
      TIntegra = TIntegraAnt + (Kp(i)/Ti(i))*Tmuest*ErrorV+(1/Tt)*(Ucontrol(n-1)-
u(n-1));
      uci(i) = TProporc + TDerivad + TIntegra;
      if uci(i) > VoltMax
         uci(i) = VoltMax;
      elseif uci(i) < 0
```

```
        uci(i) = 0;
      end
      Uaux(n - 2) = Ucontrol(n - 2);
      Uaux(n - 3) = Ucontrol(n - 3);
      for j = n - 1:NumPred + (n - 1)
         Uaux(j) = uci(i);
         YiDis(j + 1) = (a1 * Uaux(j) + a2 * Uaux(j - 1) + a3 * Uaux(j - 2)) - b1 *
YiDis(j) - b2 * YiDis(j - 1) - b3 * YiDis(j - 2);
      end
      ErrorModelN = Yref(NumPred + (n - 1)) - YiDis(NumPred + (n - 1));
      ErrorModelP = (ErrorModelN - ErrorModel) / (NumPred * Tmuest);
      Ji = ErrorModel + BETA * ErrorModelP;
      Fitness(i) = ALFA / (Ji^2 + ALFA);
      if minimo >= Fitness(i)
         minimo = Fitness(i);
      end
      if maximo <= Fitness(i)
         maximo = Fitness(i);
         nmax = i;
      end
      suma = suma + Fitness(i);
   end
   TProporc = Kp(nmax) * ErrorV;
   Tau=Td(nmax)/10;
   Tt=Ti(nmax);
   TDerivad = (Tau/(Tau+Tmuest))*TDerivadAnt +
Kp(nmax)*Td(nmax)*Tmuest*(veloReal(n) - veloReal(n-1));
   TIntegra = TIntegraAnt +
(Kp(nmax)/Ti(nmax))*Tmuest*ErrorV+(1/Tt)*(Ucontrol(n-1)-u(n-1));
   Ucontrol(n) = TProporc + TDerivad + TIntegra;
   TDerivadAnt = TDerivad;
   TIntegraAnt = TIntegra;
   u(n)=Ucontrol(n);
   if Ucontrol(n) > VoltMax
      Ucontrol(n) = VoltMax;
   elseif Ucontrol(n) < 0
      Ucontrol(n) = 0;
   else
      Ucontrol(n) = Ucontrol(n);
   end
   veloReal(n+1) = (a1 * Ucontrol(n) + a2 * Ucontrol(n-1) + a3 * Ucontrol(n-2)) - b1 *
veloReal(n) - b2 * veloReal(n - 1) - b3 * veloReal(n - 2);
   sumFitness = suma;
   maxFitness = maximo;
```

```
    minFitness = minimo;
    avgFitness = sumFitness / numpobl;
    elitismo=1;

[ncruza,nmuta,Kp,Ti,Td,pobla]=generacionR(numpobl,sumFitness,Fitness,lcromo,ncru
za,nmuta,pcruza,pmuta,pobla_ant,nmax,elitismo);
    %Población Anterior = Población Nueva para siguiente Generación
    pobla_ant=pobla;
    KpsMax(n)=Kp(nmax);
    TisMax(n)=Ti(nmax);
    TdsMax(n)=Td(nmax);
    N_Generaciones=n;
end
N  Cruzamientos=ncruza
N  Mutaciones=nmuta
for i=1:numpobl
    if (Fitness(i)==max(Fitness))
        n=i;
        break
    end
end
figure(1)
subplot(311)
plot(KpsMax,'b')
ylabel('Kp')
grid
subplot(312)
plot(TisMax,'r')
ylabel('Ti')
grid
subplot(313)
plot(TdsMax,'g')
ylabel('Td')
grid
t=0:Tmuest:(numgen-3)*Tmuest;
figure(2)
plot(t,veloReal(4:numgen+1),'r')
hold on
plot(t,Yref(3:numgen),'--b')
xlabel('Tiempo(s)')
ylabel('Ym(- -) e Yp(-)')
grid
hold off
```

