Departamento de Física Fundamental y Experimental
Centro Superior de Informática
Universidad de La Laguna

DESARROLLO DE ESTRATEGIAS DE CONTROL PREDICTIVAS DE ALTO RENDIMIENTO MEDIANTE LA INCORPORACIÓN DE TÉCNICAS DE CONTROL ROBUSTO Y DE REDES NEURONALES

JUAN ALBINO MÉNDEZ PÉREZ

La Laguna, febrero de 1998

Departamento de Física Fundamental y Experimental
Centro Superior de Informática
Universidad de La Laguna

DESARROLLO DE ESTRATEGIAS DE CONTROL PREDICTIVAS DE ALTO RENDIMIENTO MEDIANTE LA INCORPORACIÓN DE TÉCNICAS DE CONTROL ROBUSTO Y DE REDES NEURONALES

Autor: Juan Albino Méndez Pérez
Director: Leopoldo Acosta Sánchez

La Laguna, febrero de 1998

D. LEOPOLDO ACOSTA SÁNCHEZ, Doctor en Ciencias Físicas y Profesor Titular de Física Aplicada perteneciente al Departamento de Física Fundamental y Experimental de la Universidad de La Laguna,

CERTIFICA:

Que D. Juan Albino Méndez Pérez, Licenciado en Ciencias Físicas, ha realizado bajo mi dirección la presente Tesis Doctoral: DESARROLLO DE ESTRATEGIAS DE CONTROL PREDICTIVAS DE ALTO RENDIMIENTO MEDIANTE LA INCORPORACIÓN DE TÉCNICAS DE CONTROL ROBUSTO Y DE REDES NEURONALES, para optar al grado de Doctor en Informática.

Con esta fecha autorizo la presentación de la misma.

La Laguna, febrero de 1998

El Director

Leopoldo Acosta Sánchez

A la chica que quiero,

Alicia.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento, en primer lugar, al Dr. D. Lorenzo Moreno Ruiz por haber apostado por mi para afrontar junto a él esta carrera como investigador y docente. Además le quiero agradecer sus aportaciones y acertados puntos de vista sobre este trabajo que han ayudado a incrementar la solidez de la investigación desarrollada.

Al Dr. D. Leopoldo Acosta Sánchez a quien le debo gran parte del conocimiento y formación que he alcanzado durante estos años de trabajo y junto al cual las dificultades que iban surgiendo parecían siempre menores.

A todos mis compañeros del grupo de computadoras y control: a D. José Ignacio Estévez Damas, a D. José Sigut Saavedra, al Dr. D. Demetrio Piñeiro Vera, a Dña. Rosa María Aguilar Chinea, al Dr. D. Alberto Hamilton Castro, a D. Graciliano Nicolás Marichal Plasencia, a D. Juan Julian Merino Rubio, al Dr. D. Gumersindo Vera Casanova, a Dña. Marieta Ivanova Dimitrova, a Dña Marta Sigut Saavedra, a D. Roberto Marichal Plasencia, a D. Carlos Martín Galán y a D. Roberto Bentancor Bonilla por su ayuda desinteresada en los aspectos técnicos del trabajo.

A D. Santiago Torres Álvarez cuya ayuda resultó imprescindible en el tramo final del trabajo.

A D. Sergio Hernández, y en general a todos los miembros del Servicio Electrónico de la Universidad, por su inestimable ayuda en la construcción del prototipo del puente de grúa.

Quiero expresar también mi agradecimiento a mi hermana Marisol por dos razones. La primera es por su asesoramiento constante en todas los aspectos administrativos que rodearon esta tesis. Y la segunda, y más importante, por ser tan buena hermana.

En general estoy muy agradecido a mis padres, Carmen y Francisco, por haber puesto todo de su parte para que nunca me faltara nada en mi carrera como estudiante y luego como Profesor. Asimismo quiero recordar a todos mis hermanos/as y sobrinos/as por ser tan buena gente conmigo.

Quiero agradecer especialmente a Alicia que siempre estuviese a mi lado en este proyecto, infundiéndome el sosiego y la dulzura necesaria para sacarlo adelante.

INTRODUCCIÓN

En la presente memoria se resume el trabajo consistente en la formulación e implementación de algoritmos de control que añadan nuevas características en cuanto a robustez, estabilidad y seguimiento de consignas variables en relación a las propuestas existentes. Todo el trabajo realizado presenta un punto de referencia común: el controlador predictivo generalizado. La razón para escoger este controlador como punto de partida, está en el enorme éxito que ha tenido este algoritmo en la industria. Prueba de ello son los innumerables trabajos de investigación que han ido apareciendo a lo largo de esta década. El presente trabajo pretende ser una aportación más al desarrollo al que se ha visto sometido este tipo de algoritmos.

A pesar de su eficacia para el control de procesos industriales, el algoritmo predictivo generalizado presenta una serie de puntos que desde siempre han parecido como grandes desventajas. El más importante de ellos, sin duda, es el aspecto concerniente a la estabilidad. La formulación original del controlador predictivo generalizado (GPC) carecía de una teoría sólida que permitiese garantizar condiciones de estabilidad para el sistema en lazo cerrado. Este aspecto ha sido el principal motivo de investigación en la presente década. Otro aspecto que se presenta en ocasiones como un inconveniente para el algoritmo es la complejidad computacional del mismo. Para procesos lentos, este hecho no aparece como un problema importante. Pero cuando se trata con sistemas de dinámicas rápidas, y si se incluyen en la síntesis del controlador ligaduras, la complejidad computacional del algoritmo puede poner en peligro la aplicabilidad del controlador. Por último, otro punto de interés abordado en este trabajo es el relativo a los horizontes de predicción del controlador. La mayoría de resultados

que hacen referencia a la elección de los horizontes de predicción en la síntesis de un controlador predictivo, son meramente criterios cualitativos, o condiciones límite como el caso en que el horizonte es infinito. Sin embargo, la importancia de una elección apropiada de estos horizontes es fundamental para el buen funcionamiento del controlador.

La aportación de este trabajo va encaminada hacia el tratamiento de estas grandes desventajas que presenta el algoritmo predictivo generalizado. Por ello, los objetivos marcados en este trabajo se resumen en los siguientes tres grandes puntos:

A.- **Aplicabilidad del GPC**. El primer objetivo planteado en este trabajo es el de estudiar el comportamiento en tiempo real del GPC. Para este estudio se ha considerado como planta a controlar una bancada con un motor de corriente continua. Este planta puede ser considerada como un sistema multivariable tomando como salidas la posición y la velocidad del motor. Teniendo en cuenta que un algoritmo GPC multivariable conllevaría un elevado número de operaciones, se ha propuesto un algoritmo alternativo que cumpla las especificaciones impuestas y que, además, presente una complejidad computacional inferior. Con esto se pretende presentar al GPC como una estrategia aplicable incluso para plantas con constantes de tiempo pequeñas. El algoritmo diseñado es "pseudo-multivariable", en el sentido de que se trata de un control multivariable, pero conseguido a partir de la combinación en el tiempo de dos acciones de control sobre variables independientes. Otro aspecto que se ha abordado en esta fase es el de explotar la facilidad que presenta el GPC para incorporar en el diseño del control las componentes de ruido de la planta. Así, se ha realizado una comparación entre una formulación puramente determinista con un diseño estocástico en el que se tienen en consideración las perturbaciones de la planta. Los resultados obtenidos ponen de manifiesto efectos que prevé la teoría en controles estocásticos como es el de *caution.*

B.- **Reducción de la complejidad computacional**. Los controladores predictivos presentan una dependencia directa entre el número de operaciones por intervalo de muestreo y el horizonte de predicción elegido. En general, cuanto mayores son los horizontes de predicción, mayores son las dimensiones de las matrices que intervienen en el algoritmo y, por lo tanto, mayor es la complejidad computacional del mismo. Asimismo los horizontes de predicción mayores se requieren cuanto más compleja es la dinámica de la planta. Basándonos en este hecho, se ha presentado una estructura de control que pretende reducir los horizontes de control de la planta, y con ello la carga computacional. Para ello, se ha diseñado un esquema basado en técnicas de control

robusto para garantizar un buen comportamiento del sistema. Este esquema consiste en la introducción de un bucle interno de estabilización robusta, y sobre la planta resultante aplicar el controlador GPC. Los resultados obtenidos demuestran la efectividad de esta estrategia, garantizando por un lado un comportamiento estable para el sistema, y además, con horizontes de predicción bajos.

C.- **Autosintonización inteligente de los horizontes de predicción**. Otra gran parte de esta memoria está dedicada a la descripción de algoritmos basados en la aplicación de técnicas inteligentes a controladores predictivos. El objetivo final en esta línea era el de buscar un mecanismo que permitiera sintonizar de forma automática los parámetros de los controladores GPC. Debido a las posibilidades que presentan las redes neuronales para el control, se ha diseñado un esquema adaptativo basado en redes neuronales artificiales que permite la sintonización de un controlador a partir de información sobre el estado de la planta bajo control. Para comprobar la eficacia de este controlador neuronal, se ha aplicado a una planta no lineal. La planta es un puente de grúa, muy habitual en la industria. El objetivo planteado es el de sintonizar *on-line* los parámetros de un controlador que actúa sobre este sistema. En una primera fase se realizó un estudio en simulación para estudiar la eficacia del controlador y posteriormente se implementó sobre un prototipo a escala de un puente de grúa construido con las mismas características que el modelo estudiado en simulación. Los resultados obtenidos ponen de manifiesto las buenas prestaciones que ofrece este controlador para la sintonía de parámetros. La aplicación de este esquema a los controladores predictivos ha ido encaminada a la sintonización de los horizontes de control en el GPC. Como se mencionó anteriormente, estos parámetros juegan un papel fundamental para obtener buenas prestaciones en el sistema. El controlador propuesto (NAGPC) se basa en un controlador GPC estándar, y una o varias redes neuronales que actúan paralelamente al controlador. Estas redes neuronales reciben información del estado del sistema y a partir de ella van aprendiendo cuáles son los horizontes de predicción más adecuados en cada instante. Las características más importantes de este controlador neuronal son que, por un lado no necesitan información sobre el modelo de la planta, además el entrenamiento se realiza *on-line* lo cual supone un avance importante ya que la mayoría de los controladores neuronales presentan un entrenamiento *off-line*.

Teniendo en cuenta estos puntos, el trabajo presentado en esta memoria se desarrolla siguiendo tres grandes líneas correspondientes a cada uno de los puntos anteriores. La primera de ellas (punto A), se desarrolla en los capítulos 1 y 2. La segunda cuestión que se aborda en este trabajo (punto B), se describe en los capítulos 3 y 4 y, por último, en los

capítulos 5, 6 y 7 se estudia el último de los puntos (punto C). De esta forma, la estructura por capítulos es la siguiente:

Los dos primeros capítulos están destinados al estudio del GPC y de sus propiedades y a los problemas relacionados con la implementación en tiempo real del algoritmo. El capítulo 1 es un capítulo principalmente teórico donde se da un repaso de los fundamentos del control predictivo basado en modelos. Como el resto del trabajo está desarrollado en torno al controlador GPC, existe una sección donde se describe en profundidad el GPC y sus características. Este primer capítulo termina presentando una serie de simulaciones donde se pone de manifiesto la efectividad de GPC. Para ello se realizan comparaciones de la repuesta de este controlador frente a la de un PID.

El segundo capítulo describe una implementación del controlador GPC sobre una planta real. La planta en la que se ha llevado a cabo la implementación es una bancada formada por un motor de corriente continua. Se ponen de manifiesto en este capítulo los problemas relacionados con la implementación del controlador en tiempo real, especialmente los que tienen que ver con la complejidad computacional del algoritmo. El algoritmo implementado está basado en el esquema SISO del GPC, pero modificado de tal forma que se permite un control multivariable de la planta. El controlador realizado se ha implementado de dos formas: con una estrategia puramente determinista y con una estrategia estocástica donde se tuvo en cuenta el ruido presente en la planta.

La segunda parte de la memoria la constituye el capítulos 3. En él se aborda el problema de la reducción de la complejidad computacional en el GPC. En las primeras secciones del capítulo se resumen los fundamentos del control robusto, haciendo una revisión general del problema de control robusto y estudiando en detalle el problema de la sensibilidad mixta, ya que esta estrategia es la que se emplea posteriormente conjuntamente con el GPC para mejorar el rendimiento de éste.

En la segunda parte de este capítulo se presenta el diseño de un controlador GPC de bajo coste computacional. El algoritmo está basado en el empleo de técnicas de control robusto para mejorar el rendimiento del controlador. El objetivo es aumentar las características de estabilidad del controlador para así poder disminuir los horizontes de predicción, y por consiguiente la carga computacional del algoritmo. Los resultados obtenidos en simulación son contrastados con los de un GPC clásico, pudiendo así evaluar el rendimiento de esta estrategia.

En la última parte del trabajo se aborda el problema de la sintonización de los horizontes de predicción en el GPC. La herramienta empleada para abordar este problema son las redes neuronales artificiales. En el capítulo 4 se hace un repaso del control neuronal en general. Se describen además las configuraciones de control basadas en redes neuronales más comunes.

En el capítulo 5 se presenta el diseño del controlador propuesto para abordar el problema de la sintonía de los horizontes de predicción en el GPC. Este controlador combina un bloque de control clásico cooperando con un subsistema de redes neuronales. La idea es conseguir que la redes neuronales, tomando información del estado del sistema en cada instante, sintonicen los parámetros del controlador. El algoritmo se implementa sobre algunas plantas lineales empleando controladores PI. Posteriormente, y para probar la eficacia de esta estrategia sobre una planta compleja, se considera la implementación de este esquema de control sobre una sistema no lineal. La planta es una grúa transportadora de contenedores. Este sistema, presente en muchas plantas industriales, presenta un problema de control complejo durante el transporte de los contenedores. Empleando un esquema de realimentación por variables de estado junto con dos redes neuronales actuando en paralelo, se diseña un algoritmo capaz de solucionar este problema de control que presenta la grúa. El capítulo finaliza mostrando los resultados obtenidos de la implementación de este controlador sobre un prototipo a escala de un puente de grúa.

En el último capítulo se hace uso del algoritmo de control diseñado en el capítulo 5 para resolver el problema de la sintonización de los horizontes de predicción en el GPC. En las primeras secciones de este capítulo se describe el problema de la sintonía de los horizontes de predicción, poniendo de manifiesto la importancia de la elección de estos parámetros. Especialmente se estudia el comportamiento del controlador cuando la consigna que se le impone es variable. Concretamente se realiza un estudio de la influencia del horizonte de predicción en sistemas de segundo orden. Una vez descrito el problema, se presenta un controlador basado en redes neuronales que sintoniza automáticamente el horizonte de salida del GPC. Este algoritmo es implementado sobre diferentes plantas con distintas características con el objeto de demostrar la eficacia del mismo.

ÍNDICE

1

CONTROL PREDICTIVO BASADO EN MODELOS

En este primer capítulo se describen los fundamentos del control predictivo basado en modelos. Se empieza dando una perspectiva histórica de este tipo de controladores, mostrando los avances y las distintas líneas de investigación que han ido surgiendo alrededor de esta metodología desde sus inicios hasta nuestros días. Posteriormente, se hace una descripción general de la estrategia MBPC, comentando cual es la metodología y las características principales de estos algoritmos. Como caso particular, se estudia en profundidad el controlador predictivo generalizado, analizando sus características, sus puntos débiles y las distintas propuestas que se han formulado para mejorar el algoritmo inicial. En la última sección se muestran algunos resultados obtenidos de simulaciones sobre diferentes plantas.

1.1 INTRODUCCIÓN AL MBPC

El Control Predictivo Basado en Modelos (MBPC, *Model Based Predictive Control*) es una de las estrategias de control de más éxito desarrolladas en los últimos años para

abordar problemas de control avanzado de procesos industriales. En realidad, el MBPC no es una estrategia de control determinada, sino que agrupa a toda una serie de algoritmos desarrollados bajo una misma idea central. El funcionamiento del MBPC se basa en lo siguiente: resolver en cada instante de muestreo un problema de control óptimo de horizonte finito o infinito en lazo abierto y aplicar el comando resultante de esta optimización hasta el siguiente intervalo de muestreo. Aunque la optimización es en lazo abierto, la estrategia no lo es, ya que el hecho de tomar nuevas medidas antes de cada optimización produce un control realimentado.

La optimización se lleva a cabo minimizando una función de coste en la que interviene el error futuro que se ha predicho para el sistema. Para obtener esta predicción es necesario el empleo de un modelo que permita predecir el comportamiento del sistema. La idea que subyace en las estrategias MBPC es la del horizonte móvil (*receding horizon* ó *moving horizon*), que consiste en que en cada etapa t el horizonte es desplazado hacia el futuro repitiendo la minimización en la etapa $t+1$. Este concepto fue introducido por primera vez por Propoi en 1963, aunque enmarcado en otro tipo de estrategias [Pro63].

Bajo el concepto de horizonte móvil han surgido diferentes metodologías de MBPC desde la aparición del primer algoritmo a finales de la década de los 70. Sin embargo, la mayoría de ellas difieren entre sí sólo por el tipo de modelo que emplean para representar el proceso pero comparten la misma filosofía. En este sentido se distinguen estrategias que usan modelos de repuesta impulso, respuesta escalón, función de transferencia o modelo de espacio de estados. El éxito de esta estrategia ha sido debido entre otras cosas a su sencillez y a los buenos resultados que mostró desde un principio en aplicaciones industriales. De hecho los dos primeros trabajos que dieron origen al MBPC fueron desarrollados en sendas compañías independientes. Estos autores fueron los encargados de convencer a la comunidad de empresarios y a los investigadores que este tipo de controladores era viable y ofrecía un potencial importante para aplicaciones industriales.

A partir de este momento se sucedieron las investigaciones y los trabajos publicados proponiendo nuevos algoritmos. La principal orientación de todas estas publicaciones era la de consolidar una base teórica que en su comienzo parecía bastante deficiente. Así, hubo bastantes detractores de estas técnicas ya que inicialmente no existían resultados sobre la estabilidad del MBPC. Sin embargo, a medida que fueron apareciendo algoritmos más robustos y con garantías de estabilidad, se dejó de cuestionar la aplicabilidad de estas técnicas.

El MBPC ha tenido aplicaciones en campos muy diversos. Quizás en el que más éxito tuvo inicialmente es en el campo de la industria petroquímica [Ric78], [Cut80], [Ric93]. Aunque luego ha sido aplicado al control de plantas eléctricas [Ros91], aplicaciones clínicas [Mah92], [Lin94], control de motores DC [Mor94], control de plantas de energía solar [Cam94], control de máquinas herramienta [Dum94], servosistemas [Ric93], sistemas de gran escala [Kat97], etc.

Las características principales de los controladores MBPC se pueden resumir en los siguientes puntos:

- La idea básica de los MBPC es simple, no requiere de unas matemáticas complejas y resulta bastante intuitiva.

- Permite tratar de forma sistemática y natural las ligaduras a las que está sometido el proceso.

- La extensión de la estrategia de la formulación SISO a una formulación multivariable es abordable de forma relativamente simple.

- Es una estrategia que puede ser utilizada para controlar diversidad de procesos, desde aquellos que presentan una dinámica relativamente simple hasta procesos de complejidad elevada. Puede tratar procesos con retardo, de fase no-mínima o inestables en lazo abierto.

- El controlador resultante del diseño es un controlador lineal que puede ser implementado de forma sencilla.

- En campos como la robótica pueden ser muy útiles ya que se dispone de antemano de la trayectoria de referencia deseada para la salida.

1.2 REVISIÓN HISTÓRICA DEL CONTROL PREDICTIVO BASADO EN MODELOS

Es a finales de la década de los 70 cuando se puede situar el comienzo del MBPC. Esto fue debido fundamentalmente a la aparición de los trabajos desarrollados por Richalet *et al* [Ric76],[Ric78] donde se presentó el IDCOM (Identificación/Comando) y los trabajos de Cutler y Ramaker [Cut80] en los que se propuso el Control de Matriz Dinámico (DMC,

Dynamic Matrix Control). Otro planteamiento basado en el IDCOM es el Algoritmo de Control Predictivo (PCA, *Predictive Control Algorithm*) [Bru80]. En estos algoritmos se consideraba explícitamente un modelo del proceso dinámico para predecir la respuesta del sistema debido a las futuras acciones de control. Tras su formulación, este tipo de estrategias tuvieron gran auge dentro de la industria. Sin duda, el sector que más ayudó a este popularidad fue el de los procesos químicos, especialmente la industria petroquímica. Algunas de las ventajas que ofrecían estas estrategias era la sencillez para abordar problemas de control multivariable y la inclusión de las ligaduras del propio proceso a controlar. De hecho la mayoría de las aplicaciones desarrolladas fueron sobre plantas multivariables. Sin embargo, muchos investigadores, sobre todo los que trabajaban en control adaptativo, presentaban reservas hacia estos algoritmos debido principalmente a problemas de inestabilidad [Bit90].

Ya en la década de los 80 surgieron otras formulaciones de control predictivo basadas en conceptos de control adaptativo. Especial importancia tuvo el controlador MUSMAR (*Multistep Multivariable Adaptive Regulator*) propuesto por Menga y Mosca [Men80]. También es importante el Control Self-tuning Basado en Predictores (PBSTC, *Predictor-Based Self Tuning Control*) propuesto por Peterka en el año 84 [Pet84]. Por esta misma fecha se presentó el Control Adaptivo de Horizonte Extendido (EHAC, *Extended Horizon Adaptive Control*) por Ydstie [Yds84] y también el Control Auto-adaptivo de predicción extendida (EPSAC, *Extended Prediction Self Adaptive Control*) por De Keyser y Van Cuawenberghe [Dek85]. Una diferencia fundamental entre ambos métodos era que el EHAC calculaba la salida predicha a partir de la resolución de una ecuación diofántica, mientras que el EPSAC utilizaba un predictor subóptimo.

Fue en 1987 cuando se propuso uno de los controladores predictivos que más éxito ha tenido: el Controlador Predictivo Generalizado (GPC, *Generalized Predictive Control*) [Cla87a], [Cla87b]. Este controlador propuesto por Clarke *et al* estaba inspirado en el Controlador de Mínima Varianza Generalizado (GMV, *Generalized Minimum Variance*) [Cla79]. Este método dotó de mayor robustez a los MBPC pudiendo abordar procesos con retardo variable, de fase no-mínima, inestables en lazo abierto o con modelos sobreparametrizados.

Existen otros métodos que han surgido siguiendo ideas similares entre los que cabe destacar el Control Multipredictor Adaptativo de Horizonte Móvil (MURHAC, *Multipredictor Receding Horizon Adaptive Control*) [Lem85] o el Control Funcional Predictivo (PFC, *Predictive Functional Control*) [Ric87] que fue formulado en el marco del espacio de estados. El éxito de esta estrategia fue su fácil generalización para abordar

sistemas multivariables, aunque presenta una complejidad computacional importante. Otro método de gran popularidad es el Control Predictivo Unificado propuesto por Söeterboek (UPC, *Unified Predictive Control*) [Soe92]. Este esquema utiliza un modelo de proceso similar al GPC, pero emplea una función de costo diferente inspirada en la del controlador GMV. Se puede comprobar que el UPC se puede transformar, eligiendo adecuadamente los parámetros de diseño, en controladores bien conocidos como el GMV, dead-beat, control por asignación de polos, etc.

A pesar del éxito que tenían estos controladores, la principal desventaja que ofrecían estaba relacionada con cuestiones de estabilidad. Las estrategias de control predictivo de horizonte finito presentaban gran dificultad de análisis, de ahí que apenas existieran resultados sobre su estabilidad en lazo cerrado. En la década de los noventa la mayoría de los nuevos trabajos presentados iban encaminados a resolver el problema de la estabilidad de los controladores predictivos sujetos a ligaduras. La mayoría de las soluciones propuestas consideraban la misma idea: la imposición de restricciones en el estado del sistema en el punto final de la trayectoria garantiza estabilidad en el proceso. De los primeros trabajos en este sentido cabe destacar en 1990, la propuesta del Control de Horizonte Móvil de Entrada/salida Estabilizado (SIORHC, *Stabilizing Input-Output Receding-Horizon Control*) [Mos90]. El SIORHC fue obtenido resolviendo un problema LQ dinámico sujeto a las restricciones adecuadas sobre el error de regulación o de seguimiento final y sobre las señales de comando. De esta forma el SIORHC permite garantizar estabilidad para un horizonte de predicción superior al orden de la planta. Clarke y Scattolini en 1991 presentaron el Controlador Predictivo de Horizonte Móvil Restringido (CRHPC, *Constrained Receding-horizon Predictive Control*) [Cla91]. En este último se impone la condición de que la salida alcance la consigna dentro de un rango de tiempo en el futuro. Esto es suficiente para garantizar la monotonicidad de la ecuación de Riccatti asociada, lo que supone alcanzar una respuesta estable.

Un algoritmo más desarrollado fue propuesto en 1993: el Control Predictivo Generalizado con Peso en el Estado Final (GPCW, *Generalised Predictive Control with End-point State Weighting*) [Dem93a]. Este método se basaba en la inclusión de un peso sobre el vector de estado final, de forma que si este peso es cero se recupera el algoritmo GPC y si el peso es infinito se transforma en un CRHPC. Este algoritmo probó tener mejores prestaciones que el CRHPC debido al relajamiento en las condiciones impuestas al final de la trayectoria.

Otro trabajo dirigido a asegurar la estabilidad en el GPC es el de Kouvaritakis *et al* en 1992 [Kou92]. En esta propuesta se alcanzaba la estabilidad del sistema mediante la inclusión de un controlador de estabilización interna en un bucle de realimentación dando lugar al SGPC (*Stable Generalized Predictive Control*). En 1993 estos mismos autores refinaron este método proponiendo el Control Predictivo Generalizado Estable Restringido (CSGPC, *Constrained Stable Generalized Predictive Control*) [Ros93].

Otras líneas de investigación han ido encaminadas a la formulación de la estrategia GPC en el dominio continuo y hacia la generalización del GPC para sistemas multivariables. Cabe destacar en este sentido el Controlador Predictivo Generalizado Continuo (CGPC, *Continuous* GPC) [Dem91] donde se formulaba el GPC para tiempo continuo. Un año más tarde aparecía la extensión al caso multivariable, el MCGPC (*Multivariable* CGPC) [Dem92]. También se propuso la formulación del Control Multivariable Predictivo Unificado (MUPC, *Multivariable Unified Predictive Control*) [Vri94].

Los trabajos sobre estabilidad llevados a cabo por Kouvaritakis y Rossiter dieron lugar a la aparición en 1993 del Control Predictivo Generalizado Estable Multivariable (MSGPC, *Multivariable Stable Generalized Predictive Control*) [Kou93]. Estos mismos autores han dirigido en los últimos años su investigación hacia garantizar la viabilidad del CSGPC como proponen en [Ros95], [Ros96]. En estos trabajos presentan un algoritmo que evita la pérdida de viabilidad de la estrategia debido a posibles violaciones en las ligaduras de los comandos durante la trayectoria como consecuencia de las ligaduras impuestas en los estados finales de la trayectoria. Este algoritmo, conocido como Control Predictivo Generalizado Estable Restringido Modificado (MCSGPC, *Modified* CSGPC), garantiza estabilidad y seguimiento asintótico incluso en presencia de perturbaciones [Gos97].

Como recientes avances en el control predictivo adaptivo hay que resaltar la propuesta de Nicolao *et al* [Nic96]. Estos autores presentaron dos planteamientos para el control predictivo adaptivo de sistemas SISO lineales e invariantes en el tiempo con saturación tanto en la variable de control como en sus incrementos.

En la figura 1.1 se representa gráficamente la evolución cronológica de los controladores MBPC desde la década de los setenta hasta nuestros días.

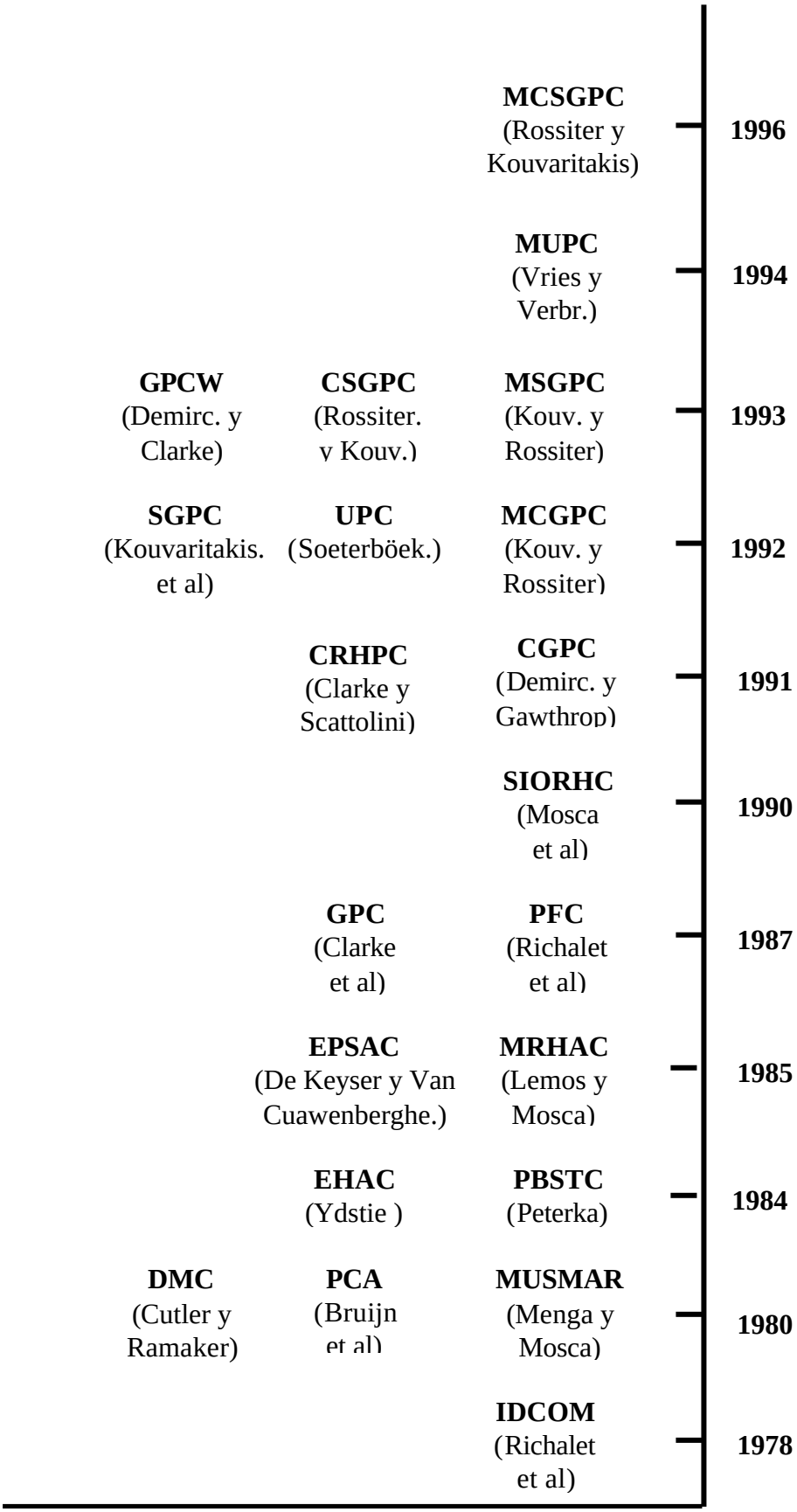


Figura 1.1. Evolución histórica de los controladores predictivos basados en modelos.

1.3 DESCRIPCIÓN DEL MBPC

El esquema general de los algoritmos MBPC es el que se muestra en la figura 1.2 [Cla94], [Cam95]. Como se observa en esta figura, se utiliza un modelo para predecir las salidas de la planta. Para ello se utiliza información de salidas y comandos hasta el instante actual y de los comandos a aplicar en el futuro obtenidos del proceso de optimización. La complejidad computacional del algoritmo depende del bloque optimizador. Normalmente la función de costo, o criterio de optimización suele ser cuadrática. Esto permite obtener su mínimo como una función lineal de las entradas y salidas anteriores y de la trayectoria de referencia marcada en el futuro. Si existen ligaduras en el problema, entonces la solución es más compleja. De hecho, se encuentra que la cantidad de tiempo necesaria para resolver problemas en los que existen ligaduras o se imponen condiciones de robustez puede ser varios órdenes de magnitud superior al tiempo requerido para un problema sin ligaduras. Esto provoca un notable descenso en el ancho de banda del sistema.

Un factor que interviene directamente en la carga computacional del problema es el de los horizontes de predicción considerados. Así, se encuentra que un aumento en estos horizontes (que generalmente conduce a una mejor respuesta del controlador) se paga con un notable incremento en la dimensionalidad del problema.

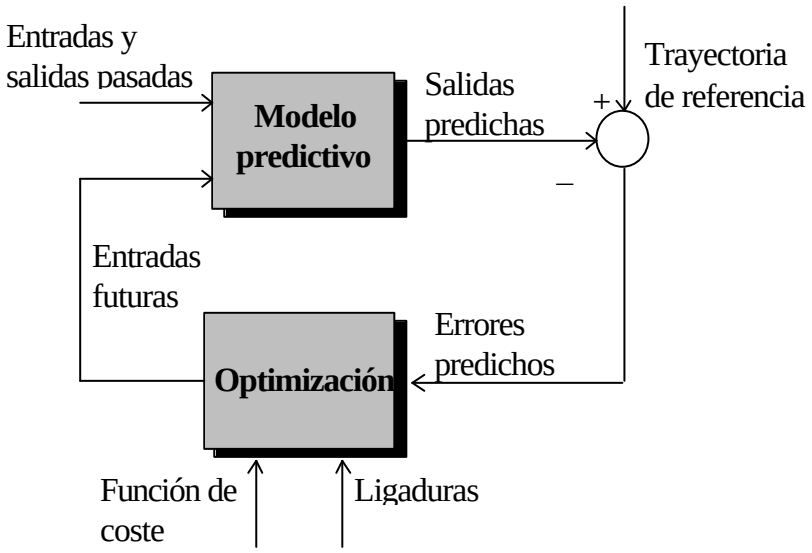


Figura 1.2. Estructura de un MBPC.

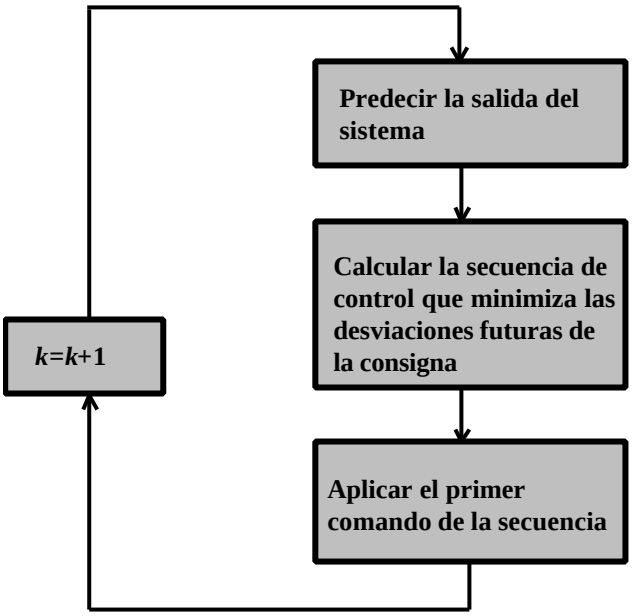


Figura 1.3. Funcionamiento básico de un MBPC.

En el algoritmo empleado por todos los controladores MBPC se pueden distinguir tres pasos (ver figura 1.3) [Cla87a], [Cam95]:

1.- Fijado el horizonte N, llamado horizonte de salida u horizonte de predicción, se predice el valor de la salida en los instantes $t+1, \ldots, t+N$. Esta salida predicha, $\hat{y}(t+k|t)$, se obtiene haciendo uso del modelo del proceso, de las entradas y salidas obtenidas hasta el instante t y de la secuencia de control $u(t+k|t)$, $k=0, \ldots, N-1$ que resulta del proceso de optimización (ver figura 1.4).

2.- Cálculo de la secuencia de control a aplicar en el futuro. Se debe fijar una función de costo que penalice las desviaciones futuras de la trayectoria de referencia. Generalmente, esta función suele ser cuadrática en el error entre la salida predicha y la referencia futura. En la mayoría de los casos se suele incluir un término cuadrático en el comando, de forma que se puedan penalizar acciones demasiado energéticas. Con este criterio, considerando una planta lineal y sin tener en cuenta ligaduras, la ley de control se puede obtener de forma explícita. De otra forma, es necesario la aplicación de un método numérico para la optimización.

3.- Se aplica la idea de horizonte móvil: en el instante t, se aplica únicamente el primer comando de la secuencia de control obtenida del optimizador, $u(t|t)$, despreciando el resto de los comandos $u(t+k|t)$. El proceso se vuelve a repetir en el instante $t+1$, de forma que en este instante se aplica el comando $u(t+1|t+1)$ (que puede ser diferente de $u(t+1|t)$).

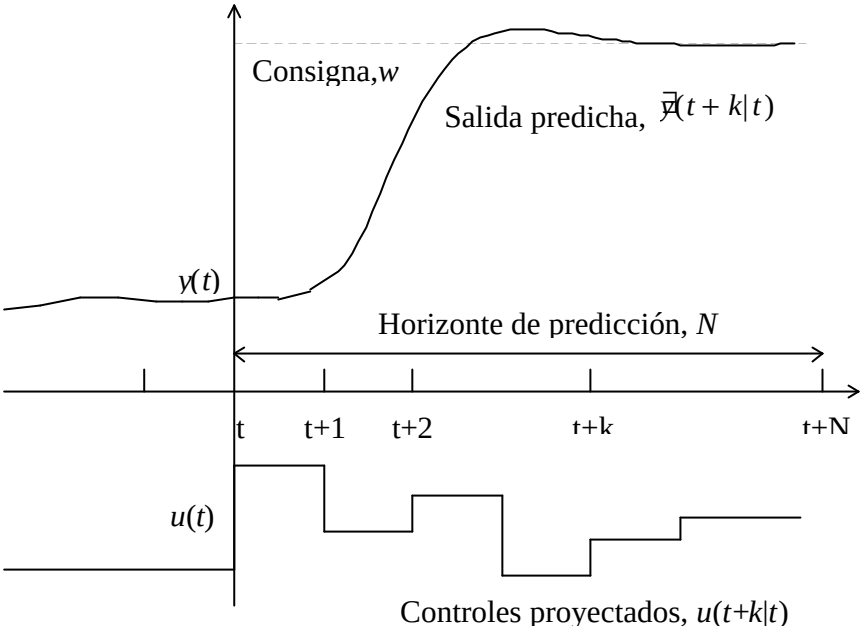


Figura 1.4. Consigna, salida predicha y secuencia de control en un MBPC.

1.3.1 MODELO PREDICTIVO

En el planteamiento MBPC la elección del modelo del proceso juega un papel muy importante en el rendimiento del algoritmo. Este es el principal factor de diferenciación entre las distintas formulaciones de MBPC conocidas. Para que el modelo elegido sea considerado como un buen modelo, éste debe ser capaz de recoger toda la información posible sobre la dinámica del sistema de tal forma que las predicciones generadas sean lo más veraces posibles. Al mismo tiempo, se debe intentar que el modelo sea simple de entender e implementar.

Existen propuestas que utilizan modelos predictivos no-lineales de la forma

$$\hat{y}(t+j) = f(t, q, D(t)) \tag{1.1}$$

que dan una predicción de la salida en el instante $t+j$ a partir de la función no lineal $f(.)$. Siendo q el vector de parámetros del modelo y $D(t)$ la colección de datos a partir de los

cuales se calcula la predicción. Sin embargo, la mayoría de los MBPC consideran un modelo lineal para la planta. Esto permite modelar por separado la dinámica del proceso y las perturbaciones del mismo. Aplicando el principio de superposición, se encuentra que la salida del modelo global del sistema se puede obtener como la suma de la salida de los dos submodelos por separado (ver figura 1.5).

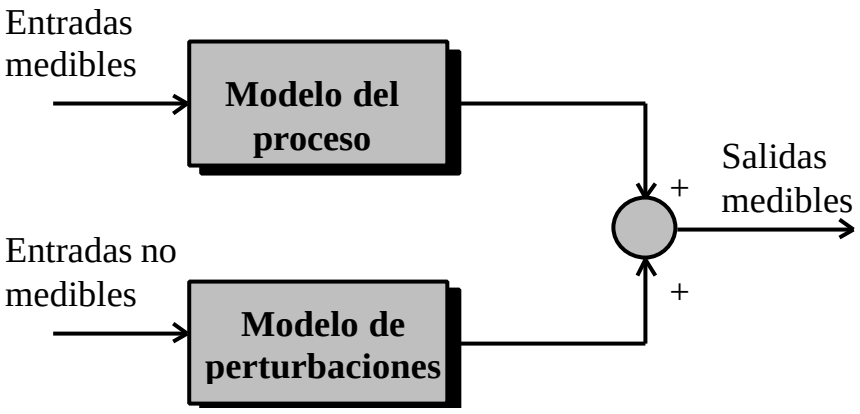


Figura 1.5. Modelo predictivo en el MBPC.

1.3.1.1 ECUACIONES DE PREDICCIÓN

Las ecuaciones de predicción en un algoritmo MBPC dependen del modelo empleado para la planta. Los modelos de proceso más comúnmente utilizados son:

- Modelos de respuesta a impulso.
- Modelos de respuesta a escalón.
- Modelos de función de transferencia.
- Modelos de espacio de estados.

Modelos de respuesta a impulso

Este tipo de modelos son empleados en las estrategias IDCOM y PCA y en casos especiales del GPC y EPSAC. En estos modelos se toma la siguiente aproximación para la salida del sistema:

$$y(t) = \sum_{k=1}^{\infty} h_k u(t-k), \tag{1.2}$$

donde h_k son los valores en los instantes de muestreo de la respuesta a un impulso unitario de anchura igual que el periodo de muestreo considerado. De esta suma sólo se consideran los N primeros valores y los demás se desprecian (si se supone estabilidad, $h_k \rightarrow 0$, $t \rightarrow \infty$). Se tiene, entonces

$$y(t) = \sum_{k=1}^{N} h_k u(t-k) = H(q^{-1})u(t), \tag{1.3}$$

donde $H(q^{-1}) = h_1 q^{-1} + h_2 q^{-2} + \ldots + h_N q^{-N}$, siendo q^{-1} el operador retardo unitario. La ecuación de predicción está dada entonces por

$$\hat{y}(t+j|t) = \sum_{k=1}^{N} h_k u(t+j-k|t) = H(q^{-1})u(t+k|t). \tag{1.4}$$

La notación $\hat{y}(t+k|t)$ indica que es la predicción en el instante $t+k$ obtenida haciendo uso de la información disponible hasta el instante t.

Las ventajas de este modelo son principalmente su sencillez para la descripción de la dinámica del proceso y el hecho de que no aparezcan términos recursivos. En cuanto a las desventajas, las más importantes son que no puede representar procesos inestables y el gran número de parámetros que aparecen en el modelo.

Modelos de respuesta a escalón

Estos modelos se emplean en el DMC, QDMC y LDMC (estos dos últimos son variantes del DMC). Este método, que es muy similar al anterior, presenta las mismas ventajas y desventajas que éste.

La salida en este modelo está dada por

$$y(t) = y_0 + \sum_{k=1}^{N} g_k \Delta u(t-k) = y_0 + G(q^{-1})(1-z^{-1})u(t) \tag{1.5}$$

donde g_k son los valores en los instantes de muestreo de la respuesta de la planta a una entrada escalón y $\Delta u(t) = u(t) - u(t-1)$. Considerando que el valor inicial, y_0, puede

considerarse igual a cero sin pérdida de generalidad, se puede tomar como predictor para la salida la siguiente expresión

$$y(t+j|t) = \sum_{k=1}^{N} g_k \Delta u(t+j-k|t) \tag{1.6}$$

Modelos de función de transferencia

El modelo basado en función de transferencia es usado, entre otros, por el UPC, EPSAC, EHAC, MUSMAR y GPC. En este caso la expresión para la salida del sistema se obtiene a partir de la función de transferencia, de forma que se puede escribir

$$y(t) = \frac{B(q^{-1})}{A(q^{-1})} u(t) \tag{1.7}$$

donde

$$A(q^{-1}) = 1 + a_1 q^{-1} + a_2 q^{-2} + \ldots + a_{na} q^{-na} \tag{1.8a}$$

$$B(z^{-1}) = b_0 + b_1 q^{-1} + b_2 q^{-2} + \ldots + b_{nb} q^{-nb} \tag{1.8b}$$

Entonces, la ecuación de predicción se puede escribir como

$$\hat{y}(t+j|t) = \frac{B(q^{-1})}{A(q^{-1})} u(t+j|t) \tag{1.9}$$

Esta representación tiene la desventaja de que es necesario cierto conocimiento previo del sistema, como por ejemplo el orden de los polinomios. Otro desventaja es que al ser una expresión recursiva, las predicciones son más susceptibles de errores. Por el contrario, la ventaja de este modelo en relación a los anteriores es que admite la representación de plantas inestables. Y además, el número de parámetros que intervienen en el modelo es mínimo.

Modelos de espacio de estados

El principal MBPC que utiliza esta representación es el PFC. Las ecuaciones que definen el estado y la salida del proceso son:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{F}\mathbf{x}(t-1) + \mathbf{G}\mathbf{u}(t-1) \\ y(t) &= \mathbf{H}\mathbf{x}(t) \end{aligned} \tag{1.10}$$

siendo **x** el estado del sistema y **F**, **G** y **H** las matrices de estado del sistema. La ecuación de predicción para este sistema es [Ast97]:

$$\hat{y}(t+j|t) = \mathbf{H}\hat{\mathbf{x}}(t+j|t) = \mathbf{H}\left[\mathbf{F}^{j}\mathbf{x}(t) + \sum_{k=1}^{j}\mathbf{F}^{k-1}\mathbf{G}\mathbf{u}(t+j-k|t)\right] \tag{1.11}$$

Las principales ventajas que ofrece esta representación son, por un lado, que permite obtener una buena representación interna del sistema y, por otro, que la extensión al caso multivariable se simplifica tomando este modelo. Como desventaja se puede citar la alta complejidad computacional que puede requerir esta representación. Además presenta el problema de la estimación del estado del sistema, para lo cual en ocasiones es necesario emplear un observador. Un ejemplo de un MBPC bajo la perspectiva del espacio de estados es el método presentado en [Lee94].

1.3.1.2 MODELO DE PERTURBACIONES

La elección del modelo de perturbaciones para el modelo predictivo tiene también una gran importancia ya que permite modelar perturbaciones ambientales, errores de modelado, ruido de medida, etc. La mayoría de los MBPC utilizan para modelar las perturbaciones uno de los siguientes tres modelos.

CARIMA

Este es el modelo de perturbaciones más utilizado. Lo emplean los esquemas GPC, EPSAC, EHAC y UPC. En el modelo CARIMA (*Controlled Auto-Regresive Integrated Moving Average*) las perturbaciones están dadas por

$$n(t) = \frac{C(q^{-1})}{D(q^{-1})}e(t) \tag{1.12}$$

donde $e(t)$ representa un ruido blanco de media cero, $C(q^{-1})$ es un polinomio que normalmente suele ser considerado igual a uno y $D(q^{-1})$ es un polinomio que incluye

explícitamente el término $\Delta = 1 - q^{-1}$. Este término es una acción integral que garantiza un control estacionario con error cero. La ecuación de predicción para este modelo resulta ser

$$\hat{n}(t+j|t) = F_j(q^{-1})n(t) \tag{1.13}$$

donde el polinomio $F_j(q^{-1})$ se obtiene resolviendo la siguiente ecuación diofántica:

$$1 = E_j(q^{-1})D(q^{-1}) + z^{-j}F_j(q^{-1}) \tag{1.14}$$

Como se observa, en esta expresión interviene el polinomio $D(z^{-1})$ que determina la dinámica de las perturbaciones. Resolviendo la ecuación se obtendrán los coeficientes de los polinomios E_j y F_j y, por lo tanto, se determina la expresión (1.13) para la ecuación de predicción.

Perturbación constante

Este es el modelo de perturbación empleado en el DMC y sus variantes, y en el PCA. La perturbación se modela mediante

$$n(t) = \frac{e(t)}{1 - z^{-1}}; \tag{1.15}$$

cuya mejor predicción es

$$\hat{n}(t+k|t) = n(t). \tag{1.16}$$

Perturbación con deriva (*drift disturbance*)

Es una extensión de la anterior que se utiliza básicamente en el PFC. El modelo de perturbación es

$$n(t) = \frac{e(t)}{(1 - z^{-1})^2}; \tag{1.17}$$

y la predicción óptima es

$$\hat{n}(t+k|t) = n(t) + k(n(t) - n(t-1)) \tag{1.18}$$

1.3.2 FUNCIÓN DE COSTO

La determinación de la política de control en los MBPC se basa en la minimización de una determinada función de coste. La característica de los MBPC es que en esa función de coste intervienen predicciones hechas sobre el proceso. Aunque las diferentes metodologías emplean diferentes criterios, la función de coste más extendida es la siguiente expresión cuadrática

$$J(N_1, N_2, NU) = \sum_{j=N_1}^{N_2} g(j)\left[w(t+j) - \hat{y}(t+j|t)\right]^2 + \sum_{j=1}^{NU} l(j)\left[\Delta u(t+j-1)\right]^2 \quad (1.19)$$

donde y es la salida predicha, w es la consigna, $\Delta u = u(t+j\text{-}1) - u(t+j)$ son los incrementos de comando, N_1, N_2 y NU son los horizontes de predicción y $g(j)$ y $l(j)$ son secuencias de pesos sobre los errores futuros y sobre los incrementos de comando. En general $g(j)$ y $l(j)$ suelen ser valores constantes o bien secuencias de valores exponenciales. Es muy usual una secuencia de peso exponencial para $g(j)$ de la forma $g(j) = b^{N_2-j}$. Donde el parámetro b, es una constante que pesa qué errores en el tiempo están más penalizados. Si $0< b < 1$ se penalizan más los errores que están más distantes del instante t. Y si b es mayor que 1 entonces tienen más peso los errores cometidos inicialmente. En cuanto al parámetro l, suele utilizarse tomando valores constantes y muy pequeños. Así, es muy frecuente ver algoritmos funcionando con $l = 10^{-3} \sim 10^{-6}$.

Existen formulaciones, como el IDCOM, que no incluyen términos de peso sobre los comandos, pesando únicamente los errores predichos. En otras aparecen términos cuadráticos en el comando, en lugar de en el incremento de comando como en el caso del UPC.

En la expresión (1.19) es habitual que la trayectoria de referencia elegida ($w(t+j)$) no coincida exactamente con la referencia real ($r(t+j)$) a la cual se quiere llevar la salida del proceso. Esto se hace con el objetivo de conseguir un acercamiento suave de la salida del sistema desde su valor actual al deseado. En general se suele considerar para la trayectoria de referencia una aproximación de primer orden como la que sigue:

$$\begin{aligned} w(t) &= y(t) \\ w(t+j) &= a\,w(t+j\text{-}1) + (1\text{-}a)r(t+j); \quad j = 1,2, \ldots \end{aligned} \quad (1.20)$$

donde a es una constante que toma valores entre 0 y 1. Si se desea una transición suave desde la actual salida hasta la consigna se debe tomar a próximo a 1. Cuanto más cerca está a de cero, más brusca es la transición hasta la consigna final. Esto se pone de manifiesto en la figura 1.6, donde aparecen dos trayectorias de referencia con diferentes valores de a. Se puede observar que la trayectoria con mayor valor, α_1, presenta un acercamiento más suave hacia la consigna.

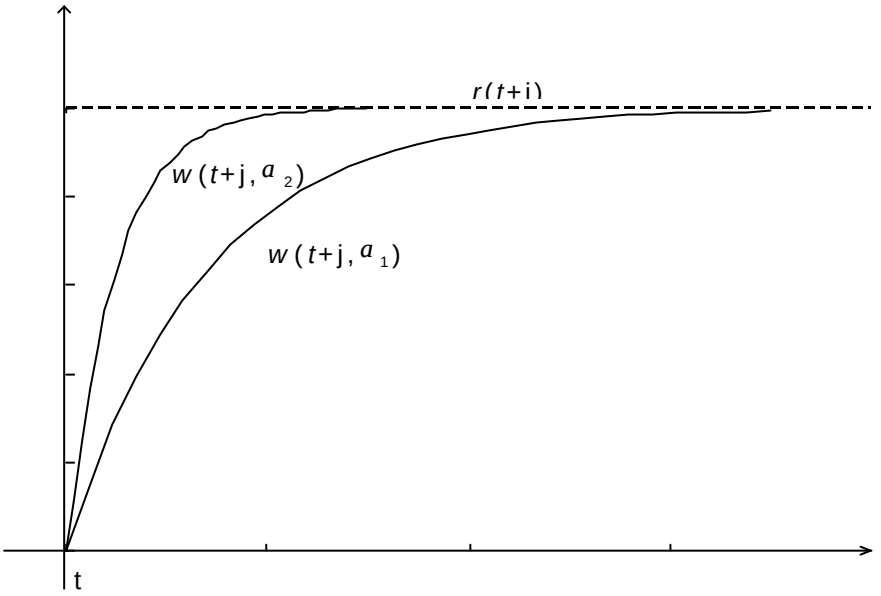


Figura 1.6. Trayectorias de referencia considerando diferentes valores de a ($a_1 > a_2$).

Uno de los aspectos más importantes en la función de costo es el de los horizontes de predicción. La elección correcta de estos parámetros juega un papel determinante en las prestaciones que alcanza el controlador. N_1 y N_2 se conocen como horizontes de salida mínimo y máximo, respectivamente. NU se conoce como horizonte de control. Los horizontes N_1 y N_2 delimitan el intervalo de tiempo en el que se desea que la salida del sistema sea igual a la de referencia. En general, N_1 no se suele tomar menor que el retardo del sistema, ya que los comandos obtenidos en la etapa t no afectarán a la salida del sistema en las etapas anteriores a $t+d$, siendo d el retardo del sistema. Por lo tanto no tiene sentido poner N_1 menor que d. La actividad del controlador estará regulada por NU, ya que $Du(t+j)=0$, para $j > NU$. Esto quiere decir que superado el horizonte de control, el comando aplicado será constante. Por lo tanto se obtiene una política de control más suave para valores de NU pequeños.

1.3.3 LIGADURAS EN EL MBPC

Una de las características más relevantes del GPC es su simplicidad para abordar problemas con ligaduras tanto en la variable de control como en la salida del proceso de forma sistemática y natural. Normalmente la aparición de estas ligaduras es debido a tres limitaciones: en el comando mínimo y máximo a aplicar, en la velocidad de variación del comando y en los valores que puede tomar la salida. En la práctica estas limitaciones pueden estar originadas por diferentes razones. Pueden ser debido a limitaciones físicas como la cantidad de energía que puede suministrar un calefactor o la cantidad de agua o la presión que puede soportar una válvula, etc. Pueden estar motivadas por cuestiones de seguridad, económicas, operativas, etc. En cualquier caso existe una necesidad de incluir estas ligaduras en el proceso de diseño. Algunas estrategias de control predictivo tienen en cuenta intrínsecamente estas ligaduras (como el DMC o el IDCOM), mientras que otros las pueden incluir a posteriori como es el caso del GPC [Cam93].

Las condiciones impuestas por las ligaduras en el valor de los comandos, en la velocidad de variación de los comandos y en la salida se pueden representar por las siguientes desigualdades [Cam93], [Cam95]:

$$\begin{aligned} & u_m \le u(t) \le u_M\,,\ \forall t \\ & du_m \le u(t) - u(t-1) \le du_M;\quad \forall t \\ & y_m \le y(t) \le y_M;\ \ \forall t \end{aligned} \tag{1.21}$$

Es posible obtener una expresión matricial simplificada que recoja todas estas ligaduras (ver Apéndice A):

$$\mathbf{Ru} \le \mathbf{c} \tag{1.22}$$

Con las ligaduras expresadas de esta forma, el cálculo de la ley de control del GPC se reduce a resolver un problema de programación cuadrática (ver Apéndice A).

Existen otro tipo de condiciones que suelen aparecer en los procesos aparte de las que se expresan en (1.21). Es común que aparezcan ligaduras de sobrepasamiento, como por ejemplo en manipuladores, ya que para estas plantas un sobrepasamiento en la salida puede provocar un deterioro de la pieza manipulada. Otra restricción se impone cuando se desea un comportamiento monótono para la salida del proceso (para evitar así oscilaciones indeseadas). Para plantas de fase no-mínima es habitual imponer ligaduras que restrinjan el

movimiento reverso en las primeras etapas del control. Otras ligaduras están relacionadas con las no-linealidades que exhiben la mayoría de los actuadores en la industria. Para compensar los efectos negativos de estas no-linealidades se imponen condiciones para evitar por ejemplo las zonas muertas, histéresis en los componentes. Otra ligadura que tiene especial importancia en la formulación de los MBPC es el de las ligaduras en el estado final. Como ya se ha comentado éstas son introducidas para garantizar la estabilidad del sistema . Aunque el origen de las condiciones de ligadura en todos estos casos es muy diverso, para todas ellas se puede encontrar una representación igual a (1.22) que permite sistematizar la resolución de estos problemas.

1.4 CONTROLADOR PREDICTIVO GENERALIZADO

El controlador predictivo generalizado fue propuesto por Clarke et al en 1987 [Cla87]. Este controlador ha marcado el desarrollo definitivo de los controladores MBPC que hasta ese entonces aún no contaban con el apoyo de algunos sectores de académicos. Como ya se ha comentado, en los primeros años de desarrollo de los controladores MBPC, y a pesar de su éxito en la industria, los algoritmos propuestos carecían de una base teórica que permitiera asegurar robustez y estabilidad para los procesos. Con la aparición del GPC muchas dudas que había planteadas fueron disipadas debido principalmente al grado de robustez que presentaba esta nueva formulación. Así, el GPC ha sido implementado en la industria desde finales de la década de los ochenta y durante toda la década de los noventa con notable éxito.

Esta estrategia es capaz de abordar con cierta robustez problemas con plantas de fase no-mínima, plantas inestables en lazo abierto o con polos mal amortiguados, plantas con retardo variable o incluso desconocido, plantas mal modeladas, de orden desconocido, sobreparametrizadas, etc. Se pueden abordar problemas de control muy diferentes para una amplia variedad de plantas. Para ello es necesario fijar una serie de parámetros de diseño que deben ser especificados por el usuario a partir del conocimiento previo de la planta y de las especificaciones de control.

El controlador GPC está inspirado en el controlador de mínima varianza generalizado [Cla79]. Pero las prestaciones que ofrece son, en general, superiores tanto al GMV como al control por asignación de polos [Wel79]. La idea es calcular una secuencia de control futura de tal forma que se minimice una función criterio definida sobre un horizonte de predicción. En esta función aparece un término cuadrático que pesa los errores predichos sobre este

horizonte de predicción y los incrementos del comando sobre otro horizonte. El resultado de esta minimización es una solución analítica que da la secuencia de control a aplicar en cada instante.

Desde su formulación el GPC ha sido objetivo de investigadores para mejorar las prestaciones y la robustez del controlador. Por ello, en la década de los noventa se han prodigado gran variedad de algoritmos que son modificaciones o mejoras del original propuesto en 1987. Incluso, otros algoritmos ligeramente diferentes se han inspirado en la idea del GPC, como es el caso del control por asignación de polos generalizado (GPP, Generalized Pole Placement) [Wel93].

1.4.1 FORMULACIÓN DEL GPC

1.4.1.1 ECUACIONES DE PREDICCIÓN

El controlador GPC asume para el proceso un modelo CARIMA:

$$A(q^{-1})y(t) = B(q^{-1})u(t-1) + C(q^{-1})x(t)/\Delta, \tag{1.23}$$

donde $y(t)$ es la salida del proceso, $u(t)$ es la entrada y A y B son dos polinomios en el operador retardo q^{-1} ($q^{-1}y(t) = y(t\text{-}1)$):

$$A(q^{-1}) = 1 + a_1 q^{-1} + \mathrm{K} + a_{na} q^{-na}, \tag{1.24a}$$

$$B(q^{-1}) = b_0 + b_1 q^{-1} + \mathrm{K} + b_{nb} q^{-nb} \tag{1.24b}$$

El segundo sumando del segundo miembro de (1.23) representa las perturbaciones en el sistema, $e(t)$,

$$e(t) = C(q^{-1})x(t)/\Delta \tag{1.25}$$

donde Δ es el operador de diferenciación definido como $1\text{-}q^{-1}$, $x(t)$ es una secuencia aleatoria no correlada y

$$C(q^{-1}) = 1 + c_1 q^{-1} + \mathrm{K} + c_{nc} q^{-nc}. \tag{1.26}$$

Este modelo de perturbaciones resulta bastante útil pues es capaz de representar correctamente los dos principales tipos de perturbaciones presentes en la práctica: cambios aleatorios ocurridos en instantes de tiempo aleatorio (debidos principalmente a cambios en los materiales) y movimiento Browniano (que aparece en procesos con balance de energía).

Por simplicidad, en el desarrollo que sigue se considerará $C(q^{-1})=1$ de forma que el modelo (1.23) pasa a ser

$$A(q^{-1})y(t) = B(q^{-1})u(t-1) + x(t)/\Delta \tag{1.27}$$

En el desarrollo del controlador el siguiente paso es la obtención de un predictor para la salida. Consideremos la obtención de un predictor de j pasos hacia adelante. Para ello se hace uso de la siguiente ecuación diofántica

$$1 = E_j(q^{-1})A\Delta + q^{-j}F_j(q^{-1}). \tag{1.28}$$

En esta ecuación aparecen los polinomios E_j y F_j como incógnitas. Dando el valor del polinomio $A(q^{-1})$ y el intervalo de predicción j es posible determinar unívocamente los coeficientes de E_j y F_j que verifican la identidad (1.28).

Multiplicando (1.27) por $E_j\Delta q^j$ se obtiene

$$E_j A\Delta y(t+j) = E_j B\Delta u(t+j-1) + E_j x(t+j)$$

Teniendo en cuenta (1.28) se puede obtener

$$y(t+j) = E_j B\Delta u(t+j-1) + F_j y(t) + E_j x(t+j) \tag{1.29}$$

Como el grado del polinomio $E_j(q^{-1})$ es j-1, todos los términos de ruido de (1.29) están en el futuro, por lo tanto el predictor óptimo para la salida es

$$\hat{y}(t+j|t) = G_j\Delta u(t+j-1) + F_j y(t) \tag{1.30}$$

donde $G_j(q^{-1}) = E_j B$. (Recuérdese que la expresión $\hat{y}(t+j|t)$ hace referencia a la salida predicha en el instante t usando la información de las entradas y salidas medidas hasta ese instante y de las entradas que se aplicarán hasta el instante $t+j$).

1.4.1.2 SOLUCIÓN A LA ECUACIÓN DIOFÁNTICA

La obtención del predictor óptimo (1.30) conlleva la resolución de la ecuación (1.28). Existen diferentes métodos para resolver esta ecuación. Uno de los más simples y efectivos es el que permite resolver la ecuación recursivamente [Cla87a]. Este método permite obtener E_{j+1} y F_{j+1} en función de E_j y F_j. Así, se puede demostrar que

$$F_{j+1}(q^{-1}) = f_{j+1,0} + f_{j+1,1}q^{-1} + \mathrm{K} + f_{j+1,na}q^{-na} \tag{1.31}$$

donde los coeficientes vienen dados por

$$f_{j+1,i} = f_{j,i+1} - f_{j,0}\tilde{a}_{i+1};\ \ i = 0,1,\ldots,na-1 \tag{1.32}$$

siendo $\tilde{a}_{i+1}$ los coeficientes del polinomio $A\Delta$. Para inicializar las iteraciones se toma $F_1(q^{-1}) = q(1-\Delta A)$. En cuanto al polinomio E_{j+1}, se encuentra que se puede expresar como

$$E_{j+1}(q^{-1}) = e_{j+1,0} + e_{j+1,1}q^{-1} + \mathrm{K} + e_{j+1,j-1}q^{-(j-1)} + e_{j+1,j}q^{-j} = E_j(q^{-1}) + e_{j+1,j}q^{-j} \tag{1.33}$$

donde $e_{j+1,j} = f_{j,0}$. y $E_1 = 1$. Entonces, el polinomio G_{j+1} puede ser obtenido recursivamente de la siguiente forma:

$$G_{j+1} = E_{j+1}B = (E_j + f_{j,0}q^{-j})B = G_j + f_{j,0}q^{-j}B \tag{1.34}$$

Es decir, los primeros j coeficientes de G_{j+1} serán iguales a los de G_j y los restantes coeficientes son

$$g_{j+1,j+i} = g_{j,j+1} + f_{j,0}b_i, \quad i = 0,\ldots,nb \tag{1.35}$$

Una vez obtenida una expresión para F_j y G_j, la predicción (1.30) queda totalmente determinada.

1.4.1.3 OBTENCIÓN DE LA LEY DE CONTROL

La obtención de la ley de control predictiva se basa en minimizar una función de costo que permita conducir la salida del sistema hasta la trayectoria de referencia fijada. Para ello la función de costo que se toma es (1.19)

$$J(N_1, N_2, NU) = \sum_{j=N_1}^{N_2} g(j)\left[w(t+j) - \hat{y}(t+j|t)\right]^2 + \sum_{j=1}^{NU} l(j)\left[\Delta u(t+j-1)\right]^2 \quad (1.19)$$

donde N_1 y N_2 son los horizontes de coste mínimo y máximo, respectivamente (este último también se conoce como horizonte de predicción u horizonte de salida). NU es el horizonte de control. $w(t+j)$ es la trayectoria de referencia que, como se comentó en la sección 1.3.2 se suele tomar como

$$w(t) = y(t)$$
$$w(t+j) = \alpha\, w(t+j\text{-}1) + (1\text{-}\alpha) r(t+j); \quad j = 1,2, \ldots$$

En el desarrollo siguiente, por simplicidad se tomarán para los horizontes los siguientes valores $N_1 = 1$ y $N_2 = NU = N$. Además, consideraremos $g(j)$ igual a 1 y $l(j)$ se tomará constante.

Teniendo en cuenta (1.30), las predicciones futuras de la salida del sistema se pueden escribir como

$$\hat{y}(t+1) = G_1 \Delta u(t) + F_1 y(t)$$
$$\hat{y}(t+2) = G_2 \Delta u(t+1) + F_2 y(t)$$
$$\text{M}$$
$$\hat{y}(t+N) = G_N \Delta u(t+N-1) + F_N y(t)$$

Es posible reescribir la salida predicha $\hat{y}(t+j)$ de forma que aparezca como la suma de dos términos, uno dependiendo de acciones de control pasadas y otro dependiente de las acciones de control futuras que deben ser calculadas. A estos términos también se les conoce como respuesta libre y respuesta controlada (ver figura 1.7) [Cla94]. Si a la respuesta libre la llamamos $f(t+j)$ se tiene

$$f(t+1) = [G_1(q^{-1}) - g_{10}]\Delta u(t) + F_1 y(t),$$
$$f(t+2) = q[G_2(q^{-1}) - q^{-1}g_{21} - g_{20}]\Delta u(t) + F_2 y(t),$$
$$\ldots$$

donde

$$G_i(q^{-1}) = g_{i0} + g_{i1}q^{-1} + \mathrm{K}$$

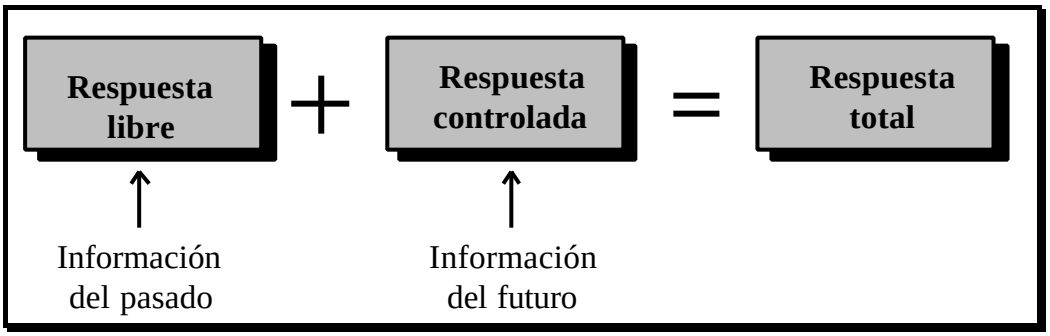


Figura 1.7. Componentes en la respuesta total del sistema.

Matricialmente se puede escribir entonces,

$$\hat{\mathbf{y}} = \mathbf{G}\mathbf{u} + \mathbf{f} \tag{1.36}$$

donde

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}(t+1|t) \\ \hat{y}(t+2|t) \\ \mathrm{M} \\ \hat{y}(t+N|t) \end{bmatrix}, \tag{1.37a}$$

$$\mathbf{u} = \begin{bmatrix} \Delta u(t) \\ \Delta u(t+1) \\ \mathrm{M} \\ \Delta u(t+N-1) \end{bmatrix}, \tag{1.37b}$$

$$\mathbf{G} = \begin{bmatrix} g_0 & 0 & \ldots & 0 \\ g_1 & g_0 & \mathrm{K} & 0 \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ g_{N-1} & g_{N-2} & \mathrm{K} & g_0 \end{bmatrix}; \tag{1.37c}$$

$$\mathbf{f} = \begin{bmatrix} f(t+1) \\ f(t+2) \\ \mathrm{M} \\ f(t+N) \end{bmatrix}. \tag{1.37d}$$

Definiendo el vector **w** como

$$\mathbf{w} = [w(t+1), w(t+2), \ldots, w(t+N)]^T,$$

la función de costo (1.19) se puede escribir como

$$J = (\mathbf{Gu} + \mathbf{f} - \mathbf{w})^T (\mathbf{Gu} + \mathbf{f} - \mathbf{w}) + l\,\mathbf{u}^T\mathbf{u} \tag{1.38}$$

La minimización de este índice suponiendo que no existen ligaduras en los controles futuros da como resultado el siguiente vector de comandos:

$$\mathbf{u} = \left[\mathbf{G}^T\mathbf{G} + l\,\mathbf{I}\right]^{-1}\mathbf{G}^T(\mathbf{w} - \mathbf{f}). \tag{1.39}$$

Teniendo en cuenta que se está aplicando una estrategia de horizonte móvil, sólo el primer comando de este vector es el que se aplica. Este comando viene dado por

$$u(t) = u(t-1) + \tilde{\mathbf{g}}^T(\mathbf{w} - \mathbf{f}) \tag{1.40}$$

donde $\tilde{\mathbf{g}}^T$ es la primera fila de $(\mathbf{G}^T\mathbf{G} + l\,\mathbf{I})^{-1}\mathbf{G}^T$.

En este desarrollo se ha supuesto $N_2 = NU = N$. Sin embargo, una de las grandes ventajas del GPC es la posibilidad de utilizar un valor para NU menor que N. Esto puede conseguir estabilizar los posibles modos inestables que se generen en el sistema. Además, el uso del horizonte de control NU, reduce significativamente la carga computacional del algoritmo. Así, si se considera el NU, entonces $\mathbf{u}$ es de dimensión NU y las ecuaciones de predicción se reducen a

$$\bar{\mathbf{y}} = \mathbf{G}_1\mathbf{u} + \mathbf{f} \tag{1.41}$$

donde

$$\mathbf{G}_1 = \begin{bmatrix} g_0 & 0 & \text{K} & 0 \\ g_1 & g_0 & \text{K} & 0 \\ \text{M} & \text{M} & & \text{M} \\ & & & g_0 \\ \text{M} & & & \text{M} \\ g_{N-1} & g_{N-2} & \text{K} & g_{N-NU} \end{bmatrix} \tag{1.42}$$

Y la ley de control correspondiente es

$$\mathbf{u} = \left[\mathbf{G}_1^T\mathbf{G}_1 + l\mathbf{I}\right]^{-1}\mathbf{G}_1^T(\mathbf{w} - \mathbf{f}) \tag{1.43}.$$

Como se aprecia en (1.42) la matriz $\mathbf{G}_1$ es de dimensión N x NU. Esto hace que el número de cálculos en la obtención de la secuencia de control sea notablemente inferior. Así, la matriz $\mathbf{G}_1^T\mathbf{G}_1 + l\mathbf{I}$ a la que hay que calcularle la inversa es ahora de dimensión NU x NU a diferencia de la que aparece en la expresión (1.39) que era de dimensión N x N. Teniendo en cuenta que en la mayoría de los situaciones una elección de NU=1 resulta apropiada, la inversión de la matriz se reduce a cálculos escalares.