```
figure(3)
plot(t,Yref(3:numgen)-veloReal(4:numgen+1),'r')
title('Error entre el Modelo de Referencia y el Motor DC')
ylabel('Velocidad(rad/s)')
xlabel('Tiempo(s)')
grid
```

B.2 PROGRAMAS EN LENGUAJE ENSAMBLADOR PARA EL PIC 16F873

A continuación se presenta el código del programa escrito en lenguaje ensamblador *pwm.asm* para el PIC 16F873, que usaremos como interfaz de control entre la Pc y el motor DC.

Código:

```
; PROGRAMA PARA CONTEO DE ENCODER DE UN MOTOR
; USANDO EL TMRO Y TRANSMITIDO A LA PC
; SU VELOCIDAD ES SETEADA DESDE LA PC POR
; PUERTOSERIAL VARIANDO EL DUTY CYCLE DEL PWM
        list p=16f873
        ERRORLEVEL -302
        #include p16f873.inc
    CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _LVP_OFF
;-----------------DIRECCIONANDO REGISTROS-----------------------------------
DESB                equ    20h
FLAG                equ    21h
PULSOS              equ    22h
DATORECIBE          equ    24h
CONTA               equ    25h
PWM_ALTA            equ    26h
PWM_BAJA            equ    27h

        org    00h
        goto   inicio
        org    04h
        goto   interr
        org    05h
inicio
```

```
;--------------------------------CONFIGURACIONES-------------------------------
        bcf     STATUS,RP0  ;selecciono BANCO 0
        bcf     STATUS,RP1
        clrf    PORTA
        clrf    PORTB
        clrf    PORTC
        clrf    TMR0
        clrf    CONTA
        clrf    DATORECIBE
        clrf    CCP1CON
        clrf    INTCON
        clrf    TMR2
        clrf    CCPR1L
        clrf    PWM_ALTA
        clrf    PWM_BAJA
        bcf     RCSTA,RX9    ;Se configura datos de 8 bits
        bcf     RCSTA,ADDEN  ;Deshabilita deteccion de direccion
        bsf     RCSTA,CREN   ;Habilita recepcion
        bsf     RCSTA,SPEN   ;Habilita puerto serial asincrono
        bsf     STATUS,RP0  ;selecciono BANCO 1
        movlw   0XFF
        movwf   PR2
        movlw   b'00000110'  ;configurando el PORTA
        movwf   ADCON1       ;como E/S digitales
        movlw   b'00111111'
        movwf   TRISA        ;PA entradas digitales
        clrf    TRISB        ;PB salidas digitales
        clrf    TRISC        ;PC salidas digitales
        bsf     TRISC,7      ;Configura PortC(0-5:Entradas Digitales)
                             ;(7:Recepcion, 6:Transmisión)
        clrf    PIE1
        movlw   b'10100000'  ;se selecciona TMR0 como reloj interno
        movwf   OPTION_REG         ;escala de 1:2
                             ;contador de pulsos externos por RA4
        bcf     TXSTA,SYNC  ;Habilita puerto serial asincrono
        bsf     TXSTA,BRGH
        movlw   d'25'        ;Configura a 9600 baudios d'25'
                             ;si deseo 2400 baudios d'25'
                             ;si deseo 1200 baudios d'51'
                             ;si deseo 4800 baudios d'12'
                             ;todos con BRGH=0
                             ;si deseo 9600 d'25' y BRGH=1
        movwf   SPBRG
        bsf     TXSTA,TXEN  ;Habilita la transmision
```

```
        movlw b'00000001'   ;interrupcion por overflow del TMR1
        movwf PIE1
        bsf   PIE1,RCIE     ;interrupcion por recepción
;                                   ---------------------------------------
        bcf   STATUS,RP0  ;selecciono BANCO 0
        clrf  PIR1
        movlw b'00111000'   ;TMR1 escala 1:8, reloj interno
        movwf T1CON
        movlw b'00001100'   ;Modo PWM
        movwf CCP1CON
        bsf   T2CON,T2CKPS1     ;escala maxima de 1:16 del TMR2
        bsf   T2CON,TMR2ON      ;TMR2 empieza a contar