1.4.1.4 MODIFICACIONES AL ALGORITMO BÁSICO

En la sección anterior se ha descrito el algoritmo básico del GPC. Sin embargo, cuando se presentó este algoritmo se propusieron algunas extensiones que permitían obtener mejores resultados en cuanto a robustez y estabilidad [Cla87b] [Cla89]. Una de estas modificaciones consiste en la inclusión de un polinomio $T(q^{-1})$ de forma que las perturbaciones se modelen

como un proceso no estacionario coloreado por este polinomio $T(q^{-1})$. El modelo CARIMA es ahora

$$A(q^{-1}) = B(q^{-1})u(t-1) + \frac{T(q^{-1})}{\Delta}x(t) \tag{1.44}$$

El polinomio $T(q^{-1})$ se conoce como polinomio observador y permite mejorar tanto la robustez como la respuesta ante perturbaciones. Este polinomio interviene como un parámetro de diseño más en el controlador. El efecto inmediato de la introducción de este polinomio en el modelo es que la predicción de la salida deja de ser óptima. Como contrapartida la política de control obtenida gana robustez en presencia de perturbaciones.

La obtención del predictor para la salida se obtiene ahora resolviendo la ecuación diofántica

$$T(q^{-1}) = E_j(q^{-1})A\Delta + q^{-j}F_j(q^{-1}) \tag{1.45}$$

Su resolución da lugar a la ecuación de predicción

$$\bar{y}(t+j|t) = G_j\Delta u^f(t+j-1) + F_j y^f(t) \tag{1.46}$$

donde "^{f}" denota una cantidad filtrada por $1/T(q^{-1})$. En la mayoría de las aplicaciones prácticas se encuentra que T puede tomarse como un polinomio de grado 1. Una expresión común para este polinomio es [Cla87b]

$$T(q^{-1}) = 1 - m\,q^{-1} \tag{1.47}$$

donde $0 \le m \le 1$. Otra posibilidad es la que se propone en [Yoo94]:

$$T(q^{-1}) = A(q^{-1})(1 - b\,q^{-1})^n, \quad n \le N_1. \tag{1.48}$$

donde b es un valor próximo a la raíz dominante de A.

Otra modificación que permite mejorar la robustez del algoritmo es la inclusión del polinomio $P(q^{-1})$. Éste se introduce para definir la siguiente salida auxiliar

$$y(t) = P(q^{-1})y(t) \tag{1.49}$$

De esta forma lo que se hace es predecir el comportamiento de la variable de salida auxiliar $y(t+j)$ en lugar de $y(t+j)$. Y se obtiene la ley de control minimizando el valor esperado de los errores cometidos sobre estas predicciones:

$$J(N_1, N_2, NU) = \sum_{j=N_1}^{N_2} g(j)[\hat{y}(t+j) - w(t+j)]^2 + \sum_{j=1}^{NU} l(j)[\Delta u(t+j-1)]^2 \tag{1.50}$$

Esta modificación en el algoritmo se traduce en una mejora en los sobrepasamientos de la salida. En aplicaciones de altas prestaciones donde NU se elige mayor que la unidad, $P(q^{-1})$ puede ser interpretado como el modelo inverso en lazo cerrado aproximado de la planta. De tal forma que si no se considera ruido, la respuesta en lazo cerrado de la planta considerando como entrada $w(t)$ es

$$y(t) \cong \frac{1}{P} w(t-k)$$

Otros resultados relacionados con la robustez del GPC se exponen en [Rob91].

1.5 AVANCES EN EL CONTROL PREDICTIVO GENERALIZADO

Como ya se ha comentado, los puntos débiles del GPC están relacionados principalmente con la estabilidad. Durante los primeros años de desarrollo de este tipo de controlador, los algoritmos no garantizaban estabilidad cuando se consideraba un horizonte de predicción finito. A principios de la década de los 90 surgieron nuevas propuestas que mejoraban el algoritmo GPC propuesto inicialmente y superaban estos problemas. En las siguientes secciones se resumen algunos de los algoritmos que han aportado avances más importantes.

1.5.1 CONTROL PREDICTIVO DE HORIZONTE MÓVIL RESTRINGIDO (CRHPC)

Uno de los primeros algoritmos que supuso un considerable avance en relación a la propuesta inicial del GPC es el controlador predictivo de horizonte móvil restringido. Este algoritmo que fue propuesto en 1991 [Cla91] solucionaba de forma relativamente sencilla los problemas de estabilidad de los MBPC propuestos hasta entonces. La idea era muy simple: se buscaba una secuencia de control que garantizara que, después de un cierto intervalo de tiempo posterior al que define el horizonte de predicción, la salida predicha coincidiera con la consigna marcada. Es decir, en el proceso de minimización se impone una ligadura adicional. De forma que la ley de control futura debe cumplir por un lado con la minimización de la ley de funcionamiento durante el periodo transitorio y por otro debe cumplir esta ligadura en el estacionario. Esta idea aparentemente simple, obtuvo un enorme éxito ya que conseguía evitar comportamientos inestables en la respuesta del sistema con la única contrapartida de obtener una ley menos óptima (debido a la pérdida de grados de libertad en el control) que el planteamiento tradicional. En la figura 1.8 se pone de manifiesto la idea del CRHPC. Como se observa la política de control debe verificar los dos requisitos: minimización de J y posteriormente la ligadura de *offset* cero.

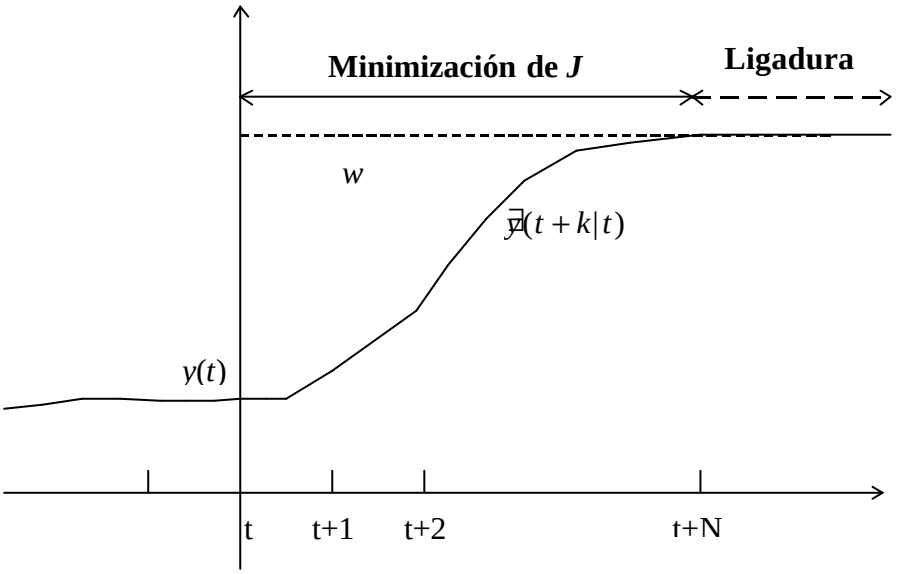


Figura 1.8. Control predictivo de horizonte móvil restringido.

1.5.1.1 ECUACIONES DE PREDICCIÓN

El CRHPC asume también un modelo CARIMA para representar el proceso a controlar. Supóngase un proceso con un retardo d, entonces el modelo se puede escribir:

$$A(q^{-1})\Delta(q^{-1})y(t) = B(q^{-1})\Delta(q^{-1})u(t-d) \tag{1.51}$$

Supóngase la siguiente función de costo:

$$J(N_1, N_2, NU) = \sum_{j=1}^{N} m(j)\left[w(t+j) - y(t+j|t)\right]^2 + \sum_{j=1}^{NU} l(j)\left[\Delta u(t+j-1)\right]^2 \tag{1.52}$$

La característica fundamental del CRHPC que lo diferencia de los MBPC´s propuestos hasta entonces es que se minimiza el índice de costo (1.52) sujeto al siguiente conjunto de m ligaduras futuras

$$\begin{cases} y(t+N+i) = w(t+N) \\ \Delta u(t+N+i) = 0 \end{cases} \quad \text{con } j = 1,2, \ldots, m \tag{1.53}$$

Por lo tanto, para calcular la ley de control es necesario predecir los valores de $y(t+j)$, j=1, …, $N+m$. La obtención de esta predicción se hace por medio de los polinomios auxiliares $E_j(q^{-1})$, $F_j(q^{-1})$ y $G_j(q^{-1})$. Los coeficientes de estos polinomios se obtienen resolviendo las dos siguientes ecuaciones polinómicas [Cla85]. [Cla87a], [Cla91]:

$$\left.\begin{aligned} 1 &= E_j(q^{-1})\Delta A(q^{-1}) + q^{-j}F_i(q^{-1}) \\ E_j(q^{-1})B(q^{-1}) &= \sum_{h=1}^{j} s_h q^{-h} + q^{-(j+1)}G_j(q^{-1}) \end{aligned}\right\} \tag{1.54}$$

siendo el grado de $E_j(q^{-1})$ igual a j-1 y donde s_h son los coeficientes de la respuesta escalón del sistema. Si se multiplica (1.51) por $E_j(q^{-1})$ y teniendo en cuenta (1.54) se llega a la siguiente expresión para la salida:

$$\begin{aligned} y(t+j) &= \sum_{h=1}^{i} s_h \Delta u(t+j-h) + f(t+j) \\ f(t+j) &= F_j(q^{-1})y(t) + G_j(q^{-1})\Delta u(t-1) \end{aligned} \tag{1.55}$$

Si se definen los siguientes vectores

$$
\begin{aligned}
Y(t) &= \left[y(t+1), y(t+2), \ldots, y(t+N) \right]^T \\
W(t) &= \left[w(t+1), w(t+2), \ldots, w(t+N) \right]^T \\
\Delta U(t) &= \left[\Delta u(t), \Delta u(t+1), \ldots, \Delta (t+NU) \right]^T \\
F(t) &= \left[f(t+1), f(t+2), \ldots, f(t+N) \right]^T \\
\overline{Y}(t) &= \left[y(t+N+1), y(t+N+2), \ldots, y(t+N+m) \right]^T \\
\overline{W}(t) &= \left[w(t+N+1), w(t+N+2), \ldots, w(t+N+m) \right]^T \\
\overline{F}(t) &= \left[f(t+N+1), f(t+N+2), \ldots, f(t+N+m) \right]^T
\end{aligned}
\tag{1.56}
$$

y las matrices

$$
\mathbf{G} = \begin{bmatrix}
s_1 & 0 & \ldots & 0 & 0 \\
s_2 & s_1 & 0 & \ldots & 0 \\
\mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{K} & \mathrm{M} \\
s_N & s_{N-1} & \ldots & s_1 & 0
\end{bmatrix}
\tag{1.57}
$$

$$
\overline{\mathbf{G}} = \begin{bmatrix}
s_{N+1} & s_N & \ldots & s_2 & s_1 \\
s_{N+2} & s_{N+1} & s_N & \ldots & s_2 \\
\mathrm{M} & \mathrm{M} & \mathrm{M} & \mathrm{K} & \mathrm{M} \\
s_{N+m} & s_{N+m-1} & \ldots & s_{m+1} & s_m
\end{bmatrix}
\tag{1.58}
$$

y si se definen las matrices de costo sobre las futuras acciones de control como

$$
\begin{aligned}
\mathbf{M}(t) &= \mathrm{diag}\{ g(t+1), g(t+2), \ldots, g(t+N) \} \\
\Lambda(t) &= \mathrm{diag}\{ l(t), l(t+1), l(t+2), \ldots, l(t+N) \}
\end{aligned}
\tag{1.59}
$$

Entonces se cumplen las siguientes relaciones

$$Y(t) = \mathbf{G}\Delta U(t) + F(t) \tag{1.60a}$$

$$\overline{Y}(t) = \overline{\mathbf{G}}\Delta U(t) + \overline{F}(t) \tag{1.60b}$$

Según esto, la función de costo (1.52) y las ligaduras definidas mediante las ecuaciones (1.53) se pueden reescribir como

$$J = \left[Y(t) - W(t) \right]^T \mathbf{M}(t) \left[Y(t) - W(t) \right] + \Delta U^T \Lambda \Delta U \tag{1.61}$$

$$\overline{\mathbf{G}}\Delta U(t) + \overline{F}(t) = \overline{W}(t) \tag{1.62}$$

Este problema de optimización se puede abordar por medio de los multiplicadores de Lagrange. La solución que se obtiene es:

$$\begin{aligned}\Delta U(t) = \tilde{\mathbf{G}}\Big[I - \overline{\mathbf{G}}^T [\tilde{\mathbf{G}}\tilde{\mathbf{G}}(t)\tilde{\mathbf{G}}^T]^{-1}\overline{\mathbf{G}}\tilde{\mathbf{G}}(t)\Big]\mathbf{G}^T M(t)\big[W - F(t)\big] + \\ +\tilde{\mathbf{G}}\overline{\mathbf{G}}^T\Big[\overline{\mathbf{G}}\tilde{\mathbf{G}}(t)\overline{\mathbf{G}}^T\Big]^{-1}\big[\overline{W} - \overline{F}(t)\big]\end{aligned} \tag{1.63}$$

donde

$$\tilde{\mathbf{G}}(t) = [\mathbf{G}^T M(t)\mathbf{G} + \Lambda(t)]^{-1} \tag{1.64}$$

Si se compara esta solución (1.63) con la solución del algoritmo básico GPC se puede comprobar el incremento de complejidad de la solución con ligaduras. Este hecho, hace que aparezcan problemas relacionados con la aplicabilidad del método, especialmente cuando se implementan estrategias adaptativas.

Una condición para que exista una solución a este problema es que la matriz $\overline{\mathbf{G}}\tilde{\mathbf{G}}(t)\overline{\mathbf{G}}^T$ sea invertible. Para ello se debe verificar

i) $m \leq NU+1$;

ii) $m \leq n+1$, (siendo n el orden del sistema).

Es decir, el número de ligaduras debe ser menor que el numero de variaciones en el control más uno, y debe ser también menor que el orden del sistema más uno.

1.5.1.2 CARACTERÍSTICAS DE ESTABILIDAD DEL CRHPC

Existen una serie de teoremas sobre la de estabilidad de este controlador. Los enunciados de estos teoremas se exponen a continuación (la demostración se puede encontrar en [Nic94]):

Teorema 1.- Si $N=NU > n+d+1$ y $m=n+1$ (d es el retardo en la planta), entonces el sistema en lazo cerrado (1.51) es asintóticamente estable .

Teorema 2.- Si $NU = n+d$ y $m = n+1$, entonces la ley de control (1.63) se convierte en la de un controlador *dead-beat* estable.

Teorema 3.- Si el sistema bajo control es asintóticamente estable, $\gamma=0$, $m=1$ y existe un ν que cumple

$$s_\nu \le s_{\nu+1} \le \ldots \le s_\infty,\ s_\nu > s_\infty / 2,\ s_\infty > 0$$

o bien,

$$s_\nu \ge s_{\nu+1} \ge \ldots \ge s_\infty,\ s_\nu < s_\infty / 2,\ s_\infty < 0$$

Entonces, para cualquier $N = NU \ge \nu-1$ el sistema en lazo cerrado (1.51) con la ley de control (1.63) es asintóticamente estable.

Teorema 4.- Suponiendo ciertas las condiciones del teorema 3, el sistema en lazo cerrado [(1.51), (1.63)] es asintóticamente estable tomando $NU = 0$ y $N = \ge n-1$.

Teorema 5.- Si el sistema a controlar es asintóticamente estable, $m=0$, m=1, NU=0 y existen dos constantes $K > 0$ y $0 < h < 1$ que cumplen

$$|s_i - s_\infty| \le K h^i,\ \ i \ge 0,$$

entonces, para cualquier N que cumpla

$$|s_{N+1}| > K \frac{1+h}{1-h} h^{N+1}$$

el sistema es estable en lazo cerrado.

Como se pone de manifiesto con estos teoremas, la estabilidad de este esquema de control queda garantizada mediante la correcta elección de las ligaduras en el estado final de la trayectoria y con una apropiada elección de los horizontes de predicción. La trascendencia de

este algoritmo ha sido muy elevada debido a que con él se resolvió uno de los problemas de los algoritmos MBPC que generó mayor escepticismo entre la comunidad científica.

Otras propuestas que garantizan estabilidad en el GPC son el SIORHC [Mos90] [Mos95], el algoritmo propuesto por Rawlings y Muske (a veces referenciado como algoritmo RM) [Raw93], o el SGPC que se describe en la siguiente sección.

1.5.2 CONTROL PREDICTIVO GENERALIZADO ESTABLE (SGPC)

Otra de las soluciones aportadas para mejorar la estabilidad de los algoritmos GPC fue introducida por Kouvaritakis *et al* en 1992 [Kou92]. El algoritmo propuesto presentaba una solución basada en el empleo de controladores de estabilización realimentando el proceso. Estos controladores están definidos por operadores polinomiales en lugar de por funciones de transferencia, con lo cual la carga computacional del algoritmo disminuye notablemente.

1.5.2.1 ESTRUCTURA DEL CONTROLADOR

El objetivo deseado con este esquema es el de mantener lo más posible las propiedades deseables del GPC y evitar al mismo tiempo los problemas de estabilidad que éste presenta. Esto se consigue aplicando el GPC posteriormente a que el sistema haya sido estabilizado mediante un bucle de realimentación interna. La estructura del bucle de estabilización es la que se muestra en la figura 1.9.

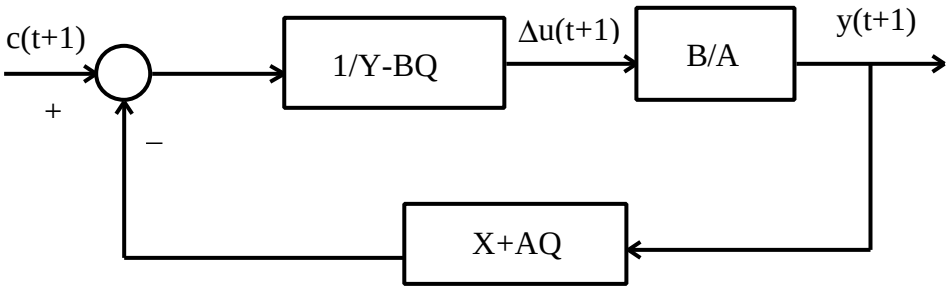


Figura 1.9. Estructura del bucle de estabilización del SGPC.

En esta figura *c*, *y* y *u* representan la señal de referencia, la salida del proceso y el comando respectivamente. *B/A* es la función de transferencia del proceso y los términos 1/*Y-BQ* y *X+AQ* son los elementos de estabilización. La determinación de estos elementos se hace teniendo en cuenta que para un sistema cuya función de transferencia sea

$$G(q^{-1}) = \frac{q^{-1}B(q^{-1})}{A(q^{-1})} = \frac{b_1 q^{-1} + b_2 q^{-2} + \ldots + b_n q^{-n}}{1 + a_1 q^{-1} + a_2 q^{-2} + \ldots + a_n q^{-n}} \tag{1.65}$$

se demuestra que el conjunto de controladores estables se puede representar como

$$K(q^{-1}) = \frac{N(q^{-1})}{M(q^{-1})} \tag{1.66}$$

siendo

$$\begin{aligned} N(q^{-1}) &= X(q^{-1}) + A(q^{-1})Q(q^{-1}) \\ M(q^{-1}) &= Y(q^{-1}) - B(q^{-1})Q(q^{-1}) \end{aligned} \tag{1.67}$$

donde $Q(q^{-1})$ es una función de transferencia estable y donde X e Y se obtienen de resolver la ecuación diofántica

$$B(q^{-1})X(q^{-1}) + A(q^{-1})Y(q^{-1}) = 1 \tag{1.68}$$

La función de transferencia $Q(q^{-1})$ es un parámetro de diseño y juega un papel similar al del polinomio $T(q^{-1})$ en el algoritmo básico [Kou92].

Este controlador (1.66) se introduce en el bucle de realimentación tal y como se muestra en la figura 1.9. Una propiedad importante que tiene esta configuración es que las relaciones entre la entrada *c* y la salida *y* y entre la entrada *c* y el incremento de comando vienen dadas por

$$\frac{y(q^{-1})}{c(q^{-1})} = B(q^{-1}); \quad \frac{\Delta u(q^{-1})}{c(q^{-1})} = A(q^{-1}) \tag{1.69}$$

Es decir, las transferencias de la entrada a la salida y de la entrada al comando vienen dadas por funciones polinómicas en vez de por un cociente de polinomios. Esto supone una notable reducción en la complejidad de cálculo.

1.5.2.2 ECUACIONES DE PREDICCIÓN Y LEY DE CONTROL

Según la figura 1.9, el modelo de la planta es

$$A(q^{-1})y(t+1) = B(q^{-1})\Delta u(t) \tag{1.70}$$

y la ecuación que define la señal de control es

$$M(q^{-1})\Delta u(t) = w(t) - N(q^{-1})y(t) = c(t) - q^{-1}N(q^{-1})y(t+1) \tag{1.71}$$

Si se considera un horizonte de salida N, y definiendo los vectores

$$\mathbf{y}^{+} = [y_{t+1}, \ldots, y_{t+N}]^T$$
$$\mathbf{y}^{-} = [y_t, y_{t-1}, \ldots, y_{t-N+1}]^T$$
$$\Delta\mathbf{u}^{+} = [\Delta u_t, \Delta u_{t+1}, \ldots, \Delta u_{t+N-1}]^T$$
$$\mathbf{c}_0 = [c_t, c_{t+1}, \ldots, c_{t+N-1}]^T$$

Con esta notación se puede demostrar que la predicción para la salida es

$$\mathbf{y}^{+} = C_B\mathbf{c}_0 - (C_b H_{z^{-1}N} + C_M H_A)\mathbf{y}^{-} - (C_B H_M - C_M H_B)\Delta\mathbf{u}^{+}$$

donde la notación $\mathbf{C}_B$ se emplea para hacer referencia a la matriz de convolución creada a partir de los coeficientes del polinomio B. En general esta matriz, para un polinomio $f(q^{-1}) = f_0 + f_1q^{-1} + f_2q^{-2} + \ldots$, se define como $[\mathbf{C}_f]_{ij} = f_{i-j}$. La matriz $\mathbf{H}$ es la matriz de Hankel y, para un polinomio $f(q^{-1})$ se define como $[\mathbf{H}_f]_{ij} = f_{i-1+j}$.

De igual forma que en el GPC se impone la condición de que de todos los controles futuros $\Delta\mathbf{u}$ sólo los NU primeros son distintos de cero, aquí también se impone una condición sobre la entrada $\mathbf{c}_o$. Así, se considera que este vector puede tomar cualquier valor hasta una

determinada etapa, a partir de la cual se le impone una condición a $\mathbf{c}_o$ para garantizar que se alcance la consigna impuesta. Es decir, $\mathbf{c}_o$ se toma igual a

$$\mathbf{c}_0 = [\mathbf{c}^+, \mathbf{c}^\infty]$$

donde $\mathbf{c}^+ = [c_t, c_{t+1}, \ldots, c_{t+nc}]^T \in \mathbf{R}^{nc}$ y $\mathbf{c}^\infty = [c_{t+nc+1}, \ldots, c_{t-nc-1}] \in \mathbf{R}^{N-nc}$. Los valores de c^∞ deben ser elegidos de tal forma que se verifique que, en el estacionario, la salida y debe seguir a la trayectoria de referencia $\mathbf{w} = [w_{t+1}, w_{t+2}, \ldots, w_{t+N}]$. Teniendo en cuenta la figura 1.9, esta condición es equivalente a que, en el estado estacionario, $\mathbf{c}$ sea igual a $N(q^{-1})y$, con lo cual $\mathbf{c}^\infty$ debe ser igual a

$$\mathbf{c}^\infty = \overline{\mathbf{C}}_N \mathbf{w}$$

donde $\overline{\mathbf{C}}_N$ denota la matriz formada por las *N-nc* últimas filas de $\mathbf{C}_N$. Debido a esto, el parámetro *m* se conoce como *horizonte de referencia*.

Si se define el vector $\mathbf{c}^- = [c_t, c_{t-1}, \ldots, c_{t-n}]^T$ se puede demostrar que la predicción para la salida y para Δu es

$$\mathbf{y}^+ = \Gamma_B \mathbf{c}^+ + \mathbf{y}^f ; \tag{1.72a}$$

$$\Delta\mathbf{u}^+ = \Gamma_A \mathbf{c}^+ + \Delta\mathbf{u}^f ; \tag{1.72b}$$

donde

$$\mathbf{y}^f = \mathbf{H}_B \mathbf{c}^- + \mathbf{M}_B \mathbf{c}^\infty , \tag{1.73a}$$

$$\Delta\mathbf{u}^f = \mathbf{H}_A \mathbf{c}^- + \mathbf{M}_A \mathbf{c}^\infty \tag{1.73b}$$

siendo Γ_A una matriz formada por las primeras *nc* columnas de $\mathbf{C}_A$, y donde $\mathbf{M}_A$ está formada por las restantes columnas de $\mathbf{C}_A$. De igual forma se define Γ_B y $\mathbf{M}_B$ a partir de los coeficientes del vector $B(q^{-1})$.

Considérese una ley de control que pese las errores cometidos y las variaciones de comando a la entrada del proceso,

$$J = \left\| \mathbf{r}^{+} - \mathbf{y}^{+} \right\|_2 + l \left\| \Delta \mathbf{u}^{+} \right\|_2 \tag{1.74}$$

esta función también se puede escribir como

$$J = (\mathbf{c}^{+} - \mathbf{c}')^T \mathbf{S}^2 (\mathbf{c}^{+} - \mathbf{c}') + d \tag{1.75}$$

donde

$$\mathbf{S}^2 = \Gamma_B^T \Gamma_B + l \Gamma_A^T \Gamma_A;$$
$$d = \left\| \mathbf{r}^{+} - \mathbf{y}^{f} \right\|_2 + l \left\| \Delta \mathbf{u}^{+} \right\|_2 - \left\| \mathbf{c}' \right\|_2;$$
$$\mathbf{c}' = \mathbf{S}^{-2} [\Gamma_b^T (\mathbf{r}^{+} - \mathbf{y}^{f}) - l \Gamma_A^T \Delta \mathbf{u}^{f}].$$

Se puede demostrar que si $N > nc + n$, entonces es posible buscar una ley de control que minimice este índice y que además produzca una evolución estable para la salida [Kou92].

1.5.3 CONTROL PREDICTIVO GENERALIZADO ESTABLE RESTRINGIDO (CSGPC)

El algoritmo que se presenta a continuación aborda el problema de la síntesis de una ley de control que minimice la función de costo del problema de control predictivo y que además respete las ligaduras a las que está sujeto el proceso a controlar. Generalmente, estas ligaduras están relacionadas con limitaciones en el comando, en la variación del comando y en los valores que puede tomar la salida del proceso. Uno de los trabajos que más éxito ha tenido en la resolución de este problema es el controlador predictivo estable restringido (CSGPC) [Ros93]. Este controlador está basado en el SGPC que, como se ha visto en el apartado anterior, resuelve el problema de la estabilidad de la estrategia GPC. El CSGPC plantea la resolución del problema de minimización de la función (1.75) sujeto a las condiciones de ligadura presentes en el sistema mediante un método basado en el algoritmo de Lawson [Law74].

1.5.3.1 MODELADO DE LAS LIGADURAS

A pesar de que son muchos los tipos de ligaduras que un sistema real puede, tener en la práctica, sólo se considerarán ligaduras en los valores de los comandos y en sus variaciones y en las salidas. Estas restricciones se pueden representar como

$$\left|u_{t+i} - U_0\right| \le U \tag{1.76a},$$

$$\left|\Delta u_{t+i}\right| \le R, \tag{1.76b}$$

$$\left|y_{t+i} - Y_0\right| \le Y \tag{1.76c}$$

donde se supone que las ligaduras son invariantes con el tiempo. Si se tiene en cuenta las expresiones (1.72a) y (1.72b) se pueden representar estas tres ligaduras mediante la expresión matricial

$$\left\|\mathbf{M}\mathbf{c}^{+} - \mathbf{v}(t)\right\|_{\infty} \le 1, \tag{1.77}$$

donde $\|\cdot\|_{\infty}$ indica norma infinito, $\mathbf{M} \in \mathbf{R}^{3N\times NU}$, $\mathbf{v} \in \mathbf{R}^{3N}$ es un vector dependiente de Δu^{f}, y^{f} y r. Los elementos de **M** y **v** dependen de t y del tipo de ligadura.

Se define la "región factible" (*feasible region*), $F_{nc}^{N}[v(t)]$ como el subespacio de todos los vectores $\mathbf{c}^{+}$ nc-dimensionales que, para un horizonte de salida N dado, satisfacen (1.77). Si $F_{nc}^{N}[v(t)]$ es el conjunto vacío para cualquier *nc* y *N* entonces el problema se dice que es impracticable o no factible (*infeasible*). Por el contrario, se emplea el término impracticabilidad o no factibilidad de corto alcance (*short-term infeasibility*) si $F_{nc}^{N}[v(t)]$ es vacío para un *nc* y *N* determinado.

1.5.3.2 ALGORITMO CSGPC

El controlador CSGPC se basa en la aplicación repetida del algoritmo de mínimos cuadrados con peso mixto (MWLS, *Mixed-Weights Least-Squares*) en cada instante de tiempo [Ros93]. Este algoritmo persigue la minimización del índice (1.75) sujeto a las ligaduras de la ecuación (1.77). Para ello se emplea un proceso recursivo que permite determinar la secuencia **c*** óptima mediante la minimización sobre $c^{(i+1)}$ del siguiente índice

$$J_{(i+1)}^{MWLS} = \left\| \begin{bmatrix} [w_{(i+1)}]^{1/2} e_{(i+1)} \\ [\mathbf{W}_{(i+1)}]^{1/2} e_{(i+1)} \end{bmatrix} \right\|^2 \tag{1.78}$$

siendo $\|\bullet\|$ la norma euclidea y con

$$e_{(i+1)} = S(c_{(i+1)}^{+} - c'), \quad e_{(i+1)}e_{(i+1)} = Mc_{(i+1)}^{+} - v(t),$$

donde $w_{(i+1)}$ es un peso escalar positivo y $W_{(i+1)}$ una matriz diagonal de pesos reales positivos definidos como

$$w_{(i+1)} = \frac{w_{(i)}}{\sum_{k=1}^{m} W_{(i)}^{kk} \left| e_{(i)}^{k} \right|}, \tag{1.79}$$

$$W_{(i+1)}^{jj} = \frac{W_{(i)}^{jj} \left| e_{(i)}^{j} \right|}{\sum_{k=1}^{m} W_{(i)}^{kk} \left| e_{(i)}^{k} \right|}. \tag{1.80}$$

La inicialización de los pesos se puede hacer tomando $w_{(0)}$ =1 y $W_{(0)}$ = I.

De esta forma, a medida que se incrementa *i*, el algoritmo converge hacia el vector **c*** que minimiza la función de costo (1.78). Es decir, a través de la minimización recursiva de un índice que aparece como una norma euclidea que incluye la función de costo del sistema y las ligaduras, se llega a la minimización de la función de costo del sistema cumpliéndose además la ecuación de ligaduras expresada en norma infinito (1.78). Una vez que ha sido calculado el vector óptimo de valores de referencia futuros **c***, el procedimiento para obtener el vector de incrementos de control óptimos y la implementación de la ley de control predictiva óptima se hace tal como se propone en el SGPC. Como se puede observar el CSGPC es exactamente igual al SGPC salvo en la determinación de **c*** para lo cual se hace uso del algoritmo MWLS.

1.5.3.3 PROPIEDADES DEL CSGPC

La propiedad fundamental del MWLS es que bajo la suposición de practicabilidad de corto alcance, el MWLS converge al valor de óptimo con ligaduras c*. En el caso de que no exista practicabilidad, el algoritmo converge a la solución que minimiza la máxima violación de

ligaduras. Esto último tiene importancia ya que supone disponer de un método para abordar la impracticabilidad de corto-alcance.

Algunas de las propiedades más importantes del CSGPC se resumen en los siguientes teoremas [Ros93]:

Teorema 1.- Sea un sistema lineal con función de transferencia $G(q^{-1}) = q^{-1}B(q^{-1}) / A(q^{-1})$ sujeto a las ligaduras de entrada/salida que en un instante *t*, y considerando un horizonte de salida *N* y un horizonte de control *NU*, están dadas por $\left\| \mathbf{M}\mathbf{c}^{+} - \mathbf{v}(t) \right\|_{\infty} \leq 1$. Sea $F_{nc}^{N}[v(t)]$ la región de viabilidad para $\mathbf{c}^{+}$. Entonces, si la región de viabilidad $F_{nc}^{\infty}[v(t)]$ definida para el *NU* considerado y con $N = \infty$ es no vacía, el CSGPC (para el valor dado de N) hará que la salida siga asintóticamente cualquier cambio en la señal de referencia, es decir, el CSGPC es asintóticamente estable.

Teorema 2.- Sea *r* cualquier señal de referencia que tome un valor constante r^0 después de un número finito de muestras. Entonces si existe una señal de entrada practicable u^*, que conduce la salida *y* hasta r^0 dentro de un número finito de intervalos de muestreo N_1, entonces el control CSGPC con $NU \geq N_1 - n$ y $N \geq NU + n$ será estable y la salida *y* llegará a r^0 después de N_1 intervalos de muestreo.

Teorema 3.- En el caso en que sólo existan ligaduras en la salida y considerando un *r* que después de un número finito de intervalos de muestreo N_1 tome un valor constante r^0, el CSGPC será estable siempre y conducirá a la salida *y* hasta r^0.

1.6 CONSIDERACIONES SOBRE LA IMPLEMENTACIÓN PRÁCTICA DEL GPC

Los algoritmos descritos en las secciones anteriores suponen una aportación importante para resolver los problemas de estabilidad que presenta el algoritmo básico GPC. Todos estos algoritmos abordan el problema de estabilidad mediante la imposición de una serie de

ligaduras en la entrada y la salida del sistema en la parte final de la trayectoria. Sin embargo, estas ligaduras pueden entrar en conflicto con las ligaduras físicas en la entrada del proceso. Cuando esto ocurre el problema se transforma en impracticable ("*infeasible*") y como se comentó en el apartado anterior, no existe una solución que minimice el índice (1.78). Hay que distinguir entre impracticabilidad ("*infeasibility*") de corto alcance e impracticabilidad global. El problema de llevar la salida de un proceso a un valor determinado puede ser practicable, pero puede no serlo para un horizonte de salida determinado, en este caso se habla de impracticabilidad de corto alcance. Si no existiera ningún valor del horizonte de predicción para el cual el controlador sea practicable, entonces se tendría impracticabilidad global. De hecho, la impracticabilidad en los controladores GPC (la violación de las ligaduras para llevar el sistema desde un punto inicial a otro) surge siempre debido a la imposición final de que la salida alcance a la referencia dentro de un horizonte finito.

Una propuesta para resolver este problema consiste en relajar este requisito [Ros95]. El problema es que un requisito para garantizar la estabilidad en el GPC es que la salida alcance al valor estacionario al final de un determinado intervalo. Para que esto siga cumpliéndose lo que se hace es que ese valor final al que debe estabilizarse la salida se convierte en un grado de libertad más. Es decir, se permite que $\mathbf{c}^{\infty}$ varíe hasta que se recupere la practicabilidad de corto alcance. De esta forma se puede modificar la función de costo del MWLS (1.78) de forma que se penalice la desviación de este valor final del valor deseado para la salida. Este mecanismo se activará sólo cuando el CSGPC se haga impracticable, de forma que se recupere la practicabilidad de corto alcance.

Otra extensión a este algoritmo es el CSGPC modificado (MCSGPC) [Ros96] que es una extensión del algoritmo comentado. Las mejoras que introduce el MCSGPC se deben a que garantiza la practicabilidad de corto alcance consiguiendo un seguimiento de la señal de referencia óptimo (transitorio + estacionario), a diferencia del algoritmo previo que sólo garantiza la consecución de un seguimiento de la señal en estado estacionario. En 1997 Gossner *et al* proponen una mejora más que permita que el MCSGPC de forma sistemática tenga en cuenta las perturbaciones y las condiciones de factibilidad de forma que se sigan manteniendo las características de estabilidad y de seguimiento asintótico en presencia de perturbaciones acotadas [Gos97].

1.7 SIMULACIÓN DEL GPC CON PLANTAS DE FASE NO MÍNIMA, INESTABLES EN LAZO ABIERTO Y CON CONSIGNAS VARIABLES.

Para poner de manifiesto las propiedades del controlador predictivo generalizado, se han realizado varias simulaciones sobre plantas con diferentes características. En la tabla 1.1 se resumen las características de estas plantas. Como puede observarse las plantas 1 y 2 son estables y de fase mínima. La planta 3 es una planta de fase no mínima ya que tiene un cero situado en +0.1. La función de transferencia de la planta 4 presenta un polo en +0.1 que la hace inestable. En general lo que se ha hecho es comparar los resultados obtenidos con el GPC con un controlador estándar como es el controlador PID [Ich91]. La sintonización del PID se ha realizado siguiendo el criterio de Zieglers-Nichols y refinando los parámetros que éste devuelve.

En las secciones anteriores se han expuesto las características que presenta la estrategia GPC. Entre ellas cabe destacar el buen comportamiento ante plantas de fase no mínima y plantas inestables en lazo abierto. Además, esta estrategia ofrece buenas prestaciones en el seguimiento de consignas variables en el tiempo. Los resultados obtenidos en simulación atestiguan el buen comportamiento del controlador en estas situaciones. En los siguientes apartados se resumen los resultados obtenidos.

Planta	**Función de Transferencia**	***Polos***
1	$\frac{1}{s^2+s+4}$	-0.5±1.9365j
2	$\frac{1}{s^3+5s^2+6s+1}$	-3.2470, -1.5550, -0.1981
3	$\frac{s-0.1}{s^2+3s+2}$	-1,-2
4	$\frac{1}{s^3+14.9s^2+48.5s-5}$	-10, -5, 0.1

Tabla 1.1. Funciones de transferencia sobre las que se ha simulado el GPC.

1.7.1 GPC vs PID. SEGUIMIENTO DE CONSIGNAS.

Algunas de las simulaciones realizadas han consistido en la comparación de la estrategia predictiva con una estrategia convencional como es el PID:

$$u(t) = k_p e(t) + k_i \int e(t)dt + k_d \frac{de(t)}{dt}$$

La sintonización de los parámetros de controlador se ha realizado tomando como parámetros de partida los que propone el criterio de Zieglers- Nichols [Ast97], [Fri96] y optimizando estos parámetros mediante técnicas de prueba y error en simulación. Así, para las plantas 1 y 2 los parámetros de control obtenidos son

- Planta 1: k_p=13.88, k_i=28.55, k_d = 5.
- Planta 2: k_p=15, k_i= 6.72, k_d = 8.

En cuanto al GPC, se ha implementado el algoritmo básico propuesto en [Cla87a], tomando como parámetros N=10, NU=1, l=0.001.

En las figuras 1.10 y 1.11 se muestran las respuestas de ambos controladores para las plantas 1 y 2 respectivamente cuando se considera una consigna escalón unitario. Como puede observarse, la respuesta obtenida con el GPC es muy superior que la conseguida con el PID, que presenta un comportamiento transitorio muy mal amortiguado, con un sobrepasamiento elevado y con un tiempo de establecimiento muy grande.

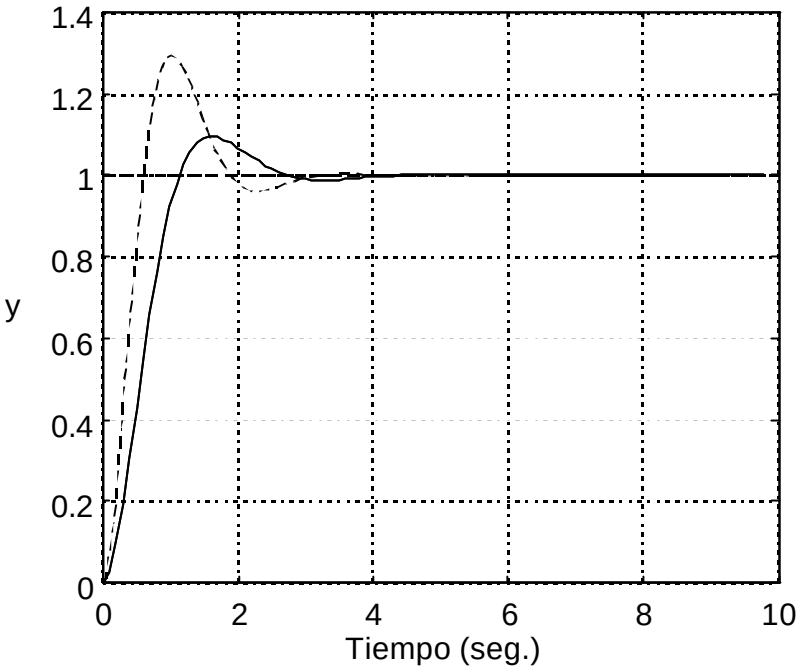


Figura 1.10. Simulación de la planta 1 con un controlador GPC con N = 10, NU=1 y λ = 0.001(trazo continuo) y con un controlador PID con k_p = 13.88, k_i =28.55, k_d = 5 (trazo discontinuo)

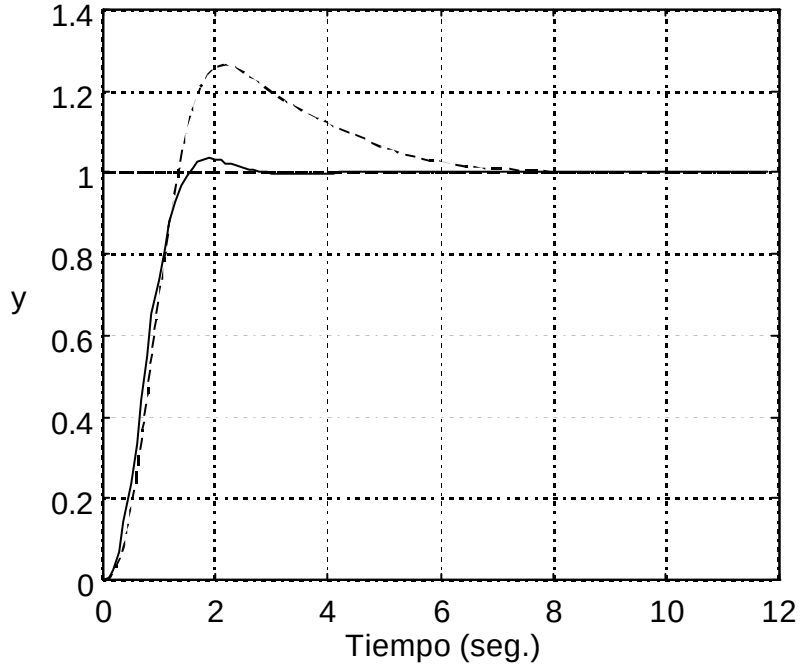


Figura 1.11. Simulación de la planta 2 con un controlador GPC con N = 10, NU = 1 y λ = 0.001 (trazo continuo) y con un PID k_p = 15, k_i =6.73, k_d = 8 (trazo discontinuo).

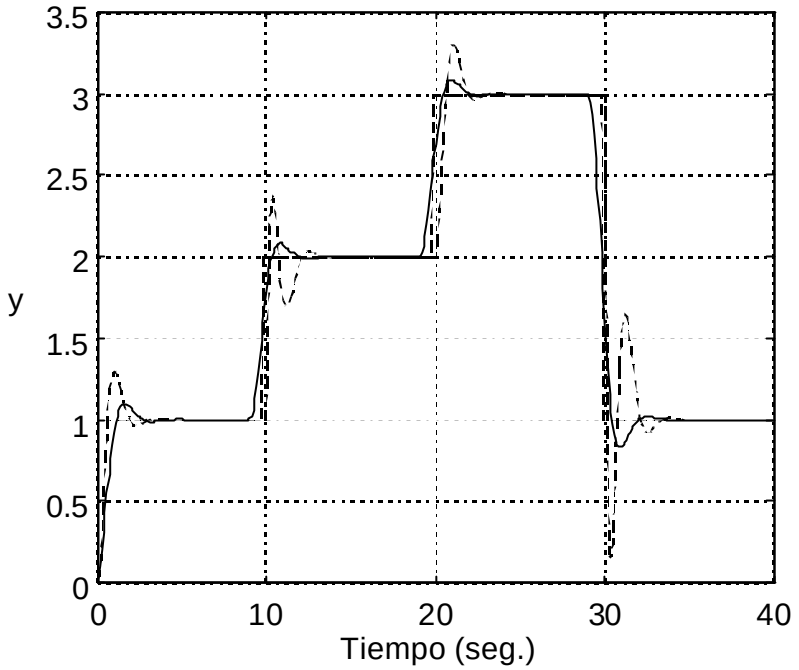


Figura 1.12. Simulación de la planta 1 con un control GPC con $N = 10$, $NU = 1$ y $l = 0.001$ (trazo continuo) y un PID con $k_p = 13.88$, $k_i = 28.55$, $k_d = 5$ (trazo discontinuo) considerando una consigna variable.

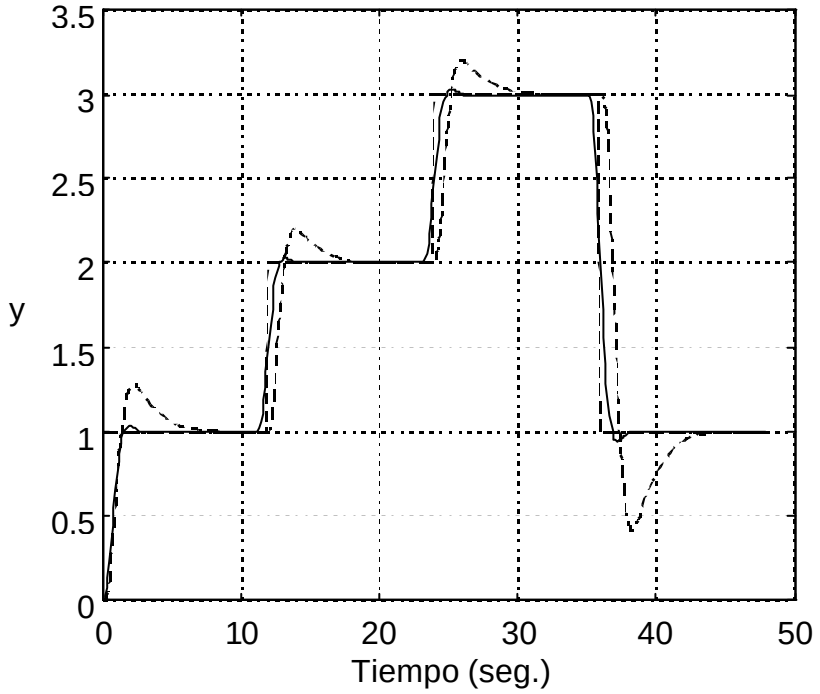


Figura 1.13. Simulación de la planta 2 con un control GPC (trazo continuo) y un PID con $k_p = 15$, $k_i = 6.73$, $k_d = 8$ (trazo discontinuo) considerando una consigna variable.

En las figuras 1.12 y 1.13 aparece una simulación en las mismas condiciones pero considerando una consigna variable. De nuevo, el comportamiento del controlador GPC es muy superior al del PID. Es interesante observar en este caso, como existe un efecto de anticipación en el controlador GPC, de forma que antes de que la consigna cambie ya el controlador tiene información de ello y se adelanta a este cambio. Esto provoca una aproximación a la nueva consigna mucho más suave.

1.7.2 PLANTA DE FASE NO MÍNIMA

Una de las situaciones en las que el GPC tiene especial interés es cuando se trata con una planta de fase no mínima, ya que las prestaciones del algoritmo en este caso son excelentes. La planta 3 de la tabla 1.1 es de fase no mínima. Se han comparado las repuestas obtenidas en simulación del GPC y de un controlador PID. Ya que la aplicación del criterio de Zieglers-Nichols para esta planta no es posible, la sintonización de los parámetros del PID se ha realizado eligiendo los parámetros que producían un mejor transitorio. Para el controlador PID los parámetros elegidos son k_p=2, k_i=-1, k_d = 0. Nótese que el valor de k_i es negativo, esto debe ser así para impedir que el sistema no se inestabilice. Los parámetros del GPC elegidos para la simulación son l=0.001 y N = 15, NU = 5.

En las figuras 1.14 y 1.15 aparecen las respuestas obtenidas con un control PID y con el GPC, respectivamente. Es interesante notar como con el controlador PID se produce el efecto característico de plantas de fase no mínima en el inicio del transitorio consistente en una repuesta en sentido contrario al de la dirección de la consigna. Sin embargo, con el GPC el comportamiento del sistema es muy superior mostrando un transitorio sin este efecto indeseado y con un sobrepasamiento mucho menor.

Se ha llevado a cabo otra simulación del GPC considerando la consigna variable a lo largo del tiempo. En la figura 1.16 se muestra una simulación de este misma planta con consigna variable tomando como horizontes N=10 y NU=1. Se puede comprobar que el seguimiento conseguido es realmente bueno, sin sobrepasamientos y con transitorios suaves.

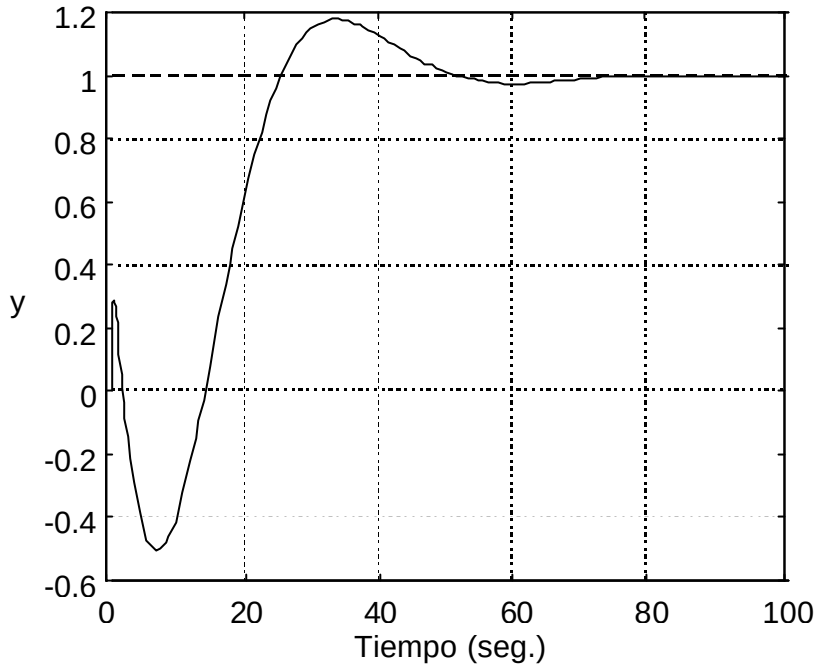


Figura 1.14. Control PID de la planta 3. k_p=2, k_i=-1, k_d = 0.

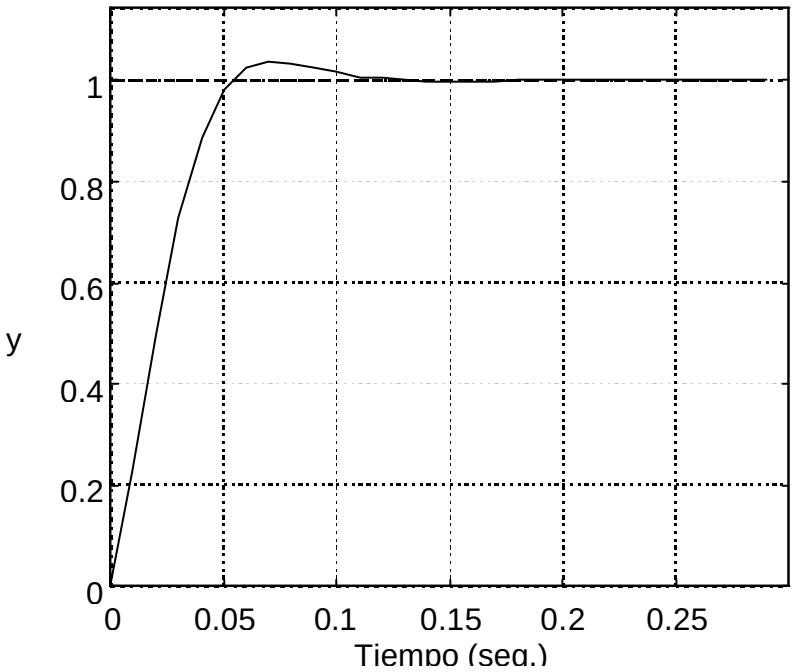


Figura 1.15. Control GPC de la planta 3. *N*=15, *NU*=5, λ=0.001.

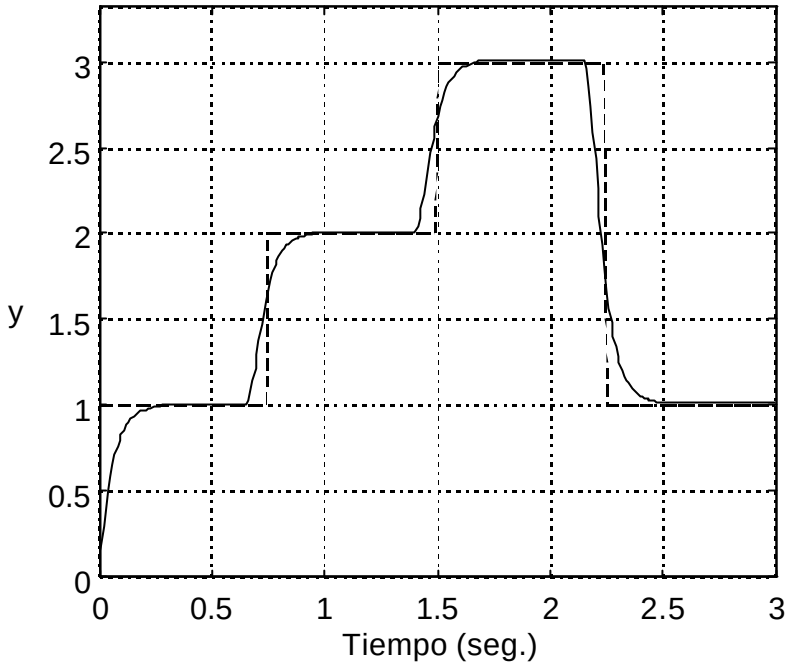


Figura 1.16. Control GPC de la planta 3 con consigna variable. N=10, NU=1, λ=0.001.

1.7.3 PLANTA INESTABLE EN LAZO ABIERTO

Otras simulaciones realizadas han sido considerando plantas inestables en lazo abierto. La planta elegida es la planta 4 de la tabla 1.1. Como se aprecia, esta planta presenta un polo que la inestabiliza y que está situado en s_1=+0.1. Optimizando en simulación los parámetros del PID que devuelve el criterio de Zieglers-Nichols, se encuentran los siguientes valores k_p=291.06, k_i=0.45, k_d = 21.28.

En la figura 1.17 se presentan las respuestas de la planta empleando un controlador PID (trazo discontinuo) y un controlador GPC (trazo continuo) con horizonte de salida igual a 15. El comportamiento del GPC es claramente superior al del PID. Es interesante observar que si el horizonte de salida en el GPC se reduce, entonces también lo hacen las prestaciones del sistema. En la figura 1.18 aparece un control de la planta 4 con un GPC con N=10. En este caso, se observa un empeoramiento notable en la respuesta en relación al caso en que N=15.

En la figuras 1.19 se muestran las simulaciones de la planta 4 con un controlador PID (trazo discontinuo) y un GPC (trazo continuo) considerando una consigna variable. Obsérvese que el comportamiento del GPC es claramente superior al del PID.

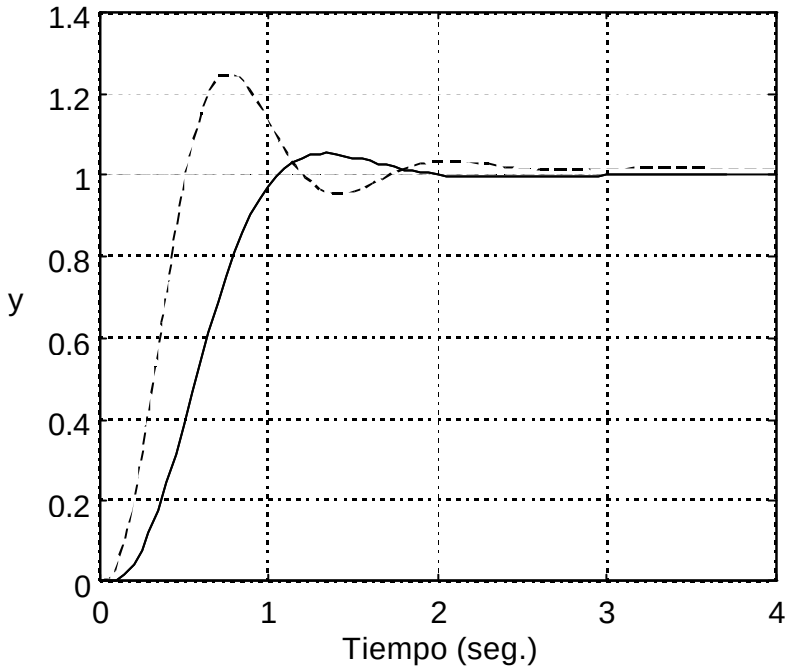


Figura 1.17. Simulación de la planta 4 con un controlador GPC con N = 15, NU = 1 y λ = 0.001 (trazo continuo) y con un PID con k_p=291.06, k_i= 0.45, k_d = 21.28 (trazo discontinuo).

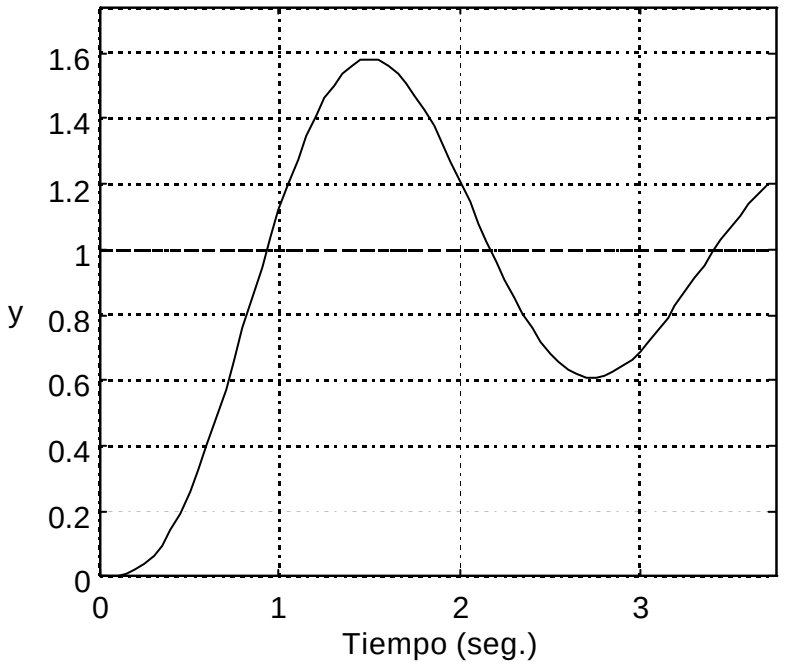


Figura 1.18. Control GPC de la planta 4. N=10, NU=1, λ=0.001.

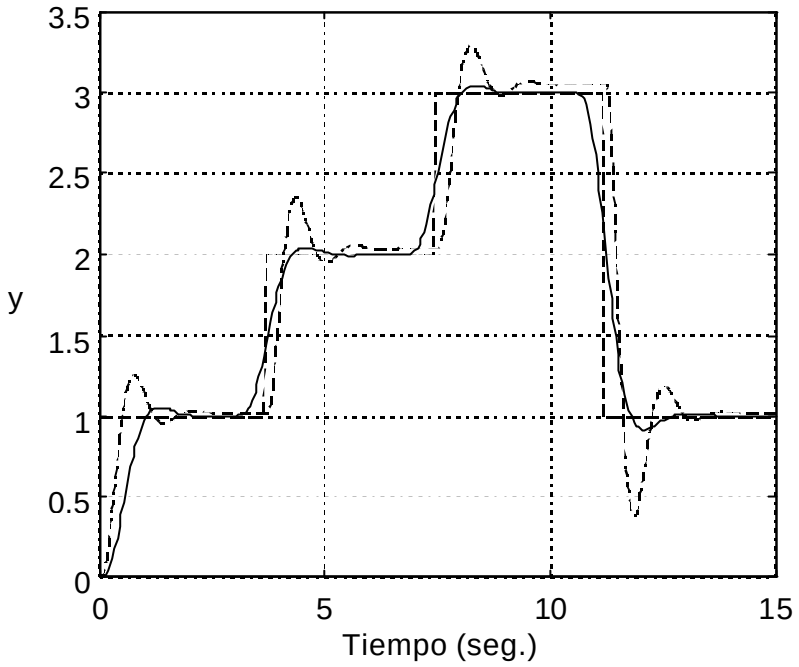


Figura 1.19. Control GPC de la planta 4 con N=15, NU=1, λ=0.001 (trazo continuo) y con un PID con k_p=291.06, k_i= 0.45, k_d = 21.28 (trazo discontinuo)..

2

DISEÑO E IMPLEMENTACIÓN DE UNA ESTRATEGIA GPC PSEUDO-MULTIVARIABLE PARA EL CONTROL EN TIEMPO REAL DE UN MOTOR DC

En este capítulo se describe la implementación de un controlador GPC en tiempo real sobre un motor de corriente continua. El algoritmo empleado es el algoritmo básico propuesto en [Cla87a]. Como se explicó en el capítulo anterior, este tipo de controladores presentan excelentes características para el control de procesos industriales. Estos procesos generalmente vienen caracterizados por constantes de tiempo relativamente grandes. Sin embargo, la planta con la que se trabaja en este capítulo tiene una constante de tiempo pequeña, con lo cual la implementación del controlador puede presentar dificultad para procesadores lentos como se refleja en un estudio elaborado sobre el tiempo de computación por periodo de muestreo. Por ello, en lugar de emplear un controlador multivariable para el motor se ha planteado un algoritmo GPC modificado basado en controladores SISO, de forma que se pueda garantizar la aplicabilidad del controlador.

Otra característica de la que se ha intentado sacar beneficio es la capacidad del GPC para incluir las componentes de ruido presentes en el sistema. Para el caso particular del motor, se ha llevado a cabo una comparación del algoritmo cuando se tienen en

cuenta en el diseño el ruido que afecta a una de las variables de estado del motor y cuando este ruido es despreciado.

2.1 DESCRIPCIÓN DE LA PLANTA

En esta sección se hará un estudio del proceso sobre el que se realizó la implementación del controlador. Se hará una descripción física del sistema y se caracterizará la planta, tanto en el dominio *s* (función de transferencia) como en el espacio de estados. Además, se presenta un estudio del ruido que afecta a la planta cuya caracterización será necesaria para plantear una estrategia de control estocástica.

2.1.1 DESCRIPCIÓN FÍSICA

La planta sobre la que se implementará el controlador es una bancada con un motor de corriente continua, una masa de inercia acoplada al eje de giro y sensores para medir la posición angular del eje y la velocidad angular. El motor DC está controlado en el inducido y la actuación sobre él se realiza aplicando voltajes en el rango -10V. → +10V. a la entrada. Este voltaje, después de pasar por una etapa de amplificación, es el que se aplica en el circuito inducido y permite mover el motor.

El estado del motor queda determinado por su posición y velocidad. Para medir la posición angular del motor se hace uso de un captador potenciómetrico que devuelve un voltaje proporcional a la posición angular del eje del motor. Por su parte, la velocidad angular se mide haciendo uso de una tacodinamo, que produce a la salida un voltaje que representa la velodicad de giro del motor.

En el apéndice C aparece una descripción detallada de todos los elementos que componen la planta.

2.1.2 REPRESENTACIÓN DEL SISTEMA

El motor es un sistema de posición y velocidad. La posición puede representarse con una función de transferencia de la forma

$$G(s) = \frac{k}{s(1+ts)} = \frac{Y_p(s)}{U(s)} \tag{2.1}$$

y para la velocidad se deriva esta expresión obteniendo la siguiente función de transferencia

$$G(s) = \frac{k}{1+ts} = \frac{Y_v(s)}{U(s)} \tag{2.2}$$

siendo k la *ganancia de la planta* y τ la *constante de tiempo,* que da cuenta de la rapidez de respuesta del motor. $U(s)$ es el comando (voltaje a la entrada), $Y_p(s)$ es la posición dada por el voltaje de salida del potenciómetro y $Y_v(s)$ es el voltaje de salida que da la tacodinamo y que representa la velocidad del motor.

Para obtener la representación del sistema en el espacio de estado se expresa (2.1) en la forma:

$$kU(s) = s(1+ts)Y(s)$$

Antitransformando para pasar al dominio temporal, se obtiene

$$ku(t) = \dot{y}(t) + t\ddot{y}(t) \tag{2.3}$$

Se pueden tomar como variables de estado el error en la posición del motor (error = consigna - posición del motor) y la derivada de este error (que coincide con la velocidad cambiada de signo). Entonces,

$$\left.\begin{aligned} x_1 &= e = r - y(t) \\ x_2 &= \dot{e} = -\dot{y}(t) \end{aligned}\right\} \tag{2.4}$$

Combinando (2.3) y (2.4) se llega a la ecuación de estado

$$\begin{pmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & \frac{-1}{t} \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{-k}{t} \end{pmatrix} u(t) \tag{2.5}$$

La solución de esta ecuación viene dada por [Oga93], [Kay80]:

$$x(t)=\begin{pmatrix} x_1(t) \\ \\ x_2(t) \end{pmatrix}=\begin{pmatrix} 1 & t(1-e^{-\frac{t}{t}}) \\ 0 & e^{-\frac{t}{t}} \end{pmatrix}\begin{pmatrix} x_{10} \\ \\ x_{20} \end{pmatrix}+ku\begin{pmatrix} -t+t(1-e^{-\frac{t}{t}}) \\ -1+e^{-\frac{t}{t}} \end{pmatrix} \tag{2.6}$$

que da la evolución en el espacio de estados del sistema.

2.1.3 DETERMINACIÓN EXPERIMENTAL DE LOS PARÁMETROS DE LA PLANTA

Viendo las expresiones de las funciones de transferencia (2.1) y (2.2) se deduce que los parámetros que se necesitan para tener perfectamente caracterizado al motor son k y t, pues estos deben ser iguales en ambas expresiones ya que corresponden a la misma planta. Se observa, sin embargo, que esto no es así, y que la ganancia del sistema k es diferente para la planta de posición y de velocidad. Obviamente, el parámetro t sí debe ser igual pues se trata de la misma planta. La razón por la que aparecen dos constantes diferentes k_v y k_p es porque los sensores tienen ganancias en voltaje diferentes que hacen que la salida de cada planta vaya multiplicada por ese factor. Por tanto, a la hora de tratar los datos obtenidos hay que tener en cuenta que los valores deben ser escalados. Por ejemplo, si se toma como referencia la ganancia de velocidad, k_v, entonces los datos de velocidad se dejarán tal como se han tomado y los de posición deberán estar multiplicados por el factor k_v/k_p .

De la ecuación (2.2) se tiene que la función de transferencia de la planta de velocidad es

$$Y_v(s)=\frac{k_v U(s)}{ts+1} \tag{2.7}$$

Por lo tanto, la determinación experimental de k_v y t se puede realizar ajustando la respuesta del motor ante una entrada escalón a esta expresión. En la tabla 2.1 se muestran algunos de los resultados obtenidos considerando entradas de distinta amplitudes. a la vista de estos resultados se decidió tomar como valores para la constante de tiempo y para la ganancia los siguientes:

$$t = 0.25 \text{ seg.}$$
$$k_v = 0.62 \tag{2.8}$$

En la figura 2.1a) aparece la gráfica correspondiente a uno de los ajustes realizados.

Altura escalón (V.)	**Periodo (seg.)**	**k_V ajustado**	t **ajustado (seg.)**
5	0.001	0.51	0.19
6	0.01	0.57	0.23
7	0.01	0.57	0.24
8	0.001	0.62	0.23
9	0.001	0.62	0.25
10	0.001	0.62	0.29

Tabla 2.1 Ajuste de velocidad.

En cuanto a la planta de posición se encuentra que su función de transferencia se puede expresar como

$$G(s) = \frac{k_p}{s(1+ts)} = \frac{Y_p(s)}{U(s)} \tag{2.9}$$

En este caso la determinación de las constantes se realizó de igual modo que para la planta de velocidad pero se consideró la planta en lazo cerrado para evitar el problema de la inestabilidad que provoca el polo en el origen. Por la tanto el procedimiento experimental consistiría en someter al motor a una entrada escalón realimentándolo con lazo unitario y estudiar la respuesta de éste. Los datos obtenidos se deben ajustar a la curva teórica que se obtiene al considerar la función de transferencia en lazo cerrado. Hay que tener en cuenta que sólo hay que ajustar la variable k_p , ya que para t se ha tomado el valor 0.25 seg. obtenido anteriormente. Después de diferentes pruebas se encontró que el valor de la ganancia que mejor ajuste produce es $k_p = 10$. En la figura 2.1b) se muestra gráficamente uno de estos ajustes. En esta figura se representa con trazo continuo la respuesta experimental y con trazo discontinuo la respuesta teórica con el ajuste

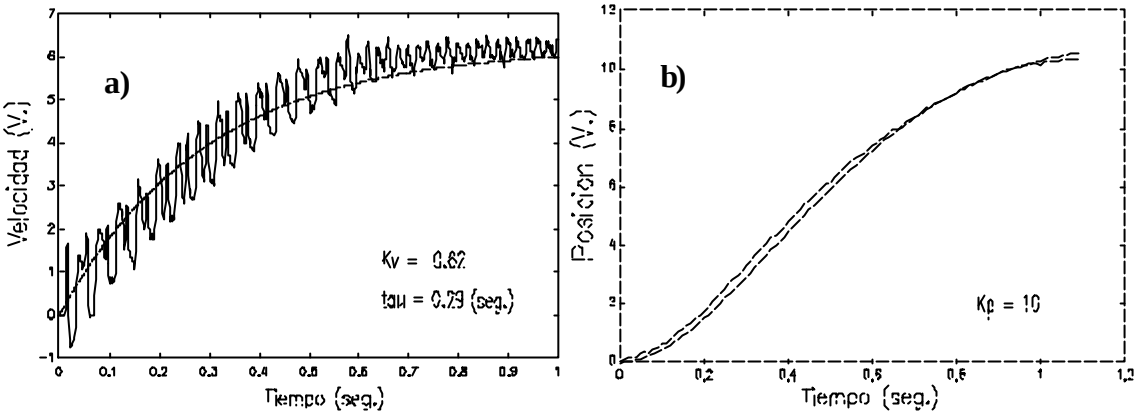


Figura 2.1. Determinación de los parámetros de la planta: a) Ajuste de velocidad. b) Ajuste de posición.

Conocido este valor, se obtiene el factor por el cual deben ir multiplicados los datos de posición:

$$\frac{k_v}{k_p} = \frac{2.5}{10} = 0.25$$

2.1.4 ANÁLISIS DEL RUIDO DE LA PLANTA

Estudiando varias muestras de la salida del sistema se llega a la conclusión de que el ruido de proceso de la planta puede ser despreciado. En cuanto al ruido de medida, se observó que una de las salidas de la planta, la de velocidad, iba acompañada de una perturbación que era, en magnitud, muy superior a la de la otra variable, la posición. Esto permitió despreciar el ruido en la posición y centrar todo el trabajo de caracterización del ruido en aquel que se produce al medir la velocidad del sistema.

En cuanto al origen de la perturbación hay que decir que es de carácter eléctrico y se produce en la tacodinamo con la que se mide la velocidad. Con lo cual su presencia es inevitable en cualquier experiencia realizada.

En este estudio, lo primero que se ha intentado es obtener una señal formada sólo por la contribución del ruido. Con este fin se somete al sistema a una entrada escalón y se resta de la curva obtenida, la curva teórica por la que evolucionaría el sistema. De este modo se consigue aislar la señal de ruido para trabajar sobre ella. En la figura 2.2 se ve la respuesta del

sistema en una de las pruebas realizadas. La zona continua será la que se tomará para el estudio.

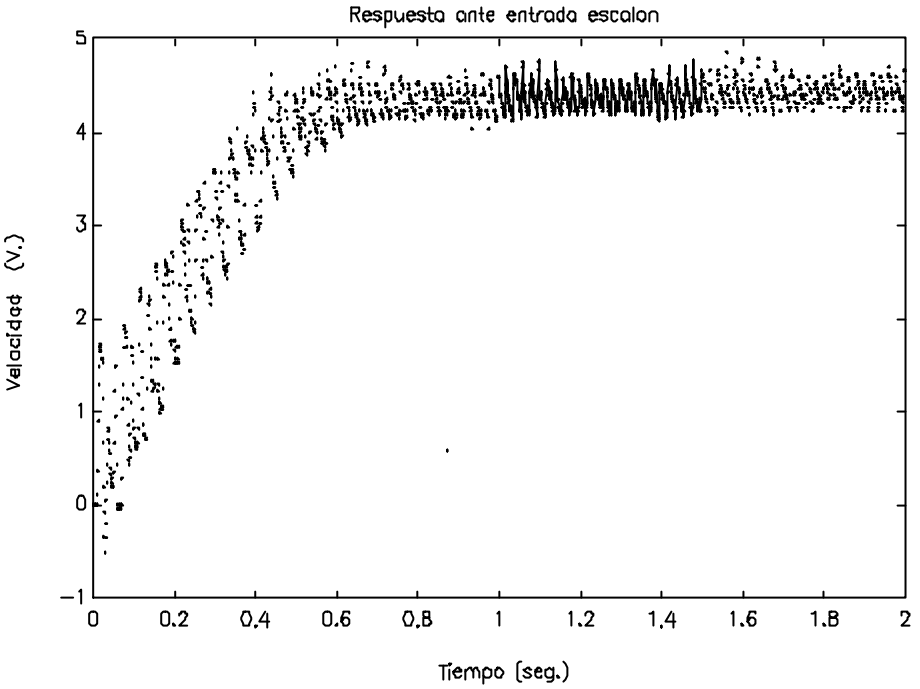


Figura 2.2. Respuesta del sistema ante entrada escalón.

Restando a todos estos valores el valor medio, es posible aislar la señal de ruido (figura 2.3). Lo que se pretende ahora es caracterizar esta señal. Supóngase, en principio que la distribución de probabilidad de la señal (figura 2.4) es gausiana. Para comprobarlo es sometida al test χ^2 para distribuciones gausianas.

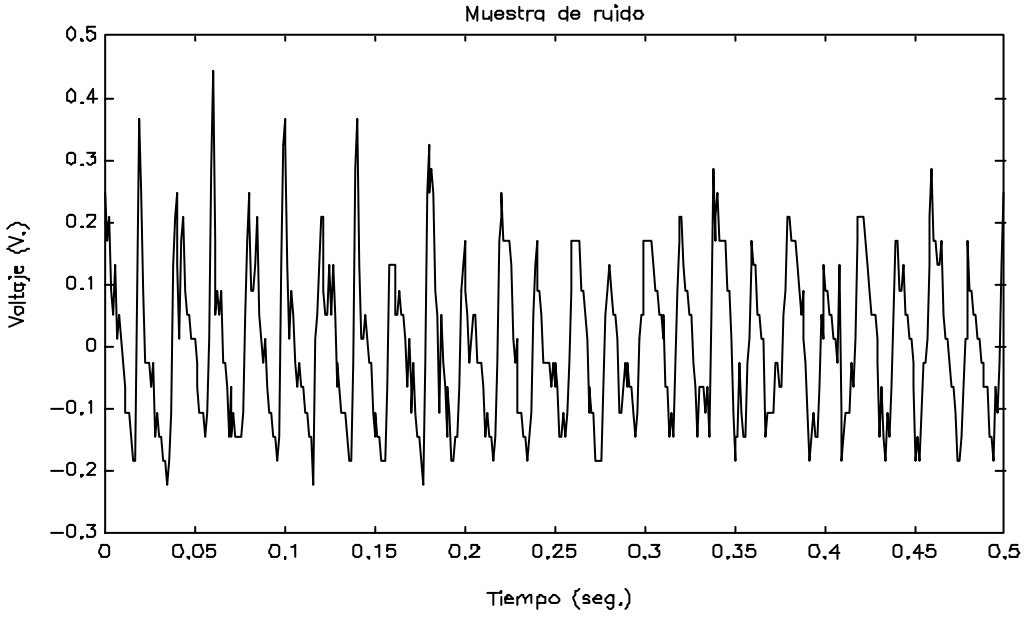


Figura 2.3. Análisis del ruido. Aspecto de la señal de ruido.

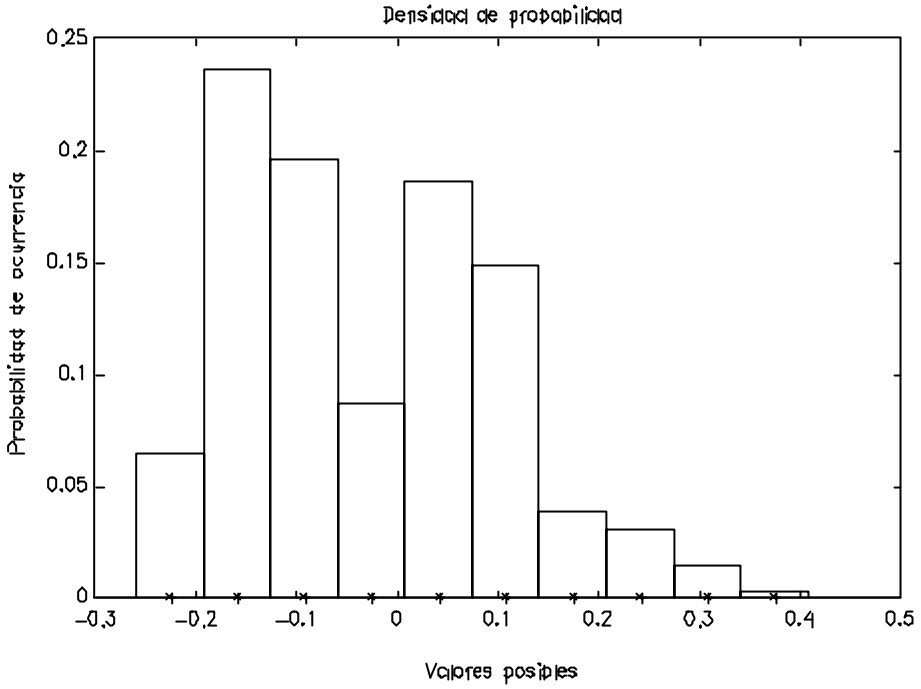


Figura 2.4. Análisis del ruido. Densidad de probabilidad de la señal de ruido.

Los resultados obtenidos son negativos no pudiendo asegurar que la distribución responde a una función de Gauss. En la tabla 2.2 se plasman los resultados de cuatro experiencias.

Nº de muestras	**Varianza**	**Media**	**K**[1]	c^2	c^2 **máximo**
200	0.010	0.000	7	33.49	14.9
200	0.017	0.003	6	14.00	12.8
200	0.021	0.003	7	47.21	14.9
500	0.020	0.000	7	35.36	14.9

Tabla 2.2. Análisis de la señal de ruido. Test χ^2.

Efectivamente, no se puede tratar la señal de ruido como si fuera gausiana pues en ningún caso se supera el test, ya que como se ve el valor de χ^2 obtenido es siempre mayor que el máximo permitido. Lo que se ha hecho es buscar un modelo para el ruido por medio del cual se pueda obtener una expresión que permita generar en simulación los valores obtenidos experimentalmente a partir de una señal gausiana. Así, se plantea el esquema que se muestra en la figura 2.5.

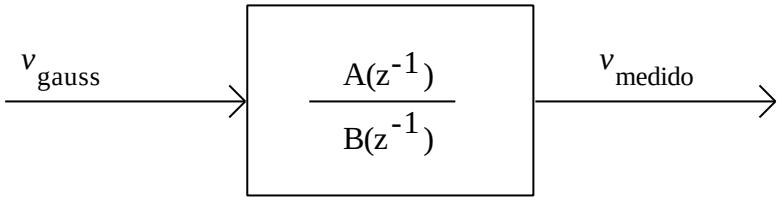


Figura 2.5. Modelo de identificación para el ruido

Es decir,

$$B(z^{-1})v_{medido}(k) = A(z^{-1})v_{gauss}(k) \tag{2.10}$$

o, desarrollando

$$(1+b_1z^{-1}+b_2z^{-2}+\ldots)v_{medido}(k) = (1+a_1z^{-1}+a_2z^{-2}+\ldots)v_{gauss(k)} \tag{2.11}$$

La señal gausiana en términos de la señal medida es

$$v_{gauss}(k) = \frac{(1+b_1z^{-1}+b_2z^{-2}+\ldots)}{(1+a_1z^{-1}+a_2z^{-2}+\ldots)}v_{medido}(k) \tag{2.12}$$

El modelo (2.10) responde a un modelo ARMA. Lo que se debe obtener es el vector de los parámetros

$$q = (b_1, b_2, \ldots, a_1, a_2, \ldots) \tag{2.13}$$

El esquema de identificación empleado está basado en los métodos de error de predicción [Lju87a], [Lju87b], [Sod89], [Sod91]. Los cálculos han sido realizados empleando el paquete de software MATLAB y su *toolbox* de identificación [Lju88]. Esta *toolbox* suministra una función llamada *armax* que implementa este método de identificación. Conjuntamente se utiliza la función *th2par* para obtener el resultado en el formato que interesa (propuesto en (2.13)).

Después de realizar muchas pruebas se llegó a la conclusión de que un valor adecuado para el grado del polinomio *A* es de 4 y para el polinomio *B* es de 2. Tomando estos valores

[1] El parámetro K se refiere al número de subintervalos de la señal tomados para hacer el test. Aquí se han tomado todos aquellos subintervalos para los que se diesen más de diez ocurrencias.

se aplicó la identificación y los valores obtenidos de los coeficientes b_1, b_2, a_1 , a_2, a_3 y a_4 para las señales de la tabla 2.2 fueron los que recoge la tabla 2.3.

$\mathbf{b_1}$	$\mathbf{b_2}$	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$	$\mathbf{a_4}$
-1.9020	0.9998	-1.1276	-0.0622	0.0060	0.3719
-1.8972	0.9957	-1.0503	-0.1078	0.0298	0.3779
-1.9008	0.9988	-1.1184	-0.1027	0.0802	0.3354
-1.8993	0.9958	-1.1717	-0.0064	0.1460	0.2227

Tabla 2.3. Coeficientes del vector θ.

Como promedio se tomarán los siguientes valores:

$b_1 = -1.90,$
$b_2 = 0.99,$
$a_1 = -1.12,$
$a_2 = 0.08,$
$a_3 = 0.33,$
$a_4 = 0.22.$

Una forma de estudiar la bondad de estos valores es comprobar si es posible obtener la señal $v_{gauss}(k)$ a partir de (2.12). Esta señal v_{gauss} obtenida tras el filtrado se muestra en la figura 2.5. Se puede comprobar que, efectivamente, su distribución (figura 2.16) se corresponde con una gausiana.

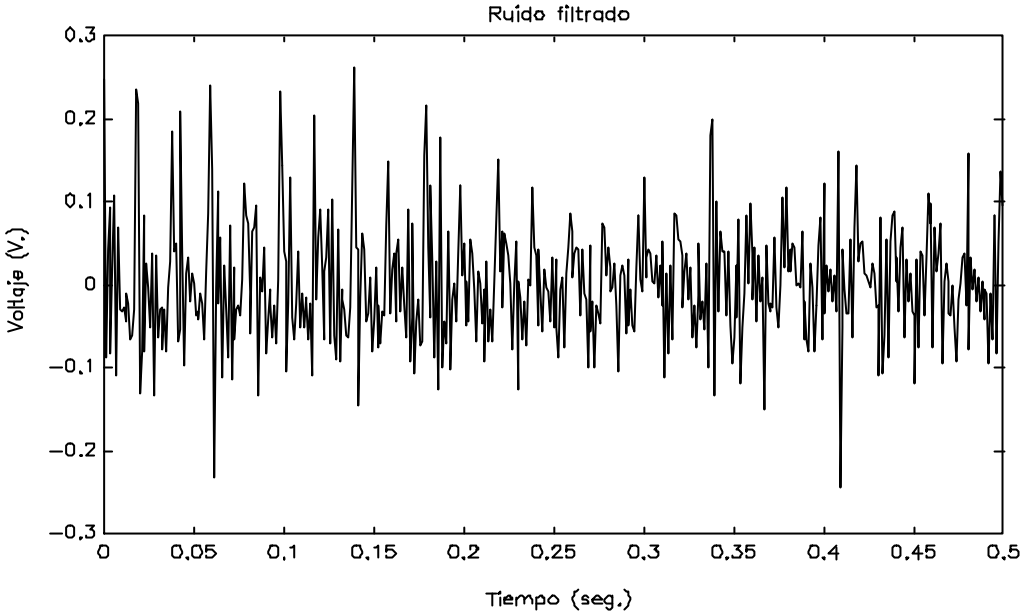


Figura 2.6. Análisis del ruido. Señal de ruido filtrada.

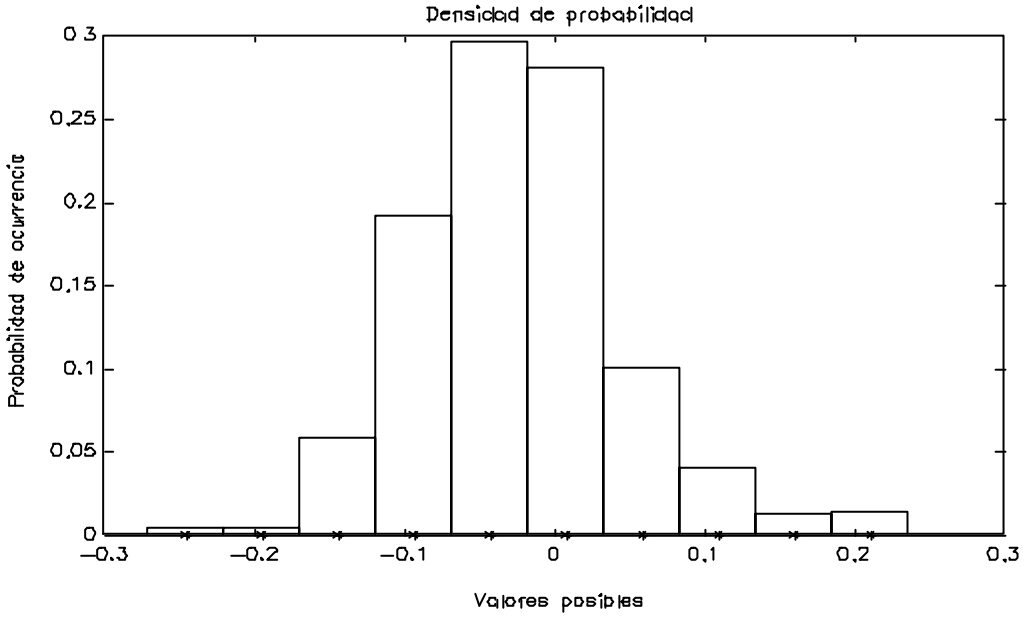


Figura 2.7. Análisis de la señal de ruido. Densidad de probabilidad de la señal filtrada.

En la tabla 2.4 se reflejan los resultados para los mismos datos de la tabla 2.2.

Nº de muestras	**Varianza**	**Media**	**K**	c^2	c^2 **máximo**
200	0.003	0.003	5	9.51	10.6
200	0.004	0.003	5	6.67	10.6
200	0.007	0.001	5	13.70	10.6

500	0.005	0.001	5	10.38	10.6

Tabla 2.4. Análisis de la señal de ruido. Test χ^2 para la señal de ruido filtrada.

Analizando estos resultados se comprueba que la señal v_{gauss} filtrada según (2.12) supera el test χ^2 , lo que permite asegurar que el vector θ calculado es fiable. Con esto se ha construido un sistema con el cual es posible generar en simulación una secuencia de ruido similar al de la planta. Para ello se sigue el siguiente proceso: partiendo de un generador de ruido gausiano, se le aplica el filtro $A(q^{-1})/B(q^{-1})$ y se obtiene una señal de ruido similar a la que presenta la planta.

2.2 DISEÑO DEL CONTROLADOR

En esta sección se describe el proceso de diseño del controlador para la planta de posición y velocidad. Como se comentó anteriormente, el motor queda representado por un modelo lineal de segundo orden. El objetivo planteado es el de controlar las dos variables, posición y velocidad, de forma simultánea. Una forma de aplicar el GPC a esta planta consistiría en realizar un planteamiento multivariable. Esto presenta un problema importante para una planta de constante de tiempo tan pequeña como la del motor. El algoritmo resultante de un planteamiento multivariable tendría una elevada carga computacional: el número de operaciones en tiempo real sería del orden de N^2, siendo N el horizonte de salida. El algoritmo propuesto en este trabajo reduce notablemente esta complejidad computacional haciéndola del orden de N. Este algoritmo se basa en desacoplar la planta de posición y velocidad por medio de una multiplexación temporal de las acciones de control.

2.2.1 DESACOPLO DEL SISTEMA

En la sección 1.4 se comentó el algoritmo básico GPC para una planta SISO. Aunque no de forma inmediata, la extensión al caso multivariable para este sistema también es posible [Dem92]. En cualquier caso, la aproximación multivariable al problema conlleva un incremento notable en las dimensiones de las matrices que intervienen en el algoritmo, y por lo tanto, en el número de operaciones. Dependiendo del sistema, este incremento puede significar que el algoritmo deje de ser aplicable en tiempo real o que se consiga un ancho de banda excesivamente corto. Una forma alternativa de abordar este problema consiste en desacoplar la planta en variables individuales, de forma que sea aplicable un esquema GPC SISO. En el caso del motor la planta se desacoplará en sus dos variables: posición y velocidad.

Para conseguir desacoplar las variables, se emplea un procedimiento indirecto que consiste en multiplexar en el tiempo las acciones correspondientes a cada variable. De forma que se apliquen secuencialmente acciones destinadas a controlar a cada variable. Para cada una de ellas, se define una referencia como objetivo de control. Para tener un mecanismo que permita definir en que puntos de la trayectoria se actúa más sobre cada variable, se define un factor de peso que determina el intervalo de tiempo en que se actúa sobre cada variable a lo largo de la trayectoria. Este factor se determina heurísticamente por medio de simulaciones en las que se ajusta su valor. En la práctica, este factor determina el número de veces que se

actúa sobre la variable de posición o de velocidad. Parece claro que, inicialmente, la acción sobre la variable de posición debe predominar sobre la acción en la velocidad. Sin embargo, se tiene la situación inversa en el último tramo de la trayectoria, porque en esta zona la evolución de la velocidad es crítica para alcanzar el objetivo.

2.2.2 COMPONENTES DE RUIDO EN EL ALGORITMO

Como se comentó anteriormente el ruido de proceso de la planta puede ser despreciado en relación con el ruido de medida. Además, el ruido de medida es sólo apreciable en la variable de velocidad. Esta componente de ruido se puede modelar mediante la expresión (2.10). Así, la salida medida de la planta real se puede describir por medio de la siguiente ecuación:

$$y(t) = y'(t) + v_m(t) = \frac{B(q^{-1})}{A(q^{-1})}u(t-1) + \frac{C(q^{-1})}{D(q^{-1})}v_g(t) \tag{2.14}$$

donde el segundo sumando del segundo miembro se puede considerar despreciable para la planta de posición. Expresado como (1.27) se obtiene:

$$P(q^{-1})y(t) = Q(q^{-1})u(t-1) + \frac{1}{\Delta}v_g(t); \tag{2.15}$$

donde

$$P(q^{-1}) = \frac{D(q^{-1})}{C(q^{-1})\Delta}; \quad Q(q^{-1}) = \frac{B(q^{-1})D(q^{-1})}{A(q^{-1})C(q^{-1})\Delta}. \tag{2.16}$$

De esta forma es posible aplicar un control GPC estocástico para la variable de velocidad, incluyendo los términos de ruido que afectan a esta variable. Es interesante tener en cuenta que los polinomios P y Q tendrán un conjunto infinito de componentes, con lo que será necesario truncar estos polinomios a partir de un orden determinado. Este proceso de truncamiento es muy importante para que el modelo tenga validez. El truncamiento en el polinomio Q debe hacerse calculando por separado los términos $B(q^{-1})/A(q^{-1})$ y $D(q^{-1})/C(q^{-1})\Delta$ y multiplicándolos posteriormente. El objetivo que se persigue es mantener tanto como sea posible las componentes que determinan la dinámica de la planta. Por lo tanto, en la expansión de $B(q^{-1})/A(q^{-1})$ deben considerarse más términos que en la expansión de $D(q^{-1})/C(q^{-1})\Delta$. En

el modelado llevado a cabo en este trabajo se ha considerando un truncamiento de sexto orden para $B(q^{-1})/A(q^{-1})$ y uno de tercer orden para $D(q^{-1})/C(q^{-1})\Delta$.

2.2.3 COMPLEJIDAD COMPUTACIONAL.

La implementación del GPC conlleva la obtención de una serie de matrices (ver figura 2.8). Estas matrices se calculan *off-line*, de forma que no representan ningún problema para la implementación en tiempo real del algoritmo. Los cálculos *on-line* del GPC pueden ser divididos en dos etapas principales. En la primera de las etapas se calcula el vector **f**, mientras que en la siguiente se calcula el valor de $u(t)$. Se ha realizado una estimación del número de operaciones para cada etapa del algoritmo. Los resultados aparecen resumidos en la tabla 2.5.

Operación	Primera etapa	Segunda etapa
Productos	$N*(na+nb+1)$	N
Sumas	$N*(na+nb+2)$	$2N$
Carga desde memoria	$2N*(na+nb+1)$	$2N$
Grabación a memoria	N	1

Tabla 2.5. Número de operaciones on-line en el GPC.

El resultado que más llama la atención de este análisis es que el número de operaciones *on-line* en el GPC no depende del horizonte de control NU, sino solamente del horizonte de salida, N. Tomando como base las hojas de datos de la familia de procesadores INTEL 80x87, se hizo una estimación de los tiempos de ejecución para varias operaciones en punto flotante. Estos tiempos se resumen en la tabla 2.6. Con estos valores se estimó el tiempo necesario para realizar las operaciones que se muestran en la tabla 2.5. Para ello se considera el caso más desfavorable que es aquel instante de muestreo en el que se ha de obtener el comando a aplicar sobre la velocidad, ya que este modelo como se vio en el apartado anterior presenta una complejidad mayor que el modelo para la planta de posición. Se tomó como horizonte de predicción N=10. Si se considera un microprocesador 80287 a 6 MHz y teniendo en cuenta que para la planta de velocidad na=2 y nb=7, el tiempo requerido es aproximadamente de 7 ms., mientras que para un procesador 80487 a 66 MHz. es de 1 ms. A estos valores hay que sumarle el tiempo para las actualizaciones de las matrices en memoria, conversiones de los datos, interrupciones periódicas del sistemas, etc.

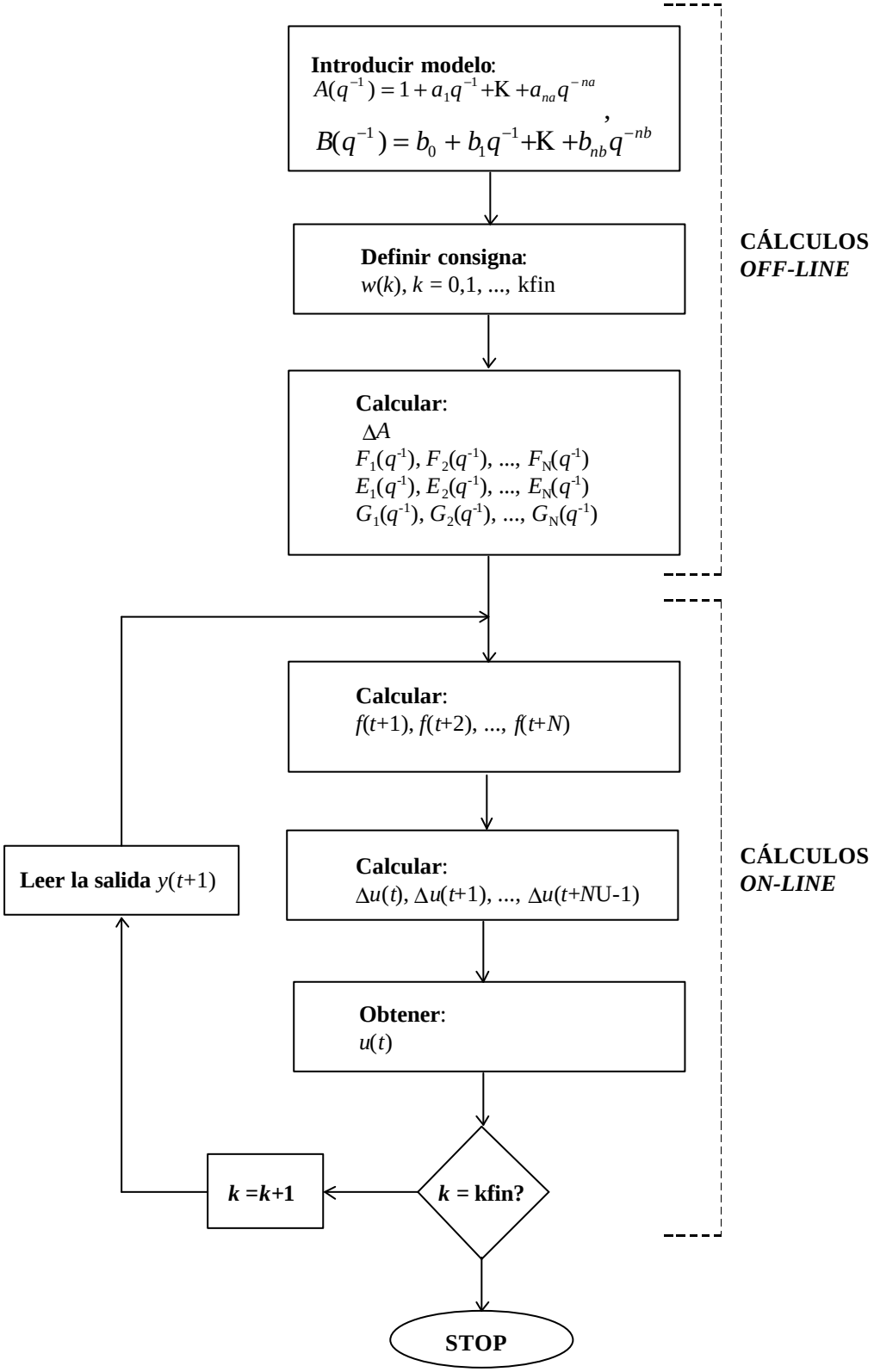


Figura 2.8. Algoritmo básico GPC.

Tomando un periodo de muestreo de 10ms. parece claro que un PC-AT con un coprocesador 80287 no es lo suficientemente rápido para llevar a cabo el control. Para comprobarlo, se intentó la implementación sobre esta computadora y, efectivamente aparecieron problemas de tiempo de cálculo que imposibilitaron la implementación del algoritmo. Cuando se empleó una computadora basada en el procesador 80486 el algoritmo pudo ser implementado sin ningún problema de tiempo. Esto es un claro ejemplo de la importancia que presenta el tiempo de computación en el GPC cuando las constantes de tiempo del sistema son pequeñas.

INSTRUCCIÓN EN PUNTO FLOTANTE	**TIEMPO DE EJECUCIÓN APROXIMADO (mseg.)**			
	8087 / 5 MHz	**80287 / 6 MHz**	**80387 / 25 MHz**	**80486 / 66 MHz**
Suma/resta	20	17	5	4
Producto	29	24	11	3
Cargar desde memoria	11	9	2	0.6
Guardar en memoria	5	4	2	0.6

Tabla 2.6. Tiempos de ejecución para diferentes procesadores.

2.3 IMPLEMENTACIÓN DEL ALGORITMO GPC PSEUDO-MULTIVARIABLE

En la implementación del controlador se han planteado dos objetivos distintos. Uno es demostrar la aplicabilidad del algoritmo GPC modificado para el control multivariable de la planta. Y otro está destinado a poner en práctica una estrategia determinista y una estocástica y comparar los resultados obtenidos con ambas, con la intención de demostrar con un proceso real el incremento en el rendimiento de la estrategia GPC cuando se modelan los ruidos de la planta [Aco89].

El diagrama de flujo del algoritmo GPC empleado se muestra en la figura 2.8. Este controlador ha sido aplicado sobre el motor DC siguiendo el esquema comentado en la sección 2.2 que permite el control simultáneo de las dos variables de estado.

Los polinomios que definen el modelo (2.15) para la planta de posición son:

$$\begin{aligned} P_p &= 1 - 1.9608q^{-1} + 0.9608q^{-2} \\ Q_p &= 0.1233*10^{-3} + 0.1217*10^{-3}q^{-1} \end{aligned} \qquad (2.17)$$

La variable velocidad despreciando el ruido, se puede aproximar con el modelo:

$$\begin{aligned} P_{vd} &= 1 - 0.9608q^{-1} \\ Q_{vd} &= 0.0245 \end{aligned} \qquad (2.18)$$

La aproximación estocástica para esta variable, truncando B/A en orden seis y $D/C\Delta$ en orden tres es

$$\begin{aligned} P_{ve} &= 1 + 0.22q^{-1} + 0.4391q^{-2}; \\ Q_{ve} &= 0.0245 + 0.0289q^{-1} + 0.0386q^{-2} + 0.0371q^{-3} \\ &+0.0356q^{-4} + .0342q^{-5} + 0.0136q^{-6} + 0.0088q^{-7}. \end{aligned} \qquad (2.19)$$

Considerando como objetivo de control llevar al sistema desde un punto inicial al origen del plano de fase (x_1=0, x_2=0) se han realizado algunas simulaciones considerando el modelo propuesto para la planta, previamente a la implementación en tiempo real. En las figuras 2.9 y 2.10 se muestran dos de las simulaciones realizadas considerando un diseño determinista (trazo discontinuo) y un modelo estocástico (trazo continuo). En las figuras se muestran las

trayectorias que sigue el sistema en el plano de fase (error de posición, x_1 y derivada del error de posición, x_2). Se comprueba claramente que las prestaciones alcanzadas con el planteamiento estocástico superan claramente a los resultados en el caso determinista. Observando las figuras se pueden distinguir dos etapas claramente en la trayectoria hacia el origen del plano de fase. En la primera de ellas las trayectorias deterministas y estocásticas son muy similares. Esto es porque en esta región las acciones de control están diseñadas para controlar principalmente la variable posición, que no presenta componentes estocásticas. Sin embargo, cuando el sistema debe empezar a disminuir la velocidad, la acción de control cambia y se intenta controlar principalmente la variable de velocidad. En las figuras se puede observar con claridad que en este tramo de la trayectoria aparece una diferencia entre las dos trayectorias. La trayectoria estocástica sube hacia el origen antes y a través de una trayectoria mucho más segura para evitar sobrepasamientos. Por el contrario, la trayectoria determinista se desvía de esta trayectoria alcanzando el origen de posición con un error de velocidad muy elevado, provocando un considerable sobrepasamiento. Otro aspecto a destacar de estas simulaciones es que el comportamiento del sistema de control es mucho mejor a medida que los horizontes de control aumentan. Se puede observar como en la simulación de la figura 2.9 en la que se consideraban horizontes N=4, NU=1, las prestaciones son muy inferiores a la experiencia de la figura 2.10 donde se toman los horizontes N=8, NU=4.

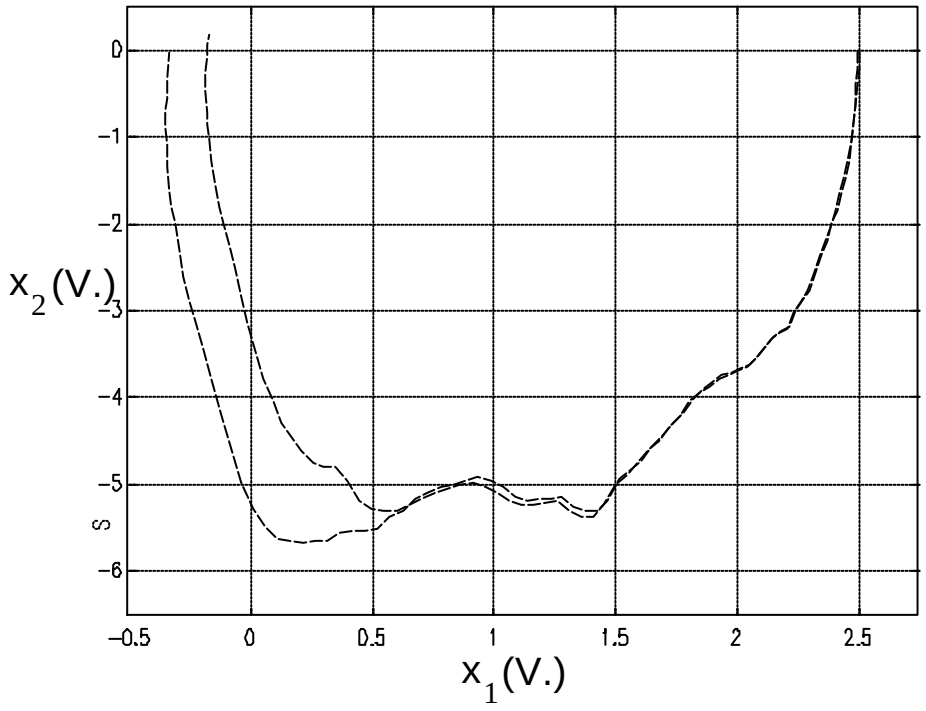


Figura 2.9. Simulación del controlador GPC. Con trazo continuo se muestra la aproximación estocástica, y con trazo discontinuo la determinista. N=4, NU=1.

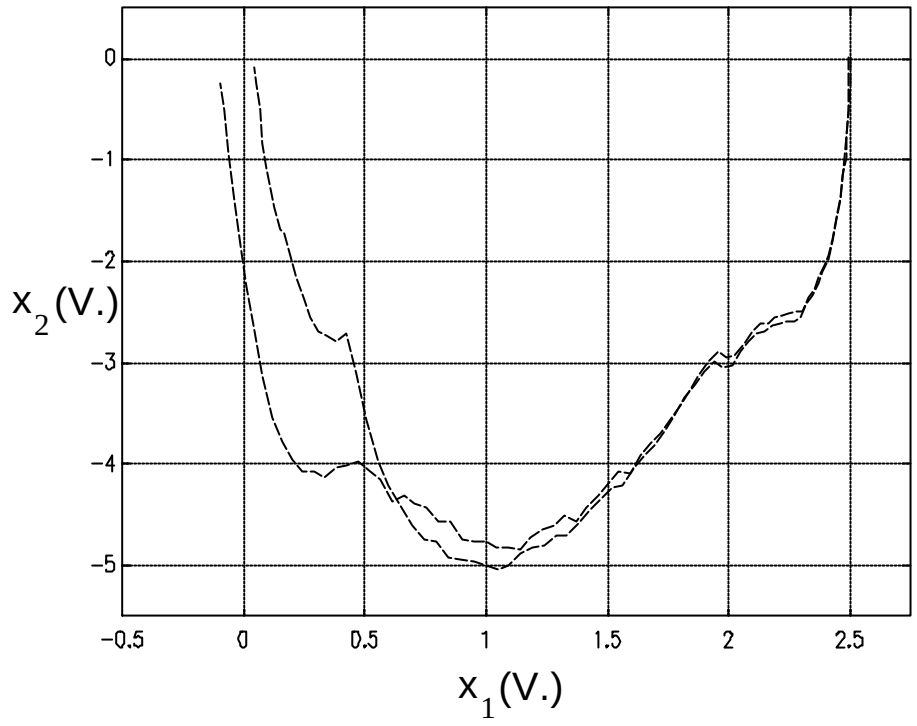


Figura 2.10. Simulación del controlador GPC. Con trazo continuo se muestra la aproximación estocástica, y con trazo discontinuo la determinista. *N*=8, *NU*=4.

Una vez estudiado el algoritmo en simulación se pasó a implementarlo en tiempo real sobre la bancada con el motor DC. En la implementación del controlador se han considerado las dos aproximaciones: estocástica y determinista. El objetivo de control es llevar al sistema hasta el interior de un área alrededor del origen del plano de fase. Una vez dentro de este área la acción de control se conmuta a una acción proporcional actuando sólo sobre la posición. Esto se hace así para garantizar un acercamiento suave hasta el origen del plano de fase, es decir, hasta el punto con error de posición cero y velocidad cero. También se conmutará a una acción proporcional si el sistema abandona el cuadrante de errores de posición positivos.

Las figuras 2.11 y 2.12 muestran la evolución del sistema bajo los planteamientos determinista y estocástico. Se puede comprobar un comportamiento similar al obtenido en simulación (figuras 2.9 y 2.10). Es decir, aparece una primera etapa en la que las dos trayectorias no presentan apenas diferencias, hasta que, al llegar al punto en que se pesan más las acciones en velocidad, la trayectoria determinista se desvía alcanzando el origen de posiciones con un elevado error en velocidad, y la trayectoria estocástica llega directamente hasta el objetivo de control. En la figura 2.13 aparece la evolución del comando correspondiente a la trayectoria estocástica de la figura 2.11.

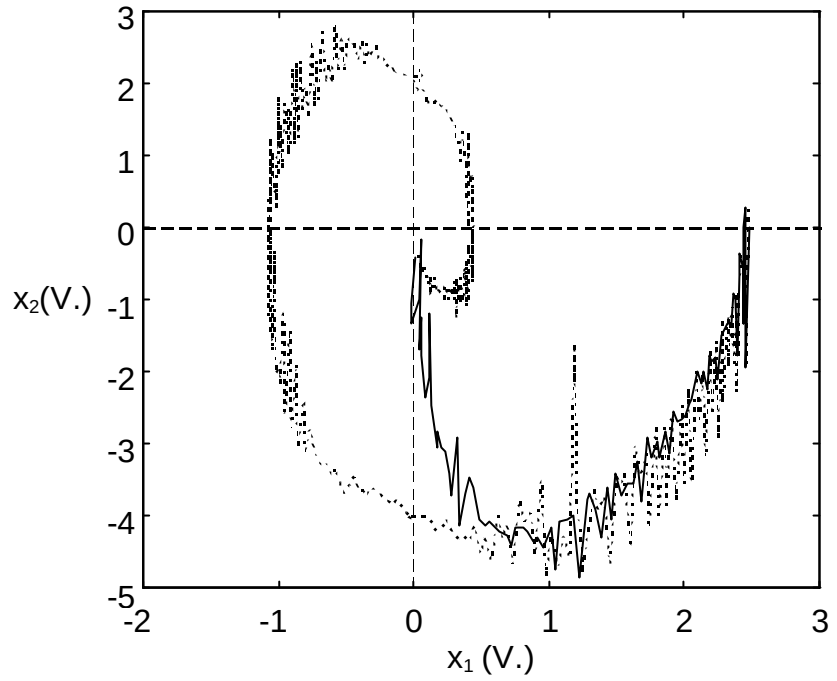


Figura 2.11. Control del motor mediante un GPC: diseño estocástico (trazo continuo), diseño determinista (trazo discontinuo).

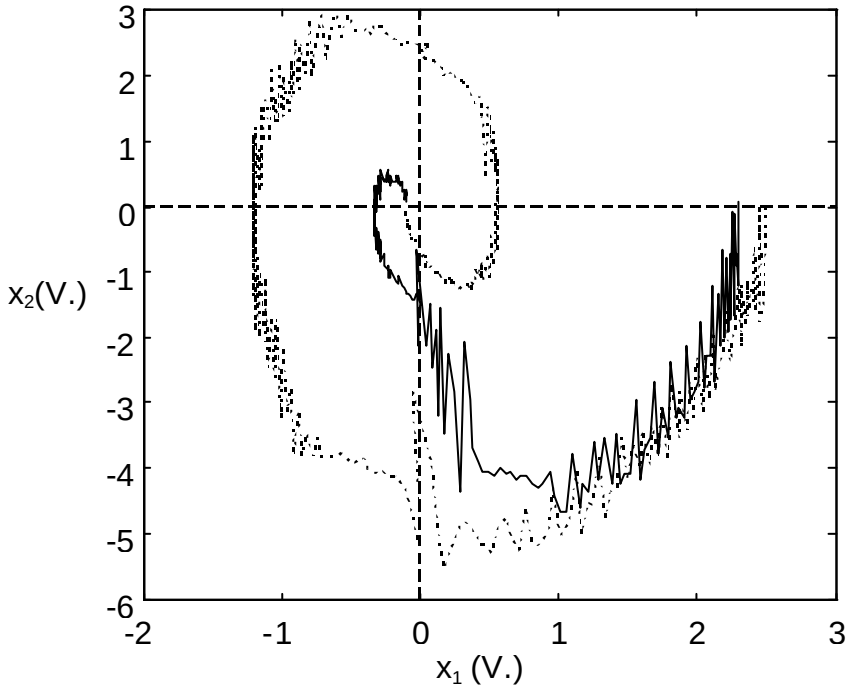


Figura 2.12. Control del motor mediante un GPC: diseño estocástico (trazo continuo), diseño determinista (trazo discontinuo).

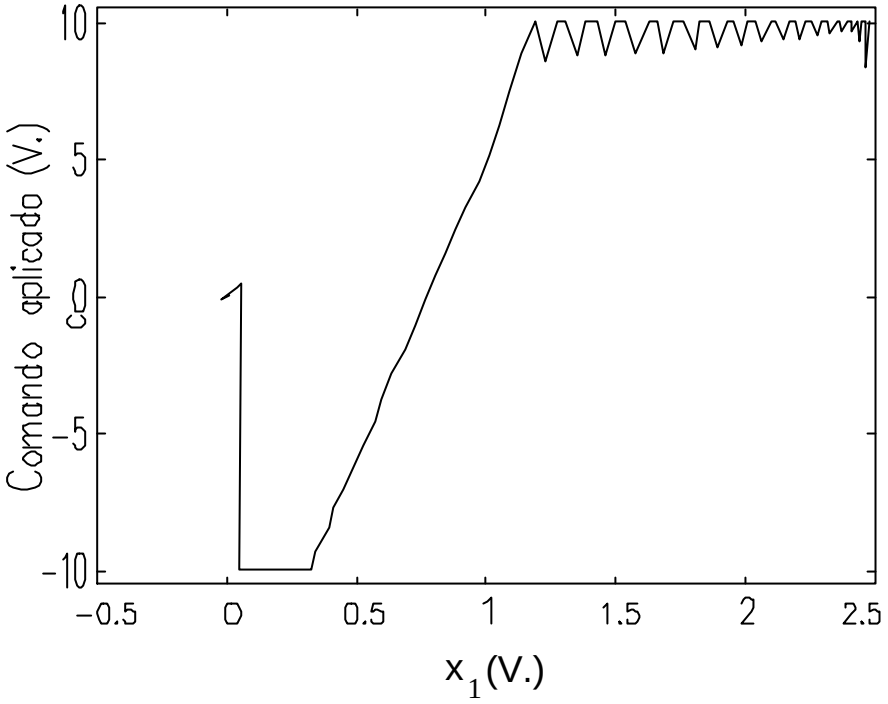


Figura 2.13. Representación del comando en función de la variable de estado de posición correspondiente al control estocástico representado en la figura 2.11.

La explicación teórica al comportamiento mostrado con las dos aproximaciones está en que cuando se considera el planteamiento estocástico, aparece en el sistema el efecto conocido como *caution* [Har85], [Aco91], [Mor95]. Este efecto se traduce en que las acciones de control que se aplican al sistema son menos enérgicas, más precavidas, para llevar al sistema por una trayectoria segura hasta el punto final. En el caso del motor, la política estocástica tiende a disminuir la derivada del error de posición en el origen de posiciones, $x_1 = 0$, asegurándose así que la zona objetivo alrededor del origen se alcance antes. Esta situación puede ser resumida como sigue: supóngase que T_σ representa la política de control óptima asumiendo que existe un ruido con varianza σ presente en el proceso.

1. - Si T_σ se aplica a un sistema que no presenta ruido, se obtienen las trayectorias que se muestran en la figura 2.14.
2. - Si T_σ se aplica a un sistema con un nivel de ruido σ_1, se pueden producir desviaciones de las trayectorias deterministas. En la figura 2.14 se ilustran las desviaciones que puede causar el ruido para los puntos A, B y C. Para el punto A el ruido puede hacer caer al sistema dentro del tercer cuadrante, esto implica un sobrepasamiento y por lo tanto un incremento de costo. Para el punto C las posibles desviaciones causadas por un nivel de ruido σ_1 son más pequeñas que las esperadas por la política de control $T_{\sigma 2}$. Por lo tanto, la trayectoria queda completamente en el cuarto cuadrante, pero con un costo mayor. Para el

punto B se alcanza el compromiso óptimo. En este caso la trayectoria está dentro del cuarto cuadrante y además con un costo bajo.

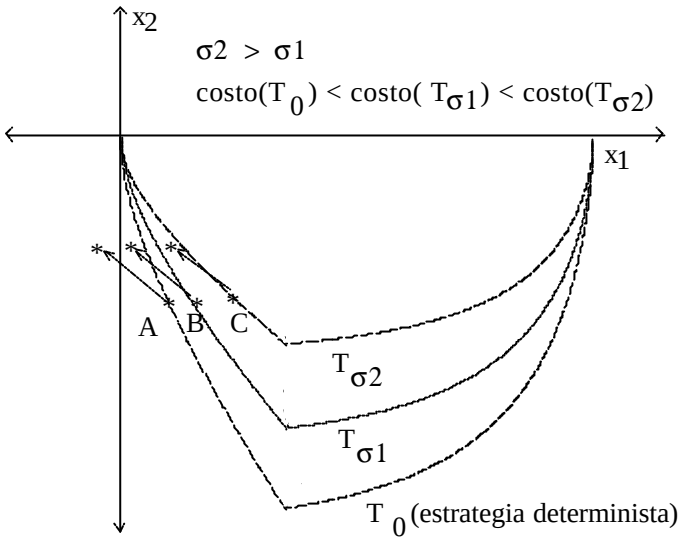


Figura 2.14. Trayectorias para las políticas de control T_0, $T_{\sigma 1}$, $T_{\sigma 2}$.

3

DISEÑO DE UN CONTROLADOR GPC COMPUTACIONALMENTE EFICIENTE MEDIANTE ESTABILIZACIÓN ROBUSTA

En este capítulo se describe el diseño y la implementación de un controlador GPC de bajo costo computacional. El objetivo planteado con el diseño de este controlador es el de reducir los horizontes de predicción, y por lo tanto la carga computacional, por medio de un lazo de estabilización robusta en la planta. En el capítulo anterior se comprobó con un experimento sobre una planta real que la implementación del GPC en plantas con constantes de tiempo pequeñas requiere el uso de un procesador relativamente potente. Con el esquema que se propone en este capítulo se intenta garantizar la aplicabilidad del GPC, incluso para plantas con constantes de tiempo pequeñas por medio de una reducción en el cálculo computacional.

El capítulo comienza exponiendo los conceptos fundamentales sobre control robusto, dando una visión general tanto desde la perspectiva del análisis como la del diseño de controladores. Con mayor profundidad se aborda el problema de control H_∞, dando una descripción detallada del problema de sensibilidad mixta. Esta técnica será utilizada en la implementación del controlador GPC con estabilización robusta que se describe en las siguientes secciones del capítulo. Para estudiar el comportamiento de este controlador, se han realizado diferentes experiencias en simulación actuando sobre plantas con distintas características. Se han comparado los resultados obtenidos con este controlador y con el GPC sin estabilizar para comprobar el rendimiento que ofrece.

3.1 INTRODUCCIÓN AL CONTROL ROBUSTO

El diseño de aplicaciones de control basadas en técnicas estándar de respuesta en frecuencia dan un resultado satisfactorio en gran parte de los casos. Aunque, en ocasiones, cuando el objetivo de diseño es complejo o se exigen especificaciones sobre márgenes de ganancia o anchos de banda para el sistema en lazo cerrado, la labor de síntesis del controlador suele ser bastante ardua. Sin embargo, en la mayoría de los casos no es necesario el empleo de otras técnicas más avanzadas. El problema aparece cuando la planta presenta una dinámica compleja e insatisfactoriamente modelada y cuando existen perturbaciones sobre ella. En estos casos las técnicas tradicionales no satisfacen los objetivos de diseño y resultan en aplicaciones con un rendimiento bastante pobre. Para afrontar este tipo de problemas surgen los métodos de control conocidos como *control robusto* [Lun88], [Mor89], [Zaf80]. Estas técnicas se basan en alcanzar el siguiente objetivo

> *"Sintetizar una ley de control que mantenga la respuesta del sistema y las señales de error dentro de unas tolerancias pre-especificadas aún en presencia de incertidumbre en el sistema y bajo las ligaduras de diseño impuestas por la tecnología*".

Con el término "incertidumbre en el sistema" se hace referencia a perturbaciones externas, incertidumbres en el modelo de la planta, y ruido de medida. La incertidumbre en el modelo suele tener su origen en linealizaciones de plantas no-lineales, procesos de deterioro de la planta debido a la antigüedad, empleo de modelos simplificados, etc. Como se observa la incertidumbre es el elemento fundamental con el que hay que operar en el diseño de un controlador robusto. Parece claro, pues la necesidad de buscar un índice que sirva de medida de esta incertidumbre. La medida cuantitativa que ha mostrado mejor versatilidad para este problema ha sido la norma H_∞. Esta norma ha dado origen a lo que se conoce como control óptimo H_∞ que ha sido la técnica que más éxito ha tenido dentro del diseño robusto.

En general, las técnicas de control robusto se pueden enfocar desde dos perspectivas. Desde el punto de vista del análisis, el objetivo es calcular el margen de estabilidad multivariable del sistema. Por el contrario, si el enfoque se hace desde un punto de vista de síntesis, el objetivo es buscar un controlador $F(s)$ que cumpla unos requisitos de prestaciones y robustez impuestos a priori. En la figura 3.1 aparece la representación con la que se trabaja en un problema de control robusto. Se distinguen tres elementos en el diagrama. Por un lado aparece la planta objeto de control, $P(s)$; esta planta está sometida a una serie de perturbaciones que se representan mediante Δ_i, (recuérdese que con estas perturbaciones se

incluyen principalmente señales de ruido y errores de modelado en la planta); por último se incorpora a este esquema un controlador $F(s)$ que realimenta a la planta. Las perturbaciones $\Delta_1, \ldots, \Delta_{n-1}$ permiten definir las especificaciones de estabilidad del sistema. Es interesante observar la presencia de la componente de *incertidumbre ficticia* Δ_n en el diagrama. Esta componente recibe su nombre porque realmente no representa ninguna perturbación, sino que se introduce para representar las especificaciones en cuanto a las prestaciones deseadas para el controlador. Con este planteamiento, el problema de control robusto es encontrar un controlador $F(s)$ de tal forma que la función de transferencia de u_1 a y_1 verifique unos requisitos de estabilidad y especificaciones.

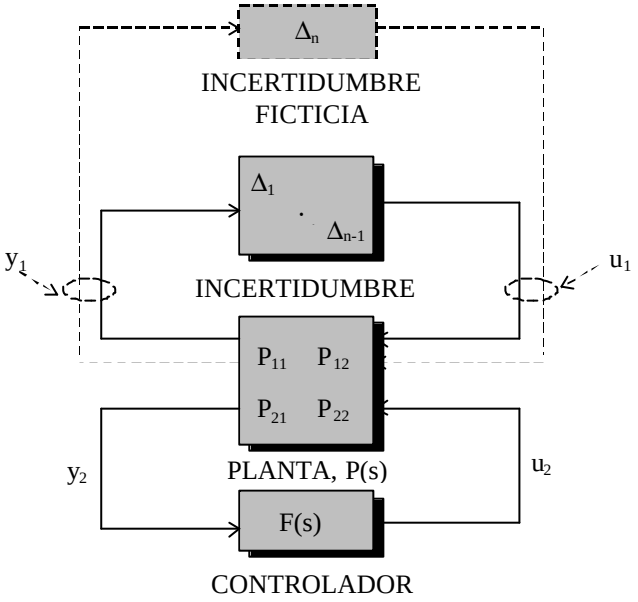


Figura 3.1. Formulación del problema de control robusto.

3.2 SISTEMAS MIMO. CONCEPTOS BÁSICOS

Considérese la representación de estado de un sistema MIMO invariante en el tiempo [Pat82]:

$$\left.\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned}\right\} \tag{3.1}$$

donde, si se considera un sistema de orden n con p entradas y m salidas, $\mathbf{x}$ tendrá dimensión $\mathbf{n}\times 1$, $\mathbf{u}$ $p\times 1$, $\mathbf{y}$ $m\times 1$, $\mathbf{A}$ $n\times n$, $\mathbf{B}$ $n\times p$, $\mathbf{C}$ $m\times n$ y $\mathbf{D}$ $m\times p$.

3.2.1 CEROS Y POLOS

En representación frecuencial podemos definir una matriz de transferencia, sin más que aplicar la Transformada de Laplace a la representación de estado:

$$\mathbf{Y}(s)= \mathbf{G}(s)\,\mathbf{U}(s);$$
$$\mathbf{G}(s)=\mathbf{C}(s\mathrm{I}-\mathbf{A})^{-1}\,\mathbf{B}+\mathbf{D};$$

siendo la dimensión de $\mathbf{G}(s)$ $m\times p$. De la misma manera que un sistema SISO, un sistema MIMO vendrá caracterizado por sus polos y sus ceros. Así, definiremos los polos como los autovalores π_i de la matriz **A**. La ecuación característica será

$$P(s)=\prod_{i=1}^{n}(s-p_i) \tag{3.2}$$

Se puede demostrar que este polinomio es el mínimo común denominador de todos los menores no nulos de cualquier orden de la matriz $\mathbf{G}(s)$. Los polos determinan la dinámica del sistema. De tal forma que, dependiendo de donde estén ubicados, el sistema será estable o no igual que en el caso SISO.

Se define z, como *cero de transmisión* de $\mathbf{G}(s)$ si el rango de $\mathbf{G}(z)$ es menor que el *rango normal*[2] de $\mathbf{G}(s)$. El polinomio que determina los ceros de transmisión es

$$Q(s)=\prod_{i=1}^{m}(s-z_i) \tag{3.3}$$

Se puede demostrar que este polinomio es el mayor común divisor de los numeradores de todos los menores de orden r, siendo r el orden normal de $\mathbf{G}(s)$. Los ceros de transmisión son una generalización multivariable de los ceros SISO clásicos. Físicamente corresponden a la situación que se produce cuando la salida de un sistema MIMO es cero, mientras los estados y las entradas del sistema no lo son.

---

[2] Una matriz polinomial $Q(s)$ tiene rango normal r si r es el orden mayor de todos los menores de $Q(s)$ tales que su determinante no es idénticamente cero. Cuando se dice que el determinante no sea idénticamente cero se quiere decir que el polinomio del determinante no sea cero, aunque pueda existir algún valor de s para el cual este polinomio se anule. Así, una matriz $Q(s)$ de dimensión $r\times r$ puede tener rango normal igual a r, y sin embargo puede ocurrir que para $s=s_1$ $\det Q(s_1)=0$.

3.2.2 VALORES SINGULARES

Considérese una matriz compleja **A** de rango *r* y dimensión *m´n*, se definen los *valores singulares*, s_i, como las raíces cuadradas no negativas de los autovalores de $\mathbf{A}^*\mathbf{A}$ ordenados tal que $s_1 > s_2 > ... > s_n$ donde $\mathbf{A}^*$ es la traspuesta conjugada de A. Si $r<n$ entonces hay $n-r$ valores singulares que son cero.

Denotaremos como $\overline{s}(\mathbf{A}) \equiv s_1$ al mayor valor singular y como $\underline{s}(\mathbf{A}) \equiv s_n$ al menor valor singular. Debido a las propiedades de los valores singulares (ver apéndice B), estos juegan un papel decisivo en el diseño robusto.

3.2.3 NORMAS H_2 Y H_∞

Considérese una matriz de transferencia estable $\mathbf{G}(s) \in C^{mxn}$, y sean $s_i(jw)$ los valores singulares de $\mathbf{G}(jw)$ dependientes de la frecuencia. Entonces se pueden definir las normas matriciales H_2 y H_∞ para una matriz de funciones como:

$$\|\mathbf{G}\|_2 \equiv \left[\int_{-\infty}^{+\infty} \sum_{i=1}^{n} (s_i(jw))^2 \, dw \right]^{1/2} \tag{3.4}$$

$$\|\mathbf{G}\|_\infty \equiv \sup_w \overline{s}(\mathbf{G}(jw)) \equiv \overline{s}(\mathbf{G}) \tag{3.5}$$

La norma $\|\mathbf{G}\|_\infty$ de un sistema estable con matriz de transferencia **G** se encontrará calculando para cada frecuencia el mayor valor singular de la matriz $\mathbf{G}(jw)$, y entonces tomando el máximo de estos valores para todas las frecuencias.

La definición de estas normas tiene como objetivo el de cuantificar los distintos elementos de un sistema MIMO. Así, la siguiente relación establece una cota para la salida de un sistema MIMO.

$$\underline{s}(\mathbf{G}(jw))\|\mathbf{u}(jw)\| \le \|\mathbf{G}(jw)\mathbf{u}(jw)\| \le \overline{s}(\mathbf{G}(jw))\|\mathbf{u}(jw)\| \tag{3.6}$$

Es decir, la ganancia (en norma 2) de un sistema MIMO queda acotada por sus valores singulares mínimo y máximo.

3.3 ANÁLISIS ROBUSTO

El objetivo del análisis robusto es la medida del margen de estabilidad multivariable. En definitiva, esto supone averiguar cuánto puede crecer Δ antes de que el sistema se haga inestable. El cálculo del margen de estabilidad multivariable exige definir previamente un modelo de incertidumbre. En la mayoría de los casos, sólo es posible obtener un modelo en términos de cotas superiores e inferiores o de distribuciones de probabilidad. Una vez definido este modelo se debe construir un diagrama M-Δ como el que se muestra en la figura 3.2.

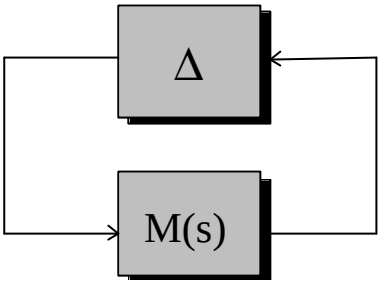


Figura 3.2. Diagrama M-Δ.

Un teorema que permite analizar la estabilidad de sistemas como el que se muestra en la figura 3.2 establece que

El sistema M-D es estable para cualquier D(s) que satisfaga

$$\overline{s}(\Delta(jw)) < \frac{1}{\overline{s}(M(jw))} \tag{3.7}$$

para todo w Î R,

donde Δ=diag(Δ_1,…, Δ_n) y $\overline{s}$ hace referencia al mayor valor singular. Igualmente este teorema se puede formular diciendo que el sistema es estable si se verifica

$$\|\Delta\|_\infty < \frac{1}{\|M\|_\infty} \tag{3.8}$$

En este marco se define el margen de estabilidad multivariable como [Saf82]:

$$K_m(M) \stackrel{def}{=} 1/\,m(M) = \inf_{\Delta}\{\overline{s}(\Delta) | \det(I - M\Delta) = 0\} \tag{3.9}$$

K_m es una función que depende de M y de la estructura de Δ y se puede interpretar como la incertidumbre más pequeña que puede desestabilizar al sistema en lazo cerrado.

3.4 DISEÑO ROBUSTO. CONTROL ÓPTIMO H_∞

En el marco del control robusto se distinguen principalmente los siguientes problemas de diseño:

- Diseño LQG robusto [Doy81].
- Diseño óptimo H_2.
- Diseño óptimo H_∞.

En la siguiente sección la atención estará centrada en el diseño óptimo H_∞.

3.4.1 CONTROL ÓPTIMO H_∞

El control óptimo H_∞ es una teoría de síntesis y optimización en el dominio de la frecuencia que fue desarrollada en respuesta a la necesidad de un procedimiento que abordara explícitamente los problemas de errores en el modelado. La idea básica de esta metodología es tratar el problema como si la planta fuera a operar en la situación más desfavorable. Las propiedades que tiene el diseño robusto H_∞ son:

- Es capaz de tratar problemas con errores de modelado de la planta y con perturbaciones desconocidas.
- Es una extensión natural de la teoría de realimentación existente, lo cual facilita trasladar conceptos intuitivos desde el punto de vista clásico al planteamiento robusto.
- Permite abordar de manera natural problemas de control multivariable.

Los requerimientos que se imponen a un controlador diseñado por estas técnicas son:

i) Rechazo al ruido: La salida del sistema en lazo cerrado no debe verse afectada por las perturbaciones dentro de unas tolerancias especificadas previamente. Aquí interviene la *Función de Sensibilidad*.
ii) Estabilidad: El sistema en lazo cerrado debe ser estable para un determinado rango de incertidumbre en el sistema en lazo abierto. Aquí interviene la *Función de Sensibilidad Complementaria*.

Normalmente los sistemas son multivariables y de alto orden, por lo que al proceso de diseño le precede uno de reducción robusto del orden de los modelos. Existen diferentes métodos de reducción de modelos: reducción del modelo mediante truncamiento balanceado, método de Schur, aproximación optima mediante la norma de Hankel, etc, [Gre95].

3.5 EL PROBLEMA DE SENSIBILIDAD MIXTA

El problema de sensibilidad mixta es un método directo y efectivo de modelar la respuesta en valores singulares del sistema, de tal modo que se logren los objetivos de estabilidad y robustez deseados. Realmente, este método es un caso especial del *Problema Canónico de Control Robusto.*

Considérese el sistema de control multivariable que aparece en la figura 3.3. La cuantificación de los márgenes de estabilidad multivariable y de prestaciones de este tipo de sistemas es posible llevarla a cabo por medio de los valores singulares de las funciones matriciales de transferencia entre las distintas señales del sistema.

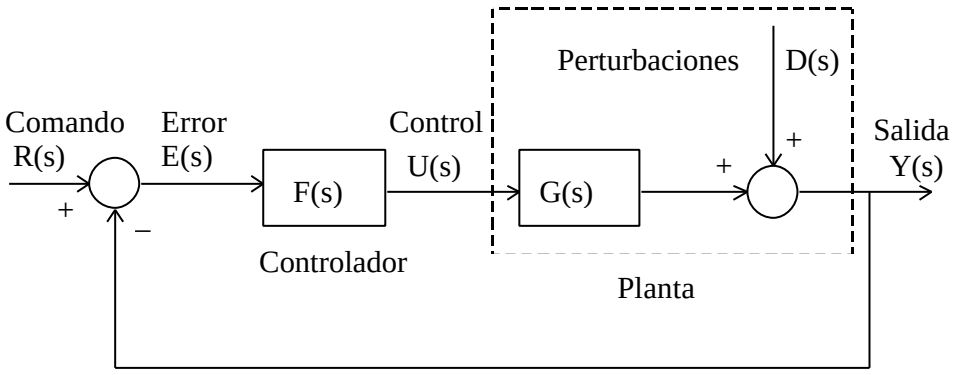


Figura 3.3. Diagrama de bloques del sistema de control multivariable.

En la figura 3.3 se distinguen los siguientes elementos:

- $G(s)$ y $F(s)$, matrices de transferencia nominales de la planta y del controlador.
- $R(s)$, $E(s)$, $U(s)$, $D(s)$,$Y(s)$ son vectores de comando, error, control, ruido y salida, respectivamente.

Aplicando técnicas del álgebra de bloques se encuentra la siguiente expresión para la salida:

$$Y(s) = D(s) + G(s)F(s)\,(R(s) - Y(s)) \tag{3.10}$$

Si definimos $L(s) = G(s)F(s)$ se tiene:

$$(I + L\,)\,Y(s) = D(s) + L\,R(s); \tag{3.11}$$

Despejando $Y(s)$,

$$Y(s) = S(s)D(s) + T(s)R(s); \tag{3.12}$$

donde

$$S(s) \equiv (\,I + L(s))^{-1} \tag{3.13}$$

que representa la matriz de transferencia entre la salida y la señal de perturbación $D(s)$. A esta función se la conoce como *Función de Sensibilidad* y

$$T(s) \equiv L(s)(\,I + L(s))^{-1} \tag{3.14}$$

representa la matriz de transferencia en lazo cerrado entre la salida y el comando. A esta función se le denomina *Función de Sensibilidad Complementaria*.

Se define además otra matriz,

$$R(s) \equiv F(s)(\,I + L(s))^{-1} \tag{3.15}$$

que es la función de transferencia entre $U(s)$ e $Y(s)$. Nótese que I-S = T.

Con estas funciones así definidas nos planteamos encontrar un controlador que verifique las condiciones i) y ii) mencionadas anteriormente. A continuación se estudian ambos problemas por separado

3.5.1 RECHAZO AL RUIDO

La formulación del problema es:

1er Problema de Minimización.- Encontrar un controlador $F(s)$ que haga al sistema en lazo cerrado estable y minimice el valor de pico de la función de sensibilidad (minimización H_∞ de la Sensibilidad).

Si el sistema es SISO la Sensibilidad es una función de transferencia escalar que para cada frecuencia nos da la ganancia frente al ruido. Si se tiene una estimación del ruido para cada frecuencia, $W_1(jw)$, entonces debe encontrarse un controlador $F(s)$ tal que produzca una función de Sensibilidad que en el peor caso atenúe de forma óptima este ruido. Esto es, se desea que el producto $W_1(jw)S(jw)$ sea pequeño en el caso más desfavorable. Esto se puede expresar formalmente como,

$$\left\|W_1(jw)S(jw)\right\|_\infty = \sup_{w\in\mathbf{R}}\left|W_1(jw)S(jw)\right| \le 1. \qquad (3.16)$$

Es decir, se trata de minimizar el efecto en la salida de la peor perturbación. Si el sistema es MIMO para la estimación del ruido $W_1(jw)$ se tomaría la norma de $D(jw)$ y $\overline{s}(S(jw))$ daría la máxima ganancia del sistema para cada frecuencia w.

El problema de diseño en este caso puede formularse como la minimización de $\left\|W_1(jw)S(jw)\right\|_\infty = \sup_{w\in\mathbf{R}}\left|W_1(jw)S(jw)\right|$, o expresándolo de otra forma, la especificación de atenuación de las perturbaciones es

$$\overline{s}(S(jw)) \le \left|W_1^{-1}(jw)\right|; \qquad (3.17)$$

donde $\left|W_1^{-1}(jw)\right|$ representa el factor de atenuación deseado. Permitiendo que $W_1(jw)$ varíe con la frecuencia se consigue especificar diferentes factores de atenuación para frecuencias distintas.

3.5.2 ESTABILIDAD FRENTE A PERTURBACIONES

2º Problema de Minimización.- Encontrar un controlador $F(s)$ que haga al sistema en lazo cerrado estable y minimice el valor de pico de la función de sensibilidad complementaria (minimización H_∞ de la Sensibilidad Complementaria).

Para el caso particular en que el sistema sea SISO, consideremos los diagramas de Nyquist para los sistemas nominal L_0 , donde $L_0 = G(s)F(s)$, y con incertidumbre, L.

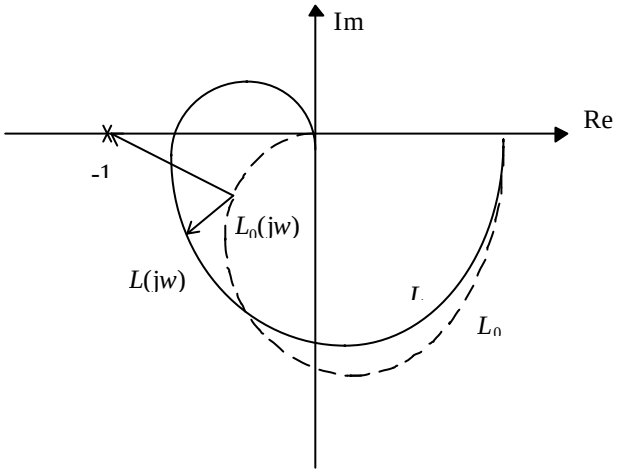


Figura 3.4.- Diagrama de Nyquist de L y L_0.

En particular, estudiamos si el sistema en lazo cerrado sigue siendo estable cuando el sistema en lazo abierto pasa de su valor nominal L_0 a un valor L como consecuencia de las perturbaciones. Por el criterio de estabilidad de Nyquist el sistema es estable si la representación de Nyquist no engloba al punto (-1,0) (suponiendo que el sistema en lazo abierto es estable). De acuerdo con la figura 3.4, el diagrama de Nyquist de L no engloba a (-1,0) si $\forall w \ |L(jw) - L_0(jw)| < |L_0(jw) + 1|$. Esto es equivalente a:

$$\frac{\left|L(jw) - L_0(jw)\right|}{\left|L_0(jw)\right|} \frac{\left|L_0(jw)\right|}{\left|L_0(jw)+1\right|} < 1, \qquad \forall w \in \mathrm{R} \tag{3.18}$$

La función de sensibilidad complementaria para el sistema nominal, T_0, es:

$$T_0 = 1 - S_0 = 1 - \frac{1}{1+L_0} = \frac{L_0}{1+L_0} \tag{3.19}$$

Expresando la condición de estabilidad para el sistema en lazo cerrado perturbado en función de T_0 se tiene

$$\frac{|L(jw) - L_0(jw)|}{|L_0(jw)|}|T_0(jw)| < 1, \qquad \forall w \in \mathrm{R} \tag{3.20}$$

El factor $|L(jw) - L_0(jw)|/|L_0(jw)|$ es el tamaño relativo de la perturbación. Considérese una cota $|W_2|$ para este tamaño relativo:

$$\frac{|L(jw) - L_0(jw)|}{|L_0(jw)|} \le |W_2(jw)| \qquad \forall w \in R \tag{3.21}$$

Entonces la condición de estabilidad se puede escribir como

$$|W_2(jw)\, T_0(jw)| < 1, \qquad \forall w \in \mathrm{R} \tag{3.22}$$

Se puede demostrar que esta condición es no sólo suficiente sino también necesaria para que el sistema en lazo cerrado sea estable para todas las perturbaciones acotadas por $W_2(jw)$.

Usando la notación de norma, la condición para la estabilidad robusta puede escribirse como

$$\|W_2 T_0\|_\infty < 1 \tag{3.23}$$

El objetivo de diseño óptimo consiste en minimizar esta norma con respecto a todos los controladores que estabilicen el sistema en lazo cerrado. Sin embargo, la estabilidad no es el único requerimiento de diseño, y puede conducir a resultados inútiles . En efecto si hacemos F=0, entonces L_0 = 0 y T_0 = 0, con lo que $\|W_2 T_0\|_\infty$ es mínimo. Esto optimiza la estabilidad del sistema pero no tiene en cuenta el buen comportamiento que debe tener la respuesta del sistema. Y, por otra parte, no minimiza la función de sensibilidad.

Considérese ahora el sistema MIMO que se muestra en la figura 3.5. Sean $\Delta_A(s)$ y $\Delta_M(s)$ perturbaciones aditivas y multiplicativas respectivamente. Entonces es posible generalizar los resultados obtenidos para los sistemas SISO sobre márgenes de estabilidad, y enunciar los siguientes teoremas:

Teorema 1.- Supóngase que el sistema de la figura 3.5 es estable si $\Delta_A(s)$ y $\Delta_M(s)$ son cero. Consideremos $\Delta_A(s) = 0$, entonces el tamaño de la perturbación más pequeña $\Delta_M(s)$ ($\Delta_M(s)$ estable) para la cual el sistema llega a ser inestable es

$$\overline{s}(\Delta_M(jw)) = \frac{1}{\overline{s}(T_0(jw))} \tag{3.24}$$

Cuanto menor sea $\overline{s}(T_0(jw))$, más grande es el tamaño de las perturbaciones multiplicativas permitidas, y por lo tanto mayor el margen de estabilidad.

Un resultado similar se obtiene para las perturbaciones aditivas:

Teorema 2.- Supóngase que el sistema de la figura 3.5 es estable si $\Delta_A(s)$ y $\Delta_M(s)$ son cero. Consideremos $\Delta_M(s) = 0$, entonces el tamaño de la perturbación más pequeña $\Delta_A(s)$ ($\Delta_A(s)$ estable) para la cual el sistema llega a ser inestable es

$$\overline{s}(\Delta_A(jw)) = \frac{1}{\overline{s}(R_0(jw))} \tag{3.25}$$

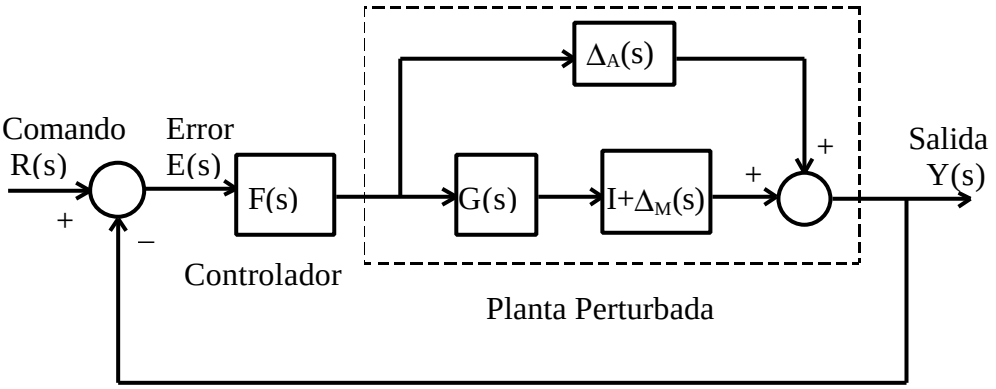


Figura 3.5. Sistema de control MIMO con perturbaciones aditivas (Δ_A) y multiplicativas (Δ_M).