repite  bsf   INTCON,GIE ;Es 0, habilita todas las interrupciones
        bsf   INTCON,PEIE       ;Habilita todas las interrupciones por periférico
        movlw h'AC'
        movwf TMR1L
        movlw h'F8'
        movwf TMR1H
        bsf   INTCON,T0IE;Activa interrupcion por overflow Timer0
        bcf   FLAG,0            ;Pone 0 a bit0 del FLAG
        bcf   INTCON,T0IF;Limpia el flag de Interrupcion de overflow del Timer0
        bcf   PIR1,TMR1IF ;Limpia el flag de Interrupcion de overflow del Timer1
        clrf  TMR0          ;Limpia TMR0
        bsf   T1CON,TMR1ON      ;Activa el Timer1
bucle   btfss FLAG,0
        goto  bucle
        bcf   INTCON,GIE ;Deshabilita todas las interrupciones
        bcf   INTCON,EEIE       ;Deshabilita las interrupciones de escritura EE
        goto  repite
;-----------------------------INTERRUPCIONES-----------------------
interr  btfsc PIR1,RCIF
        goto  serial
        btfsc INTCON,T0IF
        goto  desb  T0
        btfsc PIR1,0
        goto  desb  T1
        goto  salir

desb  T0        bcf     INTCON,T0IF
        incf    DESB,F
        goto    salir

desb  T1        movf    TMR0,W
```

```
        movwf PULSOS
        bcf   PIR1,TMR1IF ;Limpia Flag de Overflow Timer1
        bcf   T1CON,TMR1ON    ;Para el Timer1
        movlw h'AC'
        movwf TMR1L
        movlw h'F8'
        movwf TMR1H
        bsf   T1CON,TMR1ON    ;Activa el Timer1
        clrf  TMR0            ;Empieza a contar de nuevo
        bcf   PIR1,TXIF
        movf  PULSOS,W
        movwf TXREG           ;transmite la data deseada
        bsf   STATUS,RP0      ;Banco 1
check1  btfss TXSTA,TRMT      ;¿Byte transmitido?
        goto  check1          ;No transmitido
        bcf   STATUS,RP0      ;Si transmitido ir al Banco 0
        ;bsf  FLAG,0
        goto  salir

serial  bcf   PIR1,RCIF
        movf  RCREG,W
        movwf DATORECIBE
        movlw b'00000001'
        subwf CONTA,W
        btfss STATUS,Z
        goto  primero
        goto  segundo
primero movf  DATORECIBE,W
        movwf PWM_ALTA
        incf  CONTA,F
        goto  salir
segundo movf  DATORECIBE,W
        movwf PWM_BAJA
        clrf  CONTA
        movf  PWM_BAJA,W      ;Setea los valores de PWM
        movwf CCP1CON
        movf  PWM_ALTA,W
        movwf CCPR1L
salir   retfie

        end
```

Código:

```
Option Explicit
Dim Yref(0 To 4000), velocidadD(0 To 4000) As Single
Dim Ucontrol(0 To 4000), u(0 To 4000) As Single
Dim delta(0 To 4000) As Integer
Dim a1r, a2r, b1r, b2r As Single
Dim a1, a2, a3, b1, b2, b3 As Single
Dim cnn As ADODB.Connection
Dim datosGuardar As ADODB.Recordset
Dim Tmuest, velocidadR, velocidad As Single
Dim Kp(1 To 200), Ti(1 To 200), Td(1 To 200) As Single
Dim KpC(1 To 200), TiC(1 To 200), TdC(1 To 200) As Integer
Dim ErrorV, VoltMax, VoltSal As Single
Dim TProporc, TDerivad, TIntegra, TDerivadAnt, TIntegraAnt, Tau As Single
Dim uci(1 To 4000) As Single
Dim ErrorModel, ErrorModelN, ErrorModelP As Single
Dim veloReal(0 To 4000), YiDis(0 To 4000), Uaux(0 To 4000) As Single
Dim numpobl, n, total, alta, baja As Integer
Dim NumPred, i, j, NumGenMax, nmax, lcromo As Integer
Dim pobla(1 To 4000, 1 To 100), pobla_ant(1 To 4000, 1 To 100) As Integer
Dim Ji(1 To 200) As Double
Dim ALFA, BETA, Tt As Single
Dim DatoIn(1 To 6) As Single
Dim sw, desactivar, elitismo As Boolean
Dim ncruza, nmuta, pcruza, pmuta, sumFitness, maxFitness, minFitness, avgFitness As
Single