Como consecuencia de los teoremas anteriores, es común especificar los márgenes de estabilidad de los sistemas de control mediante las desigualdades,

$$\overline{s}(R(jw)) \le \left|W_2^{-1}(jw)\right| \tag{3.26}$$

$$\overline{s}(T(jw)) \le \left|W_3^{-1}(jw)\right| \tag{3.27}$$

donde $|W_2(jw)|$ y $|W_3(jw)|$ son los respectivos tamaños de las perturbaciones aditivas y multiplicativas más grandes previstas.

Es posible concentrar todos los efectos de todas las incertidumbres de la planta dentro de una perturbación ficticia Δ_M, de tal forma que los requerimientos para el problema global son:

$$\overline{s}(S(jw)) \le \left|W_1^{-1}(jw)\right|; \tag{3.28}$$

$$\overline{s}(T(jw)) \le \left|W_3^{-1}(jw)\right|. \tag{3.29}$$

Generalmente, W_1 se toma con valores altos para frecuencias bajas. De esta forma se garantiza una adecuada atenuación para las perturbaciones de baja frecuencia y un seguimiento preciso de las consignas escalón. Los errores de modelado y el ancho de banda de los actuadores generalmente imponen que W_1 a frecuencias altas tome valores bajos. En cuanto a W_3, debe ser consistente con la elección de W_1, por ello normalmente toma valores altos para frecuencias altas y valores bajos para frecuencias bajas. La imposición de las condiciones (3.28) y (3.29) hace que la forma típica de S y T sea la que se muestra en las figura 3.6a) y 3.6b), respectivamente. Este factor W_3 va a determinar el ancho de banda que tendrá el sistema de control. Cuanto más se desplace la frecuencia de corte hacia frecuencias bajas, al ancho de banda conseguido será menor, pero al mismo tiempo se consigue evitar la actividad de los actuadores a alta frecuencia y las posibles saturaciones en estos. La interpretación de este hecho es que el sistema en lazo cerrado optimizado permitirá perturbaciones relativamente grandes a bajas frecuencias y sin embargo filtrará las perturbaciones causadas por ruidos de alta frecuencia.

Un aspecto que debe ser tenido en cuenta a la hora de elegir las especificaciones de diseño para W_1 y W_3 es que la frecuencia de corte con 0 dB. del diagrama de Bode de W_1, debe estar lo suficientemente por debajo de la frecuencia de corte con 0 dB. de W_3. Esto debe ser así para que los requisitos de diseño (3.28) y (3.29) sean alcanzables, ya que hay que tener en cuenta que por la propia definición de estas funciones se debe verificar $T = I - S$. Es decir, las especificaciones de diseño deben cumplir:

$$\overline{s}(W_1^{-1}(jw)) + \overline{s}(W_3^{-1}(jw)) > 1, \qquad \forall w \tag{3.30}$$

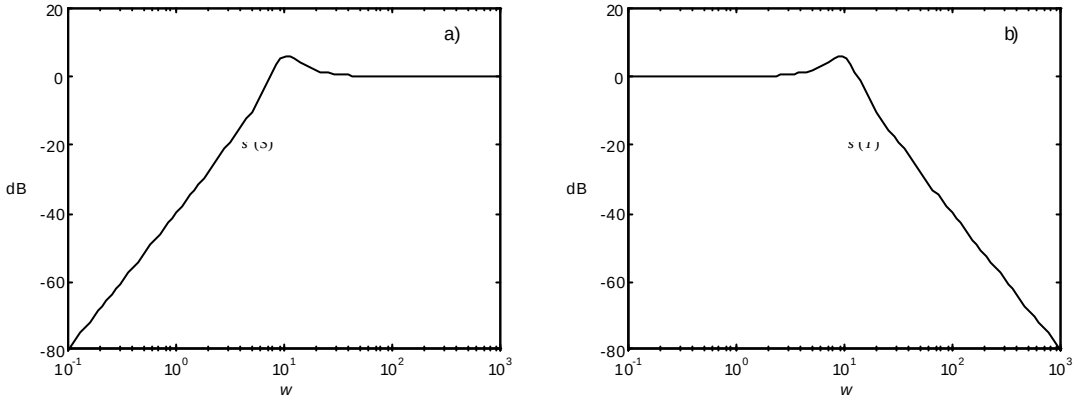


Figura 3.6. Típicas diagramas de Bode para: a) la función de sensibilidad *S*; b) la función de sensibilidad complementaria, *T*.

En la figura 3.7a) se muestra un ejemplo de diseño siguiendo esta técnica de modelado de los valores singulares de T y S. En primer lugar obsérvese como se han definido las especificaciones de diseño. W_1 presenta valores altos en baja frecuencia cayendo por debajo de cero para frecuencias altas. W_3 presenta un comportamiento inverso, toma valores altos para frecuencias altas y valores bajos para frecuencias bajas. Obsérvese que se verifica la condición (3.30) para posibilitar así la existencia de una solución al problema. El controlador que resulte del proceso de optimización debe cumplir (3.28) y (3.29). Esto es lo que se puede observar en esta figura, donde se muestra como la inversa de la cota superior de los valores singulares de S, $1/\overline{s}(S)$ (trazo sólido), están siempre por encima de la curva que define W_1; y del mismo modo, $\overline{s}(T)$ (trazo sólido) está siempre por debajo del límite que impone la especificación W_3.

Es interesante observar en la figura 3.7b) que por encima de la línea de cero dB, para frecuencias bajas, la cota inferior de los autovalores de la función de transferencia en lazo abierto, L, (representada con círculos) es igual a

$$\underline{s}\left(L(jw)\right) \approx \frac{1}{\overline{s}\,(S(jw))} \tag{3.31}$$

mientras que por debajo de la línea de cero dB, para frecuencias altas se tiene que la cota superior (representada con rectángulos) es igual a

$$\overline{s}\,(L(jw)) \approx \overline{s}\,(T(jw)) \tag{3.32}$$

Esto tiene su justificación ya que

$$S(s) \equiv (I + L(s))^{-1} \approx L(s)^{-1}, \qquad si \;\; \underline{s}(L(s)) >> 1$$
$$T(s) \equiv L(s)(I + L(s))^{-1} \approx L(s), \qquad si \;\; \overline{s}(L(s)) << 1.$$

En definitiva, lo que se consigue es modelar la respuesta en frecuencia de la función de transferencia en lazo abierto. De tal modo que permanezca acotada dentro de una zona que garantiza estabilidad y rechazo al ruido. En caso de que se penetre en la región prohibida inferior eso indicaría la pérdida de robustez en cuanto a rechazo al ruido, si se entra en la zona prohibida superior, esto implicaría la pérdida de la robustez en cuanto a estabilidad.

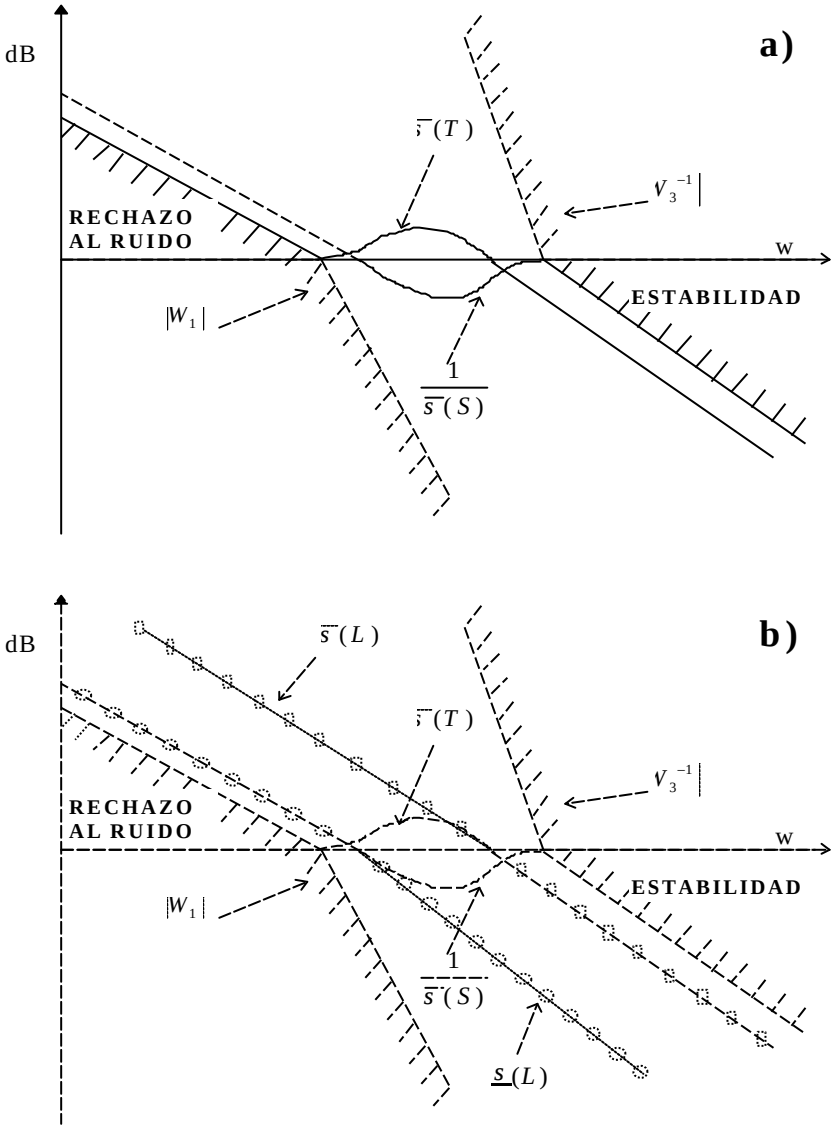


Figura 3.7 . Diagrama de valores singulares en un diseño H_∞. a) Especificaciones de diseño (W_1 y W_3) y valores singulares máximos de S y T. b) Representación de los valores singulares extremos de la función de transferencia en lazo abierto: $\overline{s}(L)$ (representado con □) y $\underline{s}(L)$ (representado con O).

3.5.3 EL PROBLEMA DE REGULACIÓN OPTIMO H¥

Como se ha comentado en la sección anterior, en el problema de la sensibilidad mixta se pretenden minimizar las funciones de sensibilidad y sensibilidad complementaria. Para ello definamos la planta aumentada *P*(s) como se indica en la figura 3.8.

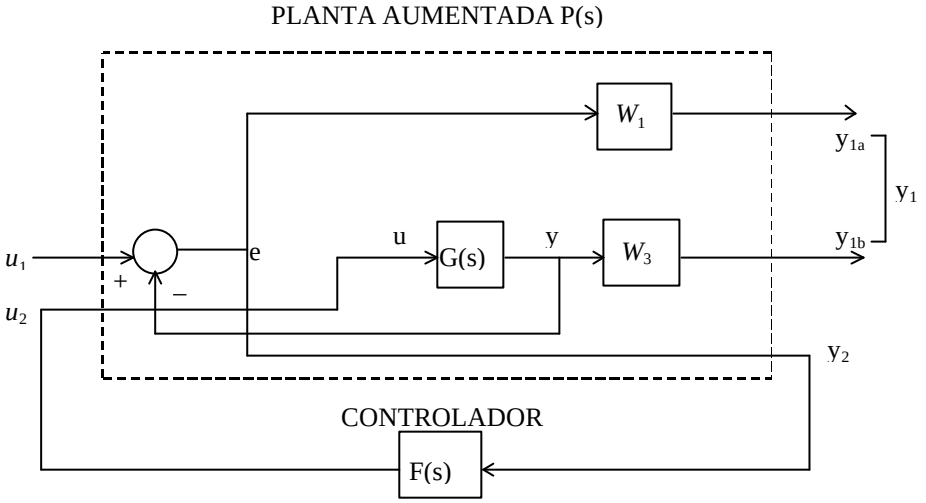


Figura 3.8. Planta aumentada en el problema de la sensibilidad mixta.

La relación entrada/salida de este sistema es:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = P(s) \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{3.33}$$

siendo la función de transferencia:

$$P(s) = \begin{bmatrix} W_1 & -W1 \\ 0 & W_3 G \\ 1 & -G \end{bmatrix} \tag{3.34}$$

Al realimentar mediante *F*(s) se encuentra:

$$y_{1a} = W_1 u_1 - W_1 G F y_2$$
$$y_{1b} = W_3 G F y_2$$

$$y_2 = u_1 - GFy_2$$

Y operando se llega a

$$y_{1a} = W_1(\mathrm{I} - L(\mathrm{I}+L)^{-1})u_1 = W_1 S u_1$$
$$y_{1b} = W_3 L(\mathrm{I}+L)^{-1}u_1 = W_3 T u_1$$
$$y_2 = (\mathrm{I}+L)^{-1}u_1$$

Así, la función de transferencia en lazo cerrado es:

$$T_{y_1 u_1} \equiv \begin{bmatrix} W_1 S \\ W_3 T \end{bmatrix} \tag{3.35}$$

Por lo tanto exigir que se cumplan las condiciones de atenuación de las perturbaciones y del margen de estabilidad (3.28) y (3.29) es equivalente a exigir $\left\| T_{y_1 u_1} \right\|_\infty \leq 1$.

Considérese la representación simplificada del sistema MIMO con el controlador $F(s)$ que se muestra en la figura 3.9. Se distinguen dos tipos de problemas según el planteamiento de diseño:

- **Control H_∞ estándar**: Dada una función matricial de transferencia $P(s)$, encontrar un controlador $F(s)$ estable tal que la función de transferencia en lazo cerrado T_{y1u1}, sea estable y su norma H_∞ sea menor o igual que 1:

  $$\left\| T_{y_1 u_1} \right\|_\infty \leq 1$$

  Este control H_∞ estándar es también conocido como *Problema de Pequeña Ganancia* (otras veces llamado problema de pequeña ganancia L_∞).

- **Control Óptimo H_∞** : Dada una función matricial de transferencia $P(s)$, encontrar un controlador $F(s)$ estable tal que la función de transferencia en lazo cerrado T_{y1u1}, sea estable y su norma H_∞ sea mínima:

  $$min \left\| T_{y_1 u_1} \right\|_\infty .$$

También existe un problema de diseño robusto llamado Control Óptimo H_2, en donde el objetivo es minimizar $\left\| T_{y_1 u_1} \right\|_2$.

.

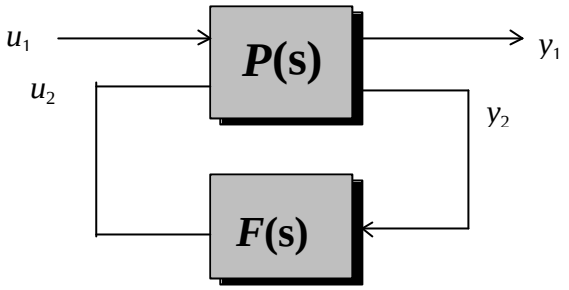


Figura 3.9. Problema de control H_∞.

3.5.4 SOLUCIÓN EN EL DOMINIO DE LA FRECUENCIA DEL PROBLEMA H¥ ÓPTIMO

Existen múltiples soluciones propuestas para resolver este problema. En la presente sección se expone una de las que presenta mayor simplicidad.

Descompóngase la planta $P(s)$ de la siguiente forma $P(s)=D^{-1}N$, donde D es una matriz cuadrada polinomial y no singular y N una matriz polinomial de la misma dimensión que P. Según (3.33) se tiene

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = P(s)\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = D^{-1}N\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{3.36}$$

lo que es equivalente a representar el sistema por el conjunto de ecuaciones diferenciales:

$$D\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = N\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{3.37}$$

Particionando las matrices polinomiales D y N en bloques de columnas con las dimensiones apropiadas se tiene:

$$D_1Y_1 + D_2Y_2 = N_1U_1 + N_2U_2 \tag{3.38}$$

Supóngase que la matriz de transferencia del controlador se puede representar en forma fraccional como $F = MX^{-1}$, siendo X y M matrices de funciones racionales estables. Por lo

tanto la función de transferencia en lazo cerrado T_{y1u1} se puede obtener como sigue: teniendo en cuenta (3.38) y que, según se muestra en la figura 3.9, $U_2 = FY_2$, se puede escribir

$$\begin{bmatrix} D_1 & D_2X - N_2M \end{bmatrix} \begin{bmatrix} Y_1 \\ X^{-1}Y_2 \end{bmatrix} = N_1U_1;$$

Multiplicando a ambos lados de esta igualdad por $\begin{bmatrix} D_1 & D_2X - N_2M \end{bmatrix}^{-1}$ se encuentra

$$\begin{bmatrix} Y_1 \\ X^{-1}Y_2 \end{bmatrix} = \begin{bmatrix} D_1 & D_2X - N_2M \end{bmatrix}^{-1} N_1U_1.$$

Con lo cual se puede despejar Y_1/U_1 para obtener la siguiente función de transferencia

$$T_{y_1u_1} = \begin{bmatrix} \mathbf{I} & 0 \end{bmatrix} \begin{bmatrix} D_1 & D_2X - N_2M \end{bmatrix}^{-1} N_1 \tag{3.39}$$

Para resolver el problema procedamos primero a determinar controladores subóptimos, esto es, aquellos que estabilizan el sistema y alcanzan $\|T_{y_1u_1}\|_\infty \le l;\ \ 0 < l < 1$. Los controladores óptimos se consiguen encontrando el valor más pequeño de λ para el cual dichos controladores existen. A continuación se expone el procedimiento para obtener estos controladores subóptimos.

Se define

$$\tilde{T}_{y_1u_1}(s) = T^T_{y_1u_1}(-s); \tag{3.40}$$

Entonces la condición $\|T_{y_1u_1}\|_\infty \le l$ es equivalente a $\tilde{T}_{y_1u_1}(s)T_{y_1u_1}(s) \le l^2\mathrm{I}$ sobre el eje imaginario. Para demostrar esto, considérese la descomposición en valores singulares de una matriz $A=U\Sigma V^*$. Entonces,

$$\mathrm{A}^*\mathrm{A} = \begin{pmatrix} s_1^2 & & \\ & \mathrm{O} & \\ & & s_n^2 \end{pmatrix}$$

y como $\|\mathrm{A}\|_\infty = \bar{s}(\mathrm{A})$, se verifica $\mathrm{A}^*\mathrm{A} \le l^2\mathrm{I} \ \Rightarrow\ \bar{s}(\mathrm{A}) \le l$.

Si se define la matriz racional,

$$\Pi_l = \begin{bmatrix} \tilde{D}_2 \\ -\tilde{N}_2 \end{bmatrix} (N_1\tilde{N}_1 - l^2 D_1\tilde{D}_1)^{-1} \begin{bmatrix} D_2 & -N_2 \end{bmatrix} \tag{3.41}$$

entonces, sin más que sustituir se ve que la desigualdad $\tilde{T}_{y_1u_1}(jw)T_{y_1u_1}(jw) \leq l^2\mathrm{I}$ es equivalente a la desigualdad:

$$\begin{bmatrix} \tilde{X} & \tilde{M} \end{bmatrix} \Pi_l \begin{bmatrix} X \\ M \end{bmatrix} \geq 0 \tag{3.42}$$

sobre el eje imaginario. La matriz Π_λ verifica que $\tilde{\Pi}_l = \Pi_l$. Si det(Π_λ) no tiene polos y ceros en el eje imaginario, Π_λ puede ser factorizada de forma *J*-espectral como:

$$\Pi_l = \tilde{Z}_l J Z_l \tag{3.43}$$

donde Z_λ es una matriz cuadrada racional, tal que ella y su inversa tienen todos sus polos en la mitad izquierda del plano y *J* es de la forma:

$$J = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \tag{3.44}$$

con los dos bloques de matrices unidad con las dimensiones apropiadas. *J* se denomina la signatura de Π_λ. Por lo tanto podemos escribir:

$$\begin{bmatrix} \tilde{X} & \tilde{M} \end{bmatrix} \tilde{Z}_l J Z_l \begin{bmatrix} X \\ M \end{bmatrix} \geq 0 \tag{3.45}$$

sobre el eje imaginario. Definiendo las matrices cuadradas estables y racionales *R* y *Q* como

$$\begin{bmatrix} R \\ Q \end{bmatrix} = Z_l \begin{bmatrix} X \\ M \end{bmatrix}, \tag{3.46}$$

se encuentra que la condición se transforma en

$$\tilde{R}(jw)R(jw) \geq \tilde{Q}(jw)Q(jw) \tag{3.47}$$

Invirtiendo Z_λ se determina la expresión del controlador:

$$\begin{bmatrix} X \\ M \end{bmatrix} = Z_l^{-1} \begin{bmatrix} R \\ Q \end{bmatrix} \tag{3.48}$$

Esta expresión, y la relación (3.47), determinan explícitamente la fórmula de todos los compensadores $F= MX^{-1}$ cumpliendo $\left\| T_{y_1 u_1} \right\|_\infty \leq 1$.

Puede demostrarse que para que el controlador estabilice al sistema es necesario además que det(R) tenga sus raíces en la mitad izquierda del plano. Hay muchas matrices que satisfacen la condición. Una elección obvia *e*s R =I, Q=0. A esta solución se la conoce como la *solución central*.

El procedimiento de diseño sería:

1. Elegir un valor de λ.
2. Determinar Z_λ y calcular un controlador mediante (3.48) tal que se satisfaga (3.47) y el det(R) tenga todos sus ceros en la mitad izquierda del plano. Una posibilidad es utilizar la solución central.
3. Verificar si el controlador estabiliza al sistema en lazo cerrado. Si lo hace, disminuir λ. Si no lo hace aumentarla.
4. Si la solución es suficientemente buena se detiene el proceso, si no lo es, se vuelve al paso 1.

No obstante, presenta muchos problemas de tipo numérico y de convergencia que aquí no tratamos. Este procedimiento está disponible en la *Toolbox* de Control Robusto integrada en el paquete MATLAB [Chi92], [Mat92], [Sim92]. Las funciones más importantes de esta *toolbox* se describen en el apéndice D.

3.6 ESTABILIZACIÓN ROBUSTA DEL GPC

El controlador GPC goza de una enorme popularidad entre la comunidad de ingenieros de control debido principalmente al éxito obtenido en las distintas aplicaciones. Sin embargo, uno de sus principales puntos débiles ha sido la estabilidad en lazo cerrado. A pesar de los avances realizados en este sentido [Cla91], [Rob89], [Kou92], [Ros93], [Zhe95], [Ros96] , aun no ha sido desarrollada completamente una teoría formal sobre la estabilidad en el controlador predictivo. Lo que sí parece claro es que, cuanto mayor sean los horizontes de control y salida, mejores serán los resultados en cuanto a estabilidad.

Sin embargo, se debe tener en cuenta que a medida que los horizontes de predicción crecen, la complejidad computacional se incrementa. Esto es muy importante puesto que, aparte de la estabilidad, el otro inconveniente que pueden presentar los algoritmos GPC en una implementación práctica es la carga computacional. Teniendo en cuenta la potencia de cálculo de muchos de los procesadores actuales este hecho pudiera carecer de importancia, pero cuando se trata con un sistema de constante de tiempo pequeña o cuando se trata de implementar una estrategia adaptativa, la carga computacional del sistema puede limitar notablemente el ancho de banda obtenido. Por lo tanto, la aplicabilidad en tiempo real del GPC parece sólo asegurada para procesos lentos.

El principal objetivo perseguido en este trabajo es el de buscar una estructura de control alternativa basada en el GPC con un bajo costo computacional en su implementación. Para lograr esto, se ha diseñado una estrategia que estabilice el sistema antes de aplicar el GPC. De esta forma se lograrán horizontes de predicción más pequeños, y por lo tanto, menores tiempos de computación. La estabilización de la planta se ha llevado a cabo mediante la inclusión de un bucle interno de estabilización robusta en el sistema. El controlador GPC actuará sobre esta planta estabilizada.

Para conseguir la estabilización de la planta se utilizarán las técnicas de sensibilidad mixta descritas en las secciones anteriores. Considérese un sistema $G(s)$ afectado por un ruido aleatorio $V(s)$, y con una incertidumbre en el modelo definida por la función de transferencia, $W(s)$. Supóngase que se realimenta esta planta tal y como se propone en la figura 3.10.

Donde $F(s)$ es la función de transferencia del controlador y $R(s)$, $E(s)$, $U(s)$ e $Y(s)$ son la señal de referencia, el error, el comando y la salida del sistema.

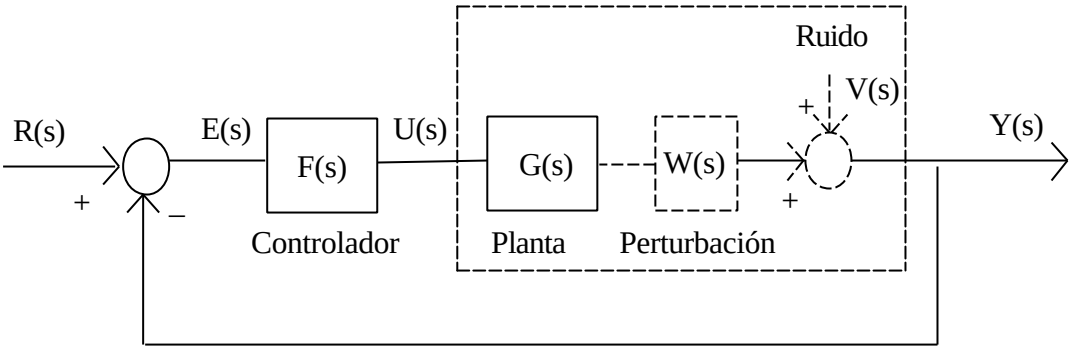


Figura 3.10. Estructura del estabilizador robusto.

Como ya es conocido, el problema de control H_∞ se formula en términos de la función de sensibilidad y sensibilidad complementaria. Recuérdese que la función de sensibilidad está definida como la función de transferencia entre la salida del sistema y la señal de ruido, es decir, como

$$S = (I + G(s)F(s))^{-1}$$

Por su parte la función de sensibilidad complementaria se define como la función de transferencia entre la entrada y la salida del sistema:

$$T = F(s)G(s)\big(I + G(s)F(s)\big)^{-1}$$

En estos términos el problema de control H_∞ puede resumirse como sigue: encontrar un controlador $F(s)$ que estabilice el sistema y que

1. Minimice el valor de pico de la función de sensibilidad. Es decir, $F(s)$ debe elegirse de forma que minimice el efecto de las perturbaciones en la salida del sistema.

2. Minimice el valor de pico de la función de sensibilidad complementaria. De esta forma se consigue que las incertidumbres en la planta no afecten a la estrategia de control y a la respuesta del sistema.

La solución a este problema se obtiene empleando el algoritmo comentado en el capítulo anterior.

3.7 ESTRUCTURA DEL GPC CON ESTABILIZACIÓN ROBUSTA

El objetivo perseguido en esta sección es el de incorporar propiedades de robustez al esquema básico del controlador GPC. Como se ha expuesto en el capítulo anterior, el número de operaciones *off-line* del algoritmo depende del horizonte de salida N y del horizonte de predicción NU. Sin embargo, el número de operaciones *on-line* sólo depende del horizonte de salida N. Por ello, es especialmente importante que el horizonte de salida en el controlador sea lo más pequeño posible, de forma que permita obtener las especificaciones de funcionamiento y que, al mismo tiempo no conlleve una carga computacional excesivamente grande.

Con el objetivo de disminuir esta carga computacional se plantea el esquema de realimentación que aparece en la figura 3.11.

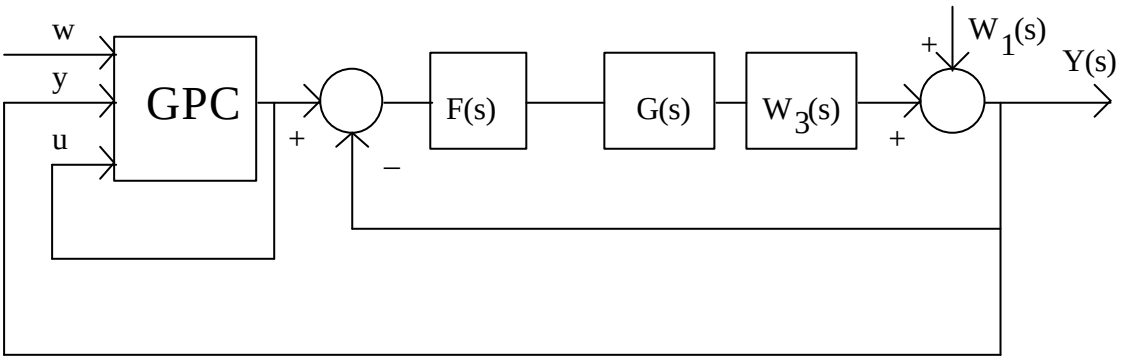


Figura 3.11. Control GPC con estabilización robusta.

Como se muestra en la figura, se incluye un bucle interno de estabilización robusta, de forma que el GPC se aplica sobre la planta ya estabilizada. En la figura, $F(s)$ es el controlador que estabiliza la planta, y que además minimiza la función de sensibilidad y sensibilidad complementaria. W_1 y W_3 representan, respectivamente, el máximo nivel de ruido estimado para la planta, y la mayor perturbación multiplicativa prevista para la planta.

La estrategia de control se puede resumir de la siguiente forma:

- *Definir las especificaciones de diseño W_1 y W_3.*
- *Encontrar el controlador que estabiliza la planta y minimiza la función de sensibilidad y la función de sensibilidad complementaria.*

- *Obtener el modelo CARIMA correspondiente a la planta estabilizada*.

- *Aplicar el controlador GPC sobre esta planta.*

Las especificaciones de diseño quedan definidas mediante las funciones W_1 y W_3 , ya que estas dan cuenta, respectivamente, del rechazo al ruido deseado y de la robustez del sistema ante perturbaciones en la planta (ver figura 3.7). En las simulaciones llevadas a cabo se ha considerado la siguiente función de transferencia para W_1 :

$$W_1^{-1} = g^{-1}\frac{(100+s/100)(1+s/100)}{100(1+s/5000)^2}$$

donde γ es un factor constante que regula el nivel requerido de rechazo al ruido. En las simulaciones se ha considerado γ=1.5. En la figura 3.12 se representa esta función de transferencia. Como el problema planteado básicamente hace referencia a la estabilidad, las restricciones impuestas en relación al rechazo al ruido han sido despreciables. De ahí que en la figura 3.12, y para frecuencias bajas, la función W_1^{-1} esté prácticamente en cero dB con lo cual apenas hay rechazo al ruido. Para frecuencias altas, la función W_1^{-1} se comporta como un filtro pasa alta.

Para las perturbaciones multiplicativas en la planta, se ha considerado la siguiente función de transferencia:

$$W_3^{-1} = 2000/s$$

De esta forma se garantiza un ancho de banda de control de aproximadamente 300 Hz. Como se observa en la figura por arriba de 300 Hz. la caída de la función W_3^{-1} es de 20 dB/década.

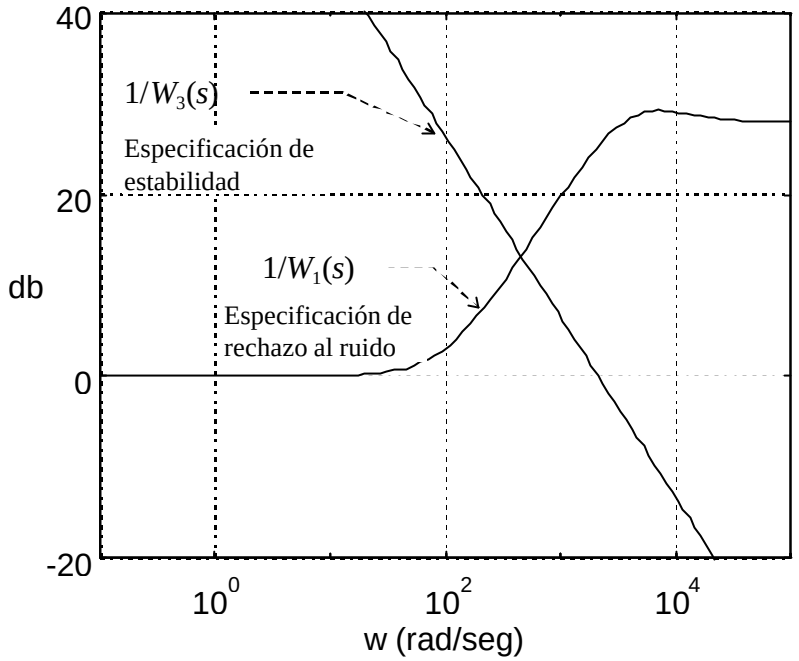


Figura 3.12. Especificaciones de diseño.

3.8 RESULTADOS

Con esta estructura de control se han llevado a cabo diferentes simulaciones sobre plantas con distintas características. En la tabla 3.1 aparecen recogidas las funciones de transferencia de las distintas plantas sobre las que se implementó el controlador GPC con estabilización robusta. Nótese que las plantas 1 y 2 son estables y que la planta 3 es inestable en lazo abierto.

Planta	Función de transferencia	Característica
1	$G(s) = \frac{s^2 + 5.05s + 4.2}{s^3 + 5.3s^2 + 7.5s + 1.8}$	Estable
2	$G(s) = \frac{s^3 + 71s^2 + 170.3s + 7017.5}{s^4 + 30s^3 + 525s^2 + 4500s + 25000}$	Estable
3	$G(s) = \frac{s^2 + 3s + 2}{s^3 + 3.3s^2 + 0.8s - 0.3}$	Inestable

Tabla 3.1. Funciones de transferencia de las plantas estudiadas en simulación.

En cuanto a la implementación práctica es importante tener en cuenta las restricciones en los comandos. Para conseguir que se respete esta ligadura se debe plantear un esquema GPC incorporando las ecuaciones de ligadura que se resuelven empleando programación cuadrática. En la simulación llevada a cabo en este trabajo no se han incorporado las ecuaciones de ligadura directamente en el diseño, sino que para acotar el valor de los comandos se ha utilizado el parámetro λ, que interviene en la función de costo y que pesa los comandos aplicados.

Las simulaciones implementadas han sido desarrolladas en MATLAB [Mat92]. Se ha hecho uso de la *toolbox* de control robusto para obtener el controlador $F(s)$ de la planta [Chi92]. A continuación se resumen los resultados obtenidos con las plantas que aparecen en la tabla 3.1. En todos los casos se han comparado los resultados de la simulación llevada a cabo con el esquema propuesto y considerando el controlador GPC actuando directamente.

3.8.1 PLANTA 1

Sobre esta planta se intentó inicialmente la implementación del GPC sin bucle de estabilización. Esto se consiguió pero obteniendo una respuesta con un transitorio con características de amortiguamiento bastante malas. En la figura 3.13 se observa la respuesta obtenida con horizonte de predicción, N, igual a 10 y horizonte de control, NU, igual a 4. Como se aprecia existen unas oscilaciones a lo largo de toda la trayectoria y un sobrepasamiento que ponen de manifiesto una baja eficiencia de esta estrategia. Para mejorar el comportamiento del sistema en lazo cerrado, una de las opciones es incrementar los horizontes de predicción, pero según se ha visto esto supondría un incremento notable en los

cálculos necesarios. Obsérvese en la figura 3.14, que si se bajan los horizontes de predicción el comportamiento del sistema, como es de esperar es aún más negativo.

Posteriormente se aplicó a esta misma planta el controlador GPC robusto. El controlador obtenido con las especificaciones impuestas fue:

$$F(s) = \frac{10^6 s^4 + 1.06*10^{10} s^3 + 5.63*10^{10} s^2 + 7.97*10^{10} s + 1.91*10^{10}}{s^5 + 1.16*10^4 s^4 + 1.62*10^7 s^3 + 1.58*10^9 s^2 + 7.66*10^9 s + 6.31*10^9}$$

Los resultados obtenidos ahora fueron completamente diferentes a los del GPC sin estabilización. Así, obsérvese la figura 3.15, donde incluso para horizontes de predicción N = 1, NU =1, la respuesta del sistema es muy satisfactoria. Como se puede ver en este caso ya no existen oscilaciones indeseadas y el sistema alcanza la consigna suavemente y con un tiempo de establecimiento mucho menor. Si en este caso se aumentan los horizontes de predicción hasta N=10 y NU=4, se obtiene la respuesta que se presenta en la figura 3.16. Se puede observar que este incremento en los horizontes de predicción no afecta prácticamente en nada a la respuesta del sistema, lo que indica que ya con N=1 y NU=1 se obtiene una respuesta óptima en el sistema. En términos de computación esto indicaría que con simples cálculos con escalares se obtiene mejor comportamiento con esta estrategia que aplicando el controlador GPC directamente sobre la planta.

3.8.2 PLANTA 2

Cuando se intentó implementar el controlador GPC sobre la planta 2 con diferentes horizontes de predicción, no fue posible observar una respuesta estable en ninguno de los casos. En la figura 3.17 se muestra una respuesta típica de la planta 2 con el GPC. Incluso en este caso en que se han tomado horizontes de predicción y control bastante elevados (N = 20, NU=10), el comportamiento del sistema sigue siendo inestable.

Sin embargo, cuando a esta planta se le aplicó el controlador GPC robusto los resultados fueron totalmente diferentes. La función de transferencia del controlador $F(s)$ obtenido fue

$$F(s) = \frac{10^6 s^5 + 1.06*10^{10} s^4 + 3.19*10^{11} s^3 + 5.58*10^{12} s^2 + 4.78*10^{13} s + 2.65*10^{14}}{s^6 + 1.17*10^4 s^5 + 1.70*10^7 s^4 + 2.65*10^9 s^3 + 1.09*10^{11} s^2 + 3.69*10^{11} s + 1.05*10^{13}}$$

En la figura 3.18 se muestra la respuesta del sistema después de incluir el controlador robusto en la estructura de control. En este caso se han tomado los horizontes N=1, NU=1. A pesar de utilizar estos horizontes tan reducidos, los resultados obtenidos son muy positivos, con una evolución suave de la salida hasta la consigna y sin problemas de estabilidad. De hecho, si se aumentan los horizontes de predicción hasta N=10, NU=4 no se observan mejorías notables en la salida del sistema (ver figura 3.19). Lo cual indica que con horizontes de N=1, NU=1 se está alcanzando una evolución muy satisfactoria en el sistema. Y esto supone una notable disminución en la complejidad computacional del algoritmo.

3.8.3 PLANTA 3

Por último, se muestran los resultado obtenidos con una planta que presenta un polo inestable en lazo abierto situado en s_1= 0.2. Para esta planta el controlador GPC presenta una evolución inestable independientemente del horizonte elegido. En la figura 3.20 aparece la evolución que presenta el sistema cuando se toma N=20 y NU=10. Como se puede observar, la salida del sistema en este caso se vuelve inestable. Por el contrario, cuando al sistema se le aplica la estabilización robusta, la respuesta del sistema en lazo cerrado con el controlador GPC es muy satisfactoria, mostrando un comportamiento excelente incluso para horizontes tan bajos como N=1, NU=1. En la figura 3.21 se muestra una de estas simulaciones. En este caso el controlador $F(s)$ aplicado es

$$F(s) = \frac{10^6 s^4 + 1.06 * 10^{10} s^3 + 3.99 * 10^{10} s^2 + 2.55 * 10^{10} s + 4.09 * 10^9}{s^5 + 1.16 * 10^4 s^4 + 1.62 * 10^7 s^3 + 1.55 * 10^9 s^2 + 4.54 * 10^9 s + 3 * 10^9}$$

Al aumentar estos horizontes no se observan cambios notables en la evolución del sistema (ver figura 3.22). Lo cual indica que se puede obtener una respuesta de altas prestaciones en el sistema incluso con horizontes bajos de predicción.

Como conclusión de las simulaciones llevadas a cabo hay que destacar que empleando el esquema combinado de estabilización robusta con el controlador GPC es posible obtener una ley de control que garantiza un comportamiento del sistema en lazo cerrado mucho más estable que el obtenido empleando una acción GPC únicamente. Este hecho hace que los horizontes de predicción necesarios para conseguir un funcionamiento adecuado del sistema sean menores. Y, por lo tanto, existe una disminución en el número de operaciones necesarias en el algoritmo, llegando en algunos casos a convertirse en simples cálculos escalares. Por ello, es posible asegurar la aplicabilidad del GPC desde el punto de vista computacional en una implementación en tiempo real.

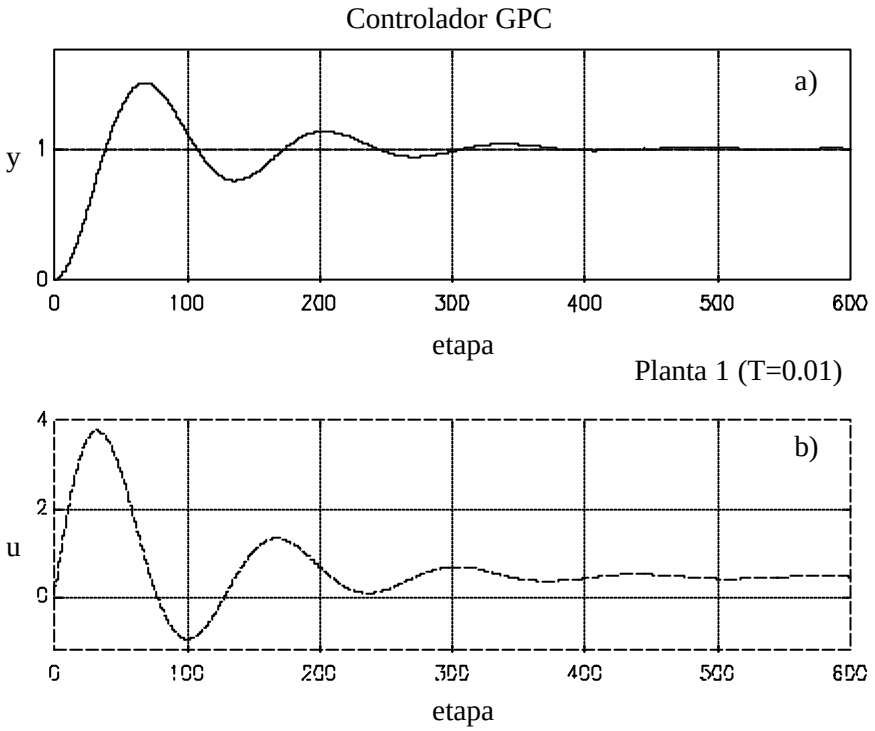


Figura 3.13. Controlador GPC aplicado sobre la planta 1 tomando como consigna un escalón unitario y con N =10, NU=4. a) Evolución de la salida de la planta. b) Comando aplicado.

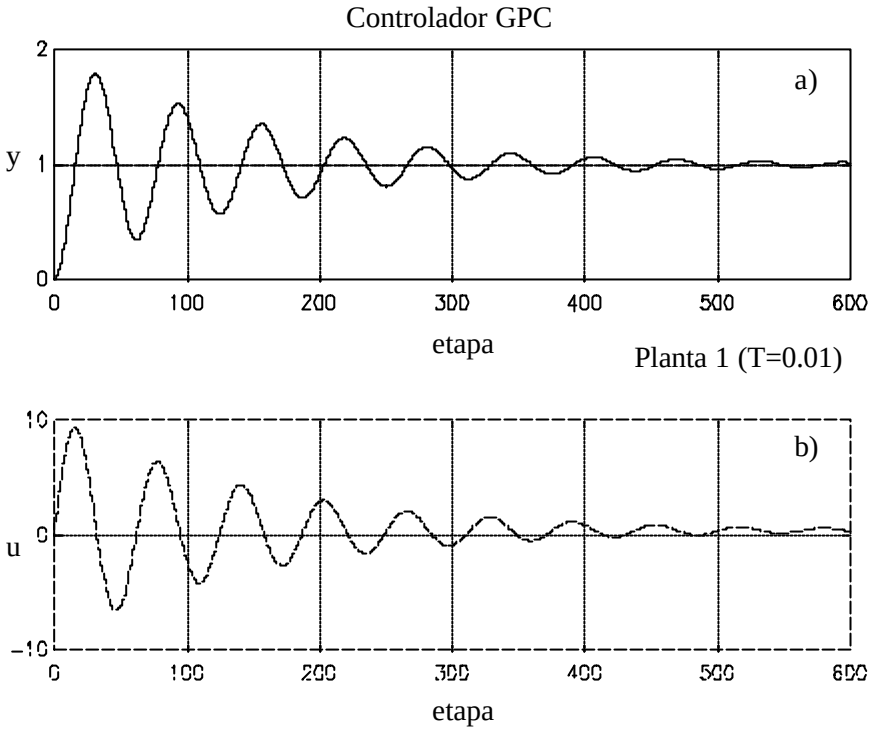


Figura 3.14. Controlador GPC aplicado sobre la planta 1 tomando como consigna un escalón unitario y con $N = 1$, $NU = 1$. a) Evolución de la salida de la planta. b) Comando aplicado.

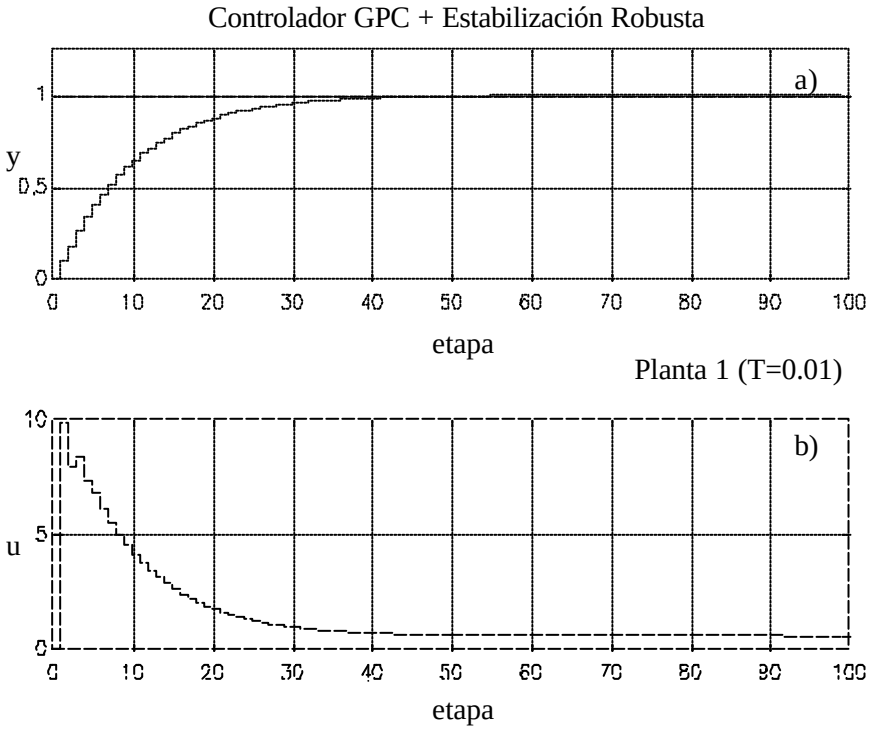


Figura 3.15. Controlador GPC con estabilización robusta aplicado sobre la planta 1 tomando como consigna un escalón unitario y con $N = 1$, $NU = 1$. a) Evolución de la salida de la planta. b) Comando aplicado.

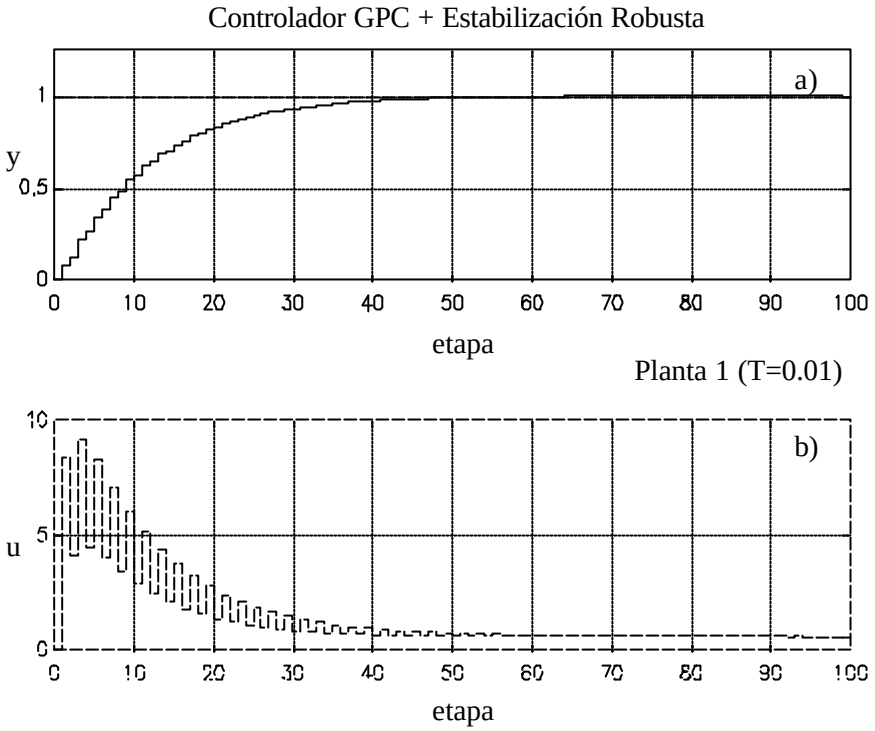


Figura 3.16. Controlador GPC con estabilización robusta aplicado sobre la planta 1 tomando $N = 10$, $NU = 4$. a) Evolución de la salida de la planta. b) Comando aplicado.

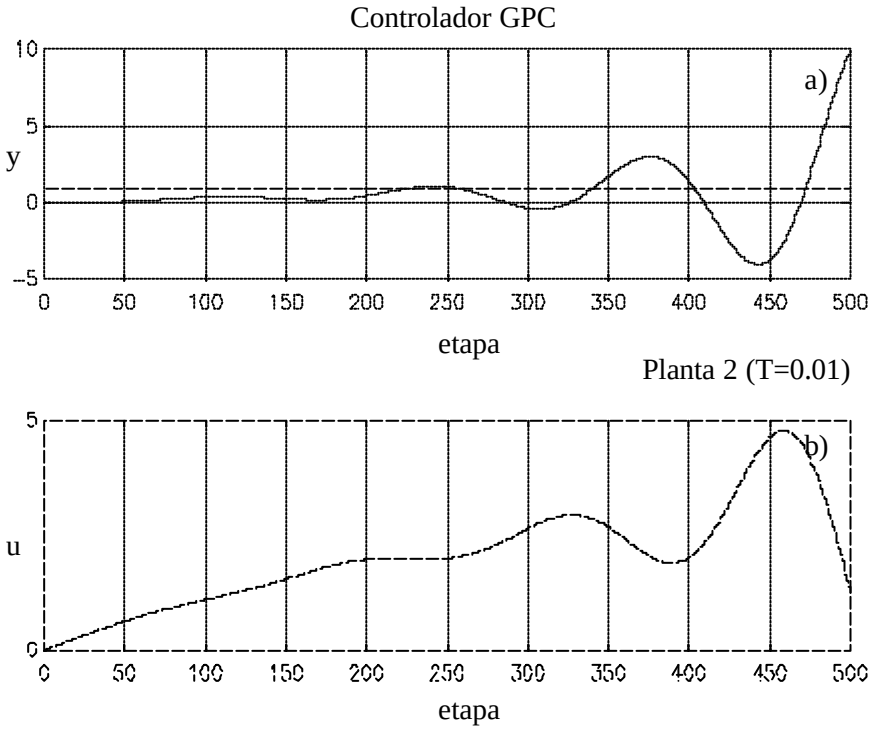


Figura 3.17. Controlador GPC aplicado sobre la planta 2 tomando como consigna un escalón unitario y con $N = 20$, $NU = 10$. a) Evolución de la salida de la planta. b) Comando aplicado.

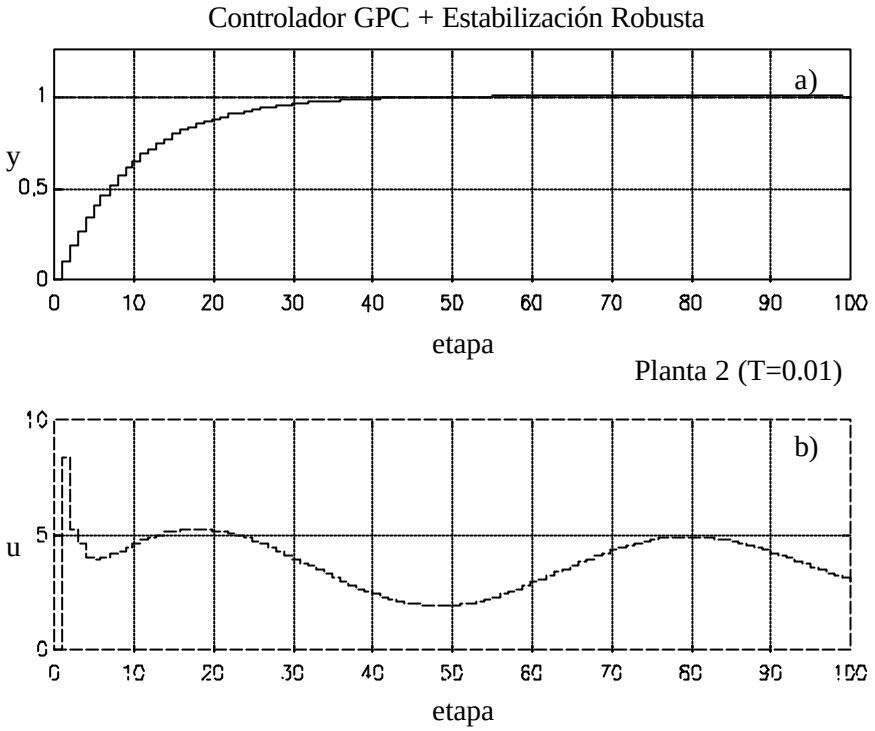


Figura 3.18. Controlador GPC con estabilización robusta aplicado sobre la planta 2 tomando como consigna un escalón unitario y con $N = 1$, $NU = 1$. a) Evolución de la salida de la planta. b) Comando aplicado.

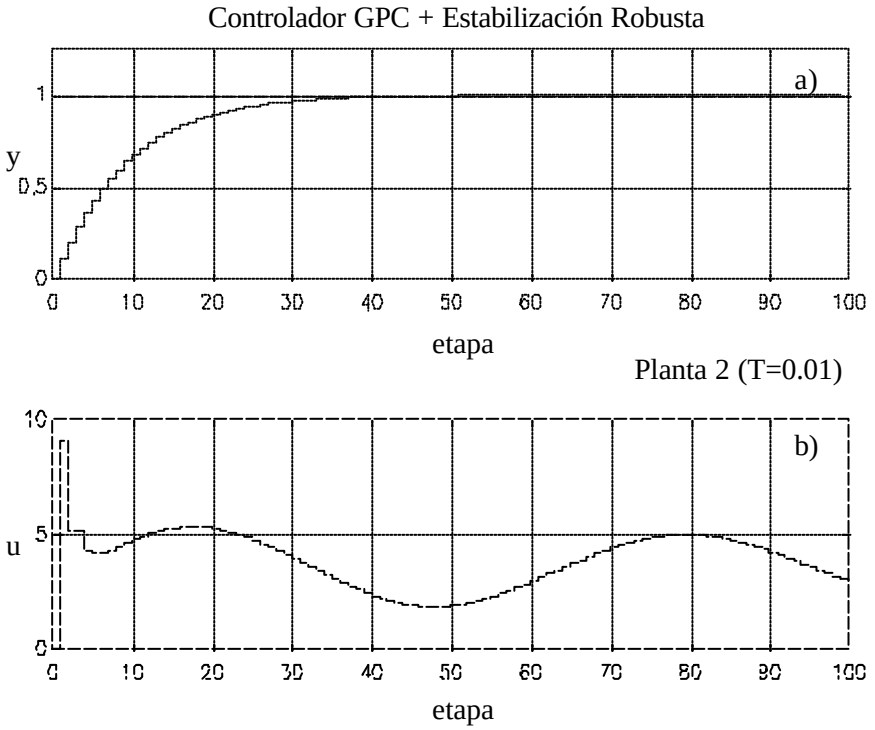


Figura 3.19. Controlador GPC con estabilización robusta aplicado sobre la planta 2 tomando como consigna un escalón unitario y con $N = 10$, $NU = 4$. a) Evolución de la salida de la planta. b) Comando aplicado.

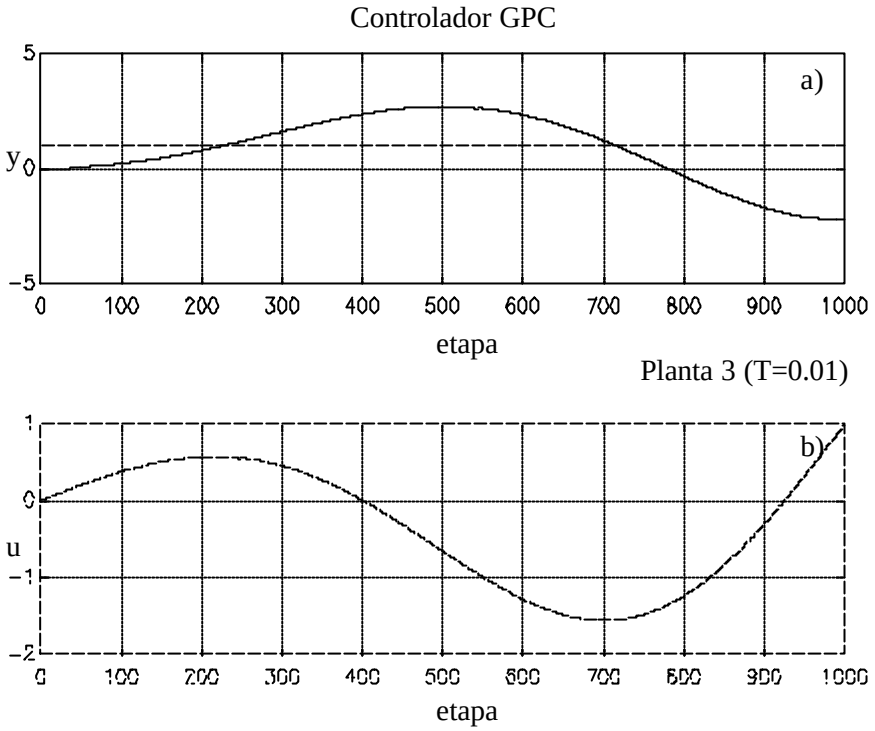


Figura 3.20. Controlador GPC aplicado sobre la planta 3 tomando como consigna un escalón unitario y con $N = 20$, $NU = 10$. a) Evolución de la salida de la planta. b) Comando aplicado.

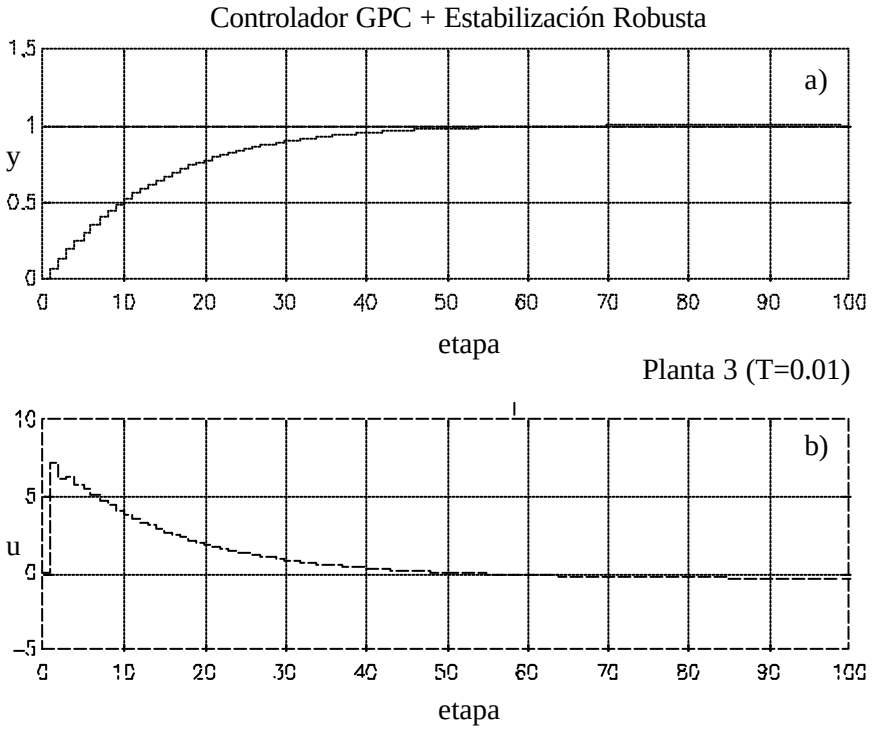


Figura 3.21. Controlador GPC con estabilización robusta aplicado sobre la planta 3 tomando como consigna un escalón unitario y con $N = 1$, $NU = 1$. a) Evolución de la salida de la planta. b) Comando aplicado.

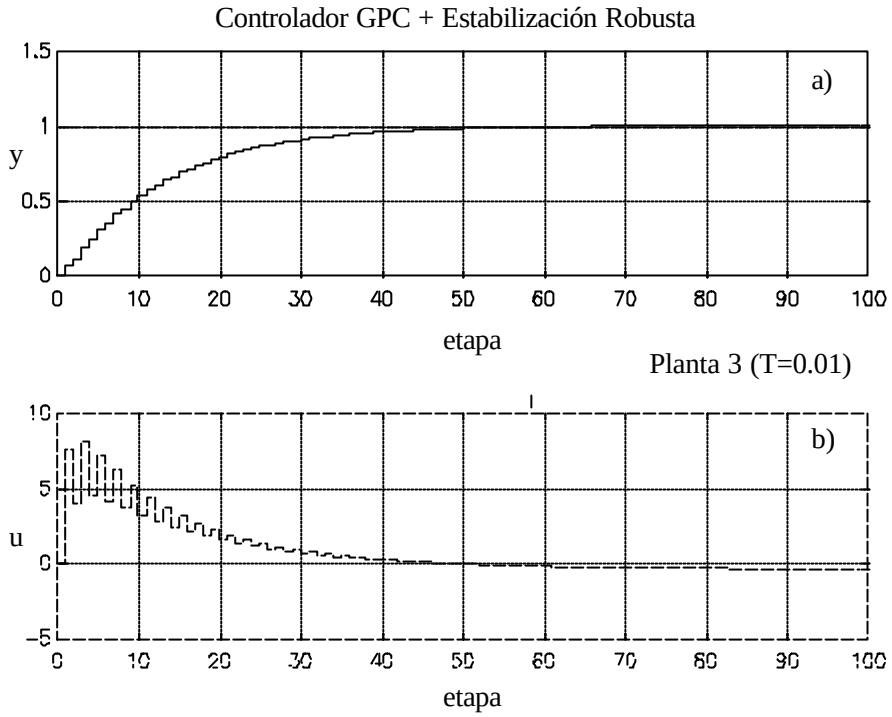


Figura 3.22. Controlador GPC con estabilización robusta aplicado sobre la planta 3 tomando como consigna un escalón unitario y con $N = 10$, $NU = 4$. a) Evolución de la salida de la planta. b) Comando aplicado.

4

REDES NEURONALES EN CONTROL

El siguiente paso a abordar es el problema del control adaptativo que permita adaptar en tiempo real los parámetros del controlador a los cambios producidos en la dinámica de la planta o a posibles perturbaciones que afecten a la misma. Para ello introduciremos el neurocontrol ya que pretendemos la utilización de redes neuronales en el proceso de control adaptativo. Lo que subyace es aplicar esta teoría adaptativa cuando el controlador a utilizar sea el GPC.

En este capítulo se pretende dar una introducción al control mediante redes neuronales. El capítulo esta dividido en dos grandes secciones. En la primera de ellas se hace un repaso de la teoría de redes neuronales y sus configuraciones más importantes, describiendo en profundidad el algoritmo de entrenamiento backpropagation. Este algoritmo es el que se emplea en el siguiente capítulo en el diseño de un controlador neuronal. En la otra sección se hace un repaso de las diferentes técnicas existentes para la aplicación de redes neuronales en control. Se comentarán las diferentes alternativas propuestas y las características de cada una de ellas.

4.1 INTRODUCCIÓN AL CONTROL NEURONAL

En los últimos años se han producido innumerables avances en el campo de la inteligencia artificial. En concreto, técnicas como los algoritmos genéticos, la lógica difusa y las redes neuronales (que a veces se engloban con el nombre común de *soft computing*) han sido aplicadas a la resolución de gran variedad de problemas: clasificación de patrones, aproximación de funciones, optimización, etc. La aplicación a los sistemas de control ha sido también objeto de investigación en los últimos años. Especialmente, las redes neuronales artificiales (RN), debido a su capacidad de aprendizaje, se han convertido en una herramienta muy potente para el desarrollo de sistemas de control. Esto ha hecho que surgiera una nueva rama en la teoría de control conocida como neurocontrol o control neuromórfico. Estos controladores se caracterizan por el empleo de RN en su estructura.

La función de las redes neuronales en estos controladores ha sido muy diversa, desde la actuación directamente como bloque controlador hasta la incorporación como un bloque adaptivo para sintonizar los parámetros del controlador. Es esta última configuración la que ha sido estudiada en este trabajo y sobre la que se han desarrollado nuevos algoritmos que han sido comprobados mediante simulación digital.

4.2 CONCEPTOS BÁSICOS SOBRE REDES NEURONALES

Las redes neuronales artificiales son sistemas matemáticos diseñados empleando los principios en los que se creen basados los sistemas biológicos nerviosos. Incorporando estos principios, se pretende emular en cierta medida las capacidades de procesamiento de información de los sistemas neurológicos. Las redes neuronales artificiales también tienen como unidad fundamental de procesamiento a la neurona. Las neuronas artificiales son emulaciones sencillas de las neuronas biológicas, en el sentido de que toman información desde sensores u otra neuronas artificiales, realizan una operación muy sencilla con estos datos y pasan el resultado a otras neuronas artificiales.

Algunas de las propiedades que presentan las redes neuronales son:

- Aprendizaje: Las redes neuronales tienen la capacidad de aprender del medio, es decir, pueden modificar su comportamiento en respuesta a su entorno.
- Generalización: Aunque las redes neuronales se entrenan para un conjunto finito de entradas, son capaces de generalizar el resultado cuando se le presentan entradas para las cuales no han sido entrenadas.
- Tolerancia a fallos: Las redes neuronales son sistemas que por su estructura presentan gran tolerancia a fallos. Esto es debido principalmente a la disgregación de la información en la red. De esta forma la destrucción de parte de la red, no produce una ruptura completa en el sistema de computación.
- Naturaleza paralela: El tiempo de cálculo en una red neuronal disminuya apreciablemente cuando se implementa en un sistema de computación paralelo. Esto es debido a la posibilidad de paralelizar las operaciones en las distintas neuronas.
- Facilidad de inserción en la tecnología existente: Las redes neuronales pueden ser usadas para mejoras y actualizaciones incrementales de sistemas, ya que una red neuronal puede ser entrenada para realizar una única tarea bien definida.

4.2.1 EL MODELO DE NEURONA

La idea básica en la que se inspira la neurona es la de imitar el comportamiento de una neurona biológica. El modelo de neurona es el que se muestra en la figura 4.1. Este modelo es conocido con el nombre de perceptrón. Cada neurona está formada por múltiples entradas **x** y una salida *y*. La salida de la neurona esta dada por

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right) \tag{4.1}$$

donde w_i son los pesos asociados a cada entrada y *f* es una función no lineal que opera sobre la suma de las entradas multiplicadas por los pesos. Obsérvese en la figura que x_0 es igual a uno. Esta entrada se conoce como entrada *bias*. La función *f* se conoce como función de activación y la única condición para *f* es que sea una función diferenciable, y preferiblemente con una expresión simple para esta derivada. Una de las funciónes de activación más usadas es la función sigmoide que está definida como

$$f(x) = \frac{1}{1+e^{-bx}} \tag{4.2}$$

Esta función es continua y varía monótonamente desde 0 hasta 1 a medida que *x* va desde -∞ hasta +∞. La ganancia de la sigmoide, β, determina la pendiente de la transición desde el 0 hasta el 1. Es usual que *b* se tome igual a 1. Otra función que se suele utilizar como función de activación es la tangente hiperbólica. La tangente hiperbólica es una función que varía monótonamente entre -1 y +1 y que está definida como

$$f(x) = \frac{1-e^{-bx}}{1+e^{-bx}} \tag{4.3}$$

Existe otra función de activación llamada función de activación umbral lógica, que es la función identidad si la suma de las entradas supera determinado valor, una función lineal si está entre este valor y un umbral inferior, o bien vale cero si la suma está por debajo de este umbral inferior. Otra posibilidad son las funciones radiales, de entre las que destaca la función gaussiana.

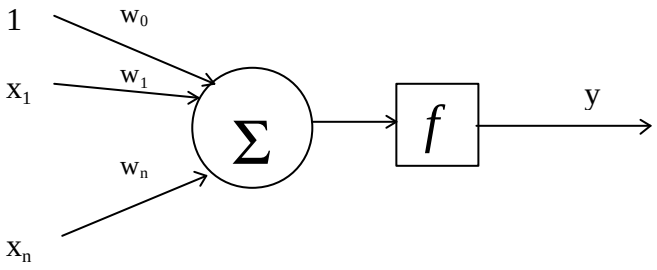


Figura 4.1. Modelo de neurona.

4.2.2 LA RED NEURONAL

Las capacidades de una neurona simple son muy limitadas, sin embargo cuando éstas se conectan formando agrupaciones de muchas neuronas es posible abordar actividades inteligentes. Estas agrupaciones se conocen como redes neuronales. Las organizaciones de redes neuronales se pueden clasificar en dos tipos principalmente: redes neuronales recurrentes y redes hacia adelante o perceptrón multicapa. La organización de una red recurrente está basada en la interconexión de múltiples neuronas entre sí con realimentaciones en las salidas de las neuronas (ver figura 4.2). Por el contrario, el perceptrón multicapa mantiene una estructura jerárquica que consiste en una serie de capas de neuronas donde cada capa de neuronas está conectada con la siguiente sin interconexiones entre las neuronas de una misma capa. En esta estructura las señales fluyen desde la capa de entrada a la capa de salida sin ninguna realimentación (ver figura 4.3).

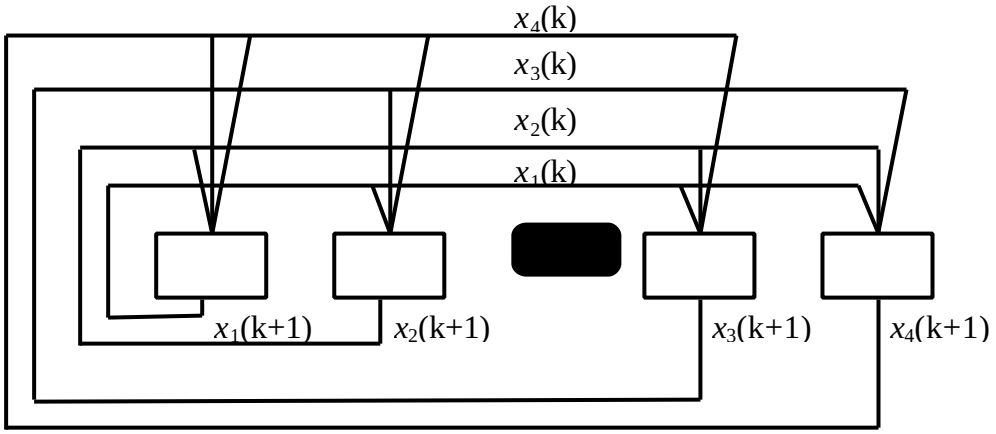


Figura 4.2. Red recurrente.

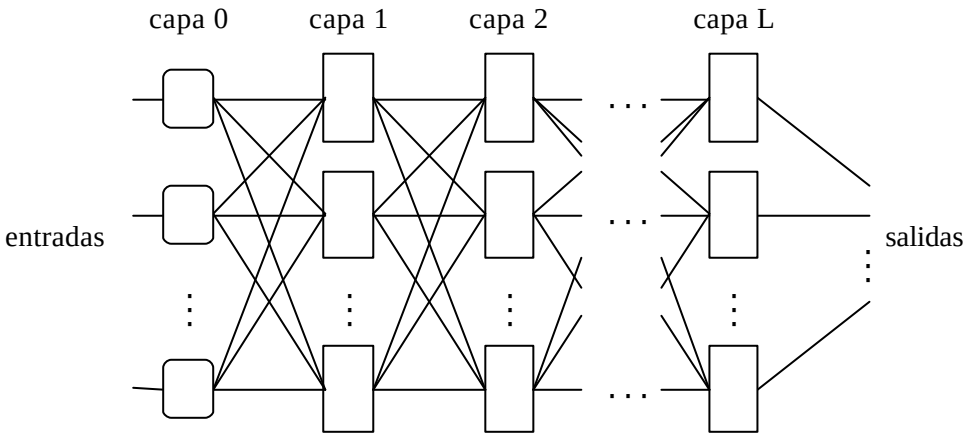


Figura 4.3. Perceptrón multicapa.

4.2.3 ENTRENAMIENTO DE LA RED: ALGORITMO BACKPROPAGATION

El objetivo final de una red es que ante la presencia de un vector de valores a la entrada, ésta responda con una salida adecuada. Para conseguir este objetivo es necesario modificar los pesos de la red para que presente la respuesta deseada. Este proceso de ajuste de los pesos de las neuronas se realiza mediante una algoritmo de entrenamiento. Durante el entrenamiento, los pesos de la red deben converger

gradualmente a valores tales que cada vector de entrada produzca el vector de salida deseado. Los algoritmos de entrenamiento se clasifican en supervisados y no supervisados:

- Entrenamiento supervisado. El proceso consiste en: aplicar un vector de entrada; calcular el vector de salida y compararlo con el vector objetivo correspondiente; alimentar la diferencia (el error) hacia atrás a través de la red y modificar los pesos según un algoritmo que tiende a minimizar el error. Los vectores del conjunto de entrenamiento se aplican de forma secuencial y se calcula el error y el ajuste de los pesos para cada vector. Esto se repite hasta que el error, para todo el conjunto de entrenamiento, sea aceptablemente bajo.

- Entrenamiento no Supervisado. Los algoritmos de este tipo no requieren vector objetivo para las salidas, y por lo tanto, no existe comparación con una respuesta ideal predeterminada. El conjunto de entrenamiento consiste únicamente en vectores de entrada. El algoritmo de entrenamiento modifica los pesos de la red para producir salidas que sean consistentes, es decir, dos aplicaciones del mismo vector de entrenamiento, o de un vector que sea suficientemente similar, debe producir el mismo vector de salida. El proceso de entrenamiento extrae las propiedades estadísticas del conjunto de entrenamiento y agrupa vectores similares dentro de clases. Aplicando a la entrada un vector de una clase dada producirá un vector de salida específico, pero no hay manera de determinar, antes del entrenamiento, que patrón de salida específico producirá una clase de vectores. Por ello, la salida de tales redes generalmente debe ser transformada a una forma comprensible tras el periodo de entrenamiento. Esto no es un serio problema y normalmente es una manera simple de identificar la relación establecida por la red.

En la figura 4.4 se muestra una clasificación de los algoritmos de entrenamiento más comúnmente utilizados.

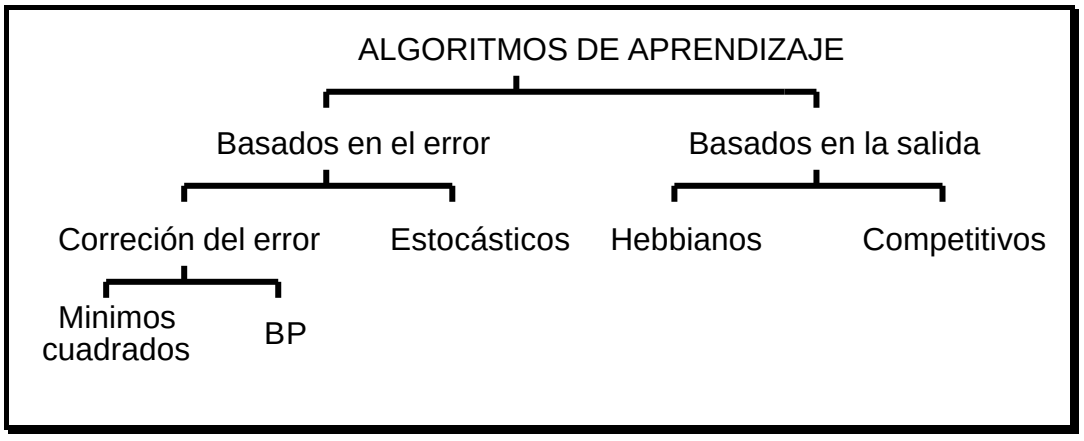


Figura 4.4. Resumen de los algoritmos de aprendizaje de las RN.

4.2.3.1 ALGORITMO BACKPROPAGATION

Uno de los principales problemas que presentó la formulación original del perceptrón multicapa fue la carencia de un algoritmo de entrenamiento adecuado. La aparición del algoritmo backpropagation jugó un papel importante en el desarrollo de las aplicaciones basadas en el perceptrón multicapa. Este algoritmo está basado en el método del gradiente descendente. El objetivo del entrenamiento de la red es ajustar los pesos de manera que la aplicación de un vector de valores a la entrada produzca el vector de valores de salida deseado. Para describir este algoritmo, considérese la siguiente notación [Hus93]:

$v_{l,j}$, salida del *j*-ésimo nodo de la capa *l*;
$w_{l,j,i}$, peso que conecta al *i*-ésimo nodo de la capa *l*-1 con el j-ésimo nodo en la capa *l*;
$\mathbf{x}_p$, *p*-ésimo patrón de entrenamiento;
$v_{0,i}$, *i*-ésima componente del vector de entrada;
$d_j(\mathbf{x})$, respuesta deseada de la *j*-ésima componente de la salida.
N_l , número de nodos en la capa *l*;
L, número de capas;
P, número de patrones de entrenamiento;

En esta notación la capa 0 de la red es la capa de entrada, es decir, $u_{0,j} = x_j$. Por otra parte, la componente número 0 del vector de entrada a cada capa es la entrada *bias* igual a 1, $u_{l,0} = 1$. El peso *bias* correspondiente es $w_{l,j,0}$. Con este convenio, la salida de un nodo en la capa *l* está dada por

$$v_{l,j} = f\left(\sum_{i=0}^{N_{l-1}} w_{l,j,i} v_{l-1,i}\right) \tag{4.4}$$

donde $f(\cdot)$ es la función sigmoide. La derivada de esta función es

$$f'(a) = \frac{df(a)}{da} = f(a)(1 - f(a)) \tag{4.5}$$

El objetivo del algoritmo es modificar los pesos de la red de forma que se minimice una función criterio suma de errores cuadrados:

$$J(\mathbf{w}) = \sum_{p=1}^{P} J_p(\mathbf{w}) \tag{4.6}$$

donde *P* es el número de patrones de entrenamiento y $J_p(w)$ es el error cuadrático total para el *p*-ésimo patrón:

$$J_p(\mathbf{w}) = \frac{1}{2}\sum_{q=1}^{N_L}\left(v_{L,q}(\mathbf{x}_p) - d_q(\mathbf{x}_p)\right)^2 \tag{4.7}$$

El algoritmo backpropagation se basa en la técnica del gradiente descendente que establece que los pesos deben ser actualizados de acuerdo con

$$w_{l,j,i}(k+1) = w_{l,j,i}(k) - m\left.\frac{\partial J(\mathbf{w})}{\partial w_{l,j,i}}\right|_{\mathbf{w}(k)} =$$

$$= w_{l,j,i}(k) - m\sum_{p=1}^{P}\left.\frac{\partial J_p(\mathbf{w})}{\partial w_{l,j,i}}\right|_{\mathbf{w}(k)} \tag{4.8}$$

donde m es una constante positiva llamada *velocidad de aprendizaje*. El cálculo de la derivada parcial se puede hacer empleando la regla de la cadena

$$\frac{\partial J_p(\mathbf{w})}{\partial w_{l,j,i}} = \frac{\partial J_p(\mathbf{w})}{\partial v_{l,j}}\frac{\partial v_{l,j}}{\partial w_{l,j,i}} \tag{4.9}$$

donde

$$\frac{\partial v_{l,j}}{\partial w_{l,j,i}} = f'\left(\sum_{m=0}^{N_{l-1}} w_{l,j,m}v_{l-1,m}\right)v_{l-1,i} \tag{4.10}$$

Teniendo en cuenta la ecuación (4.5) se llega a que

$$\frac{\partial v_{l,j}}{\partial w_{l,j,i}} = v_{l,j}(1-v_{l,j})v_{l-1,i} \tag{4.11}$$

Y, por lo tanto,

$$\frac{\partial J_p(\mathbf{w})}{\partial w_{l,j,i}} = \frac{\partial J_p(\mathbf{w})}{\partial v_{l,j}}v_{l,j}(1-v_{l,j})v_{l-1,i}. \tag{4.12}$$

El término $\partial J_p(\mathbf{w}) / \partial v_{l,j}$ representa la sensibilidad de $J_p(\mathbf{w})$ a la salida del nodo $u_{l,j}$. Es posible expresar $\partial J_p(\mathbf{w}) / \partial v_{l,j}$ en función de las sensibilidades de nodos en las capas más altas mediante la siguiente relación

$$\frac{\partial J_p(\mathbf{w})}{\partial v_{l,j}} = \sum_{m=1}^{N_{l+1}}\frac{\partial J_p(\mathbf{w})}{\partial v_{l+1,m}}v_{l+1,m}(1-v_{l+1,m})w_{l+1,m,i} \tag{4.13}$$

Este proceso se repite en las capas siguientes hasta alcanzar la capa de salida. A la salida se encuentra una condición de contorno que permite obtener las sensibilidades de esta última capa a partir de la ecuación (4.7) como

$$\frac{\partial J_p(\mathbf{w})}{\partial u_{L,j}} = u_{L,j}(\mathbf{x}_p) - d_j(\mathbf{x}_p) \tag{4.14}$$

Los pesos de la red se inicializan normalmente con valores aleatorios y pequeños. Esto hace que la búsqueda empiece en una posición relativamente segura. Las velocidades de aprendizaje se pueden escoger de diferentes formas. Puede ser un único valor constante para cada peso de la red, o bien pueden tomarse diferentes para cada nodo o para cada capa de la red. Es común añadir en cada actualización de los pesos un término *momento* de la forma $a(w_{l,j,i}(k)-w_{l,j,i}(k-1))$, siendo $0<a<1$.

El proceso de calcular el gradiente y ajustar los pesos de la red se repite hasta que se encuentra un mínimo. Existen diferentes criterios de terminación del algoritmo. Uno de ellos establece que el algoritmo termina cuando la magnitud del gradiente es suficientemente pequeña, ya que, por definición el gradiente se hará cero en el mínimo. Otra posibilidad es imponer que el algoritmo pare cuando *J* esté por debajo de cierto valor. Un criterio basado en el número de iteraciones consiste en fijar un número de iteraciones máximo, superado el cual el algoritmo debe parar. Por último, es muy conocido el criterio de validación cruzada. Este método divide los datos en dos conjuntos: un conjunto de entrenamiento (*training set*), usado para entrenar la red, y un conjunto de comprobación (*test set*), que se utiliza para medir la capacidad de generalización de la red. Durante el aprendizaje, las prestaciones de la red continuarán mejorando sobre el conjunto de entrenamiento, pero las prestaciones sobre el conjunto de test crecerán hasta cierto punto a partir del cual empezarán a decaer. Es en este punto donde debe terminar el algoritmo.

Por lo tanto, el algoritmo backpropagation se puede resumir en los siguientes pasos:

1.- *Inicializar los pesos con valores pequeños aleatorios.*
2.- *Propagar la señal de entrada hacia adelante a lo largo de toda la red.*
3.- *Calcular la función de sensibilidad para cada peso de la red.*
4.- *Actualizar los pesos.*
5.- *Ir al paso 2 y repetir el procedimiento hasta que se alcance la condición de parada.*

4.2.3.2 COMPLEJIDAD COMPUTACIONAL

Con el objetivo de medir la complejidad computacional del algoritmo *backpropagation*, se ha hecho una estimación del número de operaciones necesarias en cada periodo de muestreo en cada actualización de los pesos. Según se comentó en la sección anterior, este algoritmo tiene tres grandes tareas en cada etapa de muestreo:

- Propagar la señal de entrada a lo largo de la red hasta la salida (rutina *feed_forward*).
- Calcular el gradiente de la función de costo (rutina *compute_gradient*)
- Actualizar los pesos de la red (rutina *update_weights*).

En la tabla 4.1 se presenta una estimación del número de operaciones por periodo de muestreo para una red neuronal de tres capas, tres nodos por capa y con *no* salidas.

Procedimiento	**Multiplicaciones**	**Divisiones**	**Sumas**	**Almacenamiento**
Feed_forward	27+5**no*	6+*no*	27+5**no*	27+5**no*

Compute_gradient	90+21**no*	-	30+8**no*	30+8**no*
Update_weights	21+4**no*	-	21+4**no*	21+4**no*
Total	138+30**no*	6+*no*	78+17**no*	78+17**no*

Tabla 4.1. Número de operaciones en una red neuronal de tres capas, tres nodos por capa y con *no* salidas.

Como se observa, la rutina que conlleva mayor esfuerzo computacional es la del cálculo del gradiente. Si se considerase una red con tres salidas, *no*=3, el número de multiplicaciones, sumas y almacenamientos en la rutina *compute_gradient* sería 153, 54 y 54, respectivamente. Mientras que la rutina *feed_forward* conllevaría 42, 42 y 42.

4.3 NEUROCONTROL

La experiencia de los últimos años en el campo de las aplicaciones de control muestra que una forma de implementar sistemas de control más flexibles consiste en incorporar a las técnicas algorítmicas de la teoría de control convencional otros elementos como aprendizaje, lógica o heurísticas. Esta nueva metodología es lo que ha dado lugar al control inteligente [Nar91], [Lin96]. Con este término se hace referencia a diversas metodologías que combinan técnicas de control convencional y de inteligencia artificial como RN, sistemas expertos, lógica difusa, algoritmos genéticos y una amplia variedad de técnicas de búsqueda y optimización.

La rama de la ingeniería de control que se basa en la síntesis de sistemas de control ayudados por medio de redes neuronales se conoce como neurocontrol [Bar90], [Nar90], [Wer91]. Esta nueva vertiente dentro del control ha gozado de un gran avance en los últimos años. El principal punto de aplicación es el de resolver tareas que por su naturaleza son especialmente complejas o no tienen una solución analítica [San90], [And90], [Par94]. De hecho se pretende plantear una metodología basada en redes neuronales para resolver las dificultades que presentan las técnicas de control adaptativo cuando se abordan plantas no lineales o plantas con incertidumbre en sus parámetros. Las redes neuronales son especialmente aptas para este tipo de tareas debido a las capacidades de aprendizaje, adaptación y auto-organización que presentan. De entre las distintas estructuras de organización de las redes, el perceptrón multicapa es especialmente interesante para el control debido a que

- Las redes neuronales multicapa son esencialmente estructuras de organización hacia adelante, desde las entradas hasta las salidas, a través de capas ocultas. Esta característica es muy conveniente para los ingenieros de control, que suelen trabajar con sistemas representados mediante bloques con entradas y salidas claramente diferenciadas. Esta característica no aparece en las redes neuronales recurrentes en donde hay nodos bidireccionales y no existen entradas y salidas claramente definidas.
- El perceptrón multicapa con sólo una capa oculta y usando funciones de activación sigmoidales arbitrarias es capaz de desarrollar cualquier mapeo no lineal entre dos espacios de dimensiones finitas y con cualquier grado de precisión, empleando un número de neuronas adecuado en la capa oculta. En control, los bloques que se manejan se pueden ver como un mapeo entre un espacio de entrada y un espacio de salida, y, por lo tanto, pueden ser emulados mediante perceptrones multicapa.
- El algoritmo básico de aprendizaje del perceptrón multicapa es el algoritmo de backpropagation, que se basa en métodos de gradiente que son ampliamente aplicados en control óptimo, y, por lo tanto son bien conocidos por los ingenieros de control.

Generalmente el entrenamiento de las RN en un sistema de control se puede realizar en tiempo real (*on-line*) o previamente a la implementación de control (*off-line*), dependiendo de si las redes ejecutan o no trabajo útil mientras aprenden. Aunque el entrenamiento *off-line* es generalmente directo, las condiciones para asegurar una buena generalización de las RN a lo largo del espacio de control son difíciles de obtener. De ahí que siempre sea necesario, complementar el aprendizaje con entrenamiento *on-line* en la red. Lo deseable es que el entrenamiento de la red se llevara a cabo totalmente *on-line*, de forma que la red, partiendo de unos valores iniciales aleatorios, converja con una velocidad elevada. Sin embargo, el principal problema que presentan la mayoría de los controladores propuestos es su lentitud en la convergencia. Una convergencia lenta en un control neuromórfico implica un rendimiento y robustez pobres, especialmente durante las primeras etapas del control. Actualmente se desarrolla mucha investigación encaminada a conseguir algoritmos de control *on-line* eficientes.

4.3.1 CONTROLADOR NEURONAL

Considérese un proceso SISO discreto

$$y(k+1) = f[y(k), y(k-1), \ldots, y(k-p+1), u(k), u(k-1), \ldots, u(k-q)] \qquad (4.15)$$

donde y denota la salida, u la entrada, k la etapa, p y q son dos constantes enteras y $f(\cdot)$ una función continua y derivable. El objetivo de control consistirá en hacer que la salida y siga a una consigna r prefijada .

Supóngase que la planta (4.15) es invertible, es decir, que existe una función $g(\cdot)$ tal que

$$u(k) = g[y(k+1), y(k), \ldots, y(k-p+1), u(k-1), u(k-2), \ldots, u(k-q)] \qquad (4.16)$$

Considérese una red neuronal multicapa con un vector de entrada m-dimensional, $\mathbf{x}$, siendo $m=p+q+1$, un vector u_1 de salida, y con una relación de entrada-salida representada mediante

$$u_1 = r(\mathbf{x}) \qquad (4.17)$$

Si la salida de $r(\cdot)$ se hace igual a la salida de $g(\cdot)$ ante las mismas entradas, la red neuronal puede verse como un controlador del proceso. En cada instante k, la entrada a la planta se puede obtener a partir de (4.17) tomando el vector $\mathbf{x}$ como

$$\mathbf{x}(k) = [r(k+1), y(k), \ldots, y(k-p+1), u(k-1), \ldots, u(k-q)]^T \qquad (4.18)$$

donde se emplea la referencia $r(k+1)$ en lugar de la salida $y(k+1)$, que en ese instante es desconocida. La configuración de un neurocontrolador como éste se muestra en la figura 4.5. La señal de entrenamiento suministra la información necesaria a la red para aprender la dinámica inversa de la planta.

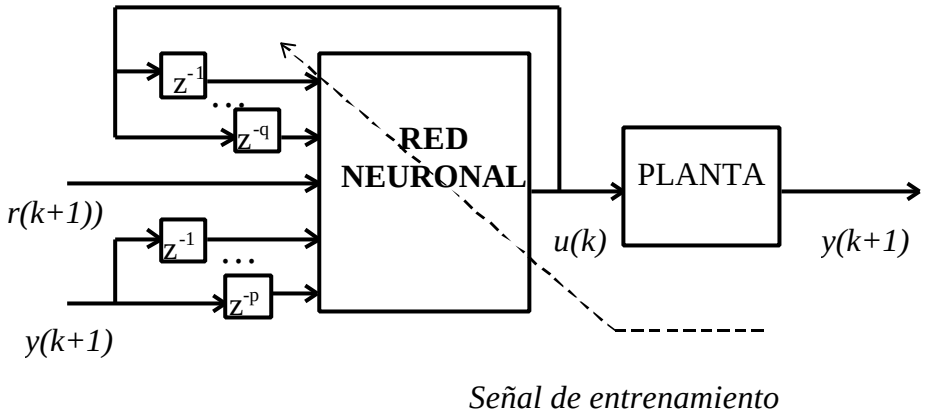


Figura 4.5. Configuración general de un neurocontrolador.

Existen diferentes estructuras de control neuronal basadas en el esquema general propuesto en la figura 4.5 [Fuk92] [Tan92]:

4.3.1.1 CONTROL INVERSO DIRECTO

En este controlador la red se entrena *off-line* para aprender la dinámica inversa de la planta, $g(\cdot)$. Para conseguir este entrenamiento se plantea el esquema mostrado en la figura 4.6. En este esquema la red recibe como señal de entrenamiento la diferencia entre el valor de salida de la red y la salida de la planta. Los pesos se ajustan para conseguir que este error sea mínimo.

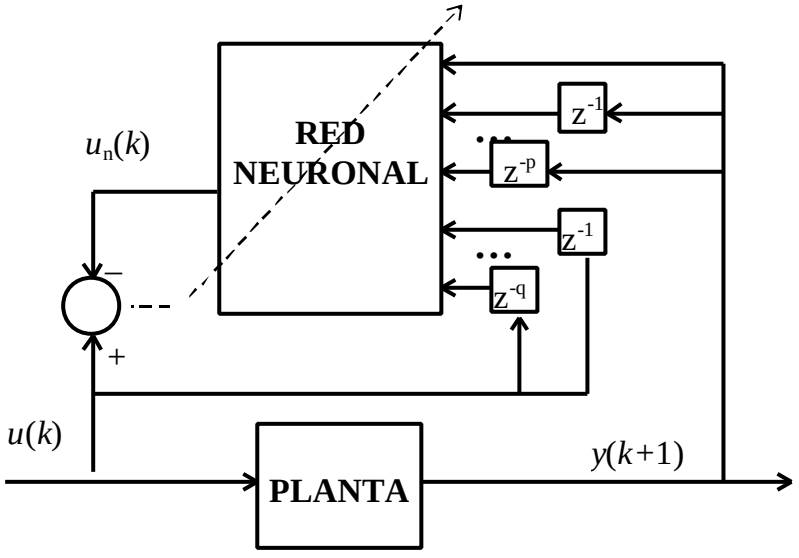


Figura 4.6. Entrenamiento del emulador de la dinámica inversa de la planta.

Una vez entrenada la red la configuración del controlador es la que se muestra en la figura 4.7. Como puede observarse para este controlador no existe realimentación en las variables de la planta, este hecho

hace que la aplicabilidad de este control esté muy restringida, debido a problemas de convergencia y robustez.

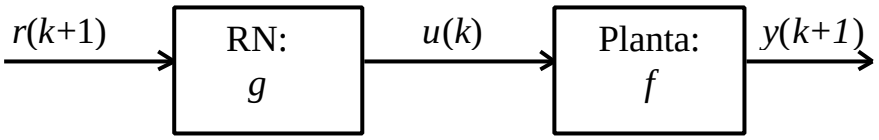


Figura 4.7. Control inverso directo.

4.3.1.2 CONTROL ADAPTIVO DIRECTO

En esta configuración el aprendizaje se realiza *on-line* y la función de error *J* que debe minimizar la red es el error entre la salida deseada *r* y la salida de la planta:

$$J(k) = \frac{1}{2}[r(k) - y(k)]^2 \tag{4.19}$$

El problema que presenta esta configuración es que el cálculo de la derivada de *J* con respecto a la salida de la red ($\partial J/\partial y$) requiere conocimiento del Jacobiano de la planta. La estructura de este controlador es la que se muestra en la figura 4.8.

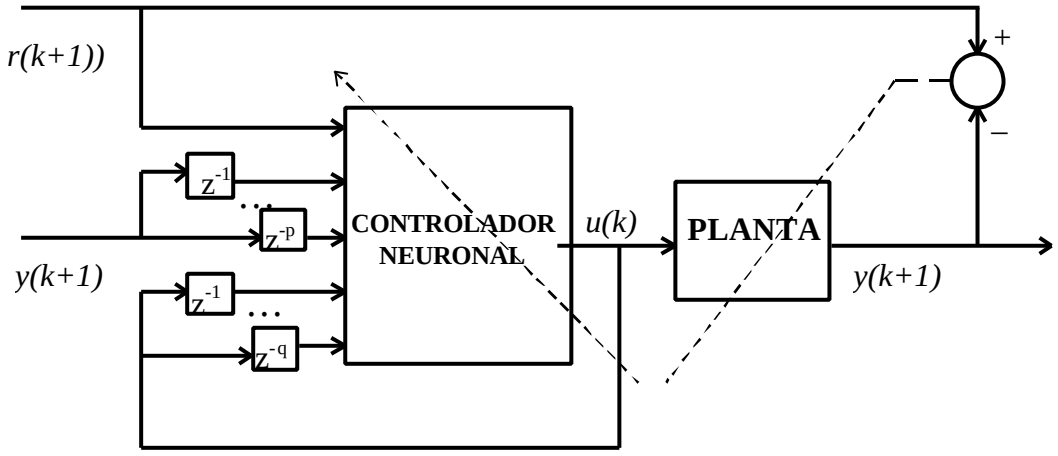


Figura 4.8. Controlador adaptivo directo.

4.3.1.3 CONTROL ADAPTIVO INDIRECTO

Este controlador tiene una estructura parecida al anterior pero en lugar de propagar el error a través de la planta, se utiliza una red neuronal que funciona como un emulador de la planta. De este forma se evitan los problemas para el cálculo de la función de sensibilidad de la planta. La estructura de este controlador se muestra en la figura 4.9.

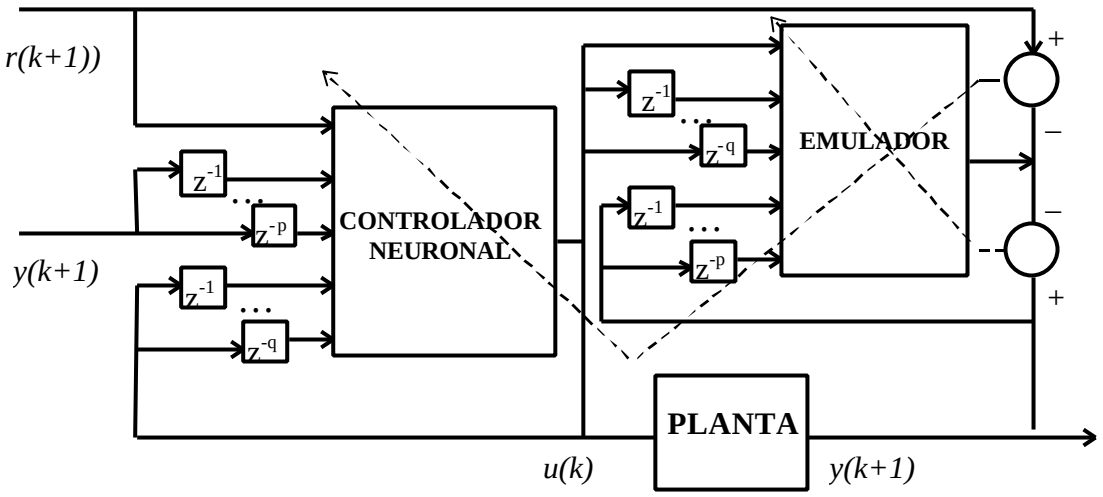


Figura 4.9. Controlador adaptivo indirecto.

Esta configuración es especialmente interesante cuando no existe una inversa para la función $f(\cdot)$ que define la planta. El entrenamiento del emulador de la planta se realiza *off-line*, y luego se entrenan *on-line* tanto el neurocontrolador como el emulador. En general, ya que el emulador realiza una tarea de identificación es recomendable actualizarlo más a menudo que el neurocontrolador para mejorar la robustez del sistema global.

5

DISEÑO DE UN CONTROLADOR ADAPTIVO NEUROMÓRFICO. APLICACIÓN AL PROBLEMA DEL PUENTE DE GRÚA

Como un avance a los conceptos comentados en el capítulo anterior sobre el uso de redes neuronales en control, en este capítulo se propone un nuevo diseño de controlador basado en redes neuronales. La idea de este controlador es hacer uso de la capacidad de aprendizaje de las redes neuronales e incluirlas como elementos de sintonización de los parámetros de un controlador convencional. El capítulo se divide en dos partes, en la primera de ellas se hace una descripción del diseño de este controlador y en la segunda se presenta una aplicación del mismo a una planta no-lineal. La planta escogida es un puente de grúa (overhead crane). Este tipo de plantas es extensivamente utilizada en muchos sectores de la industria. Se ha construido un prototipo a escala de un puente de grúa sobre el cual se ha probado la estrategia planteada. Los resultados

obtenidos tanto en simulación como sobre el prototipo atestiguan la efectividad de esta estrategia.

5.1 INTRODUCCIÓN

A pesar de que en la industria, se utilizan intensivamente controladores convencionales simples, y presentan un buen rendimiento para la mayoría de tareas, cuando la planta o el proceso a controlar es complejo o presenta altas no linealidades, el rendimiento obtenido decrece notablemente. En estos casos, surgen diferentes alternativas para mejorar la efectividad de los controladores convencionales. Una de las más empleadas es el control adaptivo. Este tipo de controladores, debido a su capacidad de adecuación a las variaciones del proceso, presenta un rendimiento muy satisfactorio.

Este capítulo trata del diseño de sistemas de control adaptativos asistidos por redes neuronales. Como ya se comentó, existen dos formas principalmente de usar redes neuronales en control. La forma más simple es emplear directamente las redes neuronales como un bloque de control que replica la dinámica inversa de la planta. La principal desventaja inherente a esta configuración es el problema de robustez que presenta. Debido a los problemas para predecir el comportamiento de la planta, es difícil estudiar la estabilidad del sistema. Una forma alternativa de emplear redes neuronales en control es utilizándolas como un sistema de control adaptivo.

El neurocontrolador propuesto aquí es un sistema de *selt-tuning* consistente en un controlador convencional combinado con una red neuronal para calcular *on-line* los coeficientes del controlador. Estos parámetros son ajustados usando una ley de adaptación pre-especificada, que normalmente será una ley lineal y cuadrática. El objetivo de este esquema es reducir el tiempo de entrenamiento del controlador de forma que sea posible una implementación en tiempo real. Además, el uso de un controlador convencional en lugar de una red neuronal inversa incrementa la robustez de este esquema.

Como ejemplo de aplicación se ha elegido una planta no lineal. Esta planta, muy típica en la industria, es un puente de grúa. Su tarea es transportar contenedores entre dos puntos de un plano. Debido a sus propiedades no lineales, la masa de carga sufre oscilaciones a lo largo de la trayectoria. Estas oscilaciones son especialmente indeseables en el punto final de la trayectoria. El objetivo de control es conseguir transportar masas de un punto a otro llegando al punto destino sin oscilaciones en la carga.

Un método famoso que trata la cancelación de oscilaciones es el control *posicast* de Smith [Smi58]. Éste es una técnica en lazo abierto que garantiza que no haya sobrepasamiento ni oscilaciones en la respuesta transitoria de un sistema de control. Sin embargo, la aplicación de este método a una planta no lineal no es simple. Otras aproximaciones propuestas recientemente para resolver el problema de control de antibalanceo están basadas en: esquemas en lazo abierto [Str88], control óptimo [Har89], control híbrido [Sak85], asignación de polos [Gus96] o técnicas de linealización [Che96]. Todos estos esquemas, aunque presentan buen rendimiento en la mayoría de las ocasiones, muestran problemas bajo ciertas circunstancias, especialmente cuando afectan a la planta dinámicas no modeladas. Aunque han surgido sistemas de control adaptivo para resolver estas dificultades [Bou92], estos suelen presentar problemas prácticos en la convergencia de los algoritmos.

5.2 DISEÑO DEL CONTROLADOR NEURONAL

El objetivo planteado en esta sección es el de proponer un método efectivo para ajustar adaptativamente los parámetros de un controlador. Para hacer esto, se utilizará una red neuronal para suministrar los parámetros del controlador, a partir de información sobre el estado del sistema.

5.2.1 ESTRUCTURA DE UN CONTROLADOR SELF-TUNER CONVENCIONAL

La estructura clásica de un controlador *self-tuning* convencional presenta dos elementos principales [Goo84], [Cla85], un controlador de realimentación, y un *self-tuner* formado por un estimador de parámetros y un algoritmo de diseño de control. En la figura 5.1 se muestra un esquema de un controlador *self-tuning* clásico. En este esquema, se asume un modelo paramétrico para la planta. El controlador se plantea normalmente como una ecuación en diferencias que viene en función normalmente de un conjunto de valores como la salida medida, la señal de referencia, etc. El estimador es recursivo y su función es monitorizar la entrada y salida de la planta y calcular una estimación de la dinámica de la planta en función de un conjunto de parámetros en un modelo estructural pre-especificado. La estimación de los

parámetros entra en el algoritmo de diseño de control, que suministra el nuevo conjunto de parámetros para el controlador.

Una versión simplificada de *self-tuner* se tiene cuando en la estructura de la figura 5.1 se omite la etapa de diseño de control. En este caso el estimador directamente genera los coeficientes de la ley de control. Este tipo de controladores *self-tuning* en los que no aparece una fase de identificación y una de diseño separadamente se conocen como *implícitos*. Por el contrario, los controladores que incluyen tanto un estimador como una etapa de diseño de control, se conocen como controladores *self-tuning explícitos*.

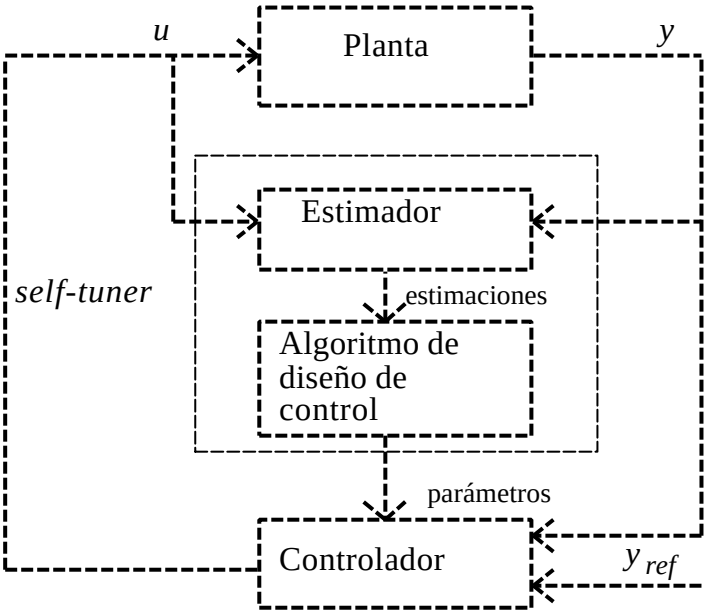


Figura 5.1. Estructura de un controlador *self-tuning* explícito.

5.2.2 ESTRUCTURA DEL CONTROLADOR NEURONAL ADAPTIVO

El *self-tuner* propuesto en este trabajo difiere de esta estructura. La principal diferencia es que no se plantea un modelo para la planta como en el caso convencional. Ahora, la red neuronal realiza la tarea del *self-tuner* sin ninguna información del modelo de la planta. La red actúa como un *self-tuner* implícito suministrando directamente los parámetros del controlador. En realidad, no existe una etapa de estimación y otra de diseño. La red neuronal en cada instante simplemente busca los parámetros óptimos minimizando una función de coste prefijada usando información del estado del sistema.

La estructura básica de este controlador para una planta con una entrada y una salida (SISO) es la que se muestra en la figura 5.2. En este esquema las entradas a la red neuronal son el error y la derivada del error. Las salidas son los parámetros del controlador. En cada instante de tiempo la red suministra un conjunto de parámetros q_1, …, q_n al controlador. La red parte de un estado inicial con valores aleatorios para los pesos y se entrena para minimizar una función de costo de la forma

$$J(k) = \frac{1}{2}(yref(k) - y(k))^2 . \tag{5.1}$$

Como se ve, se trata de una función lineal cuadrática que penaliza los errores cometidos en la salida. Por lo tanto, se espera que los parámetros de la red neuronal hagan que la salida se acerque a la consigna prefijada.

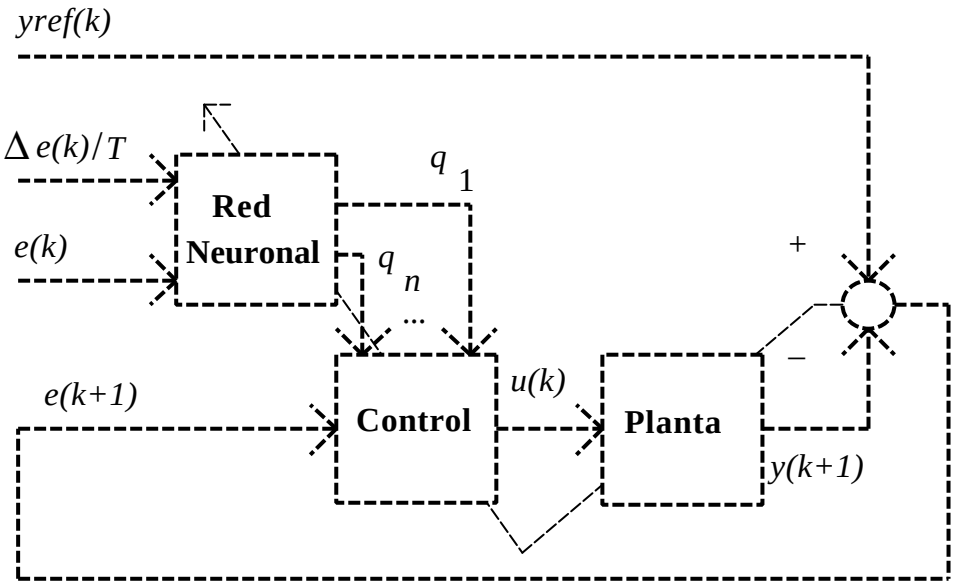


Figura 5.2. Estructura del controlador neuronal.

El aprendizaje de la red se realiza siguiendo el algoritmo *Backpropagation* (ver sección 4.2.3). El proceso para actualizar los pesos se realiza de acuerdo con

$$w_{l,j,i}(k+1) = w_{l,j,i}(k) - m\frac{\partial J(k)}{\partial w_{l,j,i}} \tag{5.2}$$

Por lo tanto, es necesario calcular el término $\partial J(k)/\partial w_{l,j,i}$. Si la salida de la red neuronal fuese directamente la variable *y*, esta derivada para la capa de salida sería

$$\frac{\P J(k)}{\P w_{L,j,i}} = \frac{\P J(k)}{\P y(k)} \frac{\P y(k)}{\P w_{L,j,i}} = -e(k) \frac{\P y(k)}{\P w_{L,j,i}} \tag{5.3}$$

siendo $e(k)$ el error de salida. En el controlador propuesto aquí, las salidas de la red, q_j, no aparecen en la función de costo explícitamente. Entonces, la función de sensibilidad para la capa de salida resulta ser

$$\frac{\P J(k)}{\P w_{Lji}} = \frac{\P J(k)}{\P q_j} \frac{\P q_j}{\P w_{L,j,i}} \tag{5.4}$$

donde

$$\frac{\P J(k)}{\P q_j} = -e(k) \frac{\P y(k)}{\P u} \frac{\P u(k)}{\P q_j} \tag{5.5}$$

El factor $\P y(k) / \P u$ se obtiene inmediatamente si se conoce el modelo de la planta. Si el modelo de la planta es desconocido, este término se puede obtener mediante la aproximación en diferencias

$$\frac{\P y(k)}{\P u} \approx \frac{y(k) - y(k-1)}{u(k-1) - u(k-2)}. \tag{5.6}$$

El factor $\partial u(k) / \partial q_j$ se calcula a partir de la expresión del controlador. Si el controlador se cambia sólo es necesario cambiar la función de sensibilidad, y el resto del algoritmo permanece igual.

5.3 CONTROL NEURONAL ADAPTIVO CON CONTROLADORES PID

Como ejemplo de aplicación para este controlador neuronal, se ha considerado este esquema para un sistema de control basado en un controlador PID. El objetivo planteado es el de ajustar dinámicamente los parámetros del controlador mediante una red neuronal de forma que se minimice una función de costo predeterminada. En este caso se ha planteado la siguiente función de costo:

$$J(k) = \frac{Q}{2}(y_{ref}(k) - y(k))^2 + \frac{R}{2}(\Delta u(k))^2 + \frac{S}{2}u(k)^2 \tag{5.7}$$

donde $\Delta u(k) = u(k) - u(k\text{-}1)$. Se trata de una función lineal cuadrática en el error, en el incremento de comando y en el comando. De esta forma se trata de conseguir, por un lado que la salida alcance la consigna, y por otro se imponen penalizaciones en el comando. Al pesar el incremento en el comando se trata de evitar oscilaciones bruscas en el comando que en general son indeseables para un sistema. Además, el peso en el comando aporta un mecanismo para controlar la magnitud de las acciones de control aplicadas. Los parámetros que definen el peso en el error, el incremento de comando y en el comando son Q, R y S, respectivamente.

Como es bien conocido, el controlador PID está definido por la ecuación continua:

$$u(t) = k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \tag{5.8}$$

Aproximando por diferencias finitas se puede escribir la salida del controlador como:

$$u(k) = u(k-1) + Ae(k) + Be(k-1) + Ce(k-2) \tag{5.9}$$

donde

$$\begin{aligned} A &= k_p + \frac{k_d}{T}; \\ B &= -(k_p - k_i T - 2\frac{k_d}{T}); \\ C &= \frac{k_d}{T}; \end{aligned} \tag{5.10}$$

siendo T el periodo de muestreo. El problema entonces para que la minimización de la función (5.7) se lleve a cabo satisfactoriamente es calcular la función de sensibilidad (5.4), $\partial J / \partial w$, para cada salida. Esta derivada se puede obtener como:

$$\frac{\partial J(k)}{\partial w_{L,j,i}} = \frac{\partial J(k)}{\partial k_j}\frac{\partial k_j}{\partial w_{L,j,i}} \tag{5.11}$$

donde k_j representa las constantes k_p, k_i y k_d del controlador. El factor $\P k_j / \P w_{L,j,i}$ se obtiene mediante la expresión (4.13) y el otro término se calcula aplicando la regla de la cadena a la expresión (5.7)

$$\frac{\P J(k)}{\P k_j} = \frac{\P J(k)}{\P y(k)} \frac{\P y(k)}{\P u(k)} \frac{\P u(k)}{\P k_j} + \frac{\P J(k)}{\P \Delta u(k)} \frac{\P \Delta u(k)}{\P k_j} + \frac{\P J(k)}{\P u(k)} \frac{\P u(k)}{\P k_j} \tag{5.12}$$

donde, teniendo en cuenta (5.7), se tiene

$$\frac{\P J(k)}{\P y(k)} = -Qe(k); \tag{5.13}$$

$$\frac{\P J(k)}{\P \Delta u(k)} = R\Delta u(k); \tag{5.14}$$

$$\frac{\P J(k)}{\P u(k)} = Su(k); \tag{5.15}$$

El término $\frac{\P y(k)}{\P u}$ puede ser aproximado según (5.6):

$$\frac{\P y(k)}{\P u} \approx \frac{y(k) - y(k-1)}{u(k-1) - u(k-2)}$$

El término $\P u(k) / \P k_j$ se obtiene de la expresión del controlador, ecuación (5.9), como

$$\frac{\P u(k)}{\P k_p} = e(k); \tag{5.16}$$

$$\frac{\P u(k)}{\P k_i} = \frac{\P u(k-1)}{\P k_i} + Te(k-1); \tag{5.17}$$

$$\frac{\P u(k)}{\P k_d} = \frac{e(k) - e(k-1)}{T}. \tag{5.18}$$

En cuanto al término $\P \Delta u(k) / \P k_j$, se puede calcular teniendo en cuenta que

$$\Delta u(k) = u(k) - u(k-1) = k_p(e(k) - e(k-1)) + k_i Te(k-1) + \frac{k_d}{T}(e(k) - \\ - 2e(k-1) + e(k-2)) \tag{5.19}$$

Por lo tanto, se encuentra que

$$\frac{\partial \Delta u(k)}{\partial k_p} = e(k) - e(k-1); \tag{5.20}$$

$$\frac{\partial \Delta u(k)}{\partial k_i} = Te(k-1); \tag{5.21}$$

$$\frac{\partial \Delta u(k)}{\partial k_d} = \frac{e(k) - 2e(k-1) + e(k-2)}{T} \tag{5.22}$$

La estructura de este controlador es la que aparece en la figura 5.3. Las entradas a la red son el error y la derivada del error que darán cuenta del estado del sistema. En este caso la señal de entrenamiento (con línea discontinua) incluye información además del error, del comando aplicado. La red parte de un estado inicial donde todos los pesos toman valores aleatorios. A partir de ahí el entrenamiento de la red se realiza simultáneamente con las acciones de control.

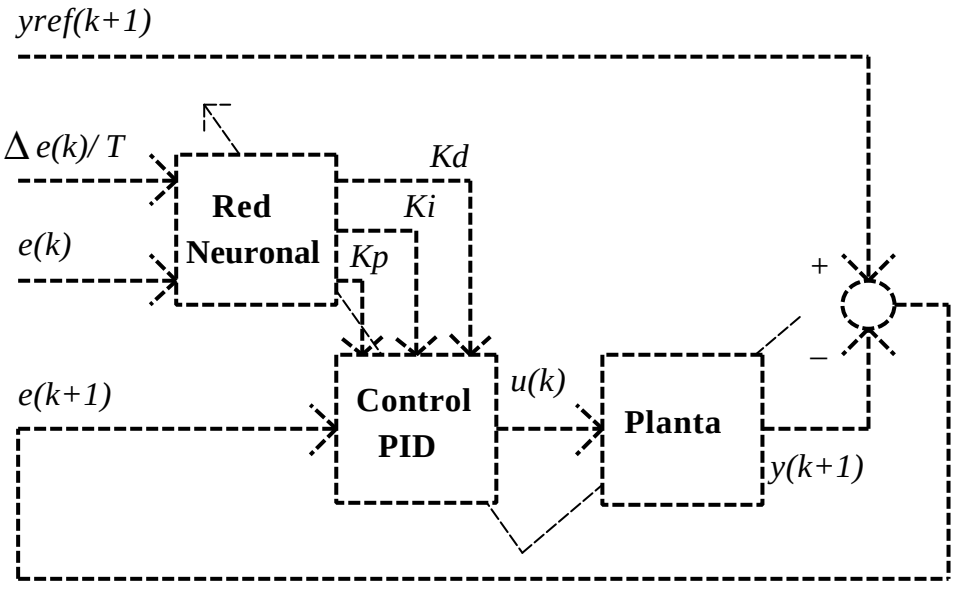


Figura 5.3. Estructura del controlador neuronal self-tuning actuando sobre un PID.

5.3.1 RESULTADOS CON CONTROLADOR PI

Siguiendo el esquema comentado, se han realizado diferentes simulaciones aplicando el controlador neuronal adaptivo sobre una planta lineal. La estructura de control considerada es similar a la de la figura 5.3. Por simplificación, en la simulación llevada a cabo se ha eliminado

la acción derivativa y sólo se ha considerado un control PI. La red empleada contiene 3 capas de neuronas con 3 neuronas en cada capa oculta y dos neuronas en la capa de salida que producen la salida de k_p y k_i. Inicialmente los pesos han sido tomados aleatoriamente a partir de una distribución normal con varianza uno en torno al cero.

En la figura 5.4 aparece una prueba realizada con un controlador PI. En este caso los pesos tomados para la función de costo fueron Q=20, R=0, S=0 y el periodo de muestreo T= 0.01 segundos. La planta bajo control es una planta de primer orden con un polo en s=-0.4. Se observa como la salida del sistema presenta un comportamiento muy satisfactorio alcanzando la consigna suavemente y sin sobrepasamiento. En las figuras 5.5 y 5.6 se muestra la evolución de las constantes k_p y k_i del controlador PI. En ambos casos se observa una primera etapa en la que las constantes sufren oscilaciones hasta que después se estabilizan en torno a un valor. En la figura 5.7 se muestra la curva con los valores que toma el comando en esta experiencia. Se observa que las acciones de control permanecen siempre dentro de valores admisibles.

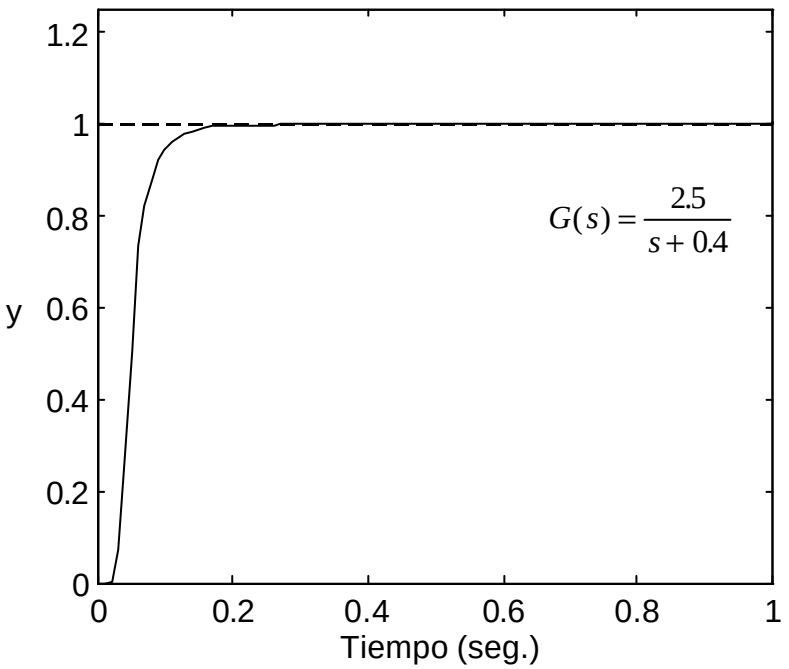


Figura 5.4. Control adaptivo neuronal. Evolución de la salida del sistema.

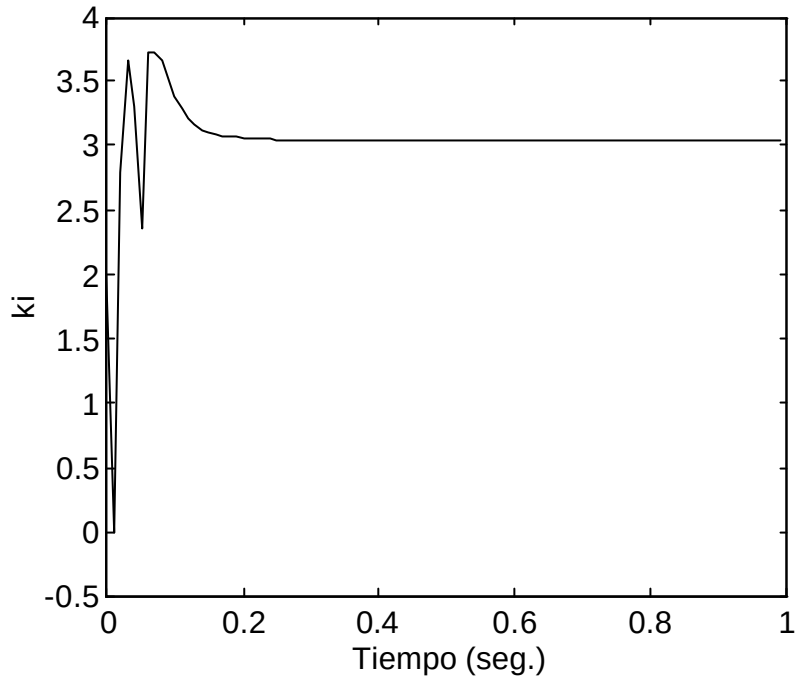


Figura 5.5. Control adaptivo neuronal. Evolución de la constante de integración, k_i en la simulación de la figura 5.4.

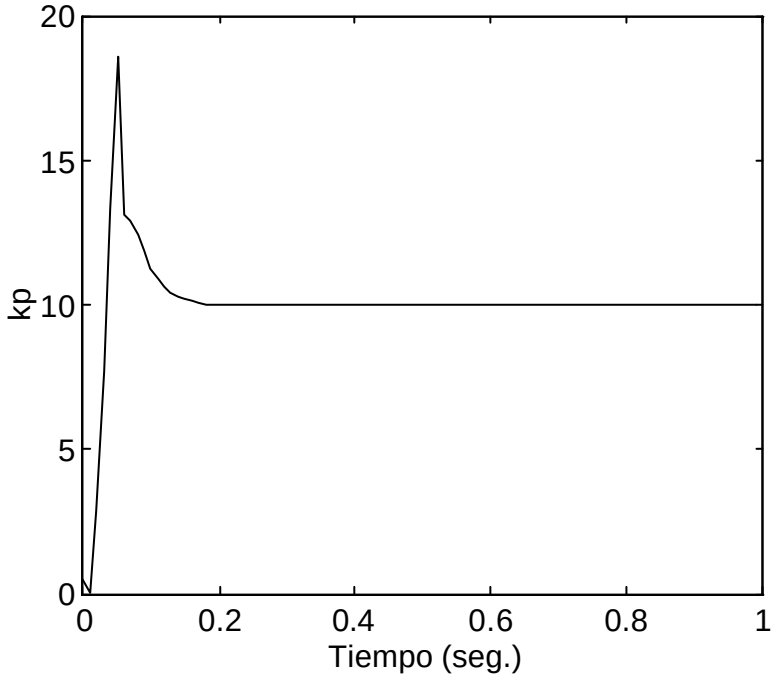


Figura 5.6. Control adaptivo neuronal. Evolución de la constante proporcional, k_p en la simulación de la figura 5.4.

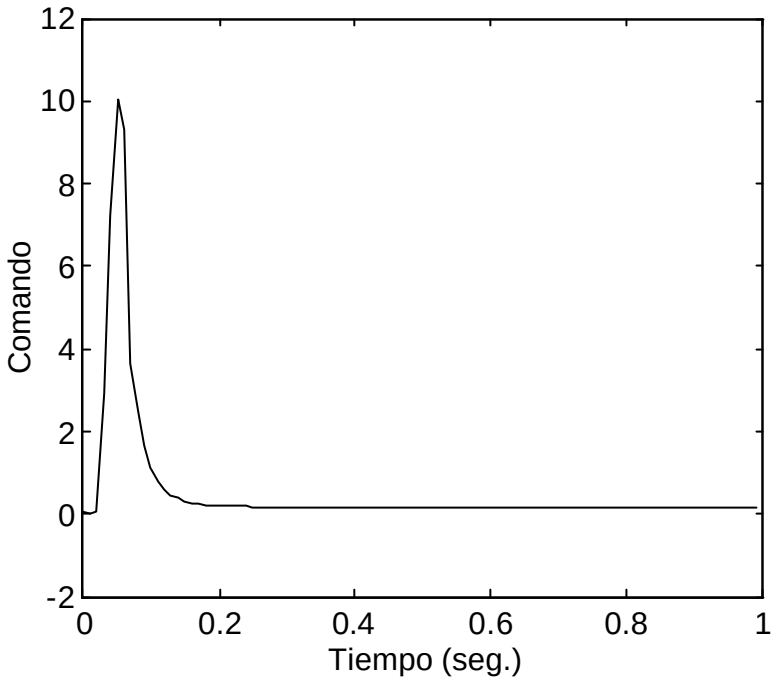


Figura 5.7. Control adaptivo neuronal. Evolución del comando en la simulación de la figura 5.4.

Para comprobar el efecto de las penalizaciones en el incremento del comando, se muestran en las figuras 5.8 y 5.9, la evolución de los comandos para una experiencia con R= 0 y otra experiencia con las mismas condiciones pero con R=7e-6. En esta experiencia se ha considerado como consigna una señal sinusoidal. Tal como se puede observar, tras una primera fase de entrenamiento en la red, el comando con R=0 presenta un comportamiento con variaciones bruscas en el comando. Sin embargo, en la experiencia con R distinto de cero se consigue que, después de una primera etapa en la que aparecen oscilaciones y una vez entrenada la red, el comando varíe de forma más suave. Este hecho es el que justifica el uso de la función de costo con peso en el incremento de comando.

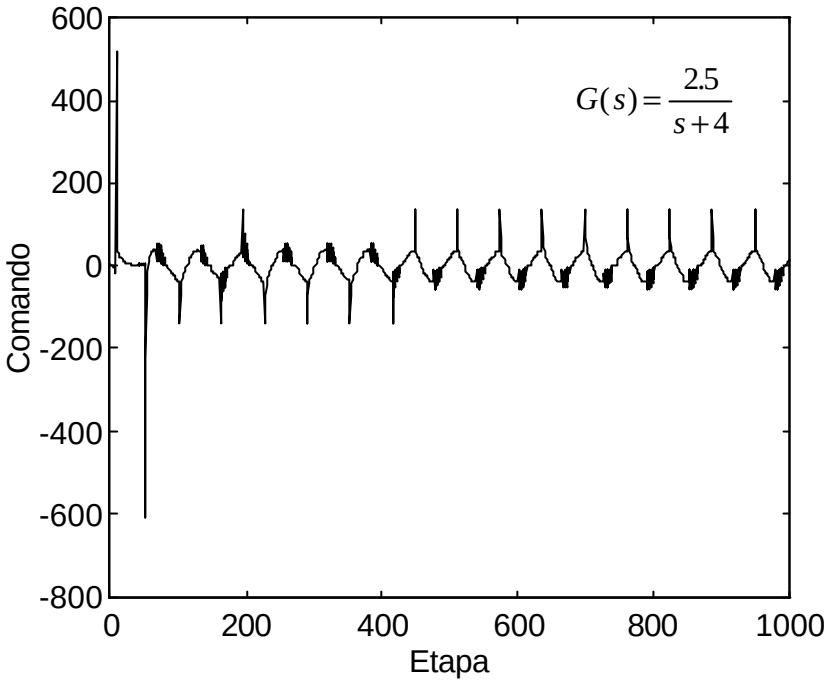


Figura 5.8. Evolución del comando con R=0.

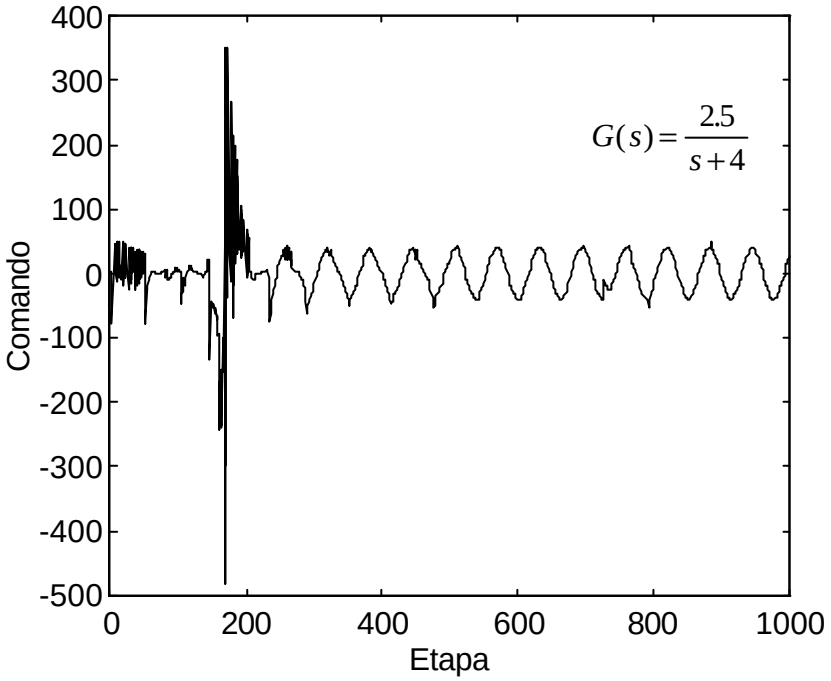


Figura 5.9. Evolución del comando con R=7e-6.

En la figura 5.10 se muestra una simulación con periodo de muestreo T=0.001 en la que se varía la consigna de forma suave desde 0 hasta 5. En este caso los parámetros de la función de costo son Q=10, R=0, S=0. Se puede comprobar como el sistema sigue la consigna con error prácticamente nulo. Otra simulación con consigna variable es la que se

muestra en la figura 5.11, donde se desea que el sistema siga una sucesión de escalones. Los resultados obtenidos con Q=100, R=1e-3 y S=0 y T=0.01 son igualmente positivos.

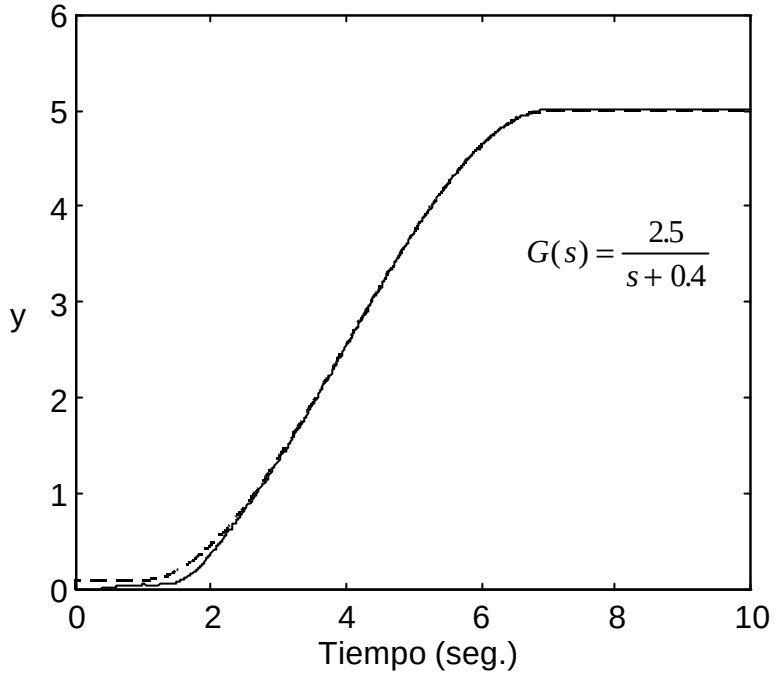


Figura 5.10. Control neuronal adaptivo con consigna variable .

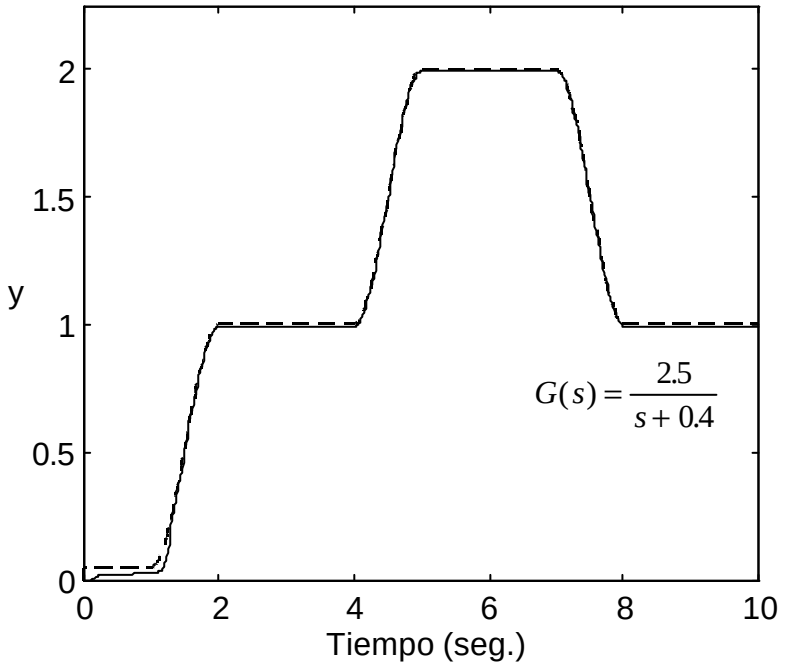


Figura 5.11. Control neuronal adaptivo con consigna variable.

En la figura 5.12 se muestra una aplicación de este controlador a una planta de fase no mínima con T=0.01. Puede observarse que el sistema después de una etapa de entrenamiento inicial, en la que muestra un comportamiento indeseable, evoluciona hasta la consigna de forma suave y sin sobrepasamiento.

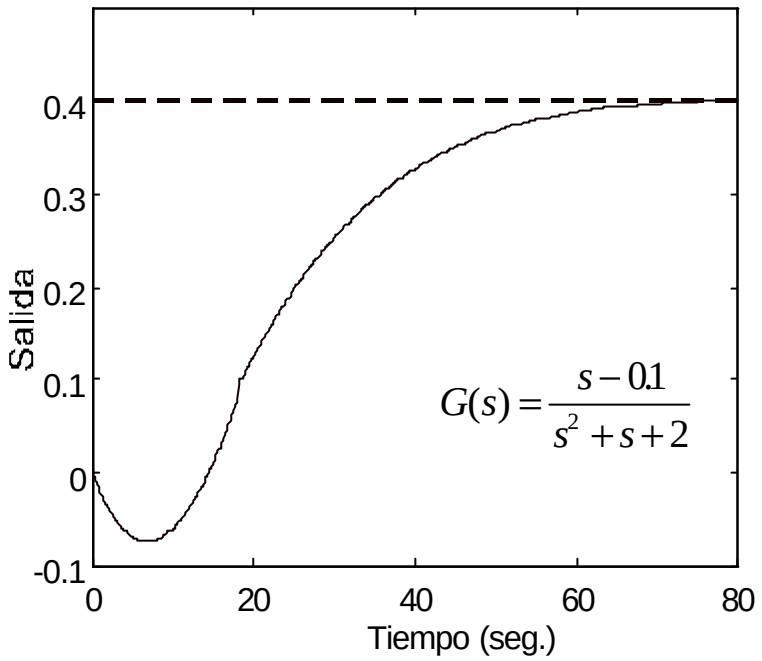


Figura 5.12. Control neuronal adaptivo variable aplicado a una planta de fase no mínima.

Por último, se presenta una experiencia donde se aplica este esquema a una planta inestable. Esta planta tiene su polo inestable en s = +0.4. En las mismas condiciones que en las experiencias anteriores, se ha aplicado la estructura de control neuronal adaptivo, considerando una consigna variable y un periodo de muestreo igual a T=0.01. En la figura 5.13 se presenta la evolución de la salida para este sistema. Se puede observar como la salida consigue un seguimiento con error casi nulo de la señal de referencia. La evolución de los parámetros del controlador, k_p y k_i, se muestra en las figuras 5.14 y 5.15, respectivamente.

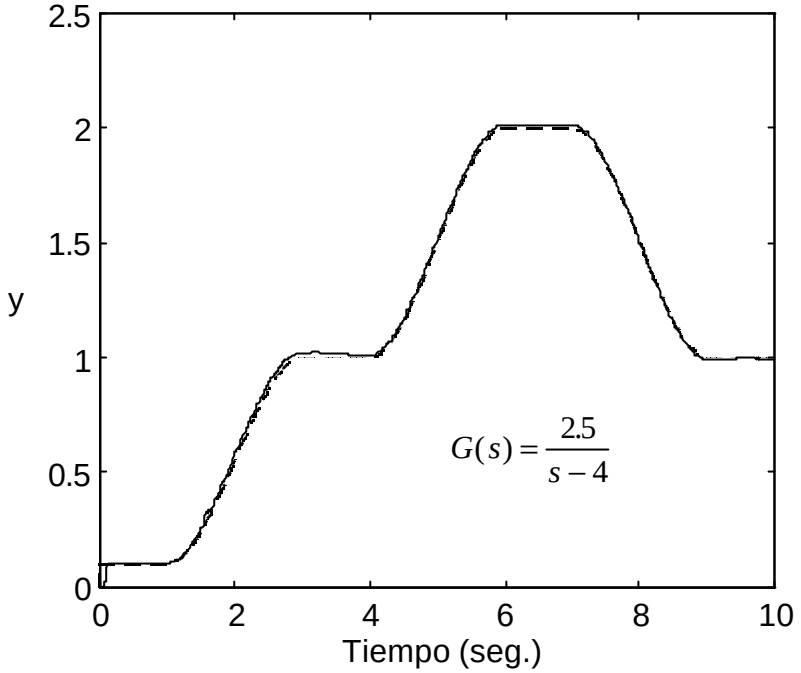


Figura 5.13. Control neuronal adaptivo con consigna variable aplicado a una planta inestable.

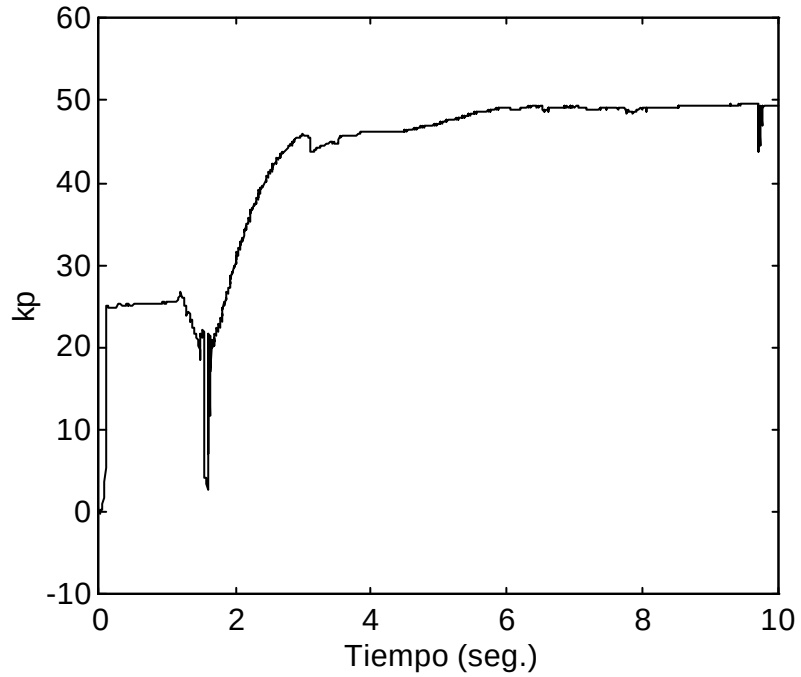


Figura 5.14. Evolución de k_p para la experiencia de la figura 5.12.

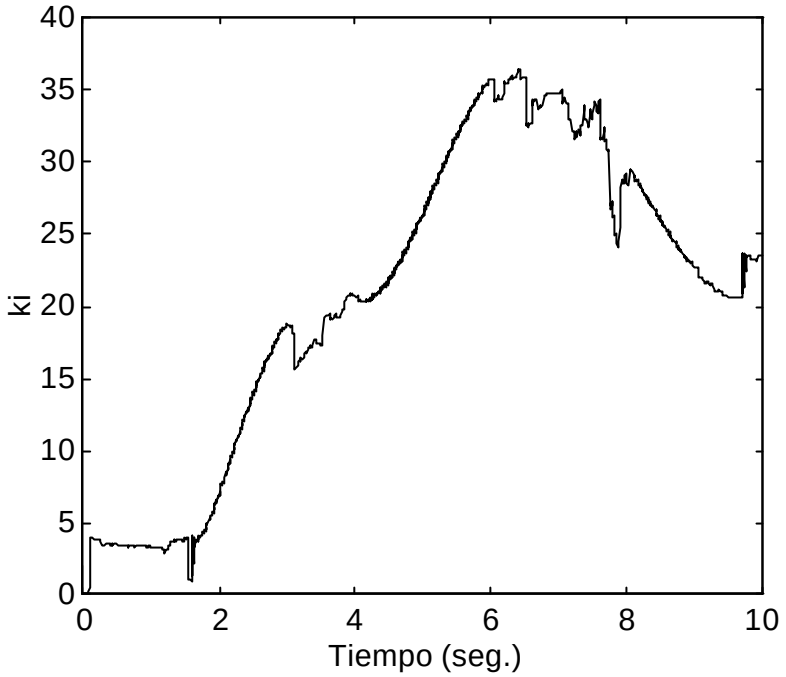


Figura 5.15. Evolución de k_i para la experiencia de la figura 5.12.

5.4 APLICACIÓN A UNA PLANTA NO-LINEAL: EL PROBLEMA DE LA GRÚA.

5.4.1 DESCRIPCIÓN DE LA PLANTA

Para la aplicación del controlador neuronal se ha elegido un sistema no lineal muy frecuente en la industria. La planta es un puente de grúa. En la figura 5.16 se representa un diagrama esquemático de esta planta. La tarea de la grúa es transportar contenedores desde un punto a otro en un espacio unidimensional.

Existen cuatro grados de libertad en este sistema, la posición del vagón (x), la velocidad del vagón ($\dot{x}$), el ángulo de la cuerda (θ) y su velocidad angular ($\dot{\theta}$). Se supone que la masa del vagón es M, la longitud de la cuerda l y la masa de la carga de transporte es m. Se asumirá que la cuerda tiene una masa despreciable.

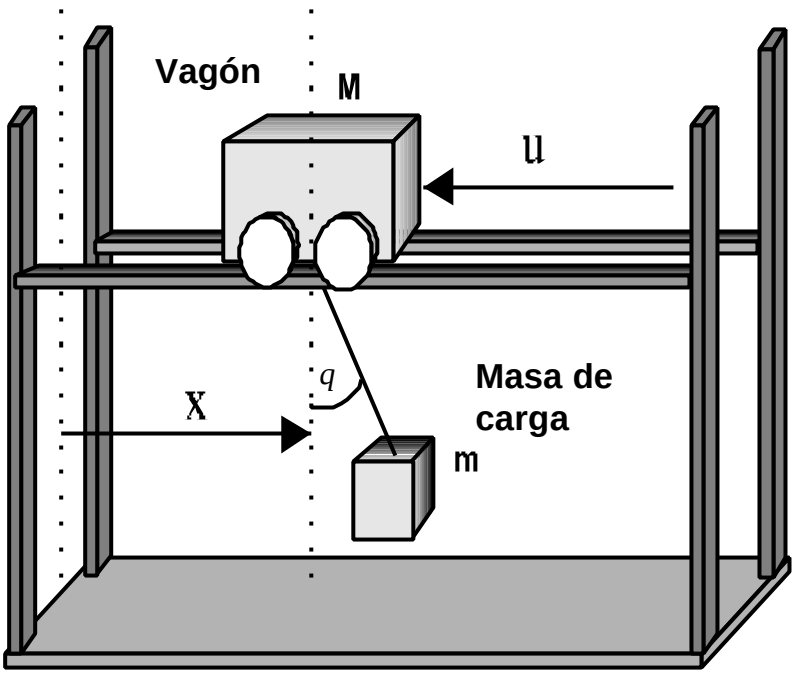


Figura 5.16. Vista esquemática de la grúa.