Private Sub cmdControlar_Click()
Call IniPobla

velocidad = CSng(txtVeloDeseada.Text)
''''''''Parámetros del modelo de Referencia''''''''
a1r = 0.3908
a2r = 0.1579
b1r = -0.5185
b2r = 0.0672
''''''''Parámetros del modelo aproximado del Motor''''''''
a1 = 13.7958
a2 = 9.0707
a3 = 0.2998
b1 = -0.1066
b2 = -0.0068423
b3 = -0.0001769
```

```
''''''''Parámetros de iniciación''''''''''
velocidadD(0) = 0
velocidadD(1) = 0
velocidadD(2) = velocidad
Yref(0) = 0
Yref(1) = 0
Yref(2) = 0
Yref(3) = a1r * velocidadD(2) + a2r * velocidadD(1) - b1r * Yref(2) - b2r * Yref(1)

Ucontrol(0) = 0
Ucontrol(1) = 0
Ucontrol(2) = 0
u(0) = 0
u(1) = 0
u(2) = 0
YiDis(0) = 0
YiDis(1) = 0
YiDis(2) = 0
n = 3
sw = True
Tmuest = 0.09
velocidadD(n) = velocidad
VoltMax = Val(txtVoltMax.Text)
NumPred = 10
ncruza = 0
nmuta = 0
ErrorV = 0
TDerivadAnt = 0
TIntegraAnt = 0
MSComm1.PortOpen = True
End Sub
Private Sub cmdDesconectar_Click()
MSComm1.PortOpen = False
End Sub

Private Sub cmdIngresarAG_Click()
numpobl = CInt(txtNumPobla.Text)
NumGenMax = CInt(txtNumMaxGen.Text)
pcruza = CSng(txtProbCruz.Text)
pmuta = CSng(txtProbMut.Text)
ALFA = CSng(txtALFA.Text)
BETA = CSng(txtBETA.Text)
cmdControlar.Enabled = True
End Sub
```

```
Private Sub cmdModificar_Click()
velocidad = CSng(txtVeloDeseada.Text)
End Sub

Private Sub cmdSalir_Click()
Unload Me
End Sub

Private Sub Form_Load()
Set cnn = New ADODB.Connection
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\WINDOWS\Escritorio\controlmotor\adaptivo4\datos.mdb;Persist Security
Info=False"
cnn.Open
Set datosGuardar = New ADODB.Recordset
datosGuardar.CursorType = adOpenKeyset
datosGuardar.LockType = adLockOptimistic
datosGuardar.Open "DatosModelo", cnn, , , adCmdTable
MSComm1.CommPort = 1
MSComm1.Settings = "9600,N,8,1"
MSComm1.Handshaking = comNone
MSComm1.RThreshold = 6
MSComm1.InputLen = 1
MSComm1.InputMode = comInputModeBinary
MSComm1.InBufferCount = 0
MSComm1.OutBufferCount = 0
lcromo = 12
End Sub

Private Sub Form_Terminate()
datosGuardar.Close
cnn.Close
Set cnn = Nothing
Set datosGuardar = Nothing
End Sub


Private Sub MSComm1_OnComm()
Dim k As Integer
Dim minimo, maximo, suma As Single
Dim buffer As Variant
Select Case MSComm1.CommEvent
   Case comEvReceive
      If sw = False Then: MSComm1.InBufferCount = 0: Exit Sub
```