La entrada de la planta, u, representa la fuerza horizontal aplicada al vagón. La aplicación de una fuerza distinta de cero sobre la planta produce oscilaciones indeseables en la masa de carga. El objetivo de control es hacer que el vagón alcance el punto final de la trayectoria con θ y $\dot{\theta}$ igual a cero. Definiendo las variables de estado $x_1 = x$, $x_2 = \dot{x}$, $x_3 = \theta$ y $x_4 = \dot{\theta}$, la dinámica de la planta puede ser descrita por medio de las siguientes dos ecuaciones diferenciales de segundo orden

$$\left.\begin{aligned} ml(\dot{x}_3\cos x_3 - x_4^2\sin x_3) + (M+m)\dot{x}_1 &= u \\ \dot{x}_1\cos x_3 + l\dot{x}_3 &= -g\sin x_3 \end{aligned}\right\} \tag{5.23}$$

La primera ecuación describe el movimiento horizontal del vagón y la segunda el movimiento de la masa de carga.

5.4.2 CONTROLADOR NEURONAL ADAPTIVO PARA LA GRÚA

El objetivo del controlador es llevar el vagón desde un punto inicial a un punto final deseado, de forma que cuando se alcance ese punto la masa de carga deje de oscilar, es decir que llegue con ángulo de oscilación igual a cero y con velocidad angular cero. En términos de

las variables de estado, el objetivo de control consiste en llevar a la grúa desde el estado inicial $x_1 = x_{1i}$, $x_2 = 0$, $x_3 = 0$, $x_4 = 0$, al estado final definido por $x_1 = x_{1ref}$, $x_2 = 0$, $x_3 = 0$ y $x_4 = 0$. Para esta planta un simple controlador como un PID, no es capaz de cumplir el objetivo de control debido a la especificación multivariable impuesta de regular tanto el vagón como la carga. En este trabajo se utiliza como controlador un regulador LQ actuando sobre las variables de estado x_2 , x_3 y x_4 y sobre la variable de error $e_1 = x_{1ref} - x_1$. Así, el comando aplicado es

$$u(k) = -\begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix} \begin{bmatrix} e_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{5.24}$$

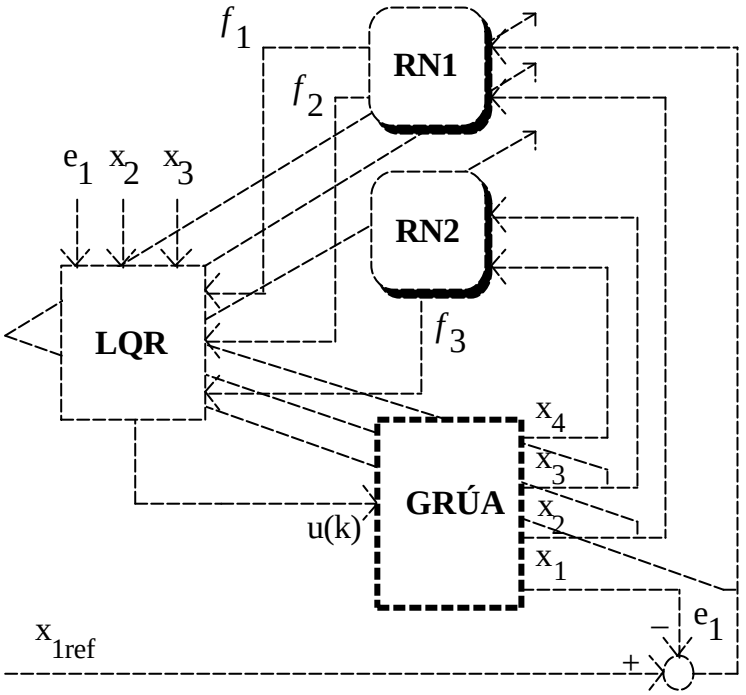


Figura 5.17. Estructura del controlador neuronal self-tuning.

Para simplificar el entrenamiento de las redes en las simulaciones llevadas a cabo se ha despreciado la realimentación a través de la variable de estado x_4. Con esta suposición, el esquema del controlador es el que se muestra en la figura 5.17. Como se observa, existen tres parámetros para sintonizar: la ganancia f_1 actuando sobre la variable e_1 y las ganancias f_2 y f_3 que actúan sobre las variables x_2 y x_3, respectivamente.

Para mejorar la convergencia de las redes neuronales, la estructura del sintonizador se ha dividido en dos subsistemas cada uno con una red neuronal. Una de las redes (NN1) se usa para sintonizar los parámetros relacionados con las variables del vagón, f_1 y f_2, y la otra red neuronal (RN2) se emplea para obtener los parámetros relacionados con las variables de la

masa de carga, f_3. Las entradas a la red RN1 son el error en x_1 y la derivada del error en x_1. En cuanto a la red RN2 tiene como entradas el error en x_3 y la derivada del error en x_3. La función de costo propuesta es

$$J(k) = \frac{1}{2} p_1(k) e_1(k)^2 + \frac{1}{2} p_2(k) x_2^2(k) + \frac{1}{2} p_3(k) x_3^2(k) + \frac{1}{2} p_4(k) x_4^2(k) \tag{5.26}$$

donde $p_i(k)$ son funciones variables con el tiempo que pesan las distintas variables de estado. Como se ve lo que se está penalizando en esta función es el error cometido en la variable x_1, y las desviaciones de las otras variables de cero. La dependencia temporal de las funciones de peso $p_i(k)$ permite en cada etapa modelar la función de costo y penalizar en cada instante la variable que interese.

El entrenamiento de las redes se realiza *on-line*. En cada etapa del proceso, las redes se entrenan para encontrar los valores óptimos de f_1, f_2 y f_3 que minimizan el índice (5.26). Para ello es necesario calcular el gradiente de la función de costo, que se obtiene de acuerdo con (5.4),

$$\frac{\P J(k)}{\P w_{Lji}} = \frac{\P J(k)}{\P q_j} \frac{\P q_j}{\P w_{L,j,i}}$$

donde la variable q_j representa la salida de la red. Teniendo en cuenta (5.26), el término $\P J(k) / \P q_j$ es

$$\frac{\P J(k)}{\P q_j} = \sum_{i=1}^{4} \frac{\P J(k)}{\P x_i} \frac{\P x_i}{\P u} \frac{\P u}{\P q_j} \tag{5.27}$$

teniéndose que

$$\frac{\P J(k)}{\P x_1} = - p_1(k) e_1(k); \tag{5.28a}$$

$$\frac{\P J(k)}{\P x_2} = p_2(k) x_2(k); \tag{5.28b}$$

$$\frac{\P J(k)}{\P x_3} = p_3(k) x_3(k); \tag{5.28c}$$

$$\frac{\partial J(k)}{\partial x_4} = p_4(k)x_4(k)\,. \tag{5.28d}$$

Los términos $\partial x_i / \partial u$ se pueden aproximar por

$$\frac{\partial x_i}{\partial u} \approx \frac{x_i(k) - x_i(k-1)}{u(k-1) - u(k-2)} \tag{5.29}$$

En cuanto al último factor de la expresión (5.27), se obtiene directamente de la expresión del controlador. Así, se encuentra que

$$\frac{\partial u(k)}{\partial f_1} = e_1(k) \tag{5.30a}$$

$$\frac{\partial u}{\partial f_2} = x_2(k); \tag{5.30b}$$

$$\frac{\partial u}{\partial f_3} = x_3(k). \tag{5.30c}$$

Partiendo de un instante k, el algoritmo puede resumirse de la siguiente forma:

Paso 1) LEER $x_1(k)$, $x_2(k)$ $x_3(k)$ y $x_4(k)$

Paso 2) (Entrenamiento de las redes neuronales)

Entrenamiento de la red RN1:

$$\frac{\partial u(k)}{\partial f_1} \leftarrow e_1(k)$$

$$\frac{\partial u(k)}{\partial f_2} \leftarrow x_2(k)$$

Backpropagation($\frac{\partial u(k)}{\partial f_1}$, $\frac{\partial u(k)}{\partial f_2}$; $e_1(k)$, $x_2(k)$; $p_i(k)|_{i=1,\ldots,4}$);

ACTUALIZAR f_1 y f_2

Entrenamiento de la red RN2:

$$\frac{\partial u(k)}{\partial f_3} \leftarrow x_3(k)$$

Backpropagation($\frac{\partial u(k)}{\partial f_3}$; $x_3(k)$, $x_4(k)$; $p_i(k)|_{i=1,\ldots,4}$);

ACTUALIZAR f_3

Paso 4) CALCULAR comando

$$u(k) = -\begin{pmatrix} f_1 & f_2 & f_3 \end{pmatrix}\begin{pmatrix} x_1(k) & x_2(k) & x_3(k) \end{pmatrix}^T$$

Paso 5) APLICAR $u(k)$ y ESPERAR hasta $t=(k+1)T$

Paso 6) $k \leftarrow k+1$

Paso 7) IR A *Paso 1*

5.4.3 RESULTADOS

Para poder contrastar los resultados obtenidos con el controlador neuronal, se presentan inicialmente los resultados obtenidos cuando se emplea un controlador self-tuning basado en un esquema convencional.

5.4.3.1 RESULTADOS CON UN CONTROLADOR BASADO EN UN SELF-TUNING EXPLÍCITO

En esta sección se hace un estudio de las prestaciones que ofrece un controlador self-tuning estándar. El controlador presenta la misma estructura que la que se muestra en la figura 5.1. Es decir, se implementará un algoritmo *self-tuning* explícito. La idea básica es linealizar el modelo en cada punto de la trayectoria, y combinar un proceso de identificación con una acción de control en cada etapa. El modelo lineal al que se aproxima la grúa en cada instante es:

$$\mathbf{x}(k+1) = \mathbf{F}(k)\mathbf{x}(k) + \mathbf{G}(k)u(k) \tag{5.31}$$

donde $\mathbf{x}=[x_1(k), x_2(k), x_3(k), x_4(k)]^T$, u es la entrada, y $\mathbf{F}$ y $\mathbf{G}$ son las matrices que definen la dinámica del sistema con dimensiones 4×4 y 4×1, respectivamente:

$$\mathbf{F}(k) = \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{pmatrix}; \qquad G(k) = \begin{pmatrix} g_{11} \\ g_{21} \\ g_{31} \\ g_{41} \end{pmatrix} \tag{5.32}$$

En cada instante es necesario determinar las componentes de estas dos matrices. Agrupando estos elementos, en cada instante se debe identificar la siguiente matriz:

$$\Theta(k)=\begin{pmatrix} f_{11}(k) & f_{21}(k) & f_{31}(k) & f_{41}(k) \\ f_{12}(k) & f_{22}(k) & f_{32}(k) & f_{42}(k) \\ f_{13}(k) & f_{23}(k) & f_{33}(k) & f_{43}(k) \\ f_{14}(k) & f_{24}(k) & f_{34}(k) & f_{44}(k) \\ g_{11}(k) & g_{21}(k) & g_{31}(k) & g_{41}(k) \end{pmatrix}=\left[q_1(k),q_2(k),q_3(k),q_4(k)\right] \quad (5.33)$$

Si se agrupa la salida y la entrada en un único vector,

$$\mathbf{z}^T(k)=[x_1(k),\, x_2(k),\, x_3(k),\, x_4(k),\, u(k)] \quad (5.34)$$

se puede escribir la ecuación (5.31) del sistema como

$$x_i(k+1)=\mathbf{z}^T(k)q_i(k); \qquad i=1,\ldots,4 \quad (5.35)$$

Para disponer de un índice que cuantifique la bondad de la identificación, se define el siguiente error de identificación,

$$e_i(k+1)=x_i(k+1)-\mathbf{z}^T(k)\hat{q}_i(k); \qquad i=1,\ldots,4 \quad (5.36)$$

Un esquema de identificación por mínimos cuadrados secuencial se basa en la minimización de un criterio de error ponderado exponencialmente como el siguiente

$$J_N=\sum_{j=1}^{N} r^{N-j}e_i^2(j); \quad (5.37)$$

donde N es el número de medidas utilizadas para estimar los parámetros $\hat{q}_i(N)$. Minimizando este error con respecto al vector de parámetros q_i se puede obtener el siguiente esquema de identificación recursivo en tiempo real:

$$\hat{q}_i(k+1)=\hat{q}_i(k)+g(k)\mathbf{P}(k)\mathbf{z}(k)[x_i(k+1)-\mathbf{z}^T(k)\hat{q}_i(k)]; \quad (5.38)$$

$$\mathbf{P}(k+1)=\mathbf{P}(k)-g(k)\mathbf{P}(k)\mathbf{z}(k)\mathbf{z}^T(k)\mathbf{P}(k), \quad (5.39)$$

donde

$$g(k) = [\mathbf{z}^T(k)\mathbf{P}(k)\mathbf{z}(k) + r]^{-1} \tag{5.40}$$

siendo ρ un factor de olvido $0 < r < 1$, aunque en la mayoría de las aplicaciones suele tomarse $0.9 < r < 1$.

Una vez identificada la planta, la acción de control en cada etapa se tomará como aquella que minimiza el índice:

$$J(k) = \frac{1}{2}[\mathbf{x}^T(k)\mathbf{Q}\mathbf{x}(k) + u^T(k)\mathbf{R}u(k)] \tag{5.41}$$

Por lo tanto, en cada instante la acción de control a aplicar es:

$$u = -\mathbf{k}\mathbf{x} \tag{5.42}$$

siendo

$$\mathbf{k} = -[\mathbf{R} + \bar{\mathbf{G}}^T(k)\mathbf{Q}\bar{\mathbf{G}}(k)]^{-1}\bar{\mathbf{G}}^T(k)\mathbf{Q}\bar{\mathbf{F}}(k)\,. \tag{5.43}$$

donde $\bar{\mathbf{F}}$ y $\bar{\mathbf{G}}$ las matrices resultantes de la identificación.

Siguiendo esta estrategia se han realizado diferentes simulaciones sobre la grúa. En la figura 5.18 se muestra, con trazo discontinuo, la respuesta en una de las experiencias. Como se observa, el resultado obtenido es aceptable ya que el sistema cumple con el objetivo de control establecido. El transitorio que presenta el sistema en la variable x_1 es satisfactorio aunque con un ligero sobrepasamiento. La variable x_3 presenta igualmente un comportamiento adecuado amortiguándose las oscilaciones en el punto destino como se exige en las especificaciones.

Cabe destacar en esta estrategia que para obtener un buen rendimiento del sistema en lazo cerrado, es determinante un buen ajuste de las matrices **R** y **Q** que intervienen en la función de costo. En caso contrario, la acción combinada de identificación y control puede provocar inestabilidad en el sistema.

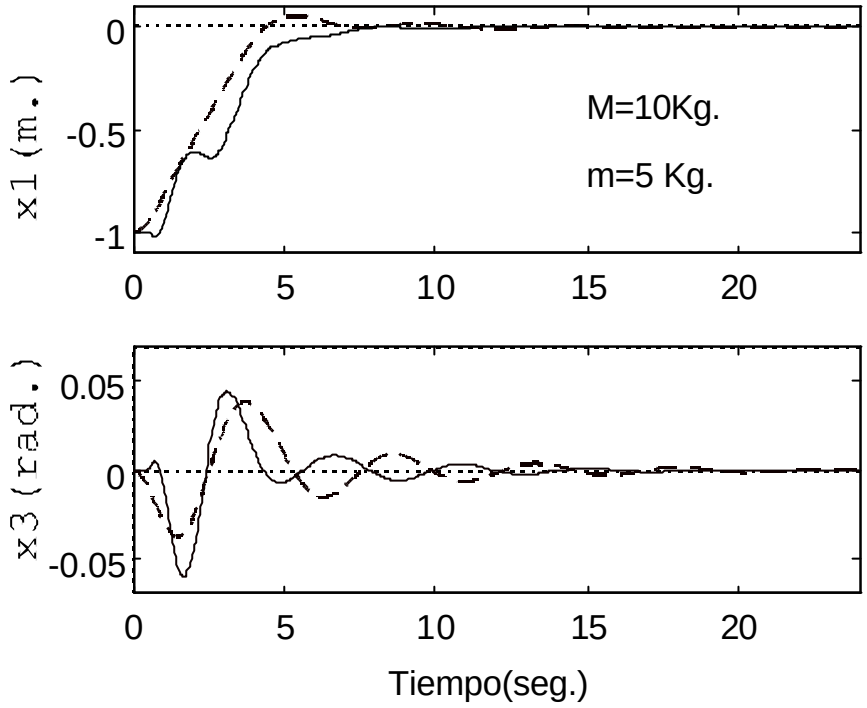


Figura 5.18. Control de la grúa. Línea continua:. Controlador self-tuning neuronal. Línea discontinua: Controlador neuronal adaptivo.

5.4.3.2 RESULTADOS CON EL CONTROLADOR NEURONAL ADAPTIVO

En la implementación del algoritmo las dos redes neuronales tienen la misma estructura: tres capas y tres nodos por capa en las capas ocultas. Las capas ocultas tienen una función de activación sigmoidal. Sin embargo, para la capa de salida se ha considerado una función de activación identidad. Los valores que toman los pesos $p_i(k)$ dependerán de la parte de la trayectoria en la que se encuentre la grúa. Sólo se ha considerado un cambio para estos pesos. Inicialmente se han elegido de forma que se pese principalmente las variables x_1 y x_2. Por el contrario, en la parte final de la trayectoria el peso debe recaer predominantemente sobre las variables x_3 y x_4.

Se han realizado diferentes simulaciones con este controlador tomando como intervalo de muestreo T=0.03 seg. En la figura 5.18 se muestra, con trazo continuo, la evolución de las variables x_1 y x_3 para una de estas experiencias. Se puede observar como la variable x_1 evoluciona desde un punto inicial x_1=-1 hasta x_1= 0, llegando suavemente y sin sobrepasamiento. Simultáneamente, la variable x_3 empieza a oscilar hasta que la grúa está cerca de la posición final. En este punto la oscilación de la masa de carga empieza a decrecer tendiendo hacia el punto de equilibrio x_3=0, x_4 =0. El comando aplicado a la grúa en esta experiencia se muestra en la figura 5.19. La evolución de los parámetros del controlador se puede observar en las figuras 5.20a), b) y c). Los coeficientes, que inicialmente toman valores pequeños, cambian adaptivamente a lo largo de la trayectoria hasta alcanzar posteriormente un estado estacionario.

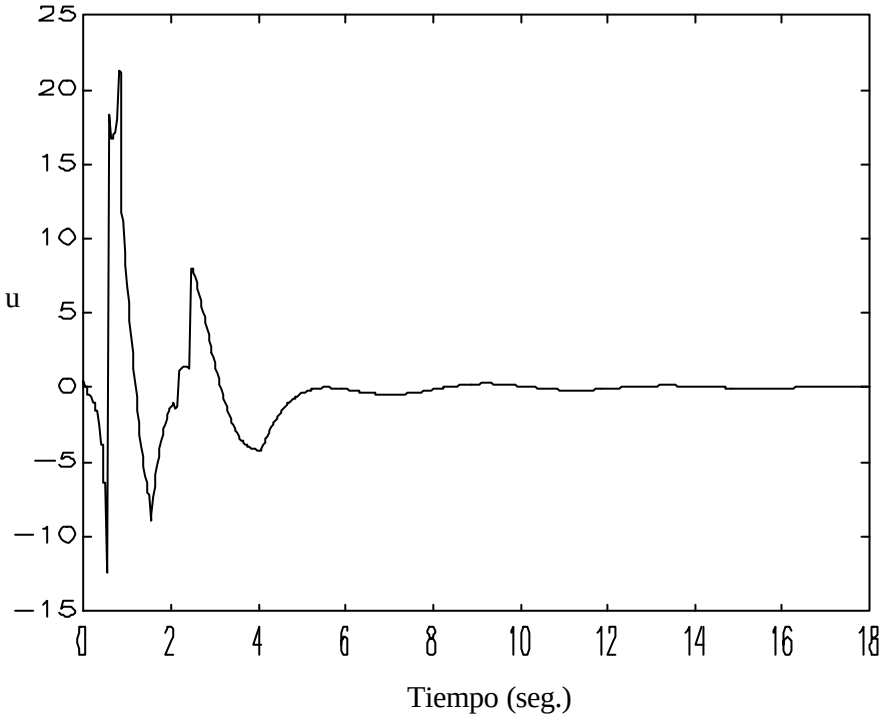


Figura 5.19. Evolución del comando aplicado en la experiencia que se muestra en la figura 17.

En la figura 5.18 se puede comparar la respuesta del controlador neuronal con la que produce el método clásico. Aparentemente, ambas estrategias tienen un rendimiento similar. Sin embargo, con el esquema neuronal, se observa un amortiguamiento algo mejor para la variable x_3 y una aproximación de x_1 a la consigna más suave.

Para verificar la robustez obtenida con uno y otro método en presencia de perturbaciones, se ha estudiado el comportamiento del sistema considerando un ruido de

medida en la variable x_3. La señal de ruido considerada se obtiene a partir de una distribución gausiana con media 0.0 y varianza 10e-6. La evolución de la variable x_3 bajo estas condiciones aparece en la figura 5.21. Ahora, se observa claramente una diferencia entre ambos métodos. En el controlador neuronal adaptivo las perturbaciones producen oscilaciones iniciales de mayor amplitud, pero las características de amortiguamiento son excelentes, asegurando una eliminación de las oscilaciones en el punto final (cuando la red sintoniza sus pesos). Por el contrario, el controlador self-tuning explícito presenta unas oscilaciones en x_3 muy poco amortiguadas que hacen que el vagón llegue a la consigna en x_1 con oscilaciones en la variable x_3. Esta situación reduce notablemente la eficiencia de este método.

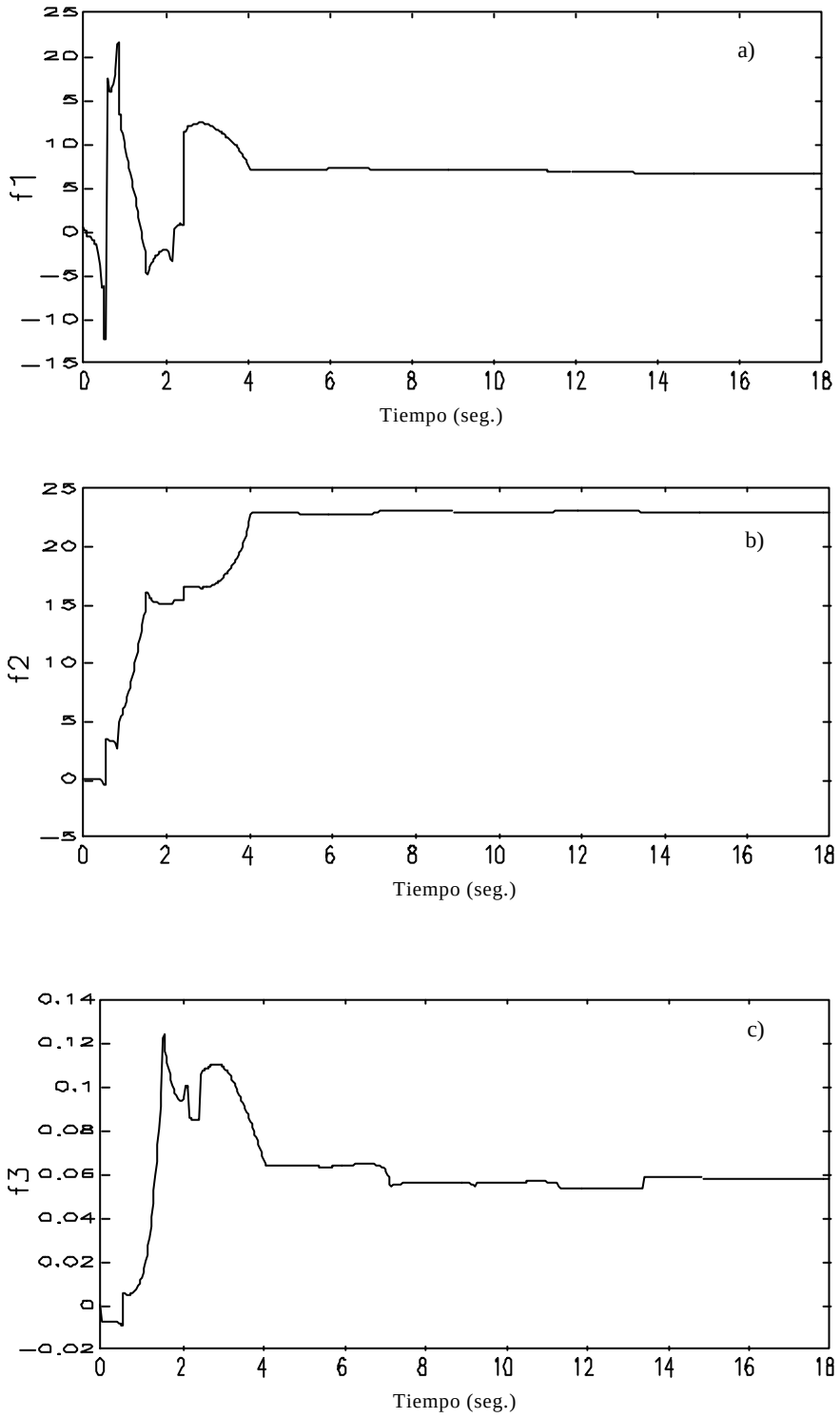


Figura 5.20. Evolución de los parámetros f_1 (a), f_2 (b) y f_3 (c) para una experiencia con M = 10Kg. y m = 5Kg.

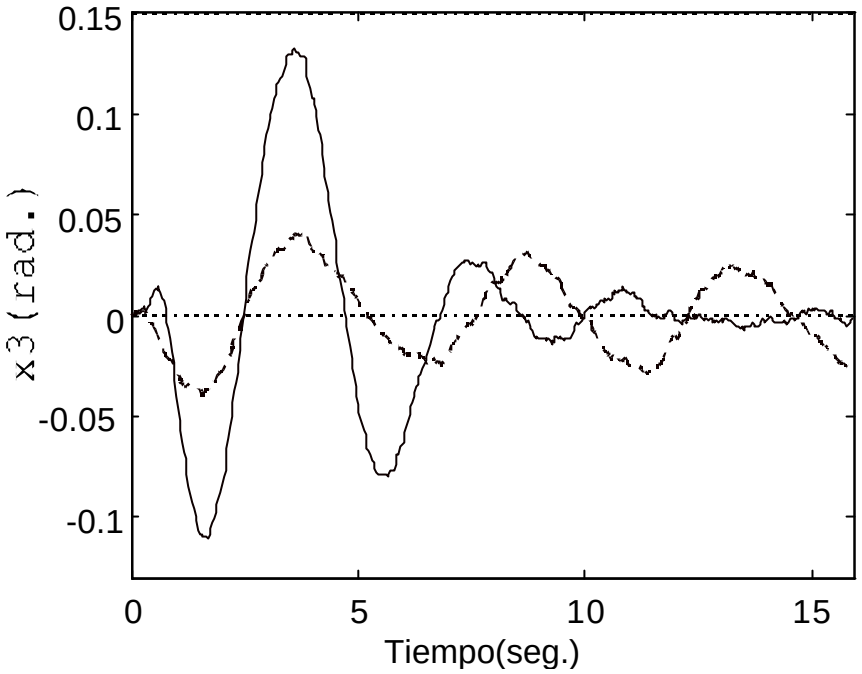


Figura 5.21. Comparación del controlador self-tuning neuronal (línea continua) y el controlador self-tuning convencional (línea discontinua) en presencia de perturbaciones.

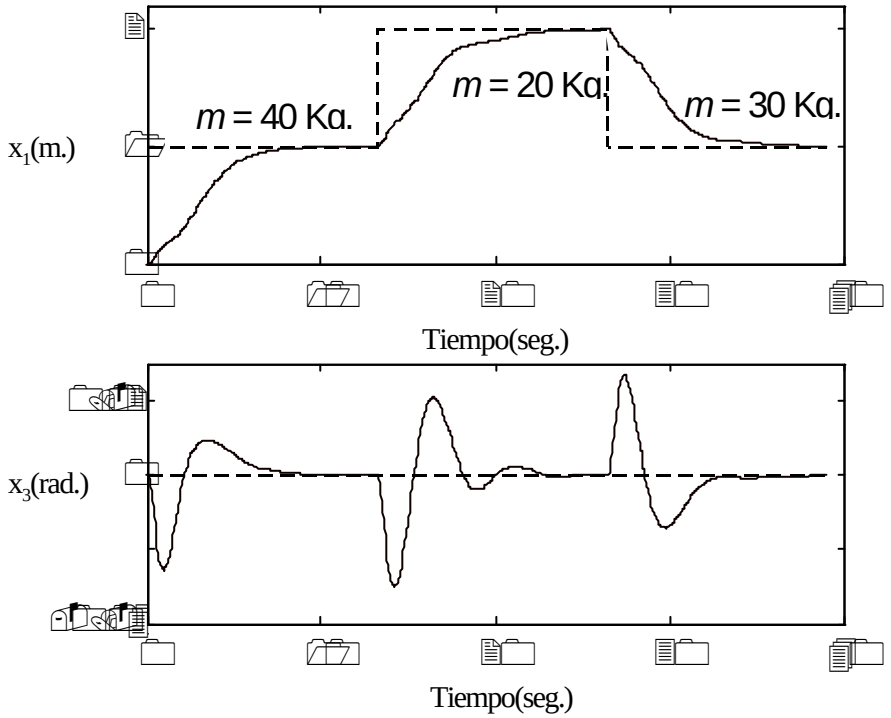


Figura 5.22. Evolución de la grúa en una tarea típica transportando cargas con diferente masa.

En la figura 5.22 se simula la grúa en una serie de tareas consecutivas de transporte de contenedores. La grúa tiene que moverse entre diferentes puntos transportando contenedores de diferente masa. En primer lugar, tiene que transportar un contenedor con masa m=40Kg.

desde un punto inicial x_1=0m. a otro punto x_1'=1m. En este punto, toma un segundo contenedor de masa m=20Kg. y lo transporta hasta un tercer punto x_1''=2m. Finalmente, la grúa transporta un tercer contenedor de masa m=30Kg. desde x_1'' hasta x_1'. Como se comprueba observando la figura 5.22 el comportamiento de la grúa en la realización de todas estas tareas es muy satisfactorio. La variable x_1 sigue a la señal de referencia con transitorios suaves y sin sobrepasamiento a lo largo de todas las tareas. Del mismo modo, la variable x_3 presenta un comportamiento muy bueno amortiguándose las oscilaciones en los puntos en los que x_1 alcanza el estado estacionario.

Un punto importante a considerar en la implementación de la aplicación en tiempo real de esta estrategia está relacionado con la fase de aprendizaje de las redes hasta que alcanzan la convergencia. En esta fase, el sistema puede mostrar un comportamiento indeseado que puede originar una pérdida importante de prestaciones. Una de las formas más sencillas para resolver este problema es llevar a la red hasta una región del espacio de pesos cercana al punto de convergencia final. Esto se puede lograr por medio de simulaciones previas realizadas *off-line* de las que se obtendrá un conjunto de pesos con las que se inicializaría la red.

El otro punto que puede ser problemático con vistas a una implementación real del método propuesto en este trabajo es el relacionado con los requerimientos de computación que exige el algoritmo. Para comprobar si esta estrategia es aplicable en tiempo real, se ha llevado a cabo un análisis computacional del tiempo requerido por el algoritmo. Este análisis ha consistido en medir el tiempo de computación total en cada intervalo de muestreo. Para hacer esto, se han implementado simulaciones del algoritmo en diferentes procesadores y se han medido estos tiempos. En la figura 5.23 se resumen los resultados obtenidos en las distintas plataformas. En la figura aparece marcado con una línea horizontal el periodo de muestro tomado en las simulaciones. Como puede observarse, para la serie de procesadores de menor potencia (PC-AT y PC-80386) el método sería inaplicable ya que el tiempo de cálculo es superior al periodo de muestreo fijado. Por el contrario, para procesadores PC-486 y superiores, es posible asegurar la aplicabilidad del método desde el punto de vista computacional ya que las operaciones de cálculo se pueden realizar dentro del periodo de muestreo establecido.

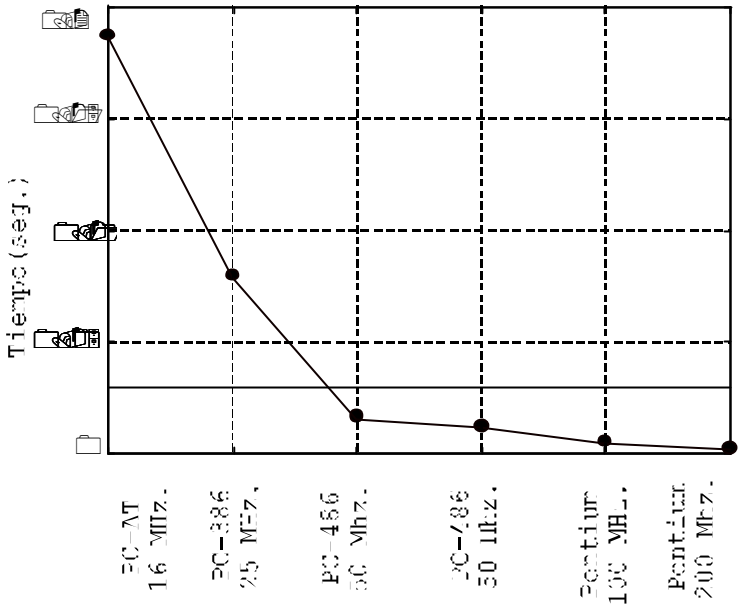


Figura 5.23. Tiempo de computación por intervalo de muestreo para diferentes procesadores.

5.5 IMPLEMENTACIÓN EN TIEMPO REAL

Con el objeto de realizar ensayos sobre una planta real, se ha diseñado y construido un prototipo a escala de un puente de grúa. Sobre él se pretende estudiar la viabilidad de la estrategia de control mencionada.

Lo primero que se ha planteado es la validación del modelo teórico propuesto. Una vez que se ha dispuesto de un modelo que se ajuste al de la planta real, se han realizado las simulaciones del algoritmo combinado de realimentación con variables de estado y de redes neuronales. Tras comprobar el funcionamiento de la estrategia, el siguiente paso ha sido trasladar estos resultados a la planta real. A continuación se resume todo el proceso de implementación.

5.5.1 DESCRIPCIÓN DE LA PLANTA

En la construcción de la planta se ha pretendido obtener un prototipo con características similares al que se ha comentado en la sección 5.4.1.

5.5.1.1 ESTRUCTURA

En la figura 5.24 se muestra una vista general de la planta. Como se aprecia , la estructura la forma un puente metálico de 1.2 m. de longitud con dos railes sobre los que se desplazará el vagón. Se incluye también una guía metálica cuya función es hacer que el vagón no descarrile.

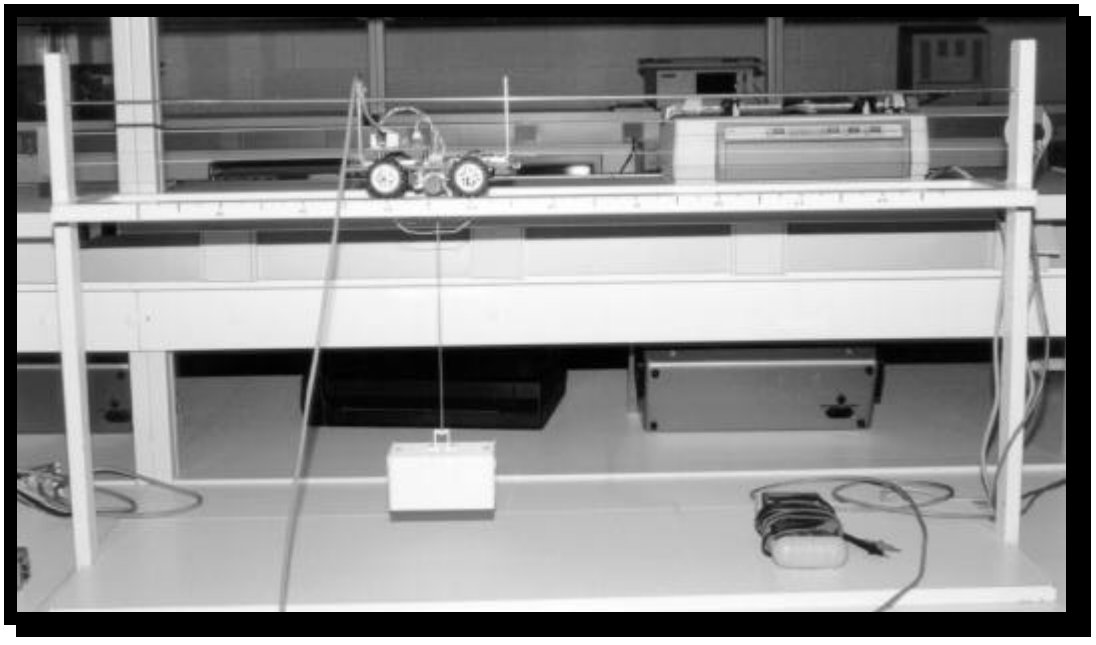


Figura 5.24. Vista general de la planta.

5.5.1.2 VAGÓN

El vagón de la grúa es un carro impulsado por dos ruedas motrices y que está guiado por una varilla para desplazarse por los railes de la estructura (ver figura 5.25 y 5.26). Para hacer girar las ruedas motrices se ha utilizado un servomotor de corriente continua acoplado directamente sobre cada rueda. Cada servomotor admite voltajes de entrada en el rango -9V. a +9V. El voltaje aplicado sobre un servomotor es proporcional a la velocidad angular de giro que alcanza el motor, presentando una velocidad angular máxima para una entrada de 9V. de 60 r.p.m. La masa de carga se halla sujeta al vagón por medio de una varilla metálica de masa despreciable. Los valores de la masa del vagón, del contenedor de carga con el que se ha trabajado y de la longitud de la varilla de carga son los siguientes:

- Masa del vagón: 0.3 Kg.
- Masa del contenedor: 0.15 Kg.
- Longitud de la varilla de carga: 0.5 m.

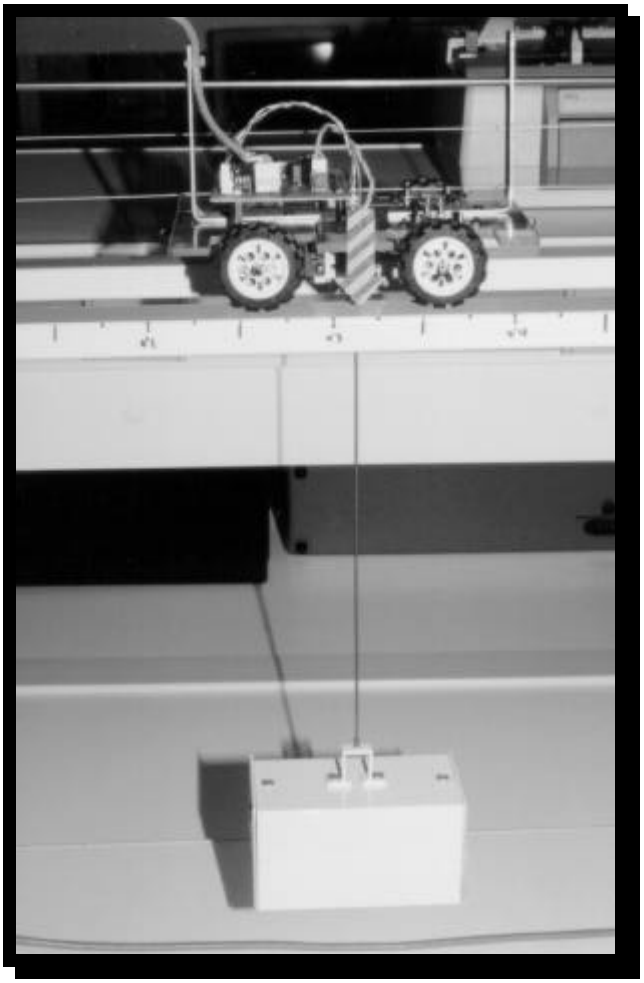


Figura 5.25. Vista del vagón y de la carga del puente de grúa.

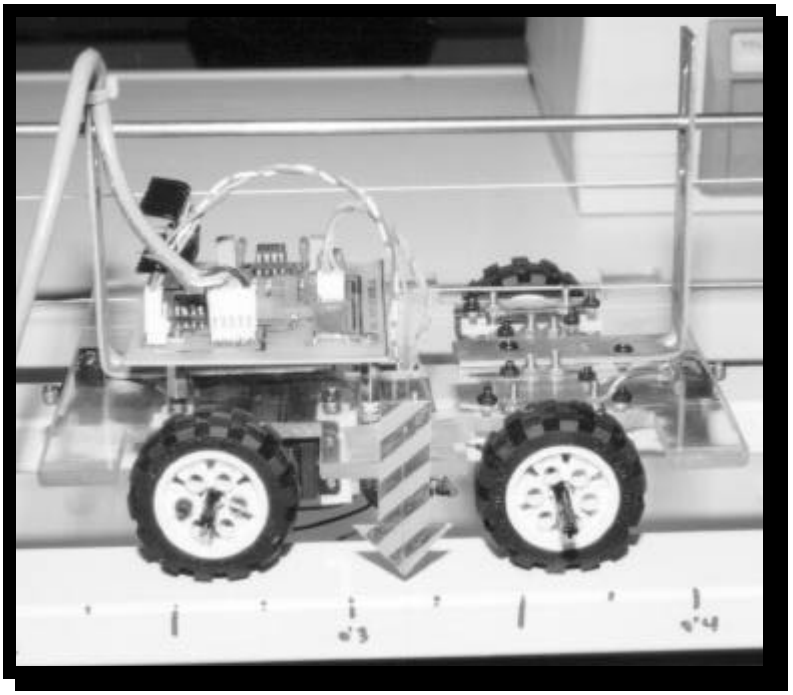


Figura 5.26. Primer plano del vagón

Para medir las variables del sistema se han instalado en la planta un sensor para medir el desplazamiento lineal del vagón, y otro para medir la posición angular de la masa de carga. El sensor de posición es un potenciómetro multivuelta acoplado a la estructura que gira a medida que el vagón avanza. De esta forma es posible disponer de un voltaje proporcional a la posición del vagón en cada momento. El sensor de ángulo es también potenciométrico y se encuentra en el centro del vagón acoplado al contenedor de carga de tal forma que el voltaje de salida que suministra sea una medida de la posición angular del contenedor. La conversión de voltios a metros y a radianes, respectivamente, se consigue sin más que hacer la transformación lineal correspondiente.

5.5.2 VALIDACIÓN DEL MODELO

El primer paso antes de llevar a cabo la implementación en tiempo real del controlador propuesto ha sido el de comprobar que el modelo teórico reproduce el comportamiento real del sistema. El modelo teórico propuesto es (5.23). Sustituyendo los valores de los parámetros del sistema, se tendría entonces

$$\left.\begin{aligned} 0.075(\ddot{x}_3\cos x_3 - x_4^2\sin x_3) + 0.45\ddot{x}_1 &= u \\ \ddot{x}_1\cos x_3 + 0.5\ddot{x}_3 &= -g\sin x_3 \end{aligned}\right\} \qquad (5.44)$$

Debe tenerse en cuenta que, tal como se dedujo el modelo, la entrada a la planta u representa la fuerza aplicada sobre el vagón en cada instante. Sin embargo, en el prototipo real se actúa mediante voltajes que resultan proporcionales a la velocidad de giro del motor, en lugar de a la aceleración, como se tendría en el caso de aplicar fuerzas directamente. Por lo tanto es necesario tener siempre presente la conversión de voltios a fuerza o viceversa. Esta conversión viene dada por la siguiente expresión

$$u(\text{Kgm}/\text{s}^2) = b\,\frac{du(\text{Volt.})}{dt};$$

$$b = 0.015\ (\text{Kgm/s}^2\text{Volt.})$$

Además de estas consideraciones hay que tener en cuenta que en el modelo propuesto tampoco se tienen en cuenta otras no linealidades que presenta la planta real como por ejemplo la limitación en los comandos, efectos de fricción mecánica, zonas muertas, etc. Sin embargo, de todas estas no linealidades la que presenta un efecto más apreciable es la zona muerta. Esta no linealidad se presenta para comandos bajos aplicados en la planta y se traduce en que el sistema no responde hasta superado un voltaje de aproximadamente 1.5V. Para comparar la respuesta del modelo con la de la planta real, ha sido necesario incluir este efecto al modelo.

Se han realizado diferentes pruebas en lazo abierto comparando la respuesta obtenida del modelo con la medida en la planta. Se ha considerado como entrada al sistema una rampa en voltios de pendiente constante. En las figuras 5.27 a 5.30 se muestran los resultados de dos experiencias realizadas. Las figura 5.27 y 5.28 corresponden a una experiencia considerando la pendiente de la rampa igual 2.75, mientras que en las figura 5.29 y 5.30 la pendiente de la rampa es igual 3.

Como puede apreciarse, el ajuste de la respuesta del sistema a la del modelo parece satisfactorio para las dos variables en ambas experiencias. Obsérvese que la respuesta medida cuando la pendiente es menor se ajusta ligeramente peor. Esto es debido a que con comandos menos activos las no linealidades no recogidas en el modelo tienen mayor influencia en la dinámica. También obsérvese el efecto antes mencionado de la zona muerta en la fase inicial del movimiento.

En virtud de estos resultados se puede considerar que el modelo (5.44) representa fielmente al puente de grúa construido.

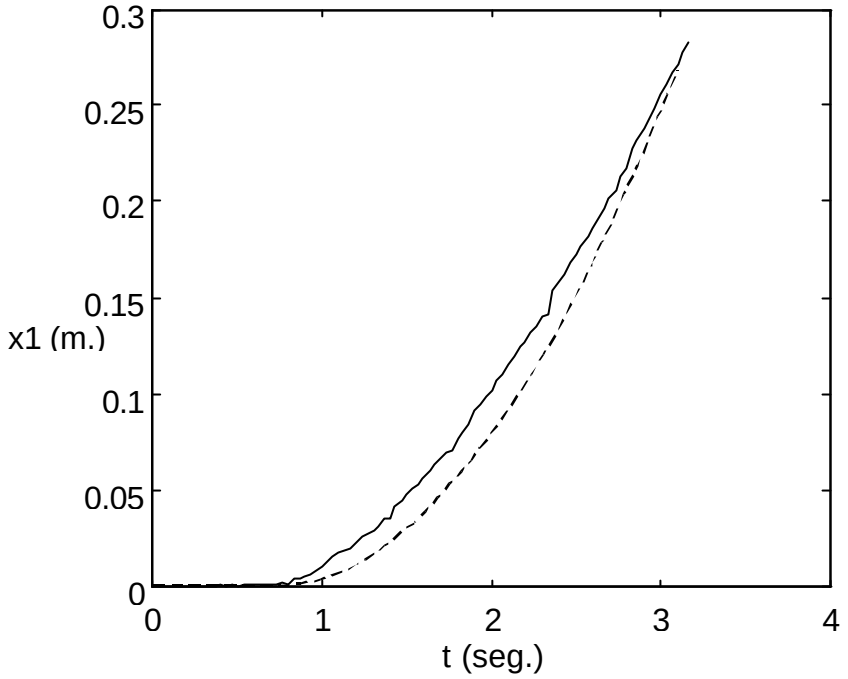


Figura 5.27. Respuesta del puente de grúa en lazo abierto ante una entrada de voltaje en rampa de pendiente 2.75. Evolución de la variable x_1.

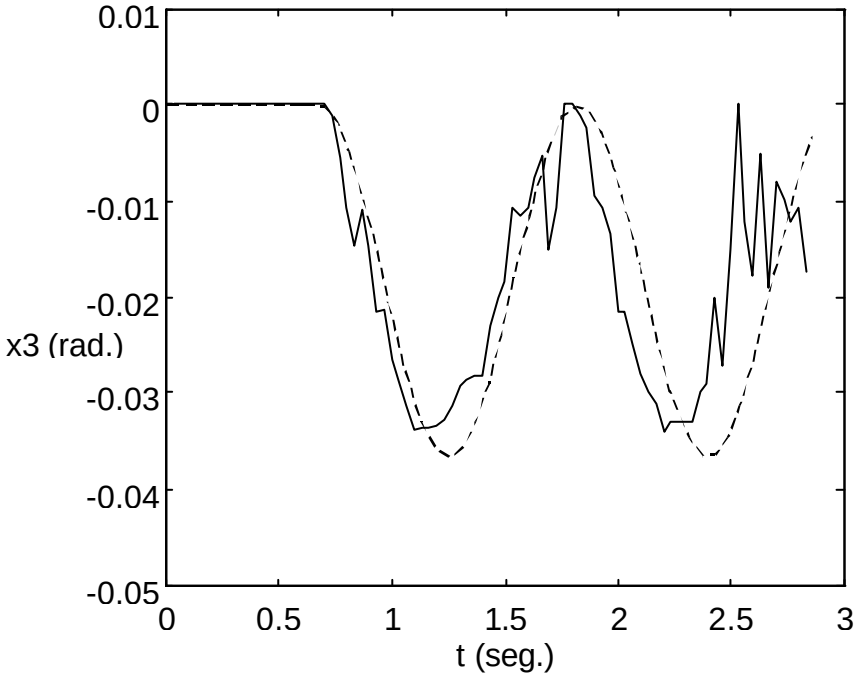


Figura 5.28. Respuesta del puente de grúa en lazo abierto ante una entrada de voltaje en rampa de pendiente 2.75. Evolución de la variable x_3.

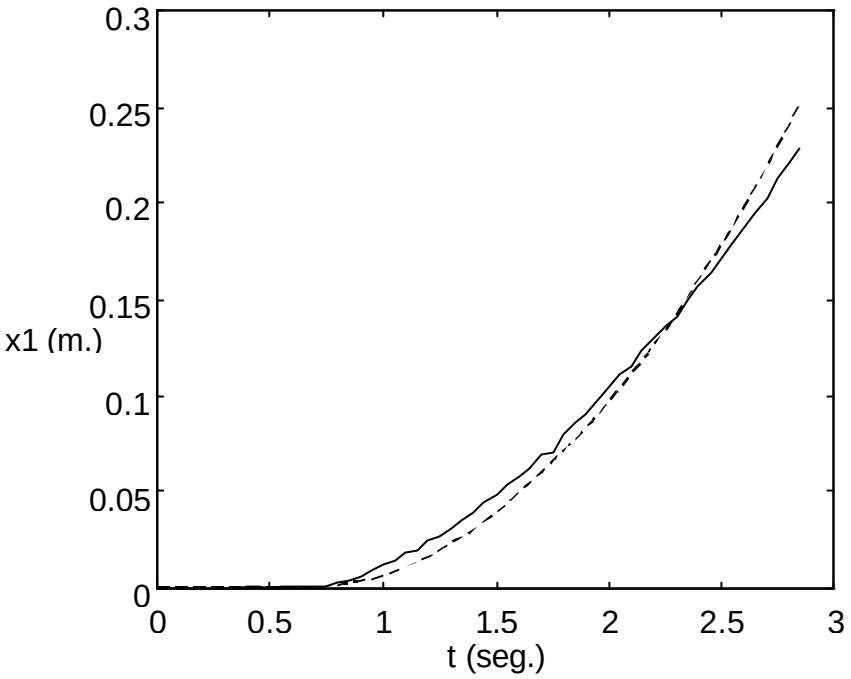


Figura 5.29. Respuesta del puente de grúa en lazo abierto ante una entrada de voltaje en rampa de pendiente 3. Evolución de la variable x_1.

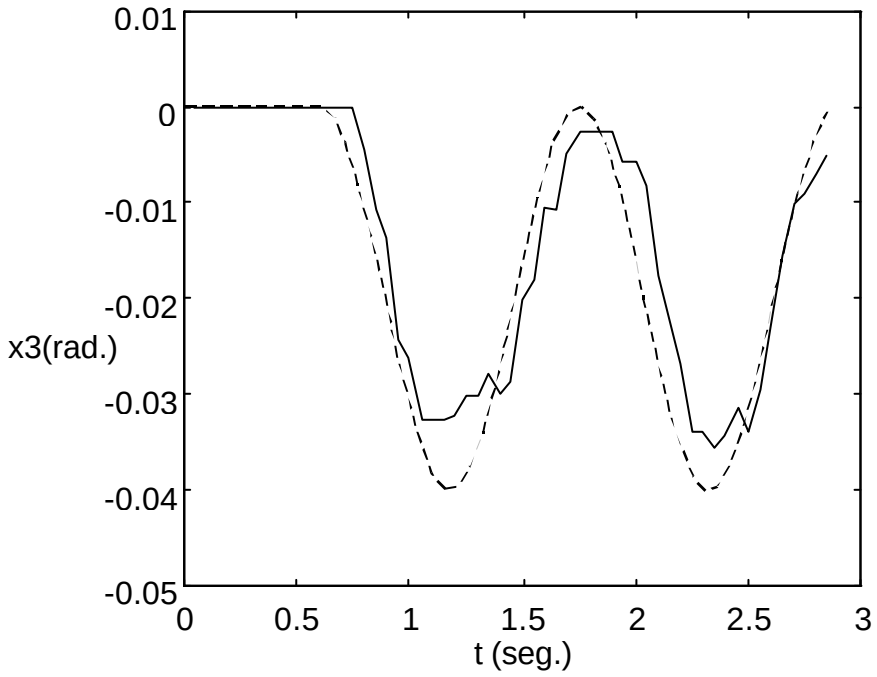


Figura 5.30. Respuesta del puente de grúa en lazo abierto ante una entrada de voltaje en rampa de pendiente 3. Evolución de la variable x_3.

5.5.3 RESULTADOS

El proceso que se ha seguido para estudiar la viabilidad de la estrategia sobre el prototipo ha sido el siguiente:

- Empleando el modelo de la planta, implementar en simulación la estrategia de control neuronal adaptiva.

- Tras haber sintonizado el controlador en simulación, utilizar los valores de los parámetros a los que convergen las redes en el instante final e incorporarlos al controlador sobre la planta real para llevar a cabo el control.

La simulación del controlador se ha realizado en idénticas condiciones que en la sección anterior. Es decir, se ha utilizado un perceptrón multicapa de tres capas y tres neuronas por capa en las capas ocultas que suministrará los parámetros del controlador (5.24): f_1, f_2 y f_3. En la figura 5.31 se muestra la evolución de las variables en una de las simulaciones desarrolladas donde se han tomado constantes los coeficientes en la función de peso: $p_1(k)$= 20, $p_2(k)$= 0.2, $p_3(k)$= 0.1 y $p_4(k)$= 0.

Como se observa, en esta simulación se consigue el objetivo de control planteado ya que la variable x_1 alcanza el valor de consigna y la x_3 después de unas oscilaciones iniciales se amortigua muy rápidamente llegando al punto de equilibrio en x_3=0. En esta simulación la convergencia de los parámetros se ha producido en los valores f_1= -8, f_2= 36, f_3 = -46. En las figuras 5.32 a) b) y c) se muestra la evolución de estos parámetros para esta experiencia. Obsérvese como los parámetros van evolucionando adaptativamente a la largo de la trayectoria hasta converger en un valor determinado.

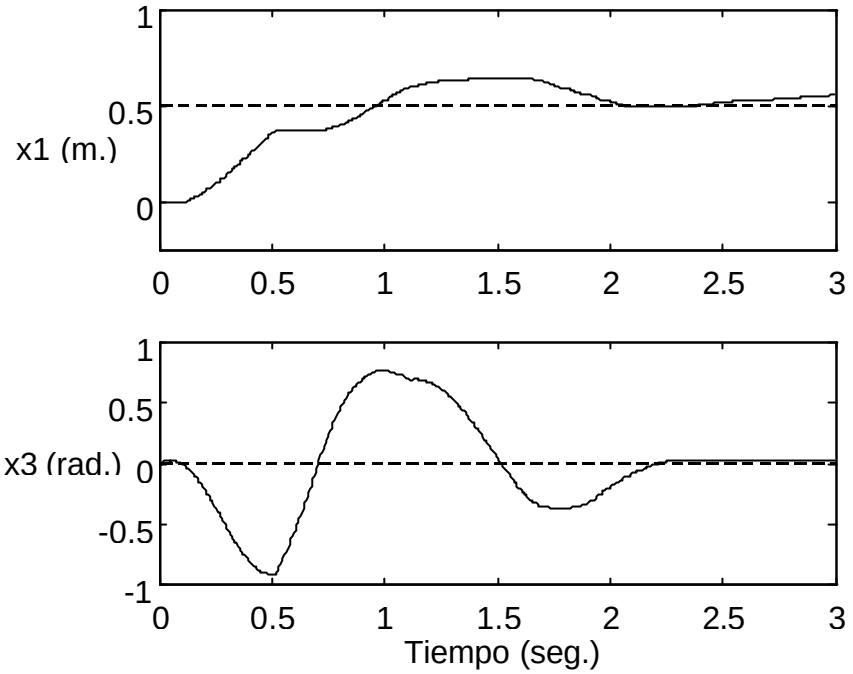
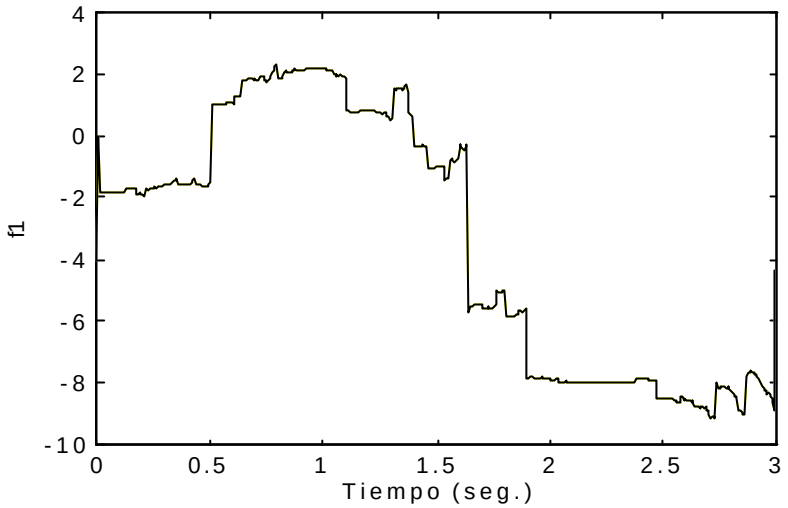


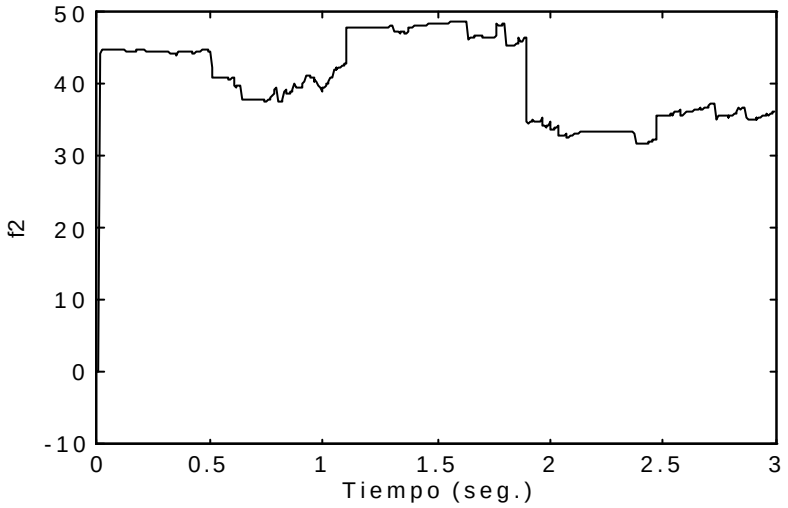
Figura 5.31. Simulación del controlador adaptivo neuronal con el modelo de la planta real.

Una vez implementado en simulación el algoritmo, se han tomado los valores de las constantes f_i a los que converge la red y se han utilizado en el controlador sobre la planta real. Se han realizado diferentes experiencias en estas condiciones. Algunos de los resultados obtenidos se muestran en las figuras 5.33, 5.34 y 5.35.

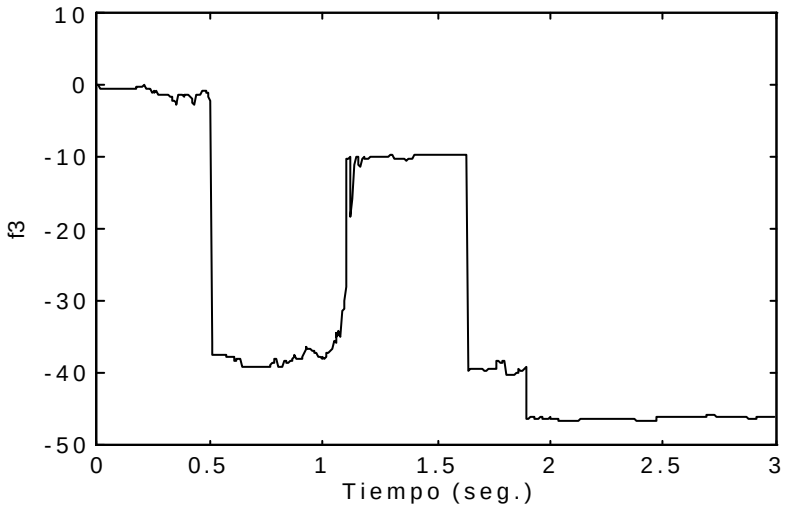
Como puede observarse la implementación sobre la planta real también ha sido satisfactoria ya que el sistema alcanza la consigna llegando sin apenas oscilación en la masa de carga. Obviamente, en las curvas que se muestran aparecen pequeños errores en la variable x_1 debido a no linealidades que no se han tenido en cuenta en el algoritmo. En cuanto a la variable x_3 se comprueba que está afectada de un ruido de medida considerable, pero sin embargo se consigue el objetivo de suprimir el balanceo cuando el sistema alcanza la consigna. En la figura 5.34 se presenta la evolución del comando en una de las experiencias apreciándose que éste permanece con valores admisibles durante toda la trayectoria.



(a)



(b)



(c)

Figura 5.32 Evolución de los parámetros f_1, f_2 y f_3 en la simulación del controlador sobre la planta real que se muestra en la figura 5.31.

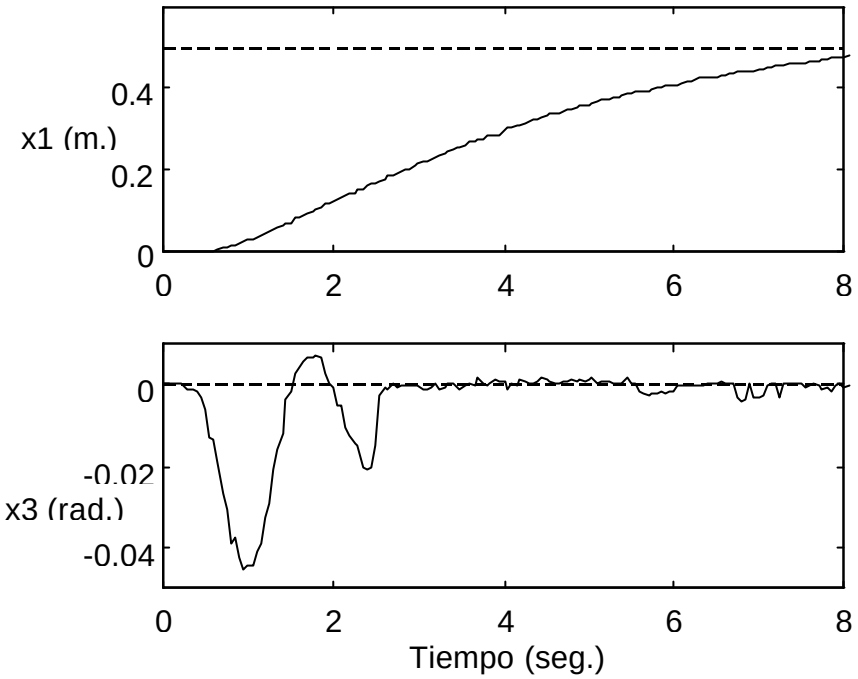


Figura 5.33. Evolución de las variables x_1 y x_3 en un control en tiempo real realizado sobre la grúa con los parámetros del controlador (5.24) ajustados mediante redes neuronales.

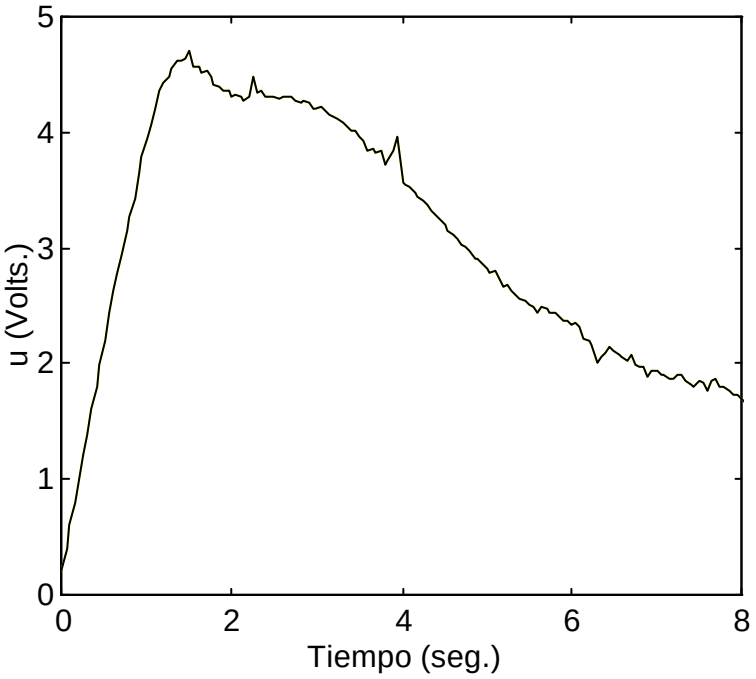


Figura 5.34. Evolución del comando en la experiencia que se muestra en la figura 5.33

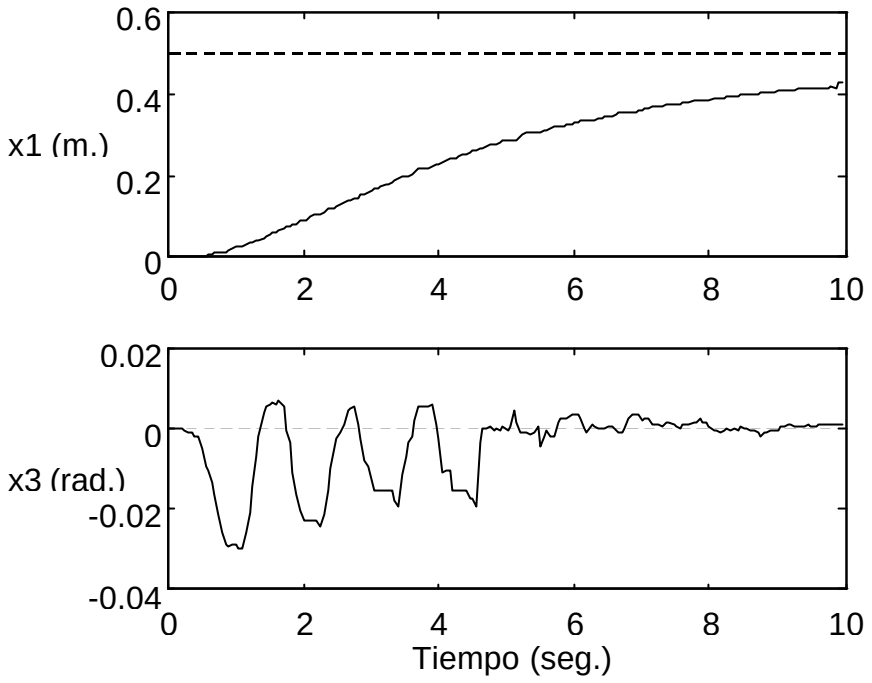


Figura 5.35. Evolución de las variables x_1 y x_3 en un control en tiempo real realizado sobre la grúa con los parámetros del controlador (5.24) ajustados mediante redes neuronales.

Como conclusión, cabe decir que los resultados obtenidos con el controlador neuronal *self-tuning* son satisfactorios. La utilización de las redes neuronales permite ajustar los parámetros del controlador sin necesidad de incluir bloques de identificación y de diseño que son habitualmente empleados en los controladores clásicos. Una de las principales ventajas que aporta el hecho de incluir redes neuronales es que no es necesario partir de un modelo paramétrico para el sistema. Las pruebas realizadas ajustando los parámetros de un controlador PI demuestran la aplicabilidad del método.

Como una aplicación más avanzada se han estudiado las prestaciones que ofrece este controlador en la resolución del problema de la grúa.. Diferentes experiencias desarrolladas ponen de manifiesto un comportamiento muy positivo del controlador neuronal propuesto. Este hecho se corrobora a través de una comparación con un controlador *self-tuning* clásico. Esta comparación muestra como el esquema neuronal es capaz de obtener una respuesta superior al método clásico. Como puntos débiles de la estrategia cabe reseñar la respuesta en las primeras etapas de entrenamiento de las redes. En esta fase se pueden observar comportamientos indeseados en la respuesta del sistema. Sin embargo, es posible mejorar

esta respuesta si se inicializan adecuadamente los pesos de la red. Como ventajas respecto al método clásico, se encuentra el hecho de que no es necesario emplear técnicas de linealización de la planta para controlarla, y sobre todo que el diseño conceptual es más simple que el que rodea a un controlador adaptativo clásico.

En cuanto a la implementación, los resultados obtenidos sobre el prototipo a escala nos permiten corroborar la eficiencia de la estrategia sobre esta planta.

6

EL PROBLEMA DE LOS HORIZONTES DE PREDICCIÓN EN EL GPC. SINTONIZACIÓN MEDIANTE REDES NEURONALES

Una de las cuestiones relacionadas con los controladores MBPC para la que aún no hay propuesta una solución eficiente es la de cómo elegir los horizontes de predicción para que un controlador GPC presente un rendimiento óptimo. A pesar de que algunos autores han dado una serie de criterios generales para esta elección, estos resultan insuficientes. En este capítulo se plantea un esquema que permite abordar este problema. La idea es emplear un controlador neuromórfico, con la misma estructura que el que se ha empleado en el capítulo 5.

En los primeros apartados del capítulo se exponen algunos criterios generales bien conocidos para la elección de los parámetros del GPC, comprobando como ante una consigna constante el rendimiento del controlador aumenta al aumentar el horizonte de salida. Se aborda también el problema de seguimiento de consignas variables. Con algunos ejemplos se pone de manifiesto la dificultad que aparece en la elección de un buen horizonte de predicción. En el siguiente apartado se hace un análisis de la respuesta de sistemas de segundo orden con un controlador GPC. En la última parte del capítulo se presenta la solución propuesta en este trabajo para resolver el problema de

la elección de los parámetros del controlador. Ésta consiste en el empleo de redes neuronales para la sintonización del horizonte.

6.1 PRESENTACIÓN DEL PROBLEMA

La elección de los parámetros N_1, N y NU en la función de costo juega un papel importante para determinar el nivel de prestaciones de un controlador GPC. Aspectos tales como los de estabilidad, error en seguimiento de consignas, carga computacional, etc., tienen una relación directa con la elección de estos parámetros. Ya desde 1987, cuando se propuso el controlador GPC, Clarke *et al* señalaban la importancia de la elección de estos parámetros y dieron algunos criterios cualitativos para una elección correcta.

La idea general para una elección acertada de los horizontes en el controlador GPC es que a medida que se aumentan los horizontes de predicción, las prestaciones del controlador aumentan, mejorando los transitorios de la respuesta. De hecho, cuando el horizonte de salida tiende a infinito, la estrategia de control coincide con la de un controlador LQ. Este resultado parece indiscutible cuando se abordan problemas con consigna constante. Sin embargo, si la referencia que se le presenta al sistema es una consigna variable, al aumentar el horizonte de salida N, existen situaciones en las que se puede observar una disminución en las prestaciones del sistema. Para estos casos, parece no ser tan sencillo la elección de este parámetro. A pesar de que existen algunos criterios que facilitan la elección de los horizontes de predicción, en determinadas ocasiones, como por ejemplo cuando se tienen consignas variables, estos criterios pierden eficacia. La solución que se plantea en este trabajo es plantear un esquema adaptativo para ajustar los horizontes por medio de redes neuronales.