```
    If MSComm1.InBufferCount = 0 Then Exit Sub
    buffer = MSComm1.Input
    DatoIn(1) = AscB(buffer)
    buffer = MSComm1.Input
    DatoIn(2) = AscB(buffer)
    buffer = MSComm1.Input
    DatoIn(3) = AscB(buffer)
    buffer = MSComm1.Input
    DatoIn(4) = AscB(buffer)
    buffer = MSComm1.Input
    DatoIn(5) = AscB(buffer)
    buffer = MSComm1.Input
    DatoIn(6) = AscB(buffer)
    MSComm1.InBufferCount = 0
    MSComm1.OutBufferCount = 0      'Limpia el buffer de transmision
    velocidadR = ((2 * (DatoIn(1) + DatoIn(2) + DatoIn(3) + DatoIn(4) + DatoIn(5)
+ DatoIn(6)) / 500) * (2 * 3.14159265359)) / Tmuest
    Label9.Caption = CStr(Round(velocidadR))
    veloReal(n) = velocidadR
    'If n <= 30 Then
    '   velocidad = 10 * n
    'ElseIf n > 30 And n <= 330 Then
    '   velocidad = 300
    'Else
    '   velocidad = -10 * n + 3600
    'End If
    If n = 200 Then
       velocidad = 350
    ElseIf n = 400 Then
       velocidad = 150
    ElseIf n = 600 Then
       velocidad = 350
    ElseIf n = 800 Then
       velocidad = 150
    End If
    'velocidad = 150 * Sin(0.09 * n * 0.5) + 200
    velocidadD(n) = velocidad
    ErrorV = velocidadD(n) - velocidadR
    'CONTROL ADAPTIVO
    delta(n - 2) = velocidadD(n - 2)
    delta(n - 3) = velocidadD(n - 3)
    For j = n - 1 To NumPred + (n - 1)
       delta(j) = velocidadD(n)
       Yref(j + 1) = (a1r * delta(j) + a2r * delta(j - 1) - b1r * Yref(j) - b2r * Yref(j - 1))
```

```
        Next j
        ErrorModel = Yref(n - 1) - veloReal(n)
        suma = 0
        minimo = 0
        maximo = 0
        nmax = 0
        For i = 1 To numpobl
           TProporc = Kp(i) * ErrorV
           Tt = Ti(i)
           Tau = Td(i) / 10
           TDerivad = (Tau / (Tau + Tmuest)) * TDerivadAnt + Kp(i) * Td(i) * Tmuest *
(veloReal(n) - veloReal(n - 1))
           TIntegra = TIntegraAnt + (Kp(i) / Ti(i)) * Tmuest * ErrorV + (1 / Tt) *
(Ucontrol(n - 1) - u(n - 1))
           uci(i) = TProporc + TDerivad + TIntegra
           If uci(i) > VoltMax Then
              uci(i) = VoltMax
           ElseIf uci(i) < 0 Then
              uci(i) = 0
           End If
           Uaux(n - 2) = Ucontrol(n - 2)
           Uaux(n - 3) = Ucontrol(n - 3)
           For j = n - 1 To NumPred + (n - 1)
              Uaux(j) = uci(i)
              YiDis(j + 1) = (a1 * Uaux(j) + a2 * Uaux(j - 1) + a3 * Uaux(j - 2)) - b1 *
YiDis(j) - b2 * YiDis(j - 1) - b3 * YiDis(j - 2)
           Next j
           ErrorModelN = Yref(NumPred + (n - 1)) - YiDis(NumPred + (n - 1))
           ErrorModelP = (ErrorModelN - ErrorModel) / (NumPred * Tmuest)
           Ji(i) = ErrorModel + BETA * ErrorModelP
           Ji(i) = ALFA / (Ji(i) ^ 2 + ALFA)
           If minimo >= Ji(i) Then
              minimo = Ji(i)
           End If
           If maximo <= Ji(i) Then
              maximo = Ji(i)
              nmax = i
           End If
           suma = suma + Ji(i)
        Next i
        TProporc = Kp(nmax) * ErrorV
        Tau = Td(nmax) / 10
        Tt = Ti(nmax)
```

```
        TDerivad = (Tau / (Tau + Tmuest)) * TDerivadAnt + Kp(nmax) * Td(nmax) *
Tmuest * (veloReal(n) - veloReal(n - 1))
        TIntegra = TIntegraAnt + (Kp(nmax) / Ti(nmax)) * Tmuest * ErrorV + (1 / Tt) *
(Ucontrol(n - 1) - u(n - 1))
        Ucontrol(n) = TProporc + TDerivad + TIntegra
        TDerivadAnt = TDerivad
        TIntegraAnt = TIntegra
        u(n) = Ucontrol(n)
        If Ucontrol(n) > VoltMax Then
           VoltSal = VoltMax
        ElseIf Ucontrol(n) < 0 Then
           VoltSal = 0
        Else
           VoltSal = Ucontrol(n)
        End If
        datosGuardar.AddNew
        datosGuardar!velocidad = velocidadR
        datosGuardar!u = Round(VoltSal, 3)
        datosGuardar!Kp = Kp(nmax)
        datosGuardar!Ti = Ti(nmax)
        datosGuardar!Td = Td(nmax)
        datosGuardar!Yreferencia = Yref(n)
        datosGuardar.Update
        total = CInt(VoltSal * 1023 / VoltMax)
        alta = (total \ 4)
        baja = total Mod 4
        baja = baja * 16 + 12
        MSComm1.OutBufferCount = 0
        MSComm1.Output = Chr(alta)
        MSComm1.Output = Chr(baja)
        MSComm1.InBufferCount = 0
        sumFitness = suma
        maxFitness = maximo
        minFitness = minimo
        avgFitness = sumFitness / numpobl
        elitismo = True
        Call NextGeneration
        Label8.Caption = CStr(n)
        n = n + 1
        End Select
End Sub

Sub IniPobla()
Dim ii, jj As Integer
```

```
Dim aux, auxKp, auxTi, auxTd As String
Randomize
For ii = 1 To numpobl
   KpC(ii) = Int((10000 * Rnd))
   TiC(ii) = Int((10000 * Rnd))
   TdC(ii) = Int((10000 * Rnd))
Next ii
For ii = 1 To numpobl
   If KpC(ii) < 10 Then
      auxKp = "000" + CStr(KpC(ii))
   ElseIf KpC(ii) < 100 Then
      auxKp = "00" + CStr(KpC(ii))
   ElseIf KpC(ii) < 1000 Then
      auxKp = "0" + CStr(KpC(ii))
   Else
      auxKp = CStr(KpC(ii))
   End If
   If TiC(ii) < 10 Then
      auxTi = "000" + CStr(TiC(ii))
   ElseIf TiC(ii) < 100 Then
      auxTi = "00" + CStr(TiC(ii))
   ElseIf TiC(ii) < 1000 Then
      auxTi = "0" + CStr(TiC(ii))
   Else
      auxTi = CStr(TiC(ii))
   End If
   If TdC(ii) < 10 Then
      auxTd = "000" + CStr(TdC(ii))
   ElseIf TdC(ii) < 100 Then
      auxTd = "00" + CStr(TdC(ii))
   ElseIf TdC(ii) < 1000 Then
      auxTd = "0" + CStr(TdC(ii))
   Else
      auxTd = CStr(TdC(ii))
   End If
   aux = auxKp + auxTi + auxTd
   For jj = 1 To lcromo
      pobla(ii, jj) = CInt(Mid(aux, jj, 1))
      pobla_ant(ii, jj) = CInt(Mid(aux, jj, 1))
   Next jj
Next ii
For ii = 1 To numpobl
   Kp(ii) = KpC(ii) * 0.00001
   Ti(ii) = TiC(ii) * 0.00001
```

```
    If Ti(ii) = 0 Then
      Ti(ii) = 0.00001
    End If
    Td(ii) = TdC(ii) * 0.00001
Next ii
End Sub