6.2 CRITERIOS GENERALES PARA LA ELECCIÓN DE LOS PARÁMETROS N_1, N Y NU EN LA FUNCIÓN DE COSTO

El diseño de un controlador GPC exige fijar a priori los valores para N_1, N y NU en la función de costo:

$$J(N_1, N, NU) = \sum_{j=N_1}^{N} g(j)\left[w(t+j) - \hat{y}(t+j|t)\right]^2 + \sum_{j=1}^{NU} l(j)\left[\Delta u(t+j-1)\right]^2 \qquad (6.1)$$

Recuérdese que N_1 es el mínimo horizonte de costo, N es el máximo horizonte de costo u horizonte de salida y NU es el horizonte de control. Es común que estos parámetros se ajusten haciendo uso de métodos de prueba y error en simulación. Sin embargo, existen unos criterios generales que ayudan a determinar los valores de los horizontes de predicción que producen un mejor comportamiento del sistema.

6.2.1 ELECCIÓN DEL HORIZONTE DE CONTROL NU

La elección de NU está relacionada con la complejidad que tenga la planta. Si el sistema que se quiere controlar no es demasiado complejo (no contiene polos inestables o cerca del límite de estabilidad y es de fase mínima) la elección de $NU = 1$, suele ofrecer buenos resultados. Un aumento en NU para este tipo de plantas provoca acciones de control más activas, pero no redunda en un aumento considerable en las prestaciones. De tal forma que, si se supera un determinado valor de NU, dejan de observarse diferencias en la respuesta del sistema.

La situación cambia cuando se trata con un sistema inestable o con polos cerca de la zona de inestabilidad, o bien el sistema es de fase no mínima. Para este tipo de sistemas el horizonte de control, NU, juega un papel importante en el diseño del controlador, observándose diferencias notables cuando se incrementa el valor de NU. Esto se puede justificar si se tiene en cuenta que, en general, el control de una planta estable y de fase no mínima ante una consigna escalón, previsiblemente presentará un comportamiento suave con una política de control que no sufre variaciones considerables entre una etapa y la siguiente. Sin embargo, cuando se trata de un sistema complejo, el seguimiento de una consigna escalón requiere la aplicación de una ley de control mucho más cambiante de una etapa a otra. Lo cual justifica la necesidad de horizontes de control mayores.

6.2.2 ELECCIÓN DEL HORIZONTE MÍNIMO DE COSTE N_1

La función de este horizonte en la función de costo es la de reducir los cálculos innecesarios en el algoritmo. El valor adecuado para N_1 debe ser igual al del retardo de la salida del sistema, d. Si se elige un valor de N_1 menor que d, entonces se estarán realizando cálculos intranscendentes, pues las salidas en esas etapas no se verán afectadas por el comando actual. Suele ocurrir que este retardo no está perfectamente identificado o es

variable. En estos casos, se debe tomar $N_1 = 1$ sin que ello tenga una influencia negativa en el controlador.

6.2.3 ELECCIÓN DEL HORIZONTE DE SALIDA N

El criterio general para la elección de este parámetro es que debe ser mayor que el grado del polinomio $B(q^{-1})$. De esta forma todos los estados contribuyen a la función de costo. En general, es deseable que en la función de costo se abarque toda la respuesta que será afectada por el valor del control actual. Por ello, se suele tomar N igual al tiempo de subida de la planta.

Como conclusión se puede extraer que para plantas sencillas, estables y de fase mínima, una elección de NU=1, N_1=1 y N igual al tiempo de subida del sistema suele producir un resultado satisfactorio en el control. Cuando la complejidad de la planta se incrementa, apareciendo términos inestables y de fase no mínima suelen ser necesarios valores mayores para NU para que el controlador estabilice la planta.

6.2.4 CRITERIOS PARA GARANTIZAR LA ESTABILIDAD EN LAZO CERRADO

Existen algunos principios que permiten fijar los horizontes de predicción más adecuados para el controlador. Algunos de estos principios se basan en teoremas que muestran que el GPC en determinadas condiciones se transforma en leyes de control standard. A continuación se enuncian tres de estos teoremas que permiten asegurar la estabilidad del sistema bajo determinadas condiciones. Estos teoremas se formulan considerando la representación de estado para la planta (1.23). Si se desprecia en este modelo la componente de ruido se tiene

$$A(q^{-1})\Delta(q^{-1})\,y(t) = B(q^{-1})\Delta(q^{-1})u(t) \tag{6.2}$$

que en el espacio de estados se puede expresar como

$$\mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{b}\Delta u(t)$$
$$y(t) = \mathbf{c}^T\mathbf{x}(t)$$

Teorema 1. El GPC resulta en un controlador de estado *dead beat* estable si se cumplen las dos condiciones siguientes:
i) El sistema (**A, b, c**) es completamente controlable y observable con dimensión de estado *n*.
ii) $N_1 = n$, $N \geq 2n\text{-}1$, $NU = n$ y $l = 0$.

Teorema 2. El sistema en lazo cerrado con un GPC es estable si se cumplen las dos condiciones siguientes:

i) El sistema (**A**,**b**,**c**) de orden *n* es estabilizable y detectable (ver apéndice D).
ii) $NU = N_1 \geq n$, $N\text{-}N_1 \geq n\text{-}1$ y $l = e \rightarrow 0$.

Teorema 3. El sistema en lazo cerrado con un GPC es estable si el sistema (**A**, **b**, **c**) es estabilizable y detectable y si se cumple alguna de las dos siguientes condiciones:

i) NU, $N \rightarrow \infty$ con $NU = N$ y $l > 0$.
ii) NU, $N \rightarrow \infty$ con $NU = N\text{-}n\text{+}1$ y $l = 0$, suponiendo que no hay ceros de la planta en el límite de estabilidad.

Existe otro criterio más genérico que se formula incluso para valores grandes de λ, y que presenta una mayor aplicabilidad que los teoremas anteriores. Este criterio se formula para facilitar le elección de los horizontes de predicción N_1 y N de forma que se pueda garantizar la estabilidad del sistema en lazo cerrado para valores altos de λ (Scattolini, 1990). Su formulación asume que la planta en lazo abierto es estable. Haciendo un análisis de la respuesta escalón del sistema, se deriva una regla simple para la elección de N_1 y N para valores grandes de λ.

Eliminando el término integral en (6.2):

$$A(q^{-1})y(t) = B(q^{-1})u(t-1) \tag{6.3}$$

La respuesta impulsional de esta planta será

$$y(t) = \sum_{k=1}^{\infty} g_j q^{-j+1} u(t-1) \tag{6.4}$$

Si la planta es asintóticamente estable se debe cumplir que $\lim_{k\to\infty} g_k = 0$, es decir, los coeficientes g_k tienden a cero cuando el valor de k aumenta. Además, se asumirá que

$$g_k = 0, \ \forall k > N. \tag{6.5}$$

Los coeficientes de la respuesta escalón de la planta son

$$s_k = \sum_{j=1}^{k} g_j$$

Por lo tanto, teniendo en cuenta (6.5) la ganancia de la planta resulta ser

$$m = \frac{B(1)}{A(1)} = \sum_{j=1}^{N} g_j = s_N \tag{6.6}$$

Como se vio anteriormente, la respuesta del sistema en el instante $t+j$ se puede poner como:

$$y(t+j) = G_j \Delta u(t+j-1) + F_j y(t) \tag{6.7}$$

donde $G_j = E_j B$ se obtienen de la ecuación diofántica (1.28). Teniendo en cuenta que G_j se puede expresar como

$$G_j(q^{-1}) = E_j(q^{-1})B(q^{-1}) = \sum_{i=1}^{j} s_i q^{-i+1} + q^{-j} G_j(q^{-1}) \tag{6.8}$$

Nótese que s_1, s_2, …, s_j, son los coeficientes de la respuesta escalón del sistema. Entonces se puede escribir

$$y(t+j) = F_j(q^{-1})y(t) + \sum_{i=1}^{j} s_i q^{-i+1} \Delta u(t+j-1) + G_j(q^{-1})\Delta u(t-1) \tag{6.9}$$

Si se tiene en cuenta que los incrementos de control proyectados $\Delta u(t+j), j > 0$, se suponen iguales a cero, entonces se puede escribir

$$\begin{pmatrix} y(t+N_1) \\ y(t+N_1+1) \\ \mathrm{M} \\ y(t+N) \end{pmatrix} = \begin{pmatrix} s_{N_1} \\ s_{N_1+1} \\ \mathrm{M} \\ s_N \end{pmatrix} \Delta u(t) + \begin{pmatrix} F_{N_1}(q^{-1}) \\ F_{N_1+1}(q^{-1}) \\ \mathrm{M} \\ F_N(q^{-1}) \end{pmatrix} y(t) + \begin{pmatrix} G_{N_1}(q^{-1}) \\ G_{N_1+1}(q^{-1}) \\ \mathrm{M} \\ G_N(q^{-1}) \end{pmatrix} \Delta u(t-1) \quad (6.10)$$

Por lo tanto el mínimo de la función de costo se alcanza haciendo

$$\left[\sum_{j=N_1}^{N} s_j^2 + l \right] \Delta u(t) = -\sum_{j=N_1}^{N} s_j \left[F_j(q^{-1}) y(t) + G_j(q^{-1}) \Delta u(t-1) \right] \quad (6.11)$$

Sustituyendo (6.11) en (6.2) se encuentra que la ecuación característica en lazo cerrado es:

$$A(q^{-1})\Delta(q^{-1})\, l + B'(q^{-1}) = 0 \quad (6.12)$$

donde

$$B'(q^{-1}) = A(q^{-1})\Delta(q^{-1}) \sum_{j=N_1}^{N} s_j^2 + q^{-1} \left[\sum_{j=N_1}^{N} s_j \left(A(q^{-1})\Delta(q^{-1}) G_j(q^{-1}) + B(q^{-1}) F_j(q^{-1}) \right) \right] \quad (6.13)$$

Se puede comprobar que la estructura de esta ecuación es similar a la de una realimentación negativa de un sistema equivalente estable con función de transferencia $B'(q^{-1})/A(q^{-1})$ y un regulador integral con función de transferencia $1/(\lambda\Delta(q^{-1}))$. Teniendo en cuenta que $\Delta(1)=0$ y $F_j(1)=1$, $\forall j = N_1+1, \ldots, N$, la ganancia de la planta equivalente es $B'(1)/A(1) = m\sum_{j=N_1}^{N} s_j$. Si se hace un análisis del lugar de las raíces de este sistema, se llega al siguiente resultado.

Criterio: Si se selecciona N_1 y N de forma que $N \geq N_1 > 0$ y $m\sum_{j=N_1}^{N} s_j > 0$, entonces existe un $\bar{l}$ tal que $\forall\, l > \bar{l}$ el sistema en lazo cerrado diseñado con el controlador GPC con estos valores de N_1 y N, es asintóticamente estable.

6.3 SEGUIMIENTO DE CONSIGNAS

Los criterios generales en el apartado anterior asumen una consigna constante de control. Otro problema interesante es el del seguimiento de una determinada señal de referencia $r(t)$ variable en el tiempo (*tracking problem*).

En general se pueden distinguir dos problemas diferentes relacionados con el seguimiento de una señal. Uno es el seguimiento de consignas que aparecen como una sucesión de escalones de determinada amplitud y anchura. Y otro es el caso en el que la consigna aparece como una señal variable conteniendo diferentes frecuencias. Centraremos nuestra atención en el estudio del horizonte de salida N. En el primero de los casos, el algoritmo GPC se muestra bastante eficaz en el seguimiento de la referencia. La elección de los horizontes de predicción en este caso no suele ser un factor crítico para el seguimiento de la referencia, aunque si tiene influencia en los transitorios del sistema. Como regla general, se puede afirmar que para horizontes de salida grandes se observan transitorios mejores, tal como se pone de manifiesto en la figura 6.1.

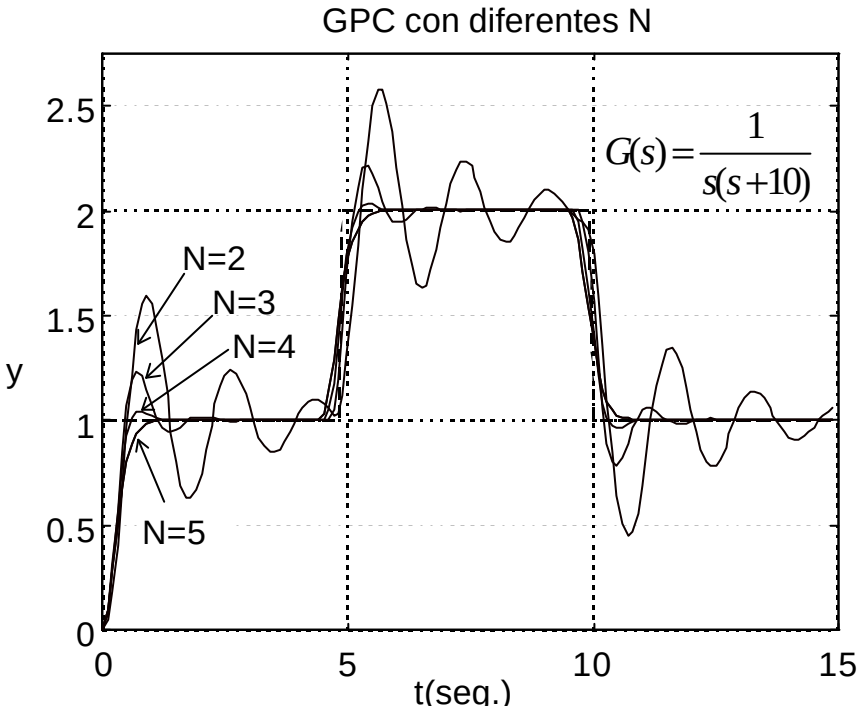
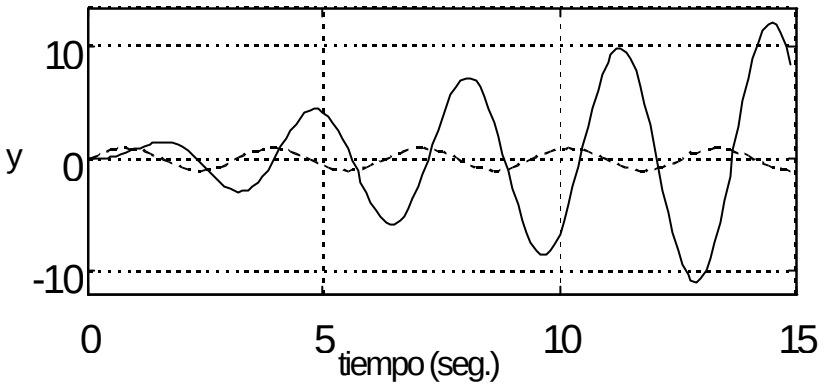


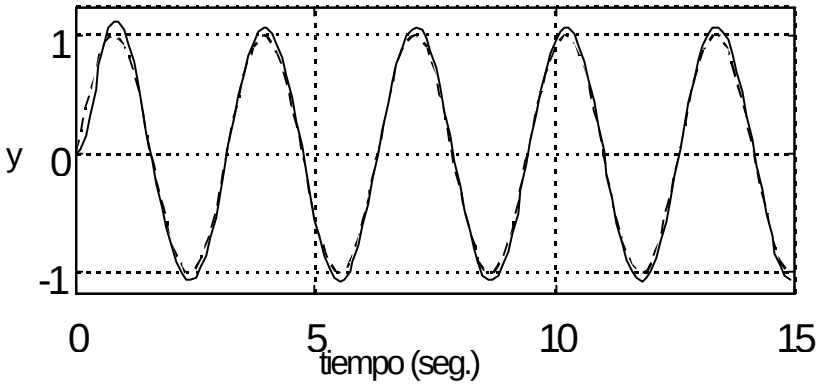
Figura 6.1. Simulación de un control GPC sobre una planta de segundo orden con diferentes valores de N.

Por el contrario, cuando el sistema trata de seguir una señal de referencia variable, juega un papel fundamental el horizonte de salida que se tome. Para simplificar el estudio se considerará el seguimiento de señales de referencia sinusoidales de frecuencia constante. En

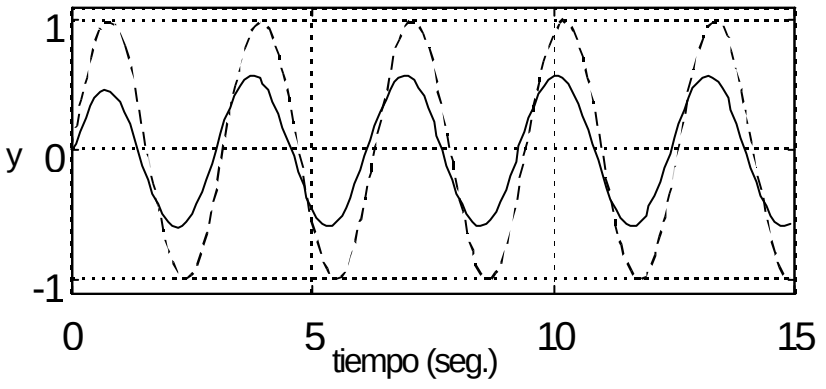
este caso, se observa que dependiendo de la frecuencia de la consigna, el horizonte de salida N que se tome en el controlador juega un papel determinante en el funcionamiento del sistema en lazo cerrado. En las figuras 6.2a), b) y c) se muestra como el rendimiento de la planta con el GPC para el seguimiento de una consigna sinusoidal se muestra especialmente sensible con el valor del parámetro N. Así, se puede observar como un valor de N=1 provoca una repuesta inestable de la planta. Un valor de N=10 produce una respuesta estable pero con un error de seguimiento importante. Mientras que un valor de N=3 produce una respuesta del sistema con un error bastante aceptable.



(a)



(b)



(c)

Figura 6.2. Simulación del GPC para la planta G(s)=1/s(s+10).a) N=1, b) N=3, c) N=10.

Las experiencias realizadas sobre diferentes plantas realizadas nos permiten corroborar el siguiente resultado:

Sea W el conjunto de todos los posibles valores que puede tomar el horizonte de predicción N. Considérese un sistema al que se le aplica un controlador GPC con consigna una señal sinusoidal r(t)=sin(wt), con w constante. Se define el índice

$$e_N^{max} = \lim_{k\to\infty}\left(|r(kp/2) - y_N(kp/2)|\right); \qquad (6.14)$$

donde |.| indica el valor absoluto e y_N es la salida del sistema obtenida aplicando un GPC con horizonte de salida N. Este índice se puede ver como un indicador del error de seguimiento de la consigna. En adelante se hará referencia a este índice como Índice de Error de Seguimiento (IES). Nótese que lo que se considera es el error estacionario en los puntos de máximo y mínimo de la consigna. En la figura 6.3 se muestra gráficamente la magnitud que representa este índice. En esta figura se ha asumido que la señal ya ha alcanzado un estado estacionario a partir de t > 6 segundos.

En estas condiciones se puede comprobar que

$$\exists\ \mathrm{N} = N_{\mathrm{opt}} \in \Omega\ /\ e_{N_{opt}}^{max} = \min_{N\in\Omega}\left(e_N^{max}\right) \qquad (6.15)$$

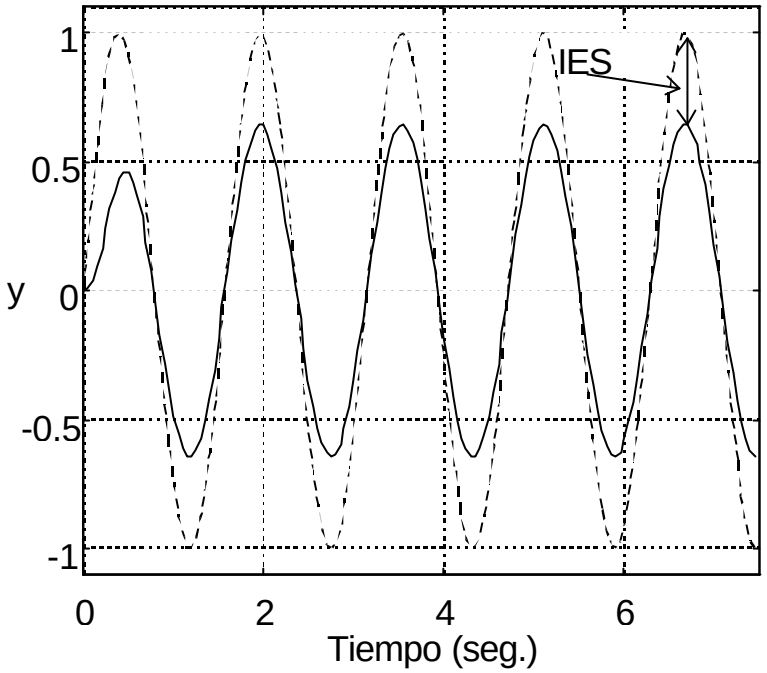


Figura 6.3. Definición del Índice de Error de Seguimiento. La salida aparece con trazo continuo y la consigna con trazo discontinuo.

Es decir, dada una planta descrita por el modelo (6.3). Si a esta planta se le aplica un controlador GPC tomando como señal de consigna una sinusoidal de amplitud y frecuencia constante, existe un horizonte de predicción N_{opt} que produce un *IES* mínimo. De tal forma que si se toma un horizonte de predicción $N \neq N_{opt}$ entonces el índice de error de seguimiento aumenta. Se puede comprobar experimentalmente que el error aumenta monótonamente a medida que N se aleja del valor N_{opt}. En la figura 6.4 aparece representada una evolución típica del *IES* con N para una planta de segundo orden. Se observa claramente la existencia de un valor de N para el cual el error de seguimiento es mínimo. Valores de N mayores o menores que este N_{opt} conducirán a un *IES* mayor.

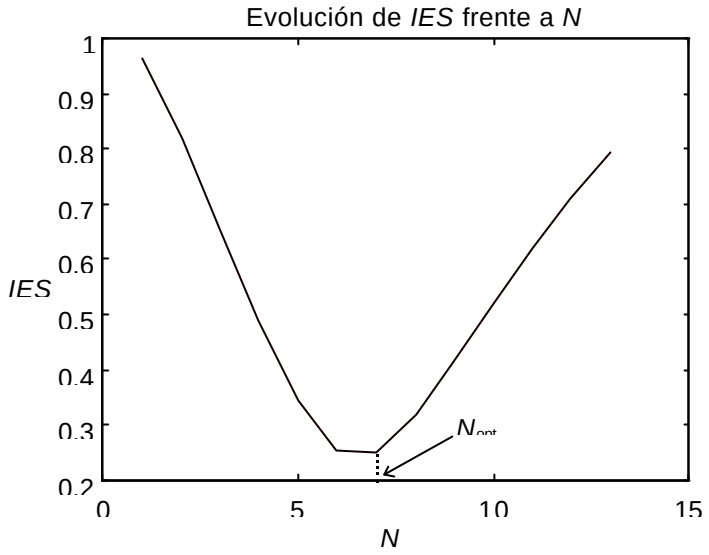


Figura 6.4. Definición del horizonte de predicción óptimo (N_{opt}).

6.4 ANÁLISIS DE LA RESPUESTA DE SISTEMAS DE SEGUNDO ORDEN CON CONTROLADOR GPC

En esta sección el estudio estará centrado en plantas de segundo orden lineales e invariantes en el tiempo. El análisis consistirá en estudiar la influencia que tiene la localización de los polos de la planta en el plano s en el tiempo de predicción óptimo.

Definición 1: Se define el periodo de predicción, P, como $P = N\,T$, donde N es el horizonte de salida y T es el periodo de muestreo del sistema.

Definición 2: Se define el periodo de predicción óptimo, P_{opt} como $P_{opt} = N_{opt}\,T$, donde N_{opt} es el horizonte de predicción óptimo (ver sección anterior).

Como resultado preliminar de este análisis cabe destacar que el valor de P_{opt} no es el mismo para una planta muestreada a diferentes frecuencias. Los estudios experimentales realizados sobre diferentes plantas nos permiten extraer la siguiente conclusión.

Sea una planta lineal al que se le aplica un controlador GPC:

i) Si los polos de la planta tienen su parte imaginaria igual a cero, entonces el valor de P_{opt} es invariante con la frecuencia de muestreo tomada.

ii) Si los polos tienen su parte imaginaria distinta de cero, el valor de P_{opt} depende del valor de la frecuencia de muestreo de la planta. En este caso, el comportamiento cualitativo del IES en función de P_{opt} es similar para la misma planta muestreada a diferentes frecuencias.

El siguiente resultado está relacionado con la influencia en la respuesta del sistema en lazo cerrado cuando la planta presenta un desplazamiento de sus polos sobre el eje real o sobre el eje imaginario.

Sea una planta con polos en $s_0 \pm jw_0$ a la que se le aplica un controlador GPC con horizontes de control NU y horizonte de salida N y con una señal de consigna sinusoidal de frecuencia constante.

Se puede comprobar que una perturbación en la planta que produzca un desplazamiento de los polos sobre el eje real, $s_0 + \Delta s_0$, se traduce en una variación en el desfase de la salida del sistema en relación a la consigna. Igualmente, si una perturbación en la planta modifica sus polos de forma que cambia el valor de su parte imaginaria, $w_0 + \Delta w_0$, entonces el efecto sobre la respuesta del sistema es un cambio en la amplitud de la señal de salida.

Es decir, las variaciones en la parte real de los polos producen cambios en el desfase medido en la salida, mientras que variaciones sobre la parte imaginaria producen cambios en la amplitud de la respuesta del sistema.

El estudio realizado se ha centrado en plantas de segundo orden con funciones de transferencia como las que aparecen en la tabla 6.1. Como puede observarse la planta de tipo A tiene ganancia constante e igual a la unidad, mientras que en las de tipo B se considera una ganancia igual a w_n^2. Los resultados obtenidos se resumen en los siguientes apartados.

TIPO	Función de transferencia
A	$\frac{1}{s^2 + 2dw_n s + w_n^2}$
B	$\frac{w_n^2}{s^2 + 2dw_n s + w_n^2}$

Tabla 6.1. Funciones de transferencia analizadas.

6.4.1 FUNCIONES DE TRANSFERENCIA DE TIPO A: $\frac{1}{s^2 + 2dw_n s + w_n^2}$

Se ha considerado el estudio de este tipo de plantas sometidas a un controlador GPC y muestreadas con un periodo T=0.05 segundos. La consigna considerada es una señal sinusoidal de frecuencia igual a 4 rad/sec ($r(t) = sen(4t)$). La localización de los polos de las plantas estudiadas es la que aparece en la figura 6.5. En la figura se representa la parte real frente al valor absoluto de la parte imaginaria del par complejo conjugado. Marcada con una x aparece la localización de los polos de las plantas estudiadas.

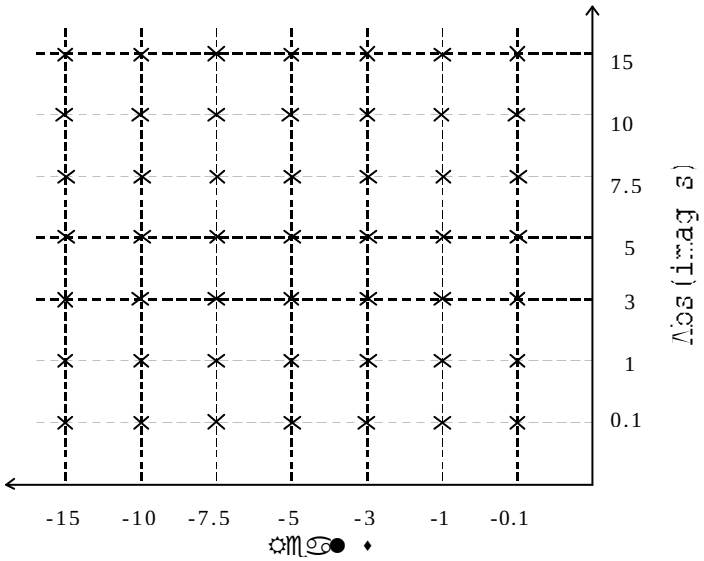


Figura 6.5. Localización de los polos de las plantas de tipo A.

6.4.1.1 VARIACIÓN DEL PERIODO DE PREDICCIÓN Y DEL IES

Después de aplicar el GPC sobre todas las plantas indicadas en la figura 6.5, se ha calculado el valor de P_{opt} para cada una de ellas. En la figura 6.6 aparece una gráfica en la que

se recoge el valor de P_{opt} en función de la localización de los polos. De nuevo, se define la localización de los polos dando al valor de la parte real y el valor absoluto de la parte imaginaria. Como se puede observar la tendencia del periodo de predicción es claramente decreciente a medida que los polos se alejan del origen. Es de destacar la simetría que existe de los valores de P_{opt} con respecto a la diagonal. Este hecho indica que el periodo de predicción óptimo sólo depende del valor de w_n y no del factor de amortiguamiento δ. Si para todas estas plantas se representa la variación del P_{opt} frente a w_n, se encuentra la evolución que se muestra en la figura 6.7, donde los valores de P_{opt} para cada frecuencia aparecen señalados con "*". Realizando un ajuste de estos puntos se puede encontrar que la curva que mejor representa su evolución es una curva de segundo orden del tipo

$$P_{opt} = aw_n^2 + b$$

En la figura aparece con trazo discontinuo esta curva ajustada donde las constantes a y b son a=0.00115, b=0.2400.

En la figura 6.8 se hace una representación de la evolución del coeficiente *IES* con los valores de P_{opt} indicados en la figura 6.6 en función de la localización de los polos. Se observa que su comportamiento es creciente a medida que los polos se alejan del origen. Igualmente, se puede comprobar un comportamiento sólo dependiente de la frecuencia natural y no de la constante de amortiguamiento. Esto se pone de manifiesto en la figura 6.9 donde aparecen valores de *IES* en función de w_n. El ajuste de estos puntos corresponde al de una parábola

$$IES = aw_n^2$$

donde a=0.0019.

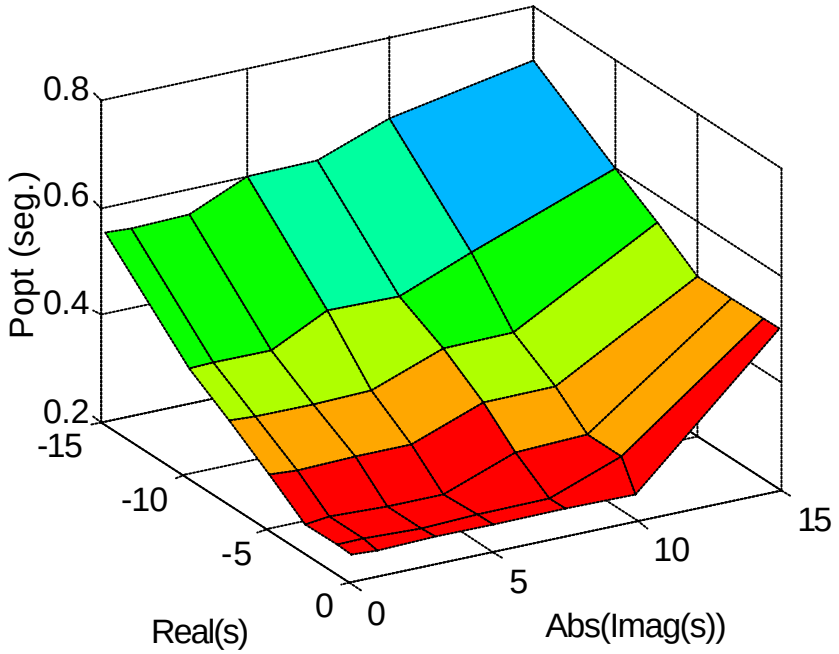


Figura 6.6. Evolución del P_{opt} para plantas de tipo A

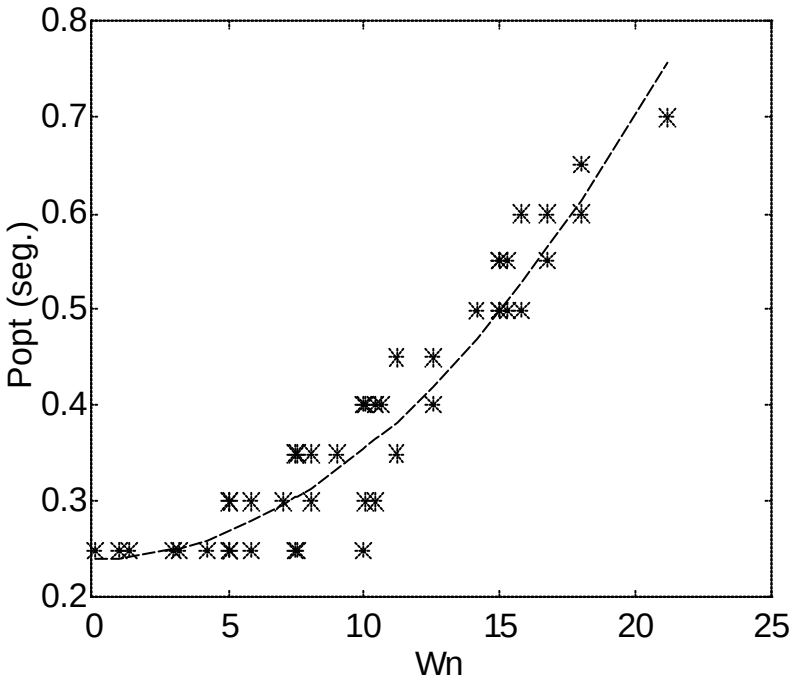


Figura 6.7. Ajuste del P_{opt} en función de w_n.

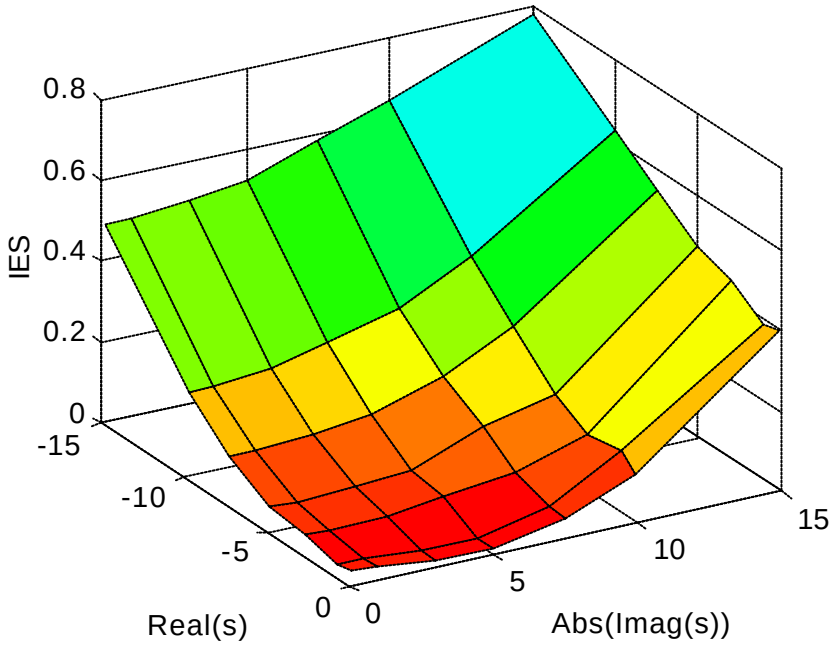


Figura 6.8. Evolución del *IES* para plantas de tipo A.

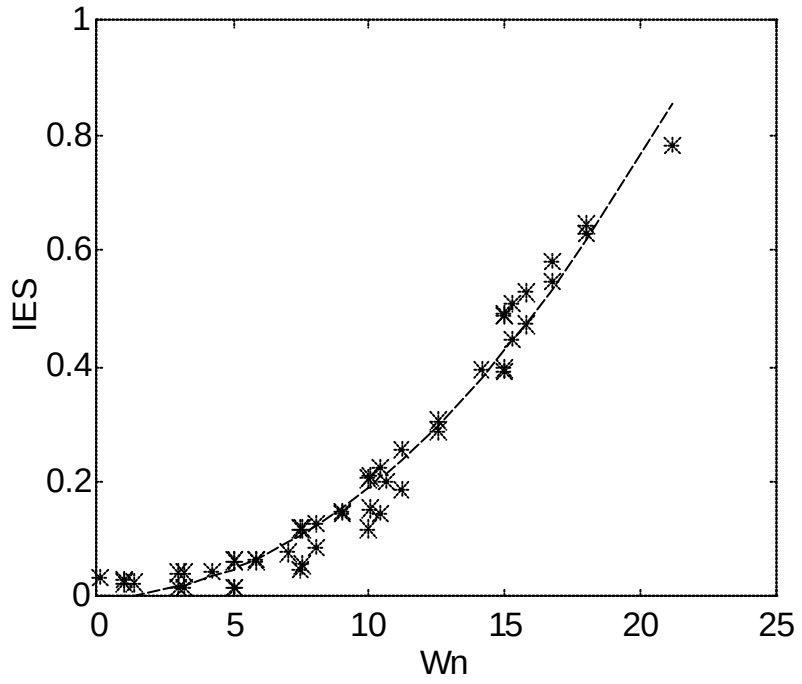


Figura 6.9. Ajuste del *IES* en función de w_n para plantas de tipo A.

6.4.1.2 VARIACIÓN CON LA GANANCIA

Las experiencias mostradas en el apartado anterior han consistido en estudiar el efecto de la variación de los polos tomando la ganancia de la planta constante. A continuación consideraremos el estudio de la influencia del valor de la ganancia en el sistema. Para ello se tomarán plantas del tipo $p \pm p$j con sus polos fijos y se variará el valor de la ganancia para ver su influencia en el horizonte de predicción. Las plantas estudiadas son las que aparecen en la tabla 6.2. Todas ellas se han muestreado con un periodo de T=0.05 seg.

	Función de Transferencia	**Polos**	a_g
1	$\frac{k}{s^2+2s+2}$	-1±1j	0.2673
2	$\frac{k}{s^2+5s+12.5}$,	-2.5±2.5j	0.2887
3	$\frac{k}{s^2+10s+50}$	-5±5j	0.3333
4	$\frac{k}{s^2+15s+112.5}$	-7.5±7.5j	0.4472
5	$\frac{k}{s^2+20s+200}$	-10±10j	0.5113

Tabla 6.2. Plantas para estudio de la variación de la ganancia.

Se ha considerado el valor de k variable entre 0.3 y 10. En estas condiciones se ha sometido al sistema a una consigna sinusoidal de frecuencia 2 rad/seg. y amplitud 1. Para cada valor de k se ha obtenido el valor del horizonte de predicción que produce un *IES* óptimo. Se puede observar en las figuras 6.10a), b), c), d) y e) , con trazo continuo, que el periodo de predicción óptimo parte de un valor relativamente grande y va descendiendo a medida que k aumenta. Es decir, el efecto que tiene un aumento en el valor de la ganancia de la planta en el controlador GPC es que el horizonte de predicción óptimo disminuye.

Si se intenta modelar el comportamiento del tiempo de predicción con la ganancia es posible observar que existe una ley que define este comportamiento. Así, se ha encontrado, aplicando técnicas de ajuste de mínimos cuadrados, que la evolución del periodo de predicción óptimo, P_{opt}, con la ganancia puede ser modelada mediante la ley

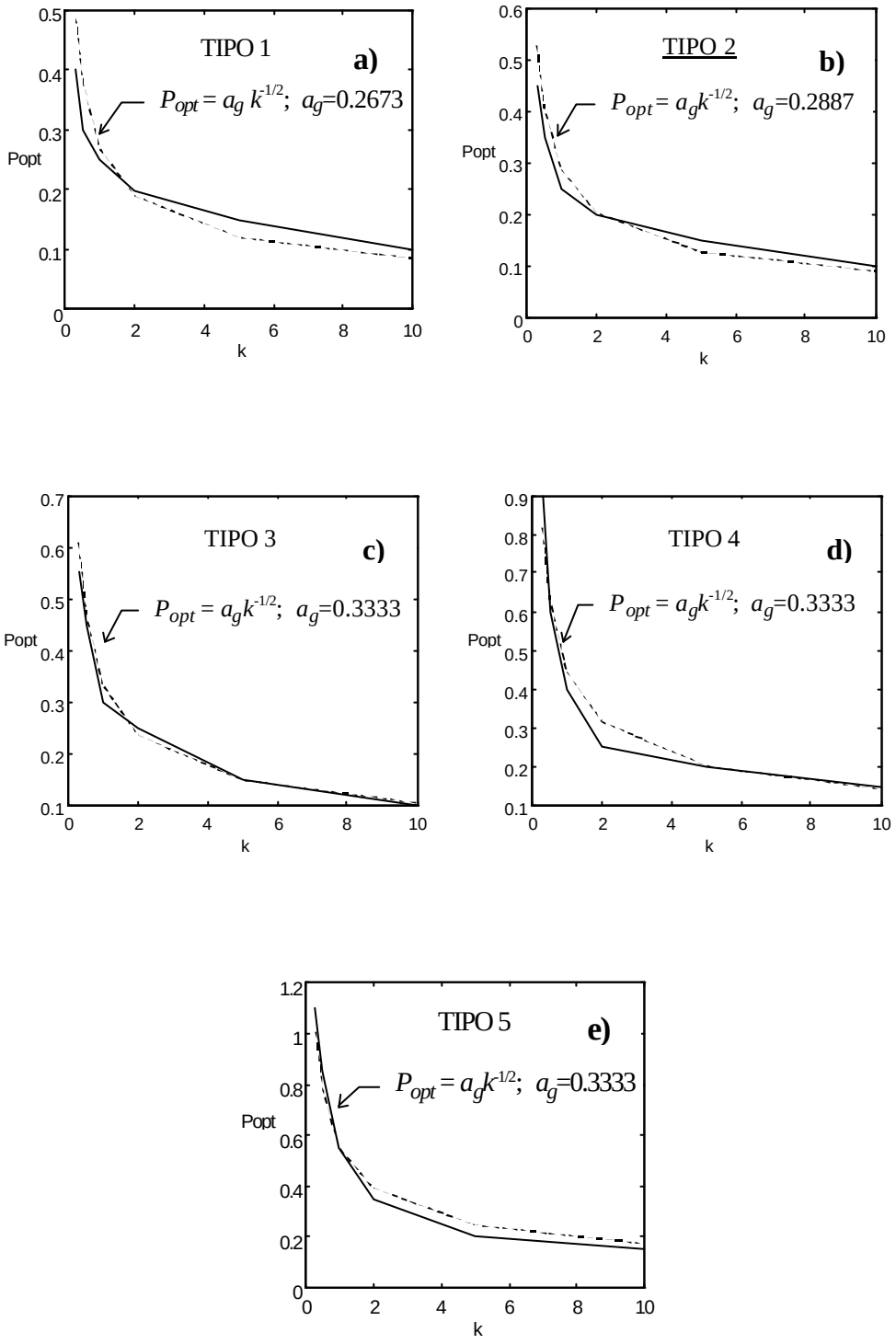


Figura 6.10. Evolución de P_{opt} para las funciones de transferencia mostradas en la tabla 6.2. Con trazo sólido se muestra la curva obtenida en simulación. Con trazo discontinuo aparece el ajuste realizado.

$P_{opt} = a_g k^{-1/2}$ siendo a_g un parámetro constante que depende de la planta en estudio y k la ganancia de la planta. En las gráficas 6.10a), b) ,c), d), y e) aparece con línea a rayas la curva ajustada mediante esta ley. El valor encontrado para el parámetro a_g para las diferentes plantas se muestra en la tercera columna de la tabla 6.2. Como se puede observar, a medida que los polos de la función de transferencia se alejan del origen, el valor del parámetro a_g va en aumento. Es posible estudiar la evolución de este parámetro en función de la localización de los polos. Si se realiza un ajuste del parámetro a_g en función del valor de p que determina la localización de los polos, se encuentra un comportamiento lineal para la evolución de este parámetro. Esto se puede confirmar observando la figura 6.15. Para las plantas consideradas el ajuste de esta recta resultó ser $a_g = 0.0292p + 0.2195$.

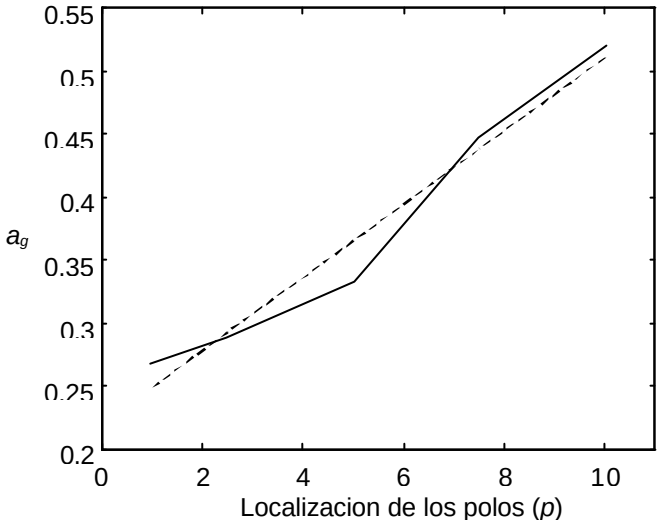


Figura 6.11. Evolución del parámetro a_g en función de la localización de los polos.

6.4.2 FUNCIONES DE TRANSFERENCIA DE TIPO B: $\dfrac{w_n^2}{s^2 + 2dw_n s + w_n^2}$

En las mismas condiciones se ha realizado un estudio con plantas de este tipo. Se ha tomado un controlador GPC con una consigna sinusoidal constante $r(t) = sen(4t)$. La localización de los polos de las plantas es la que aparece en la figura 6.12 marcada con una x. El periodo de muestreo considerado es T=0.05 segundos. Para cada una de estas plantas se ha obtenido el valor de P_{opt} y se ha medido el índice *IES* correspondiente.

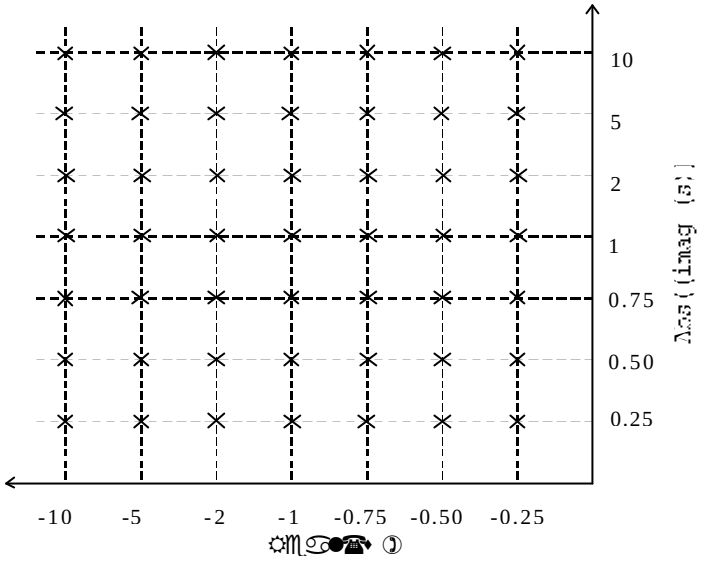


Figura 6.12. Localización de los polos de las plantas de tipo B.

6.4.2.1 VARIACIÓN DEL PERIODO DE PREDICCIÓN Y DEL IES

En la figura 6.13 se muestra la evolución del periodo de predicción P_{opt} en la superficie de ubicación de los polos. Como puede observarse se requieren valores muy grandes de P_{opt} para las plantas que tienen sus polos cerca del origen. Sin embargo, cuando los polos están localizados lejos del origen, los valores de P_{opt} van siendo cada vez menores. Este resultado contrasta con el obtenido para las plantas de segundo orden del tipo A, para las cuales los valores de P_{opt} van en aumento a medida que la situación de los polos se aleja del origen (ver figura 6.6). Esta aparente incongruencia tiene su explicación en que la influencia que tiene la ganancia w_n^2 en el P_{opt} es superior a la que tiene la situación de los polos. Por ello, la evolución del P_{opt} se ve más influenciada por el comportamiento decreciente debido a la variación de la ganancia (similar al de las figuras 6.10a) a 6.10e)) que por el comportamiento creciente debido únicamente a la ubicación de los polos del sistema.

Es interesante notar la simetría que aparece en la figura 6.13. Esto sugiere, igual que para las plantas de tipo A, que la evolución de P_{opt} sólo depende de la frecuencia de oscilación w_n. Si se intenta encontrar una curva que modele el comportamiento de P_{opt} se llega a que éste se puede representar mediante la siguiente ley:

$$P_{opt} = a w_n^{-1/2} \tag{6.16}$$

donde a es una constante que, tras un ajuste por mínimos cuadrados, resulta ser a=0.2236. En la figura 6.14 se representan los valores de P_{opt} calculados (representados con '*') en función de la frecuencia natural del sistema. En esta figura se muestra también el ajuste de estos puntos al modelo (6.16) (línea discontinua).

La representación del índice de error de seguimiento se muestra en la figura 6.15. Como se puede observar también presenta valores muy altos para plantas con polos cerca del origen, mientras que cuando estos se van alejando del origen el *IES* cae prácticamente a cero.

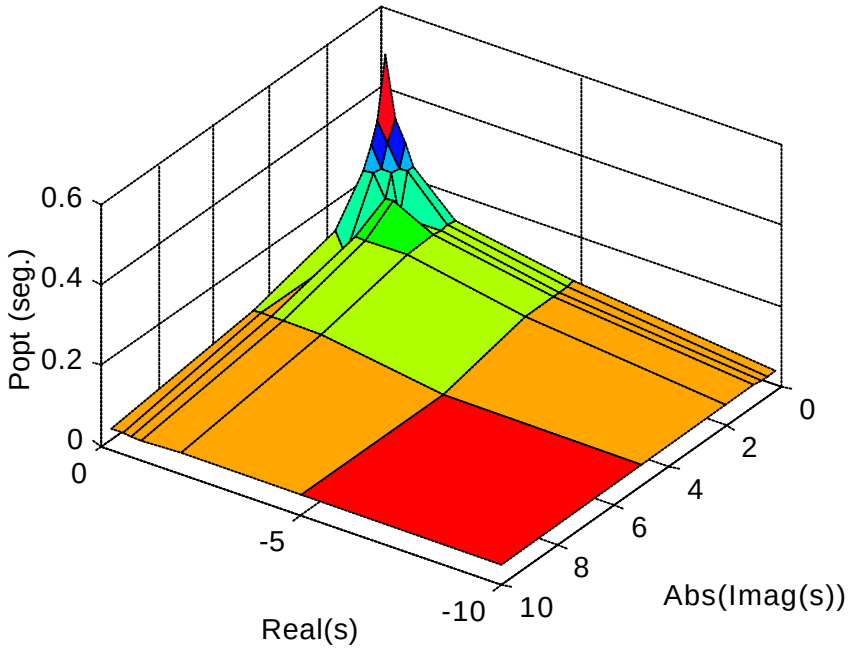


Figura 6.13. Evolución del P_{opt} para plantas de tipo A

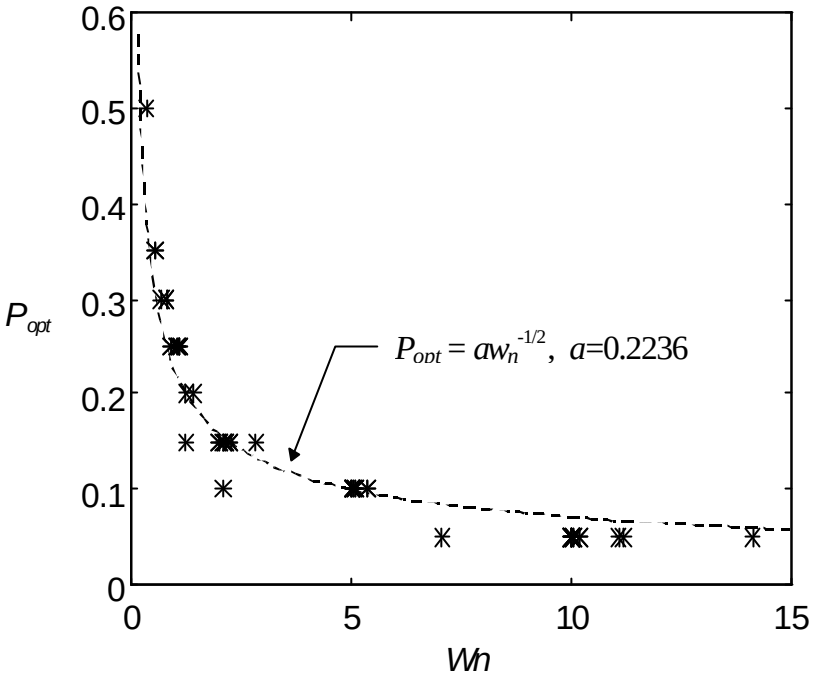


Figura 6.14. Ajuste del P_{opt} en función de w_n. para plantas de tipo A

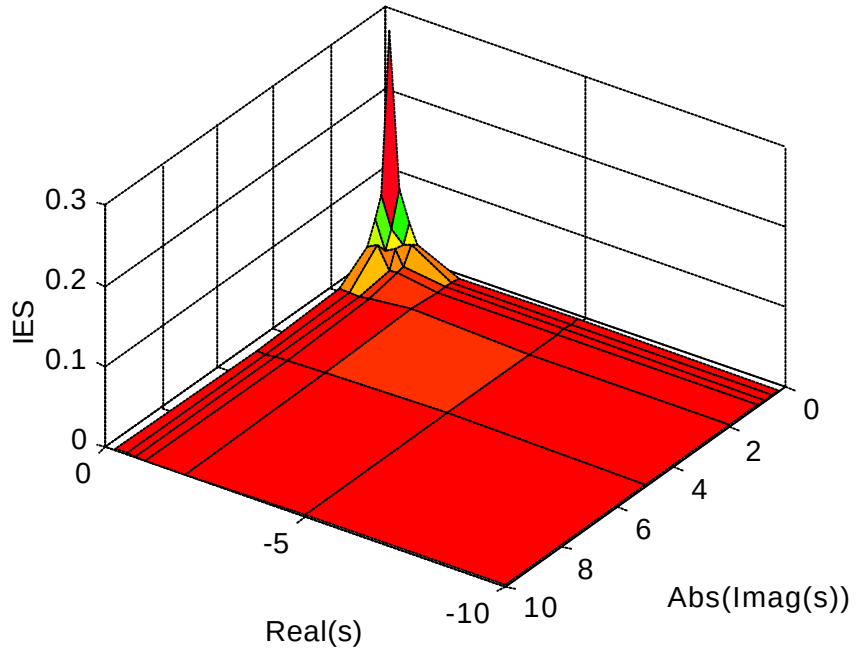


Figura 6.15. Evolución del *IES* en función de la posición de los polos para plantas de tipo B.

6.5 GPC ADAPTIVO NEUROMÓRFICO

6.5.1 INTRODUCCIÓN

Como se comentó anteriormente, en el problema de seguimiento de consignas empleando un GPC, el valor del horizonte de salida elegido juega un papel fundamental para el buen funcionamiento del sistema. Se vio que para cada planta existe un valor de N para el cual el error de seguimiento se hace mínimo. Este resultado pone de manifiesto el interés existente en encontrar para cada aplicación este valor del horizonte de predicción, N_{opt}, que garantiza un seguimiento de la consigna óptimo. La determinación de este parámetro no parece tener una solución analítica sencilla. En este apartado se expone un método que permite obtener el valor de N_{opt} de forma adaptiva: el controlador GPC adaptivo neuromórfico (NAGPC, *Neuromorphic Adaptive Generalized Predictive Controller*).

El esquema que se propone aquí se basa en el empleo de redes neuronales para sintonizar el valor de N. La función de las redes será la de un *self-tuner* que, a partir de información del estado del sistema, sintoniza el parámetro N de forma que se minimice un determinado índice de funcionamiento.

Este método presenta dos características fundamentales que lo hacen especialmente interesante. La primera de ellas está relacionada con el entrenamiento de la red. En general, cuando se incluyen redes neuronales en esquemas de control, el aprendizaje de la red suele llevarse a cabo *off-line* empleando un conjunto de patrones de entrenamiento. En el esquema que se propone aquí, el aprendizaje de la red se lleva a cabo en tiempo real. No es necesario un pre-entrenamiento antes de poner en marcha el controlador. Aunque obviamente la convergencia de la red mejora notablemente con este entrenamiento previo, que partiendo de unas condiciones iniciales aleatorias. La otra característica importante es que no se hace ninguna suposición sobre el modelo de la planta. Las redes neuronales solo necesitan la información que les llega *on-line* sobre el estado del sistema. Esto indudablemente la da más versatilidad a este esquema y redunda en un incremento en la robustez del sistema en lazo cerrado, puesto que no hay lugar a errores debidos a un modelo inadecuado de la planta en lazo abierto.

6.5.2 DESCRIPCIÓN DEL NAGPC

La estructura de este controlador es la que se muestra en la figura 6.16. En esta figura la variable $y(k)$ representa la salida del sistema, $r(k)$ es la consigna marcada al sistema, $u(k)$ es el comando aplicado y $e(k)$ es el error cometido.

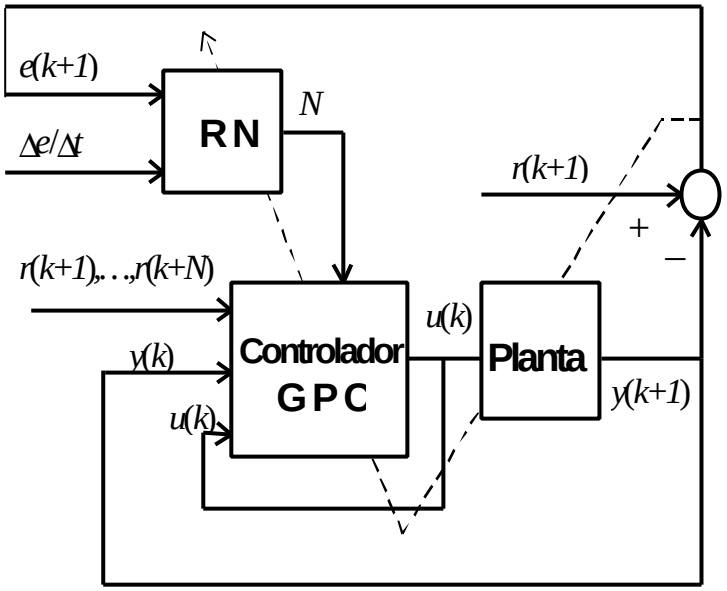


Figura 6.16. Estructura del controlador GPC adaptivo neuromórfico.

Como puede observarse las entradas a la red son el error y la derivada del error, que definen el estado del sistema. El entrenamiento de la red se lleva a cabo siguiendo el conocido algoritmo de *backpropagation*. Como se comentó en capítulos anteriores, este método consiste en ir actualizando los pesos de la red a partir de una señal de error que se genera en la última etapa y se va propagando hacia atrás a través de las diferentes capas. En nuestro caso los pesos de la red se actualizarán de tal forma que se minimice la siguiente función de costo

$$J(k) = \frac{Q}{2}(r(k) - y(k))^2 . \tag{6.17}$$

Nótese que lo que se pesa en cada instante es simplemente el error cometido. Q es un parámetro constante que actúa como factor de peso. Como se vio en la sección 5.2.3, para conseguir la minimización de este índice el algoritmo de entrenamiento emplea el método del

gradiente descendente que impone que en cada iteración los pesos deben ser actualizados de acuerdo con

$$w_{l,j,i}(k+1) = w_{l,j,i}(k) - \mu \frac{\partial J(\mathbf{w})}{\partial w_{l,j,i}} \tag{6.18}$$

donde $w_{l,j,i}$ es el peso que conecta la salida del nodo *i*-ésimo en la capa *l*-1 con la entrada del *j*-ésimo nodo de la capa *l* y *m* es la velocidad de aprendizaje. En el esquema que se propone aquí, la salida de la red *N*, no aparece en la función de costo explícitamente. Por lo tanto el término $\P J(k) / \P w_{l,j,i}$ se obtendrá aplicando la regla de la cadena como

$$\frac{\P J(k)}{\P w_{L,1,i}} = \frac{\P J(k)}{\P N} \frac{\P N}{\P w_{L,1,i}} \tag{6.19}$$

donde

$$\frac{\P J(k)}{\P N} = \frac{\P J(k)}{\P y} \frac{\P y(k)}{\P u} \frac{\P u(k)}{\P N} \tag{6.20}$$

En esta expresión el primer factor puede ser calculado como

$$\frac{\P J(k)}{\P y} = -Qe(k) \tag{6.21}$$

En cuanto al factor $\P y / \P u(k)$, éste podría ser calculado si el modelo de la planta fuera conocido. Evitando cualquier suposición sobre el modelo de la planta, este término puede ser obtenido mediante la siguiente aproximación en diferencias

$$\frac{\P y(k)}{\P u} \approx \frac{y(k) - y(k-1)}{u(k-1) - u(k-2)} \tag{6.22}$$

Igualmente, el cálculo del tercer factor de (7), $\P u / \P N(k)$, podría llevarse a cabo si se conociera la relación explícita entre el comando *u* y el horizonte de salida *N*. Como esto es imposible, se puede asumir la misma aproximación anterior para calcular este factor. Es decir,

$$\frac{\P u(k,N)}{\P N} \approx \frac{u(k,N+1) - u(k,N-1)}{2} \tag{6.23}$$

De esta forma, el gradiente de la función de costo puede ser calculado para cada peso de la red y las actualizaciones pueden ser llevadas a cabo siguiendo la expresión (6.18). Como resultado la red neuronal suministra en cada etapa el valor de N al controlador GPC. Este valor de N será tal que minimice el índice de funcionamiento (6.17), y por lo tanto minimizará el *IES,* es decir, el valor de N devuelto debe tender a N_{opt}.

Empezando en el instante k, este algoritmo puede ser resumido en las siguientes operaciones:

Paso 1) LEER $e(k)$ y $\Delta u(k)$

Paso 2) (Entrenamiento de la Red)

$$\frac{\partial J(k)}{\partial y} \leftarrow -Qe(k)$$

$$\frac{\partial y(k)}{\partial u} \leftarrow \frac{y(k)-y(k-1)}{u(k-1)-u(k-2)}$$

$$\frac{\partial u(k,N)}{\partial N} \leftarrow \frac{u(k,N+1)-u(k,N)}{2}$$

$$\text{Backpropagation}\left(\frac{\partial J(k)}{\partial y}, \frac{\partial y(k)}{\partial u}, \frac{\partial u(k)}{\partial N}; e(k)\ ,\ \Delta u(k); Q\right);$$

ACTUALIZAR N

Paso 4) CALCULAR el comando

$$u(k) = GPC(N)$$

Paso 5) APLICAR $u(k)$ y ESPERAR hasta que $t=(k+1)T$

Paso 6) $k \leftarrow k+1$

Paso 7) GOTO *Step 1*

Existe una mejora adicional que puede incluirse en el algoritmo para aumentar el rendimiento de este esquema. Esta variación consiste en realizar una transformación lineal entre la salida de la red y el conjunto Ω de valores permitidos para N (ver figura 6.17). De esta forma se garantiza que los valores de N permanezcan siempre dentro del rango permitido (entre un valor mínimo, *Nmin* y un valor máximo, *Nmax*). De esta forma se pueden evitar cálculos computacionales innecesarios (si N fuera mayor que *Nmax*) y problemas de

estabilidad (si *N* fuera menor que *Nmin*). Además, la convergencia de la red mejora notablemente cuando se incluye este mecanismo.

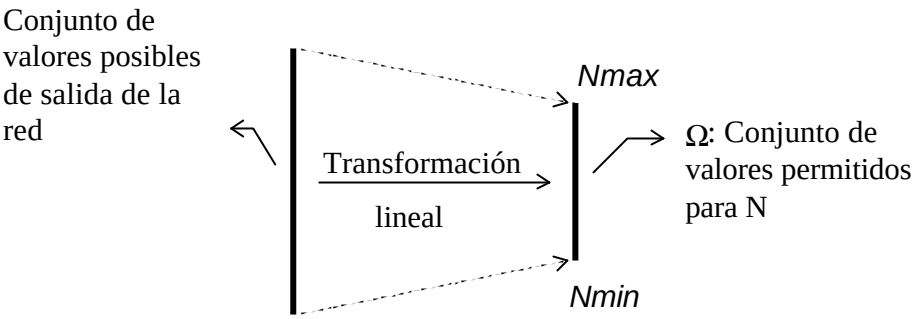


Figura 6.17. Transformación lineal a la salida de la red.

6.5.3 IMPLEMENTACIÓN DEL NAGPC

La estrategia NAGPC se ha comprobado sobre una variedad de plantas de diferentes características. El GPC utilizado emplea el algoritmo básico comentado en la sección 1.4.1, tomando λ=0.001. La estructura de RN empleada tiene tres capas con tres nodos por capa en las capas ocultas y un nodo en la capa de salida que proporciona el valor de *N*. Tanto para la capa de entrada como para la de salida se ha considerado una función de activación igual a la identidad. La inicialización de los pesos se llevó a cabo aleatoriamente con una distribución normal de media cero y varianza 10e-4. La implementación del controlador se realizó en MATLAB empleando la toolbox de redes neuronales [Mat92], [Dem93b].

En la tabla 6.3 aparecen algunas de las plantas sobre las que se han llevado a cabo simulaciones con esta estrategia de control. Obsérvese que las plantas 1,2 y 3 son plantas lineales y de fase mínima. La planta 4 es una planta de fase no mínima, mientras que la planta 5 es una planta inestable en lazo abierto. En la columna 3 de esta tabla aparece la definición del conjunto Ω para cada una de las plantas. En los siguientes apartados se resumen los resultados obtenidos con cada una de ellas.

Planta	**Función de Transferencia**	***T***	W
1	$\frac{1}{s(s+10)}$	0.1	[1,10]
2	$\frac{100}{s^2+10s+1}$,	0.1	[1,10]
3	$\frac{1}{2s^2+s+5}$	0.1	[1,10]
4	$\frac{s-0.1}{s^2+3s+2}$	0.01	[1,10]
5	$\frac{1}{s^3+14.9s^2+48.5s-5}$	0.05	[1,15]

Tabla 6.3. Funciones de transferencia sobre las que se ha simulado el NAGPC.

6.5.3.1 SIMULACIÓN CON PLANTAS LINEALES ESTABLES Y DE FASE MÍNIMA

El primer sistema estudiado es la planta 1 que tiene dos polos reales en $s_1 = 0$ y $s_2 = -10$. Se ha considerado inicialmente como consigna una sinusoidal de frecuencia 2 rad./seg. y amplitud unidad. En la figura 6.18 se muestra la simulación llevada a cabo empleando un controlador GPC con un valor constante para N (N *=2:* trazo continuo, N=10: trazo a puntos). En ambos casos se observa que la respuesta del sistema es insatisfactoria, pues se produce un error de seguimiento considerable. En el primer caso hay un sobrepasamiento considerable de la señal de referencia debido a que el horizonte de predicción es demasiado pequeño. Y en el caso en que N=10, lo que se observa es que el horizonte de salida es demasiado largo y el sistema intenta corregir la trayectoria con demasiada antelación causando un efecto de recorte en la señal y un *IES* grande.

En las mismas condiciones se ha realizado la simulación del NAGPC. Con los pesos de la red inicializados aleatoriamente, se ha implementado el algoritmo obteniendo el resultado que se muestra en la figura 6.19. El conjunto Ω de valores permitidos para el parámetro N que se ha considerado es Ω = [2, 3, …, 10]. En la figura 6.19a) aparece representada, con trazo continuo, la salida de la planta, y con trazo discontinuo la consigna. En la figura 6.19b) aparece representada la evolución del horizonte de salida N. Se puede diferenciar una primera fase en la que se está produciendo el entrenamiento de la red y en la que N toma diferentes

valores siempre dentro del rango permitido. Durante esta fase el seguimiento de la señal de referencia no es satisfactorio pues se produce un error de seguimiento apreciable. Posteriormente, una vez entrenada la red, N converge hacia un valor fijo que debe coincidir con el de N_{opt}. Esto se traduce en una respuesta satisfactoria del sistema ante la consigna aplicada. Como se puede apreciar el *IES* al final es prácticamente nulo.

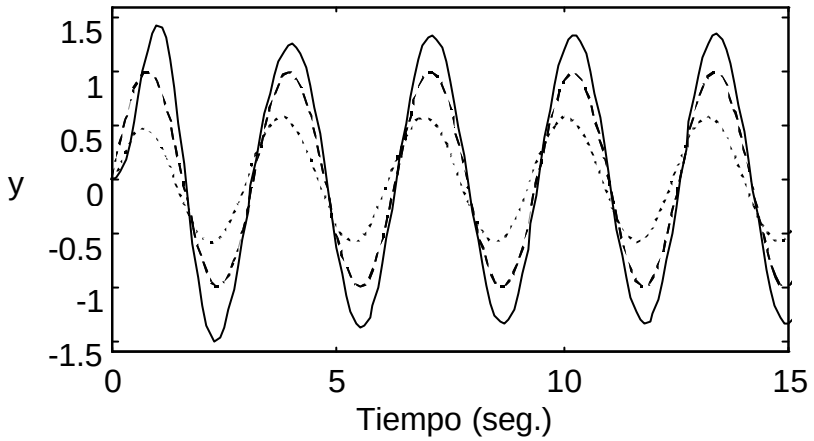


Figura 6.18. Simulación del GPC tomando valores constantes para N. Con trazo continuo N=2 y con trazo a puntos N= 10. Con línea discontinua se muestra la señal de referencia.

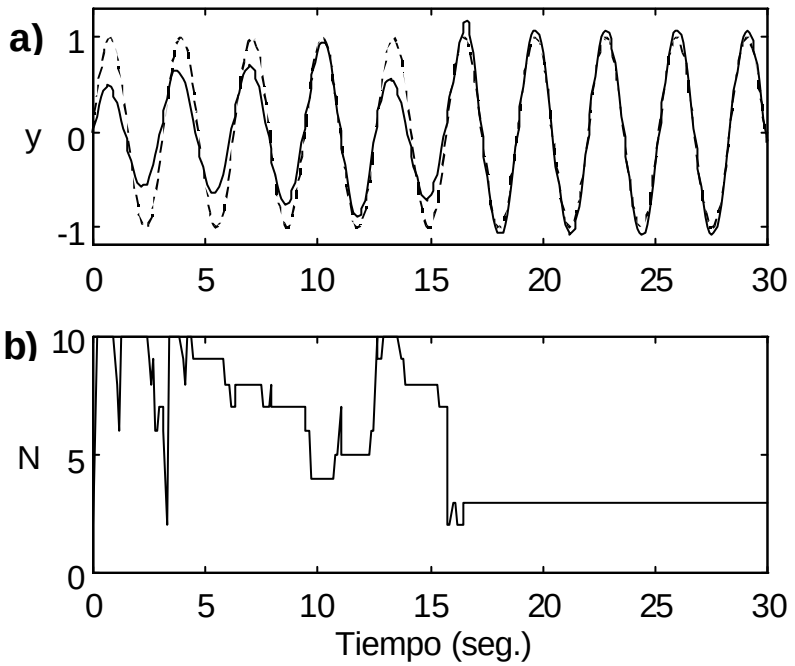


Figura 6.19. Simulación del NAGPC sobre la planta 1. a) Salida de la planta (continuo) y consigna de w=2 rad/seg.(discontinuo).b) Evolución del horizonte de salida N.

En la figura 6.20 aparece el comando aplicado en esta experiencia. Obsérvese como se diferencia una primera etapa en la que la evolución es irregular y una segunda etapa en la que su evolución alcanza un estado estacionario y permanece oscilando con una frecuencia constante.

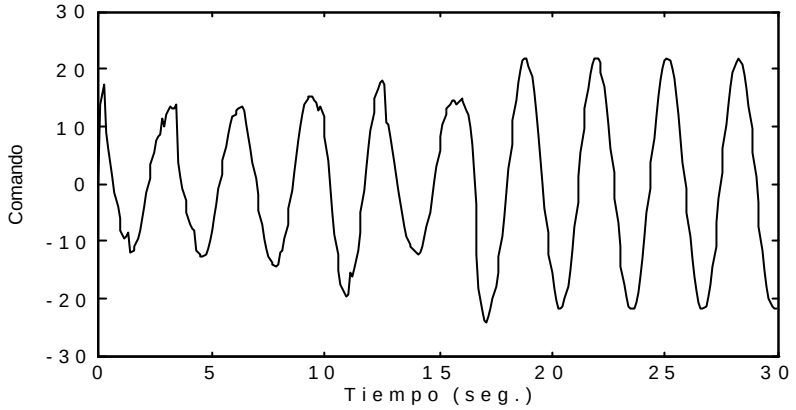


Figura 6.20. Evolución del comando de la simulación del NAGPC sobre la planta 1.