Sub NextGeneration()
Dim jj, ii, s1, s2 As Integer
Dim part1, part2 As Integer
Dim sumParcial, aleatorio As Single
Dim jjj As Integer
Randomize
jj = 1
While (jj <= numpobl)
    desactivar = False
    """CODIGO PARA SELECCION POR RUEDA DE RULETA"""
    sumParcial = 0
    jjj = 0
    aleatorio = Rnd * sumFitness
    While ((sumParcial < aleatorio) And (jjj <> numpobl)) 'And (jjj <> nmax)
      jjj = jjj + 1
      sumParcial = sumParcial + Ji(jjj)
    Wend
    s1 = jjj
    'part1 = jjj
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    sumParcial = 0
    jjj = 0
    aleatorio = Rnd * sumFitness
    While ((sumParcial < aleatorio) And (jjj <> numpobl))
      jjj = jjj + 1
      sumParcial = sumParcial + Ji(jjj)
    Wend
    s2 = jjj
    """""""CODIGO PARA SELECCION POR TORNEO"""""""
    'part2 = jjj
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    'If Ji(part1) < Ji(part2) Then
    '   s1 = part2
    'Else
    '   s1 = part1
    'End If
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
    'sumParcial = 0
    'jjj = 0
    'aleatorio = Rnd * sumFitness
    'While (sumParcial < aleatorio) And (jjj <> numpobl)
    '  jjj = jjj + 1
    '  sumParcial = sumParcial + Ji(jjj)
    'Wend
    'part1 = jjj
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    'sumParcial = 0
    'jjj = 0
    'aleatorio = Rnd * sumFitness
    'While (sumParcial < aleatorio) And (jjj <> numpobl)
    '  jjj = jjj + 1
    '  sumParcial = sumParcial + Ji(jjj)
    'Wend
    'part2 = jjj
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    'If Ji(part1) < Ji(part2) Then
    '   s2 = part2
    'Else
    '   s2 = part1
    'End If
    If elitismo Then
      If (s1 = nmax) Or (s2 = nmax) Then
         desactivar = True
         elitismo = False
      End If
    End If

   Call cruzamiento(s1, s2)
   jj = jj + 2
Wend
For ii = 1 To numpobl
   KpC(i) = CInt(CStr(pobla(ii, 1)) + CStr(pobla(ii, 2)) + CStr(pobla(ii, 3)))
   TiC(i) = CInt(CStr(pobla(ii, 4)) + CStr(pobla(ii, 5)) + CStr(pobla(ii, 6)))
   TdC(i) = CInt(CStr(pobla(ii, 7)) + CStr(pobla(ii, 8)) + CStr(pobla(ii, 9)))
Next ii
For ii = 1 To numpobl
    Kp(ii) = KpC(ii) * 0.00001
    Ti(ii) = TiC(ii) * 0.00001
    If Ti(i) = 0 Then
       Ti(i) = 0.00001
```

```
    End If
    Td(ii) = TdC(ii) * 0.00001
Next ii
For ii = 1 To numpobl
    For jj = 1 To lcromo
        pobla_ant(ii, jj) = pobla(ii, jj)
    Next jj
Next ii
End Sub
Function seleccionR() As Integer
Dim sumParcial, aleatorio As Single
Dim jj As Integer
''''''''''''''''CRUZAMIENTO''''''''''''''''''
Randomize
sumParcial = 0
jj = 0
aleatorio = Rnd * sumFitness
While (sumParcial < aleatorio) And (jj <> numpobl)
  jj = jj + 1
  sumParcial = sumParcial + Ji(jj)
Wend
seleccionR = jj
''''''''''''''''''''''''''''''''''''''''''''''''
End Function

Function mutacion(gen As Integer) As Integer
Dim llave As Boolean
Randomize
If pmuta = 1 Then
  llave = True
ElseIf Rnd <= pmuta Then
  llave = True
Else
  llave = False
End If
If llave Then
  nmuta = nmuta + 1
  mutacion = Round(Rnd * 9)
Else
  mutacion = gen
End If
End Function

Sub cruzamiento(dato1, dato2 As Integer)
```

```
Dim jcruza, jj As Integer
Dim llave As Boolean
Randomize
If pcruza = 1 Then
  llave = True
ElseIf Rnd <= pcruza Then
  llave = True
Else
  llave = False
End If
If llave Then
  jcruza = Round((lcromo - 2) * Rnd) + 1
  ncruza = ncruza + 1
Else
  jcruza = lcromo
End If
If desactivar Then
   jcruza = lcromo
End If
For jj = 1 To jcruza
   pobla(dato1, jj) = mutacion(pobla_ant(dato1, jj))
   pobla(dato2, jj) = mutacion(pobla_ant(dato2, jj))
Next jj
If jcruza <> lcromo Then
   For jj = (jcruza + 1) To lcromo
      pobla(dato1, jj) = mutacion(pobla_ant(dato2, jj))
      pobla(dato2, jj) = mutacion(pobla_ant(dato1, jj))
   Next jj
End If
End Sub
```