La planta 2 tiene sus polos en s_1=-9.9 y s_2 = -0.1010. En la figura 6.21 se muestra la simulación realizada con esta planta en las mismas condiciones que la de la figura 6.19. Se puede observar un comportamiento similar con una primera etapa de entrenamiento y una segunda en la que el horizonte de predicción se estabiliza en torno a N_{opt}. El seguimiento conseguido es igualmente satisfactorio en este caso. Para comprobar la robustez de este método se sometió al sistema a una consigna escalón unitario. Esta vez el sistema es capaz de detectar esta nueva situación y la evolución del N se mantiene constante en el máximo valor permitido que en este caso sí coincide con N_{opt} tal como se muestra en la figura 6.22.

Otra experiencia que se llevó a cabo con esta misma planta consistió en someterla a una consigna variable formada por una seria de escalones de distinta amplitud. Se ha comparado la respuesta obtenida con un GPC con N constante e igual a 10 y la obtenida con el NAGPC. En la figura 6.23 se muestran ambas respuestas. Como puede apreciarse el NAGPC produce un comportamiento mucho más satisfactorio que el GPC. Esto es debido a la sintonización *on-line* que se realiza en el NAGPC del horizonte de predicción, la cual se puede apreciar en la figura 6.24. Especialmente cuando la amplitud del escalón es muy estrecha, el NAGPC consigue un seguimiento mucho mejor que el GPC, que ni siquiera llega a alcanzar la consigna marcada.

Con la planta 3 también se implementó el GPC considerando una consigna sinusoidal $r(t)$=sen($2t$). Los resultados obtenidos se muestran en la figura 6.25, donde se comprueba

que el seguimiento de la señal de referencia se consigue tras una primera fase de entrenamiento de la red.

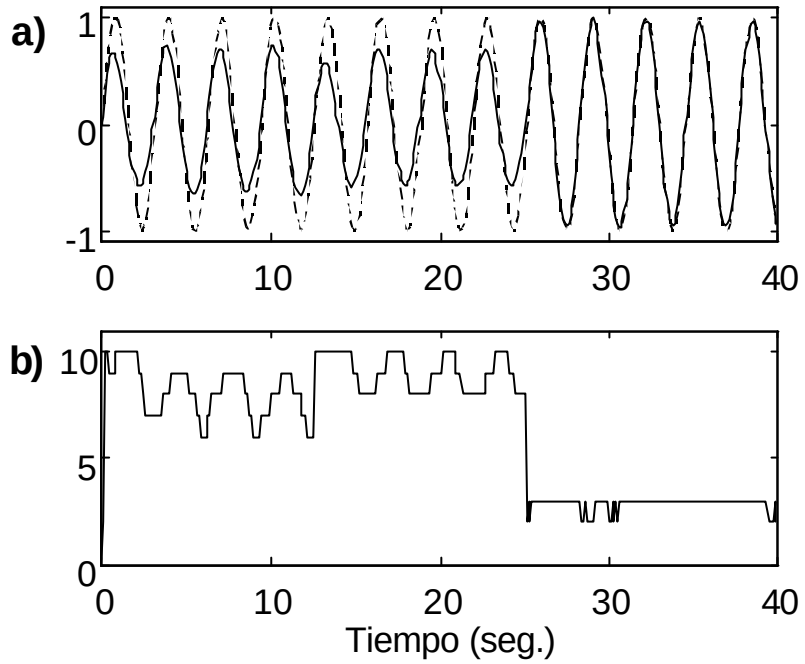


Figura 6.21. Simulación del NAGPC sobre la planta 2. a) Salida de la planta (continuo) y consigna de w=2 rad/seg.(discontinuo). b) Evolución del horizonte de salida *N*.

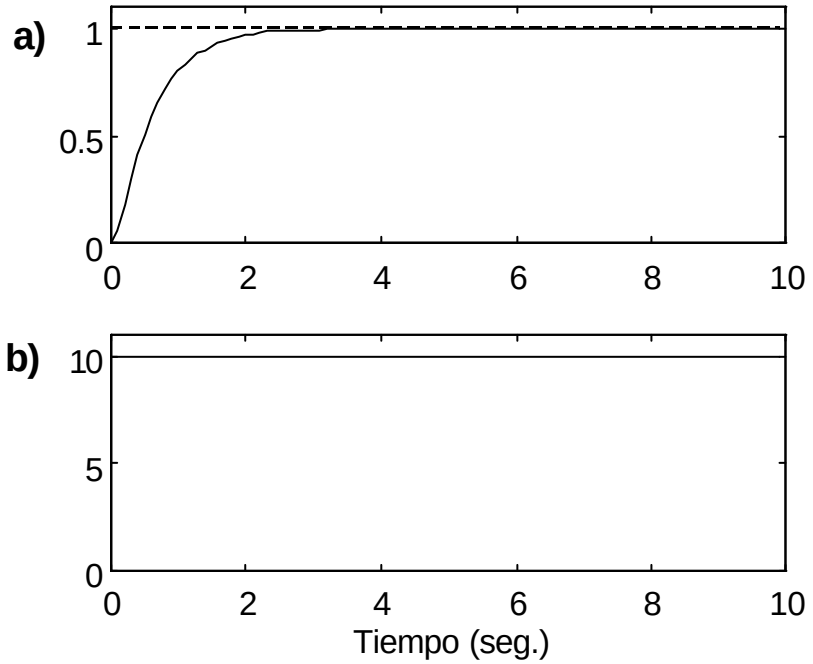


Figura 6.22. Simulación del NAGPC sobre la planta 2 considerando una consigna escalón unitario. a) Salida del sistema (continuo) y consigna (discontinuo). b) Evolución de *N*.

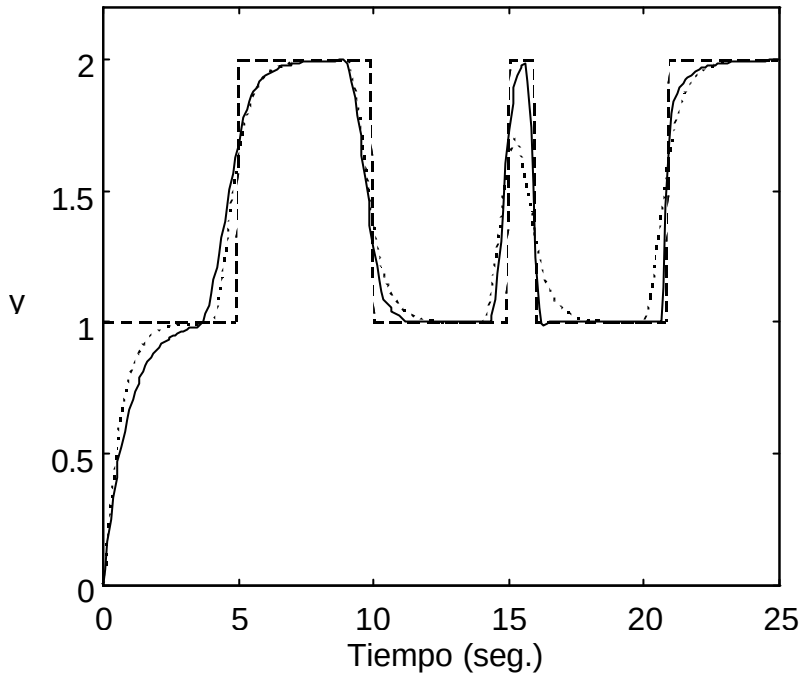


Figura 6.23. Simulación del NAGPC sobre la planta 2 (trazo continuo) frente al GPC con N=10 (trazo a puntos).

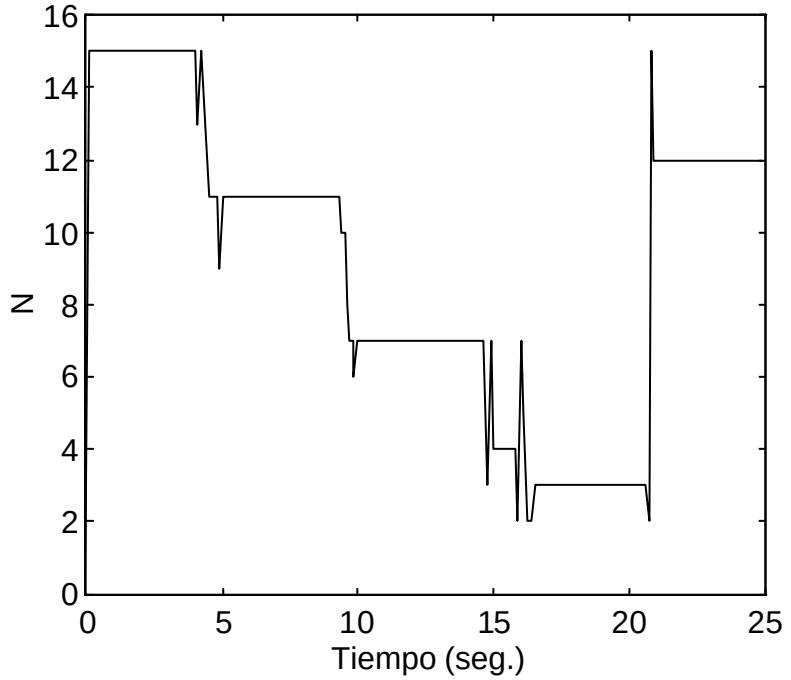


Figura 6.24. Evolución del horizonte *N* de la simulación mostrada en la figura 8.

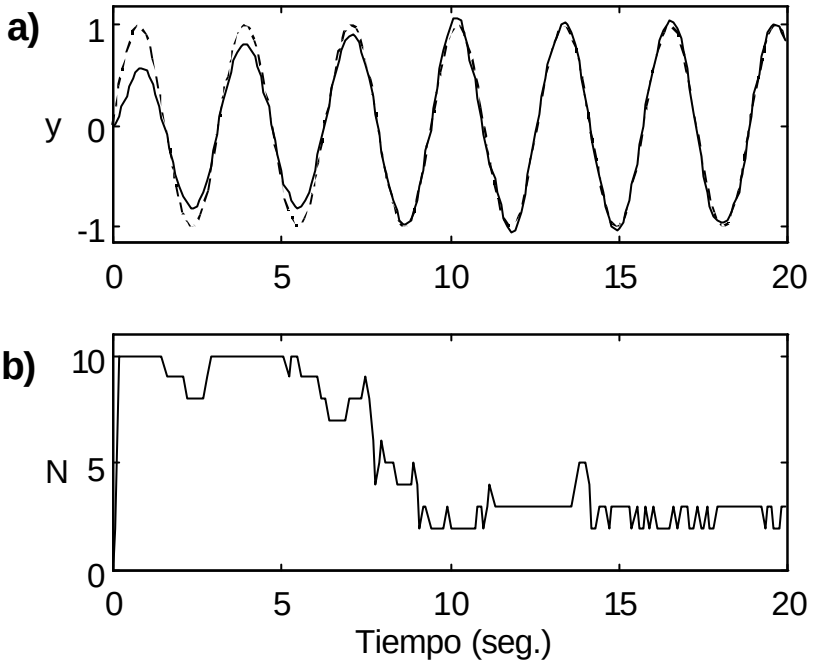


Figura 6.25. Simulación del NAGPC sobre la planta 3. a) Salida de la planta (continuo) y consigna de w=2 rad/seg.(discontinuo). b) Evolución del horizonte de salida *N*.

6.5.3.2 SIMULACIÓN CON PLANTAS DE FASE NO-MÍNIMA

Sobre la planta 4 se han llevado a cabo diferentes simulaciones del algoritmo NAGPC. Como se muestra en la tabla 6.3 esta planta es de fase no mínima ya que tiene un cero en z_1 = 0.1. Sus dos polos reales están ubicados en s_1 = -1 y s_2 = -2. Como es bien conocido el control de plantas de fase no-mínima requiere un cuidadoso diseño del controlador para que la respuesta en lazo cerrado del sistema sea satisfactoria. Como ejemplo obsérvese la figura 6.26 donde se muestra el control de esta planta empleando un PID. A pesar de que los parámetros han sido ajustados para obtener un buen régimen transitorio, la respuesta obtenida ofrece un comportamiento indeseable sobre todo en los primeros instantes de la trayectoria. Sin embargo, al insertar un controlador GPC se obtiene una mejor respuesta, como se observa en la figura 6.27 (línea discontinua). Esto pone de manifiesto el excelente comportamiento de este controlador para tratar con plantas de fase no-mínima.

En las mismas condiciones se ha aplicado el NAGPC sobre esta planta. La salida obtenida es la que se muestra en la figura 6.27 con trazo continuo. Es interesante observar la mejoría que existe en relación a la salida obtenida con un horizonte de salida

fijo. Ahora el horizonte se ajusta adaptivamente dependiendo del estado en el que se encuentre el sistema. La evolución de *N* para esta experiencia se muestra en la figura 6.28.

Al considerar como consigna sistema una señal sinusoidal $r(t) = \text{sen}(20t)$, la aplicación del GPC con un valor para *N* inadecuado conduce a una respuesta en la que el seguimiento de la consigna no es aceptable. Sin embargo, al aplicar el controlador NAGPC se encuentra la evolución que se muestra en la figura 6.29 donde se observa que tras un transitorio el sistema consigue un seguimiento de la señal de referencia con un error despreciable.

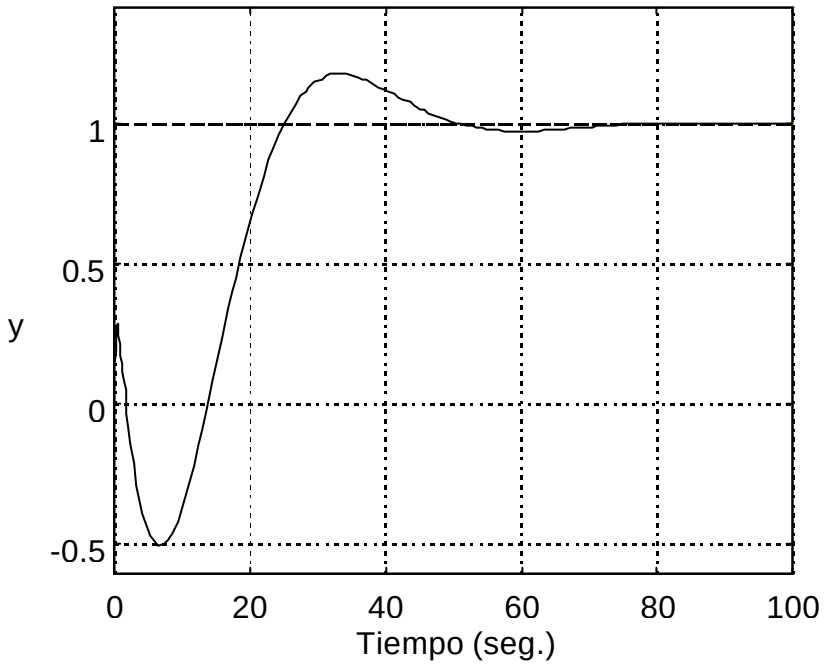


Figura 6.26. Control PI de la planta 4. $K_p = 2$, $K_i = -1$.

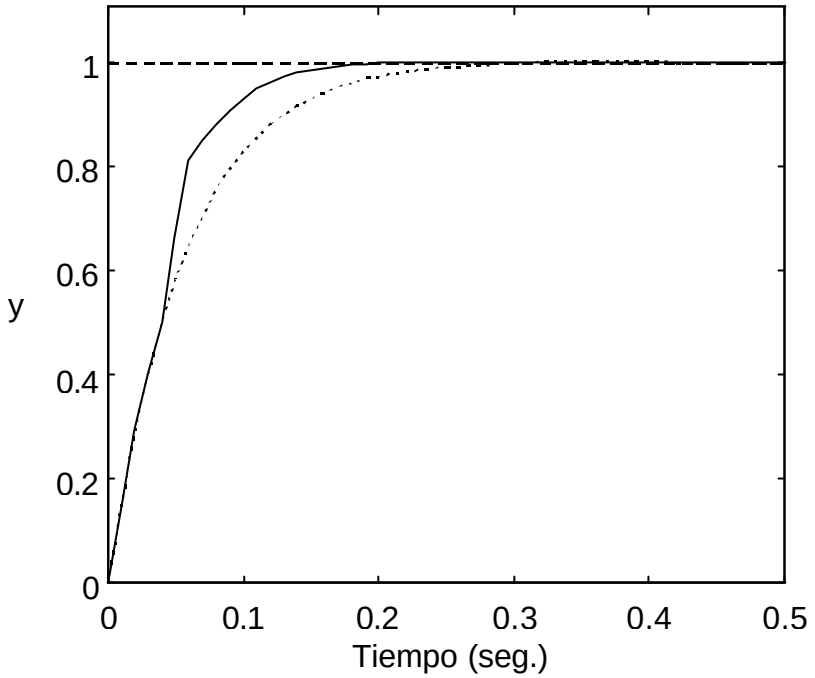


Figura 6.27. Control NAGPC (línea continua) y control GPC (línea discontinua) de la planta 4 tomando N=10.

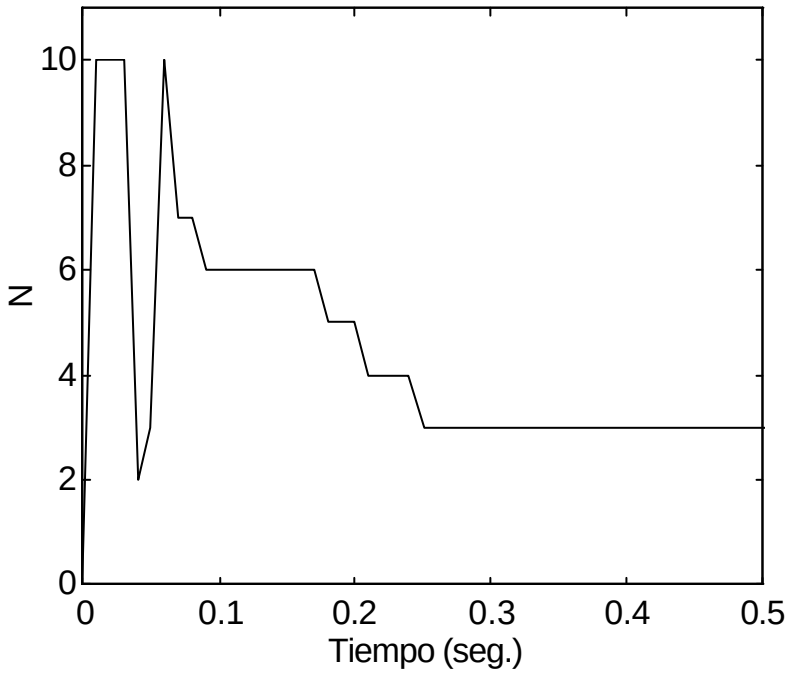


Figura 6.28. Evolución de N en la simulación de la figura 6.27.

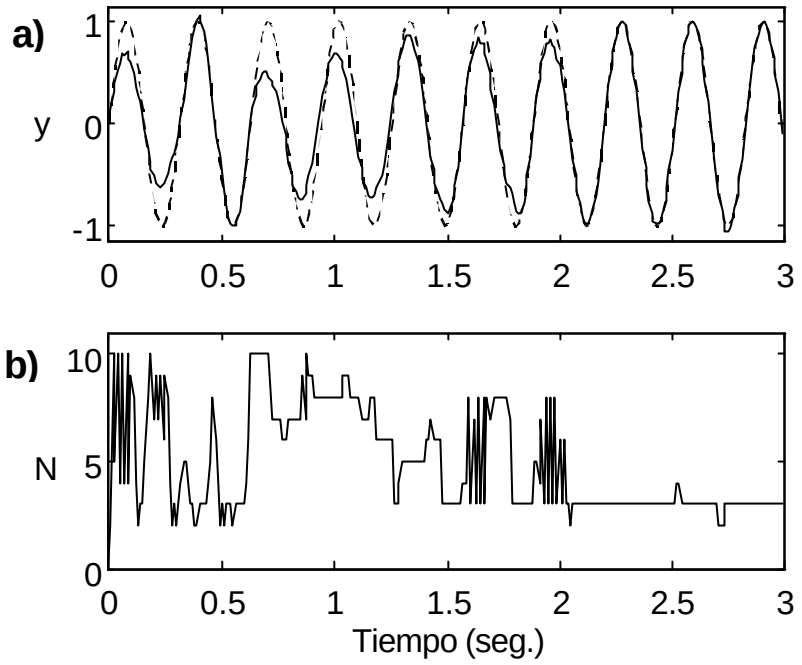


Figura 6.29. Control NAGPC con consigna $r(t)$=sen(20t). a) salida (línea continua). b) Evolución de N.

6.5.3.3 SIMULACIÓN CON PLANTAS INESTABLES EN LAZO ABIERTO

Como ejemplo de planta inestable en lazo abierto se ha tomado la planta 5 que aparece en la tabla 6.3. Esta planta es de tercer orden y presenta sus polos en s_1 = -10.0, s_2 = -5.0 y s_3 = 0.1.

Tomando como consigna $r(t)$ = sen(2t), si a esta planta se le aplica el controlador NAGPC se obtiene la respuesta mostrada en la figura 6.30. Como puede observarse, de nuevo la salida del sistema presenta una etapa transitoria terminada la cual, alcanza una situación estacionaria en la que sigue a la señal de referencia con un *IES* bajo. Es de notar en este caso que el valor de N_{opt} se encuentra para valores altos de N a diferencia de los casos anteriores.

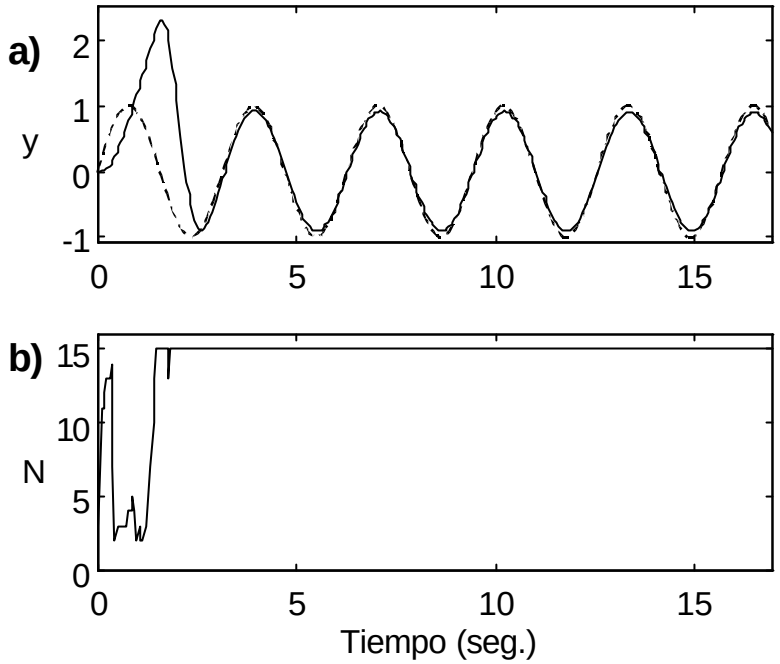


Figura 6.30. Control NAGPC con consigna $r(t)$=sen(20t). a) salida (línea continua). b) Evolución de N.

6.5.4 UN EJEMPLO DE APLICACIÓN: PUNTERO TRAZADOR

Como ejemplo de aplicación del controlador GPC adaptivo neuromórfico estudiaremos un sistema de trazado. Supóngase un dispositivo formado por un puntero que puede desplazarse en el plano por medio de dos motores de corriente continua controlados en el inducido que definen sus coordenadas x e y respectivamente. Consideraremos que los dos motores son idénticos y que están desacoplados, esto es, que pueden actuar independientemente uno del otro. El objetivo de este sistema es el de trazar figuras en el plano. Para ello se actúa sobre cada uno de los motores aplicando voltajes en el inducido que se traducen en un desplazamiento angular, y por lo tanto en un desplazamiento del puntero en esa dirección. En la figura 6.31 se muestra un diagrama esquemático de este dispositivo.

La función de transferencia considerada para los motores es $G(s)=1/s(s+10)$. Nos planteamos inicialmente el control de este sistema por medio de un control GPC con horizonte de predicción constante e igual a 10. En la figura 6.32 se muestra, con trazo continuo, el resultado obtenido. Se puede apreciar que con este horizonte de predicción fijo se comete un error importante en el seguimiento del trazado. Si se estudia independientemente cada motor se puede observar que en los puntos de mayor variación de la pendiente de la consigna la

respuesta del motor presenta errores considerables. Esto se pone de manifiesto en la figura 6.33 donde aparece la respuesta de cada motor con el tiempo.

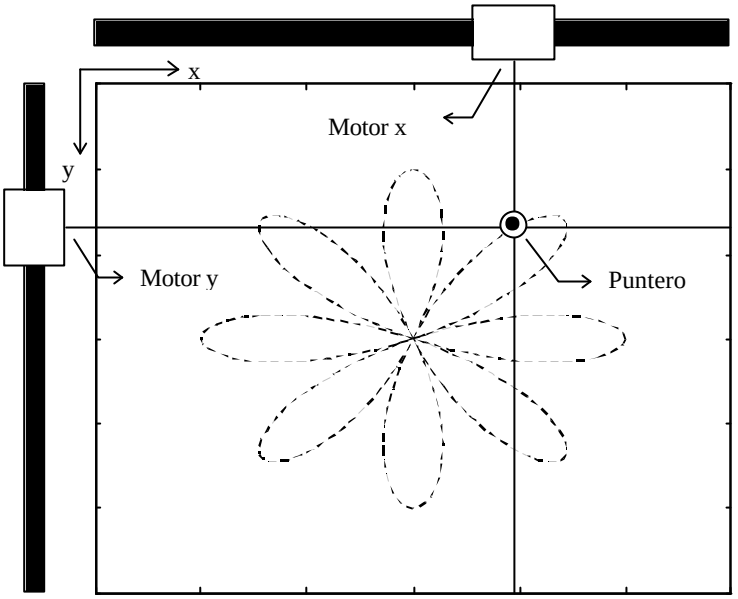


Figura 6.31. Diagrama esquemático del puntero trazador.

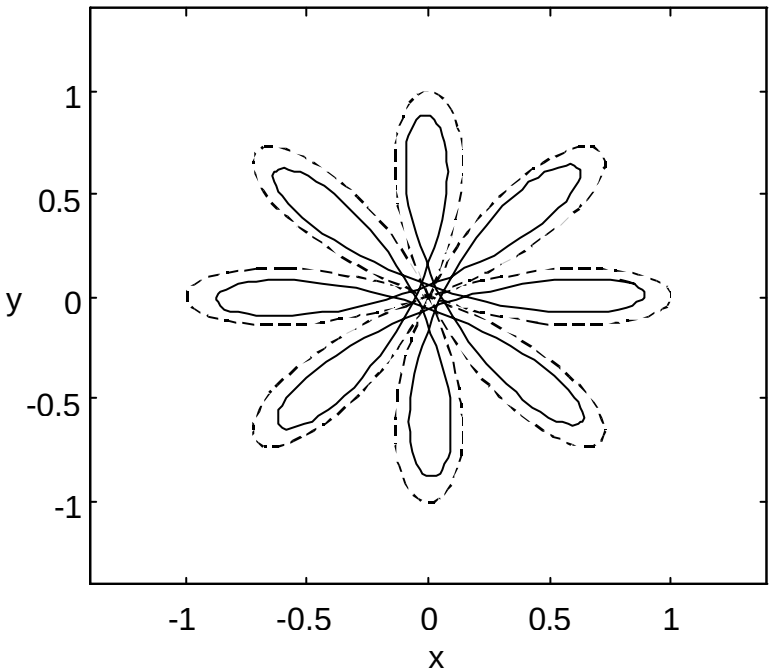


Figura 6.32. Respuesta del sistema trazador controlando los motores con un GPC con N=10.

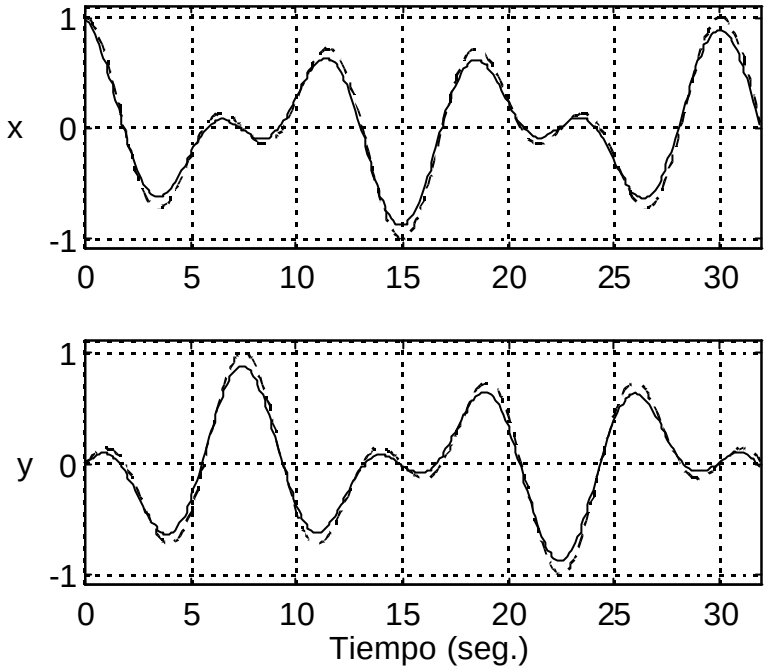


Figura 6.33. Respuesta temporal de cada motor para la experiencia de la figura 6.32.

Sobre este mismo sistema se ha aplicado el controlador NAGPC. La respuesta obtenida es la que se muestra en la figura 6.34. En este caso se puede observar una clara mejoría con respecto al GPC no adaptivo. Ahora el error cometido en el trazado de la figura es prácticamente nulo aumentando considerablemente las prestaciones del dispositivo. El estudio de los dos motores por separado pone más claramente de manifiesto un error de seguimiento muy pequeño. Esto se observa en las figuras 6.35 y 6.36 donde se muestra la evolución temporal de cada motor independientemente. En estas figuras se muestra también la evolución del horizonte de salida. Obsérvese cómo se estabiliza el valor de N en torno a un valor (N_{opt}).

A la vista de los resultados obtenidos en las diferentes simulaciones llevadas a cabo, se puede asegurar que el controlador GPC neuronal adaptivo, diseñado para la determinación en tiempo real de los horizontes de predicción, presenta unas prestaciones muy aceptables. Especialmente en el caso en que se tengan consignas variables este esquema de ajuste de los horizontes es especialmente interesante para lograr tanto un buen seguimiento de la consigna como una optimización en el número de operaciones en el algoritmo.

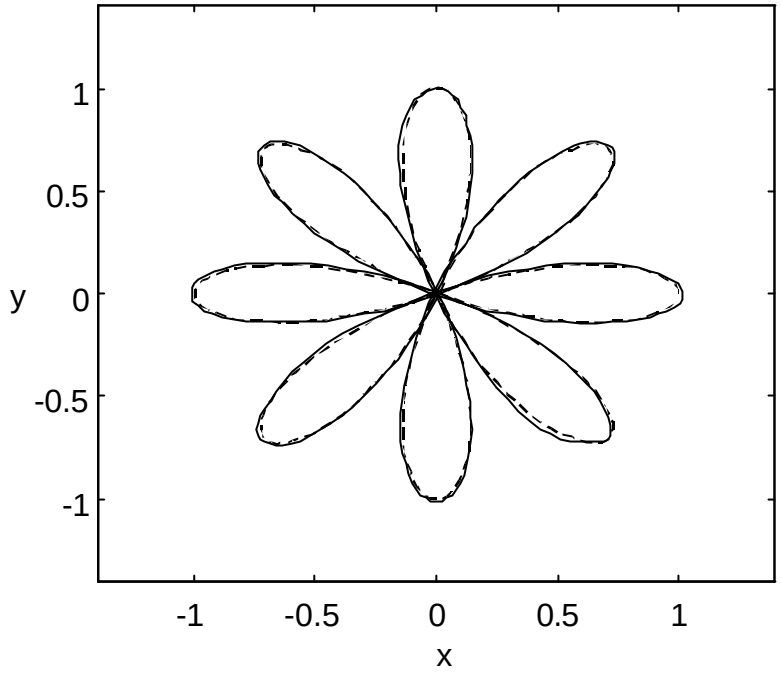


Figura 6.34. Respuesta del sistema trazador controlando los motores con NAGPC.

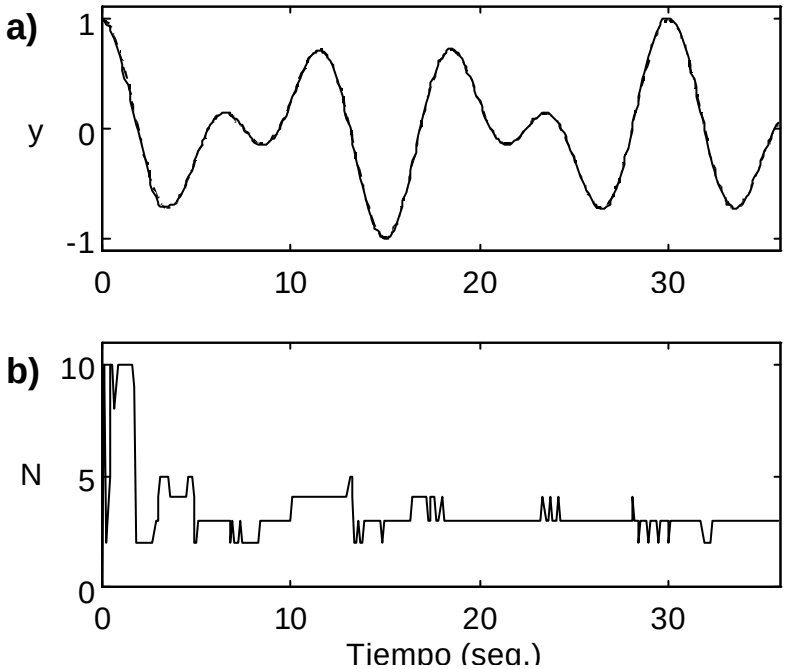


Figura 6.35. Respuesta temporal del *motor x* en la experiencia de la figura 6.34. a) salida (continuo), b) Evolución de *N*.

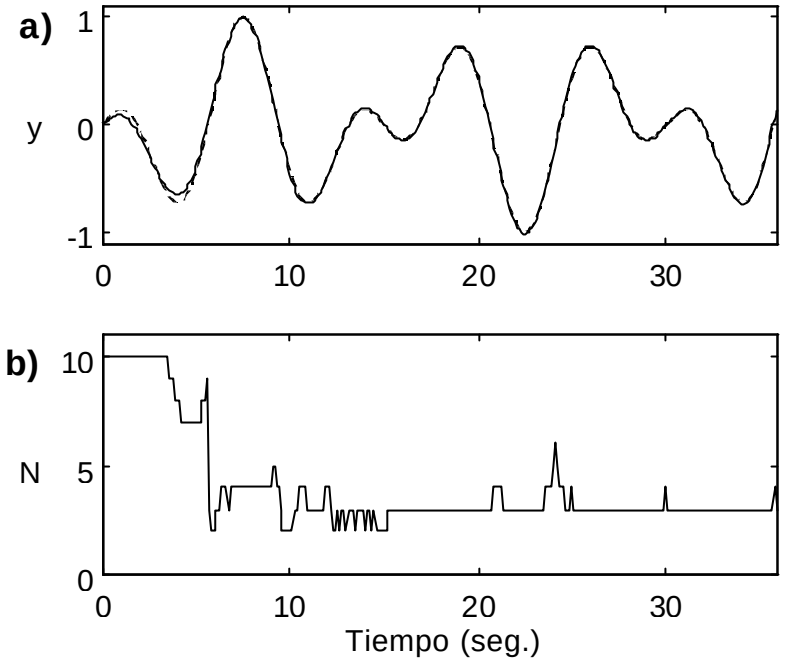


Figura 6.36. Respuesta temporal del *motor y* en la experiencia de la figura 6.34. a) salida (continuo), b) Evolución de *N*.

PRINCIPALES APORTACIONES CONCLUSIONES Y LÍNEAS ABIERTAS

En los siguientes puntos se resumen las principales aportaciones de este trabajo:

APLICABILIDAD DEL GPC E IMPLEMENTACIÓN SOBRE UNA PLANTA REAL:

- Se han analizado las características del controlador GPC poniendo de manifiesto mediante simulación su buen rendimiento para el control de plantas de fase no-mínima e inestables.

- Se ha diseñado un algoritmo de control basado en el GPC para controlar un sistema real de posición y velocidad consistente en un motor DC dispuesto sobre una bancada con masa de inercia. El algoritmo diseñado permite abordar el control multivariable pero utilizando un esquema SISO que se aplica independientemente a cada una de las variables

a controlar. La idea fundamental es multiplexar temporalmente las acciones de control de forma que en cada instante de tiempo sólo sea necesario el cálculo del comando empleando un algoritmo SISO. Globalmente con esta estrategia se consigue un control del sistema que consigue cumplir las especificaciones multivariable impuestas. La principal ventaja que presenta este método es que se puede abordar el control multivariable de un sistema por medio de un algoritmo pseudo-multivariable que conlleva menor complejidad de cálculo y, por lo tanto, permite garantizar la aplicabilidad del GPC desde el punto de vista computacional incluso para plantas con constantes de tiempo bajas.

- Se ha implementado este controlador sobre la planta del motor DC. La implementación se ha realizado considerando un planteamiento determinista y uno estocástico. Para el diseño estocástico del controlador ha sido necesario un estudio de las perturbaciones a las que está sometido el sistema y la obtención de un modelo que las represente. La implementación del GPC sobre la planta ha sido del todo satisfactoria, consiguiéndose el objetivo de control tanto en el caso determinista como en el estocástico.

- Una medida de los tiempos requeridos por el algoritmo de control planteado, pone de manifiesto la alta carga computacional que conlleva el algoritmo y la necesidad de un procesador relativamente potente para afrontar el control.

- La comparación de los resultados obtenidos con el controlador determinista y con el estocástico muestra, como era de esperar, que un diseño en el que se incorporen las características de ruido a las que está sometido el sistema presenta un rendimiento superior que una estrategia puramente determinista. El análisis de la respuesta del sistema permite descubrir efectos característicos inherentes a una estrategia estocástica, como es el efecto de *caution*.

REDUCCIÓN DE LA COMPLEJIDAD COMPUTACIONAL:

- Se ha diseñado un algoritmo GPC en el que se incluyen características de robustez con el objeto de reducir la carga computacional del algoritmo. Este esquema consiste en la introducción de un lazo de realimentación robusta en la planta antes de aplicar el controlador GPC.

- Se han realizado diferentes simulaciones sobre plantas con distintas características. Como conclusión hay que destacar que con la aplicación del esquema combinado de estabilización robusta con el controlador GPC es posible obtener una ley de control que garantiza un comportamiento del sistema en lazo cerrado mucho más estable que el obtenido empleando una acción GPC únicamente. Debido a esto, es posible reducir los horizontes de predicción sin que se pierdan prestaciones en el controlador en lazo cerrado. Como consecuencia inmediata se produce una disminución en el número de operaciones necesarias en el algoritmo, llegando en algunos casos a convertirse en simples cálculos escalares. Esto permite asegurar la aplicabilidad del GPC desde el punto de vista computacional en una implementación en tiempo real.

SINTONIZACIÓN AUTOMÁTICA DE LOS PARÁMETROS DEL CONTROLADOR GPC:

- Se ha diseñado un esquema de control que permite la sintonización inteligente de los parámetros propios del controlador. Esta estructura de control consta de un controlador convencional y otro subsistema de redes neuronales que realiza la sintonización automática de los parámetros del controlador. Para el entrenamiento de las redes se ha empleado el algoritmo *backpropagation*. Una de las principales ventajas de este esquema es que el entrenamiento de las redes se realiza *on-line*. Otra característica importante es que a diferencia de esquemas convencionales de auto-sintonización no se necesita un modelo del proceso a controlar. La eficacia de este controlador se puso de manifiesto mediante la

simulación empleando un controlador PI. Los resultados obtenidos fueron satisfactorios, observándose una evolución adaptativa de los parámetros del controlador PI actuando sobre plantas mal condicionadas.

Como una aplicación más avanzada de este controlador se planteó la aplicación a una planta no lineal. La planta en cuestión es un puente de grúa. El objetivo de control planteado fue el de controlar el balanceo en el transporte de contenedores. Para ello se utilizó un controlador por realimentación de variables de estado. El objetivo de las redes neuronales es el de sintonizar las ganancias de realimentación de este controlador. En las mismas condiciones se aplicó un esquema *self-tuning* clásico para comprobar el rendimiento del neurocontrolador. Diferentes simulaciones desarrolladas ponen de manifiesto un comportamiento muy positivo del controlador neuronal propuesto. Este hecho se corrobora a través de la comparación con el controlador *self-tuning* clásico, en la cual se puede comprobar que el esquema neuronal es capaz de obtener una respuesta superior. Como ventajas respecto al método clásico, se encuentra el hecho de que no es necesario emplear técnicas de linealización de la planta para controlarla, y sobre todo que el diseño conceptual es más simple que el que rodea a un controlador adaptativo clásico. En cuanto a la implementación, un análisis de la carga computacional requerida demuestra que la aplicabilidad de esta estrategia en tiempo real se puede garantizar para un procesador estándar.

Se ha construido un prototipo a escala de un puente de grúa para comprobar la eficiencia de la estrategia en una planta real. Para la implementación se ha seguido un proceso consistente en implementar primero el algoritmo en simulación, y posteriormente trasladar los resultados obtenidos una vez entrenada la red sobre la planta real. Los evolución obtenida en la implementación en tiempo real nos permite corroborar la eficiencia de la estrategia con este tipo de plantas.

- Con el objeto de estudiar el problema de la elección del horizonte de salida en el GPC con consignas variables, se ha realizado un análisis centrado en sistemas de segundo orden considerando como consigna una señal sinusoidal. Los resultados de este análisis son los siguientes:

♦ Existe un valor del horizonte N, que produce un error de seguimiento de la consigna mínimo. Experimentalmente se comprueba que el error de seguimiento aumenta monótonamente a medida que N se aleja de este valor óptimo.

♦ Si los polos de la planta tienen su parte imaginaria igual a cero, entonces el valor óptimo del periodo de predicción (producto del horizonte de predicción por el periodo de muestreo) es invariante con la frecuencia de muestreo tomada. Por el contrario, si los polos tienen su parte imaginaria distinta de cero, el valor del periodo de predicción óptimo depende del valor de la frecuencia de muestreo de la planta. Aunque el comportamiento cualitativo del error de seguimiento en función del periodo de predicción es similar para la misma planta muestreada a diferentes frecuencias, lo cual permite generalizar los resultados obtenidos con una frecuencia de muestreo determinada.

♦ Atendiendo al efecto de los polos de la planta en el error de seguimiento se encuentra que variaciones en la parte real de los polos producen cambios en el desfase medido entre la salida del sistema y la consigna, mientras que variaciones sobre la parte imaginaria producen cambios en la amplitud de la respuesta del sistema.

♦ Se han estudiado plantas del tipo $1/(s^2+2\delta w_n s + w_n^2)$ sujetas a una consigna de frecuencia constante encontrándose que el valor óptimo del periodo de predicción se puede considerar dependiente únicamente de w_n a través de una expresión cuadrática $aw_n^2 + b$. Para estas plantas la evolución del error de seguimiento se encontró igualmente que se podía expresar como una función cuadrática de la frecuencia natural.

♦ Se ha realizado un estudio de la evolución del periodo óptimo de predicción considerando plantas en las que sólo se varía la ganancia. Los resultados obtenidos ponen de manifiesto que es posible ajustar el periodo óptimo de predicción a una expresión del tipo $ak^{-1/2}$. siendo a una constante dependiendo de la ubicación de los polos de la planta y k la ganancia de la planta.

♦ De igual forma se ha estudiado el comportamiento de plantas del tipo $w_n^2/(s^2+2\delta w_n s + w_n^2)$. En este caso es posible ajustar la evolución del periodo de predicción óptimo a una expresión del tipo $aw_n^{-1/2}$.

- Se ha aplicado el controlador neuronal citado anteriormente a la sintonización automática del horizonte de predicción en el GPC. La estructura de este controlador (NAGPC, *Neuromorphic Adaptive Generalized Predictive Control*) se basa en el empleo de una red que se entrena *on-line* y suministra en cada instante de muestreo el valor de N adecuado para el GPC. A la vista de los resultados obtenidos en las diferentes simulaciones llevadas a cabo, se puede asegurar que el controlador GPC neuronal adaptivo, diseñado para la determinación en tiempo real de los horizontes de predicción, presenta unas prestaciones satisfactorias. La eficacia de este controlador se pone de manifiesto especialmente cuando la consigna es variable. En este caso el esquema propuesto permite lograr tanto un buen seguimiento de la consigna como una optimización en el número de operaciones en el algoritmo.

A la vista de los resultados obtenidos y como continuación a la labor realizada, surgen nuevas líneas de investigación y desarrollo con el objetivo de aportar mayores avances a las propuestas que se han realizado en este trabajo. De entre estas líneas cabe citar las siguientes:

- Aplicación de algoritmos de control predictivos a plantas complejas. En virtud del acuerdo de colaboración entre el grupo de **Computadoras y Control** de La Universidad de La Laguna y la empresa **GRANTECAN**, que afronta la construcción de un telescopio segmentado de 10m. de diámetro, se pretende aplicar algunas de las técnicas expuestas en este trabajo para el control del espejo primario del telescopio.

- Utilización de algoritmos genéticos como un sistema de optimización alternativo a las redes neuronales.

- Búsqueda de un método eficiente que permita mejorar la fase inicial de entrenamiento de las redes neuronales en el algoritmo neuronal adaptivo.

- Inclusión de las condiciones de ligadura del sistema, tales como restricciones en el comando o en variables de salida, en el algoritmo de diseño neuronal de forma eficaz.

- Optimización del código para la implementación del algoritmo neuronal adaptivo sobre el prototipo del puente de grúa entrenando las redes en tiempo real.

Apéndice A

CÁLCULO DE LA LEY DE CONTROL GPC CON LIGADURAS EN EL SISTEMA

En este apéndice se explica cómo se resuelve el problema del cálculo de la ley de control GPC cuando se consideran ligaduras en el sistema. En la primera parte se muestra como se pueden representar las condiciones de ligadura mediante una expresión matricial y posteriormente se presenta el procedimiento para calcular la política de control sujeta a estas ligaduras.

A.1 EXPRESIÓN DE LAS LIGADURAS

Considérese un sistema sujeto a las ligaduras (1.21):

$$\begin{aligned} &u_m \le u(t) \le u_M\,,\ \forall t \\ &du_m \le u(t) - u(t-1) \le du_M;\quad \forall t \\ &y_m \le y(t) \le y_M;\ \ \forall t \end{aligned} \tag{A.1}$$

Teniendo en cuenta las expresiones (1.36) y (1.37b), estas ligaduras se pueden expresar de la siguiente forma:

$$\begin{aligned} &\mathbf{I}_{N,1}u_m \le \mathbf{T}\mathbf{u} + u(t-1)\mathbf{I}_{N,1} \le \mathbf{I}_{N,1}u_M \\ &\mathbf{I}_{N,1}du_m \le \mathbf{u} \le \mathbf{I}_{N,1}du_M \\ &\mathbf{I}_{N,1}y_m \le \mathbf{G}\mathbf{u} + \mathbf{f} \le \mathbf{I}_{N,1}y_M \end{aligned} \tag{A.2}$$

donde $\mathbf{I}_{N,1}$ es un vector de dimensión N con todos sus elementos iguales a uno, $\mathbf{T}$ es una matriz triangular inferior con todos sus elementos iguales a uno. Estas condiciones se pueden expresar de forma condensada como:

$$\mathbf{R}\mathbf{u} \le \mathbf{c} \tag{A.3}$$

donde

$$\mathbf{R} = \begin{bmatrix} \mathbf{I}_{NxN} \\ -\mathbf{I}_{NxN} \\ \mathbf{T} \\ -\mathbf{T} \\ \mathbf{G} \\ -\mathbf{G} \end{bmatrix}, \ \mathbf{c} = \begin{bmatrix} \mathbf{I}_{Nx1}du_M \\ -\mathbf{I}_{Nx1}du_m \\ \mathbf{I}_{Nx1}u_M - \mathbf{I}_{Nx1}u(t-1) \\ -\mathbf{I}_{Nx1}u_m + \mathbf{I}_{Nx1}u(t-1) \\ \mathbf{I}_{Nx1}y_M - \mathbf{f} \\ -\mathbf{I}_{Nx1}y_m + \mathbf{f} \end{bmatrix} \tag{A.4}$$

De esta forma se consigue expresar todas las condiciones de ligaduras del proceso mediante una única expresión matricial.

A.2 CÁLCULO DE LA LEY DE CONTROL CON LIGADURAS

Como se estudió en el capítulo 1, la política de control en el GPC se obtiene minimizando la función de coste cuadrática:

$$J(N_1, N_2, NU) = \sum_{j=N_1}^{N_2} g(j)\left[w(t+j) - \hat{y}(t+j|t)\right]^2 + \sum_{j=1}^{NU} l(j)\left[\Delta u(t+j-1)\right]^2$$

Teniendo en cuenta (1.36), este índice también se puede escribir como

$$J = (\mathbf{Gu} + \mathbf{f} - \mathbf{w})^T (\mathbf{Gu} + \mathbf{f} - \mathbf{w}) + l\,\mathbf{u}^T\mathbf{u} \tag{A.5}$$

O, de otra forma

$$J = \frac{1}{2}\mathbf{u}^T\mathbf{H}\mathbf{u} + \mathbf{b}^T\mathbf{u} + \mathbf{f}_0 \tag{A.6}$$

donde

$$\begin{aligned} \mathbf{H} &= 2(\mathbf{G}^T\mathbf{G} + l\,\mathbf{I}) \\ \mathbf{b}^T &= 2(\mathbf{f} - \mathbf{w})^T\mathbf{G} \\ \mathbf{f}_0 &= (\mathbf{f} - \mathbf{w})^T(\mathbf{f} - \mathbf{w}) \end{aligned} \tag{A.7}$$

Si no existen ligaduras la solución de este problema se puede obtener de forma analítica mediante técnicas estándar. Por el contrario, cuando se incluyen ligaduras en el proceso el problema de optimización se traduce en un problema de programación cuadrática. Ahora el problema consiste en:

$$\textit{minimizar}\ \ J(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T\mathbf{H}\mathbf{u} + \mathbf{b}^T\mathbf{u} + \mathbf{f}_0$$

$$\textit{sujeto a la condición}\qquad \mathbf{Ru} \le \mathbf{c}$$

Existen diferentes técnicas de programación cuadrática para resolver este problema. En la siguiente sección se describe una de las más conocidas: *los métodos de conjunto activo.* Estos métodos se basan en resolver un problema de optimización con ligaduras de desigualdad transformando este problema a otro de optimización con ligaduras de igualdad.

A.2.1 RESOLUCIÓN DEL PROBLEMA DE OPTIMIZACIÓN CON LIGADURAS DE IGUALDAD

Como se comentó, los métodos de conjunto activo se basan en resolver el problema de optimización reduciéndolo a un problema con ligaduras de igualdad. El problema de

programación cuadrática con ligaduras de igualdad se puede formular como [Cam95], [Flet87]:

$$\textit{minimizar}\ \ J(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T\mathbf{H}\mathbf{u} + \mathbf{b}^T\mathbf{u} + \mathbf{f}_0$$

$$\textit{sujeto a la condición}\qquad \mathbf{A}\mathbf{u} \le \mathbf{a}$$

donde **A** es una matriz de dimensión *m*x*n*, **a** es un vector de dimensión *m* y **u** tiene dimensión *n*. Para que el problema tenga solución se asume que el rango de **A** es *m* y que $m < n$.

La forma de resolver este problema es emplear las condiciones de ligadura para eliminar variables y reducir el problema a una minimización sin ligaduras. Una forma directa de hacer esto es expresar, usando las ligaduras, *m* de las variables **u** en función de las *n*-*m* variables restantes. Así se transforma el problema en una función cuadrática de *n*-*m* variables y sin ligaduras.

Esta forma directa no es la que se suele emplear habitualmente. En su lugar se utiliza un método de eliminación generalizado. Este método consiste en expresar **u** como una función de un conjunto reducido de *n*-*m* variables, **u** = **Ya** +**Zv**. Donde **Y** y **Z** son matrices de dimensiones *n*x*m* y *n*x(*n*-*m*) que cumplen **AY** = **I**, **AZ** = 0 y que el rango de [**Y Z**] es completo. La matriz **Y** se puede interpretar como una inversa por la izquierda generalizada de $\mathbf{A}^T$. Por otra parte **Zv** es el espacio nulo de $\mathbf{A}^T$.

Sustituyendo la variable **u** en (A.6), la función criterio *J* se convierte en

$$\begin{aligned} J(\mathbf{v}) &= \frac{1}{2}[\mathbf{Y}\mathbf{a} + \mathbf{Z}\mathbf{v}]^T\mathbf{H}[\mathbf{Y}\mathbf{a} + \mathbf{Z}\mathbf{v}] + \mathbf{b}^T[\mathbf{Y}\mathbf{a} + \mathbf{Z}\mathbf{v}] + \mathbf{f}_0 = \\ &= \frac{1}{2}\mathbf{v}^T\mathbf{Z}^T\mathbf{H}\mathbf{Z}\mathbf{v} + [\mathbf{b}^T + \mathbf{a}^T\mathbf{Y}^T\mathbf{H}]\mathbf{Z}\mathbf{v} + [\frac{1}{2}\mathbf{a}^T\mathbf{Y}^T\mathbf{H} + \mathbf{b}^T]\mathbf{Y}\mathbf{a} + \mathbf{f}_0 \end{aligned} \tag{A.8}$$

De esta forma el problema se ha convertido en un problema de programación cuadrática de *n*-*m* variables sin ligaduras. Se puede garantizar que si la matriz $\mathbf{Z}^T\mathbf{H}\mathbf{Z}$ es definida positiva, existe un único punto óptimo que se encuentra resolviendo el conjunto de ecuaciones lineales:

$$\mathbf{Z}^T\mathbf{H}\mathbf{Z}\mathbf{v} = -\mathbf{Z}^T(\mathbf{b} + \mathbf{H}\mathbf{Y}\mathbf{a}) \tag{A.9}$$

Se debe tener en cuenta que si u^k es un punto que satisface las ligaduras $\mathbf{Au}^k = \mathbf{a}$, cualquier otro punto u que satisfaga las ligaduras se puede poner en la forma $u=u^k+Zv$. De esta forma, el vector **Ya** puede hacerse igual a cualquier otro punto que satisfaga las restricciones. El vector **v** se puede expresar como la solución de la siguiente ecuación lineal:

$$\mathbf{Z}^T\mathbf{HZv} = -\mathbf{Z}^T g(\mathbf{u}^k) \tag{A.10}$$

donde $g(\mathbf{u}^k) = \mathbf{Hu}^k + \mathbf{b}$ es el gradiente de $J(\mathbf{u})$ en $\mathbf{u}^k$.

Existe un procedimiento general para obtener unas matrices **Y** y **Z** apropiadas. Este método consiste en la creación de la matriz **B** de dimensión *nxn* definida como

$$\mathbf{B} = \begin{bmatrix} \mathbf{A} \\ \mathbf{W} \end{bmatrix}^{-1} \tag{A.11}$$

siendo **W** una matriz de dimensión (*n*-*m*)x*n*. La inversa de esta matriz **B** resulta ser

$$\mathbf{B}^{-1} = [\mathbf{Y}\ \mathbf{Z}] \tag{A.12}$$

Se puede comprobar entonces que, **AY** = **I** y **AZ**=0. Obviamente, para que esto sea posible **W** debe ser tal que **B** sea no-singular. Si W se escoge igual a [0 **I**], el método coincide con el de eliminación directa. En general la determinación de W se hace empleando el bien conocido método de optimización del *conjunto activo.*

A.2.2 RESOLUCIÓN DEL PROBLEMA DE OPTIMIZACIÓN CON LIGADURAS DE DESIGUALDAD

El objetivo es transformar este problema a un problema de programación cuadrática con ligaduras de igualdad.

Considérese un punto viable $\mathbf{u}^0$, es decir, un punto que verifica $\mathbf{Ru}^0 \leq \mathbf{c}$ y un conjunto de ligaduras activas (que está formado por todas las ligaduras de igualdad más las filas de **R** tales que $\mathbf{r}_i\mathbf{u}=\mathbf{c}_i$). Se forma la matriz **A** y el vector **a** añadiendo esta filas ($\mathbf{r}_i$) y los correspondientes límites ($\mathbf{c}_i$) y las ligaduras de igualdad.

Una vez creada esta matriz, el problema se resuelve de igual forma que el problema descrito en el apartado anterior. Supóngase que $\mathbf{u}^1$ es la solución al problema con ligaduras de igualdad:

Si $\mathbf{u}^1$ es viable con respecto a las ligaduras inactivas, se realiza una comprobación para verificar que se ha alcanzado un punto óptimo global. Esto se consigue verificando que los multiplicadores de Lagrange son positivos para todas las ligaduras de igualdad. Si no se ha alcanzado el óptimo, la ligadura con el multiplicador de Lagrange más negativo es eliminada del conjunto activo y se repite el proceso.

Si, por el contrario, el punto $\mathbf{u}^1$ no es viable con respecto a las ligaduras inactivas, se calcula el punto de intersección más cercano a $\mathbf{u}^0$ entre la línea que une $\mathbf{u}^0$ y $\mathbf{u}^1$ y las ligaduras inactivas. La ligadura correspondiente se añade al conjunto activo y se repiten los pasos anteriores [Cam95].

Apéndice B

CONCEPTOS TEÓRICOS

En este apéndice se presentan las definiciones de algunos de los conceptos a los que se hace mención en esta memoria. Además, se introduce el concepto de descomposición en valores singulares y se resumen las propiedades más importantes que presentan los valores singulares para sistemas multivariables.

B.1 DEFINICIONES

Considérese el siguiente sistema lineal discreto:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\Delta u(k) \\ y(k) &= \mathbf{C}^T\mathbf{x}(k) \end{aligned} \tag{D.1}$$

Definición 1. Un estado particular *x'* del sistema (D.1) se dice que es *controlable* si existe un *N* y una entrada de control u(k), *k*=0, 1, …, *N*-1, que conduce al sistema desde el estado inicial $x(0) = x'$ a $x(N) = 0$.

Definición 2. El sistema es *controlable* si todos los estados son controlables.

Definición 3. Un estado particular *x'* del sistema (1) se dice que es *inobservable* si para cualquier $N > 0$ y $u(k) = 0$, $0 \leq k \leq N$, el estado inicial $x(0) = x'$ produce una salida cero:

$$y(k) = 0; \quad 0 \leq k \leq N$$

que conduce al sistema desde el estado inicial $x(0) = x'$ a $x(N) = 0$.

Definición 4 Un sistema es *observable* si no tiene ningún estado inobservable, excepto $x=0$.

Definición 5. El sistema (1) se dice que es *estabilizable* si todos los modos incontrolables tienen sus correspondientes autovalores estrictamente dentro del circulo unidad.

Definición 6. El sistema (1) se dice que es *detectable* si todos los modos inobservables tienen sus correspondientes autovalores estrictamente dentro del circulo unidad.

Definición 7. El sistema (1) es *estable* si todos los autovalores de la matriz de estado A están dentro del circulo unidad o en la circunferencia de radio 1.

Definición 8. El sistema (1) es *asintóticamente estable* si todos sus autovalores están estrictamente dentro del circulo unidad.

B.2 DESCOMPOSICIÓN EN VALORES SINGULARES

La descomposición en valores singulares (SVD, *Singular Value Decomposition*) es una de las herramientas más importantes en el álgebra moderna y en el análisis numérico. Debido a la naturaleza algebraica en el dominio *s* de muchos problemas de control y la importancia de las características de estabilidad, los valores singulares juegan un papel muy importante en la teoría control.

Los valores singulares de una matriz **A** de rango r, $\mathbf{A}\in C^{mxn}$, se definen como las raices cuadradas no negativas de los autovalores de $\mathbf{A}^*\mathbf{A}$ (* indica traspuesta conjugada), ordenados de tal forma que,

$$s_1 \geq s_2 \geq ... \geq s_p, \qquad p = min\{m, n\} \tag{B.1}$$

Si $r < p$ entonces existen p-r valores singulares iguales a cero,

$$s_{r+1} = s_{r+2} = ... = s_p = 0 .$$

Existe un lema que permite afirmar que para cualquier matriz compleja **A**, $\mathbf{A}\in C^{mxn}$, existen dos matrices unitarias $\mathbf{Y}\in \mathbf{C}^{mxm}$ y $\mathbf{U}\,\hat{I}\,\mathbf{C}^{nxn}$ respectivamente, y una matriz Σ, tales que

$$\mathbf{A} = \mathbf{Y}\begin{bmatrix}\Sigma & 0\\ 0 & 0\end{bmatrix}\mathbf{U}^* \tag{B.2}$$

donde Σ=diag(σ_1, σ_2, …, σ_r) tiene dimensión $m\times n$. La expresión (B.2) se conoce como descomposición en valores singulares de una matriz A. Al conjunto de valores singulares se le suele denotar como

$$s(\mathbf{A}) = \{s_i : i = 1, ..., p\}$$

Al mayor valor singular σ_1 se le denota

$$\overline{s}(\mathbf{A}) \equiv s_1$$

Suponiendo que el rango de la matriz **A** es igual a n, al menor valor singular se le denota como

$$\underline{s}(\mathbf{A}) \equiv s_n$$

B.2.1 PROPIEDADES

Las propiedades más importantes que presentan los valores singulares son

1.- $\overline{s}(\mathbf{A}) = \max_{x \in C^n} \frac{\|\mathbf{A}x\|}{\|x\|}$; $\overline{s}(\mathbf{A})$ es la máxima ganancia de la matriz **A**, siendo **x** cualquier vector.

2.- $\underline{s}(\mathbf{A}) = \min_{x \in C^n} \frac{\|\mathbf{A}x\|}{\|x\|}$; $\underline{s}(\mathbf{A})$ es la mínima ganancia de la matriz **A**, siendo **x** cualquier vector.

3.- $\underline{s}(\mathbf{A}) \le |l_i(\mathbf{A})| \le \overline{s}(\mathbf{A})$, donde λ_i es el i-ésimo autovalor de la matriz **A**.

4.- Si $\mathbf{A}^{-1}$ existe, $\underline{s}(\mathbf{A}) = 1/\overline{s}(\mathbf{A}^{-1})$.

5.- Si $\mathbf{A}^{-1}$ existe, $\overline{s}(\mathbf{A}^{-1}) = 1/\underline{s}(\mathbf{A}^{-1})$.

6.- $\overline{s}(a\mathbf{A}) = |a|\overline{s}(\mathbf{A})$.

7.- $\overline{s}(\mathbf{A}+\mathbf{B}) \le \overline{s}(\mathbf{A}) + \overline{s}(\mathbf{B})$.

8.- $\overline{s}(\mathbf{AB}) \le \overline{s}(\mathbf{A})\overline{s}(\mathbf{B})$

9.- $\underline{s}(\mathbf{A}) - \overline{s}(\mathbf{E}) \le \underline{s}(\mathbf{A}+\mathbf{E}) \le \underline{s}(\mathbf{A}) + \overline{s}(\mathbf{E})$

10.- $\max\{\overline{s}(\mathbf{A}), \overline{s}(\mathbf{B})\} \le \overline{s}([\mathbf{A}\,\mathbf{B}]) \le \sqrt{2}\max\{\overline{s}(\mathbf{A}), \overline{s}(\mathbf{B})\}$

11.- $\max_{i,j} |a_{i,j}| \le \overline{s}(\mathbf{A}) \le n \max_{i,j} |a_{i,j}|$

12.- $\sum_{i=1}^{p} s_i^2 = Traza(\mathbf{A}^*\mathbf{A})$.

Apéndice C

DESCRIPCIÓN DE LA BANCADA CON EL MOTOR DC

En este apéndice se hace una descripción de la planta sobre la cual se ha implementado el controlador GPC. Esta planta es una bancada sobre la cual se haya un motor de corriente continua y una serie de elementos adicionales que la completan: masa de inercia, freno magnético, transductores, etc. En el capítulo 2 se hace una descripción de los detalles del algoritmo de control y de la implementación.

C.1 DESCRIPCIÓN FÍSICA

La planta a controlar es un sistema real de posición y velocidad (ver figura C.1). Se trata del equipo **SAD100** de ***ALECOP*** que consiste en una bancada sobre la que están dispuestos los siguientes elementos [Ale86], [Men93]:

En lo referente al **sistema,** se dispone de (ver figura C.2):

- Un motor de corriente continua excitado por imanes permanentes.

- Una masa de inercia con momento de inercia $J = 11.6 \times 10^{-4}$ Kg m^2..
- Un freno de polvo magnético. Su función es crear un par de frenado sobre el motor.

En la parte de **medida** la planta dispone de los siguientes elementos:

- Una tacodinamo (descrita posteriormente).
- Un captador potenciométrico (descrito posteriormente).
- Un *encoder* digital. Este dispositivo genera un número de impulsos dependiendo del desplazamiento angular del eje. La frecuencia de estos impulsos es una medida de la velocidad de giro del eje.

La planta dispone además de un **panel de actuación** en el que se encuentran los siguientes módulos (ver figura C.3):

- Módulo CS-100. Es un generador de consigna. Genera una tensión proporcional a un desplazamiento angular entre 0º y 360º que se puede utilizar como consigna de posición. Además dispone de un generador de funciones escalón y funciones rampa, que se utilizan como consignas de velocidad.
- Módulo GENERADOR-100. Es un generador de impulsos. Puede generar tantos impulsos como se le indique, o enviar un tren de impulsos de frecuencia proporcional a la tensión que se le haya indicado como referencia.
- Módulo ADDA-100. Es un módulo de tratamiento de datos en el cual se llevan a cabo las conversiones digital-analógico y analógico-digital. Se comentará en detalle en la siguiente sección.
- Módulo D/A-100. Convertidor digital/analógico utilizado para excitar la fuente de potencia cuando se trabaja con señales digitales.
- Módulo DSG-100. Recibe las señales generadas en el *encoder* y las acondiciona para su tratamiento.
- Módulo ALIMENTACIÓN-100. Fuente de potencia que alimenta al motor.
- Módulo VS-100. Dispone de una fuente de corriente continua de ± 15 V. Además proporciona una serie de elementos tales como potenciómetros, sumadores analógicos e inversores.
- Módulo TF-100. Actúa sobre la ganancia de la tacodinamo generando una tensión proporcional a la señal de la misma, ajustable a través de un potenciómetro.
- Módulo PI-100. Es un módulo que tiene implementado de forma analógica un controlador PI.

En las implementaciones realizadas sólo se ha empleado el Modulo ADDA-100, el ALIMENTACIÓN-100, el VS-100 y el TF-100. La alimentación del equipo se realiza a través de la red suministradora de potencia (220V., f~50/60 Hz.) .

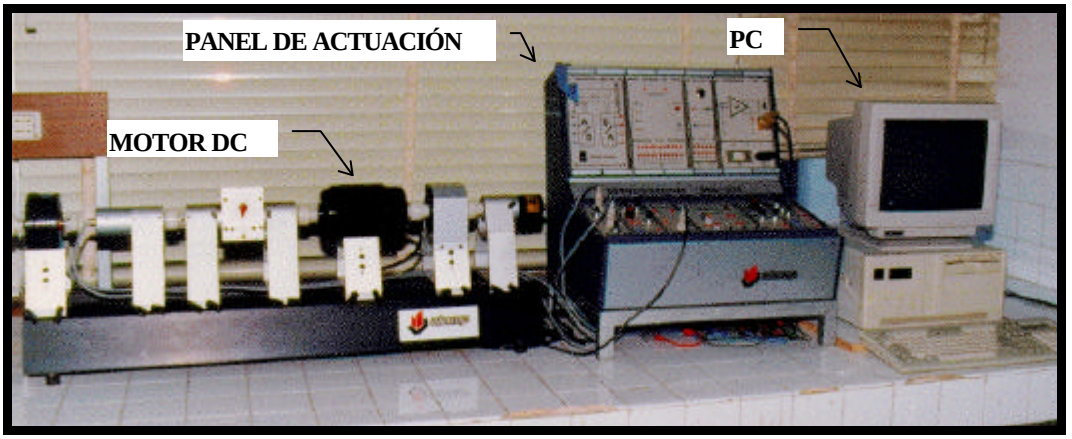


Figura C.1. Vista general del equipo SAD-100.

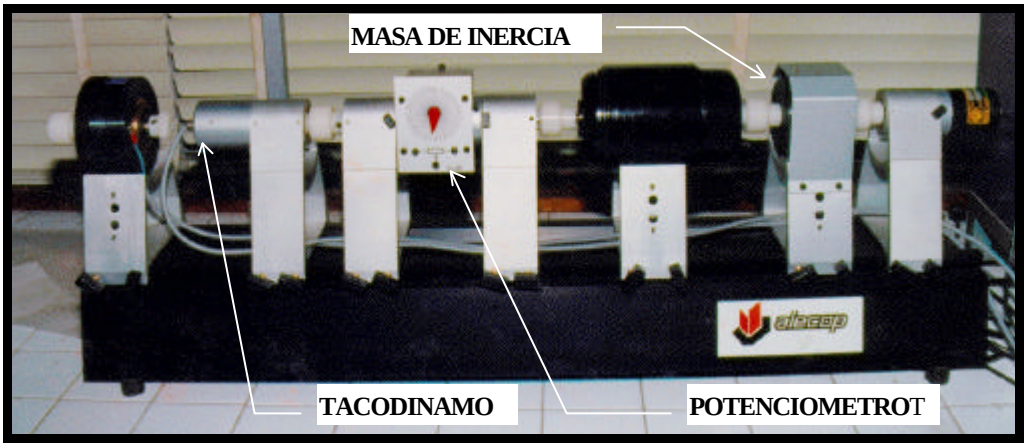


Figura C.2. Vista de la bancada con el motor DC.

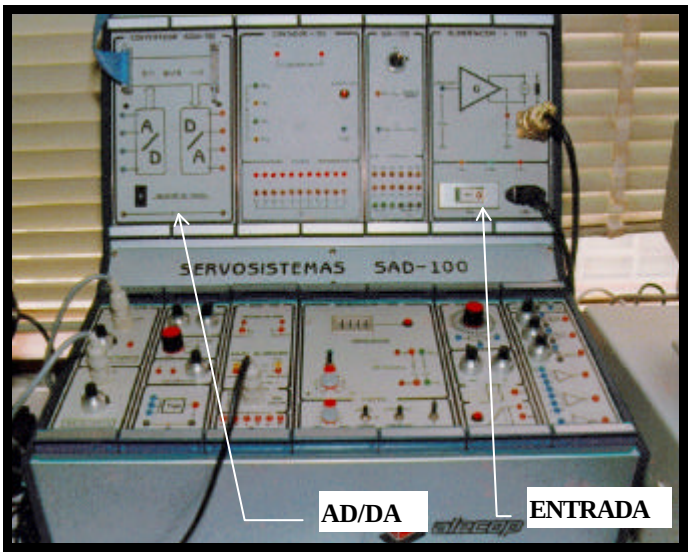


Figura C.3. Vista del panel de actuación del equipo SAD-100.

La actuación sobre el motor se realiza mediante voltajes de corriente continua (V_e) comprendidos en el rango ± 10 V aplicados a la fuente de potencia (P) del Módulo ALIMENTACIÓN-100. Ésta suministra la potencia necesaria al motor (M) para que se mueva (figura C.4). La tensión de salida de la fuente es proporcional a la tensión aplicada a su entrada. El signo de ésta hace que el motor gire hacia un lado u otro.

Otro datos del motor, funcionando en situación óptima, son

Tensión de excitación:	200V.
Tensión en el inducido:	100V.
Potencia:	250W.
Velocidad:	1500rpm.

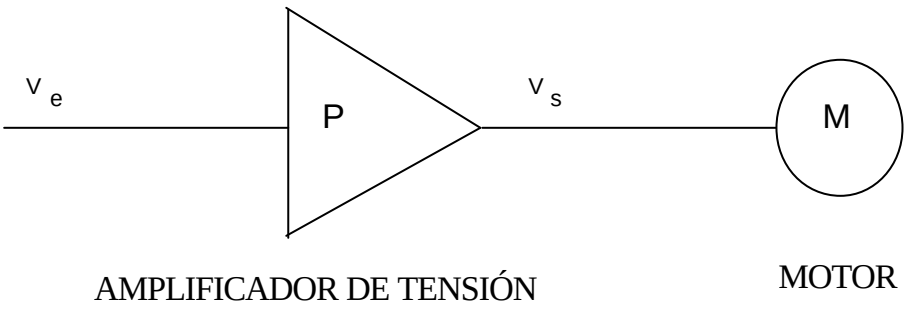


Figura C.4 Etapa de entrada del motor.

El estado del motor queda determinado por su velocidad y posición. Para poder acceder a este estado se dispone de un captador potenciométrico para la posición y una tacodinamo para la velocidad. Estos elementos se describen a continuación:

C.1.1 SENSOR DE POSICIÓN

Este dispositivo es una resistencia potenciométrica lineal, cuya misión es convertir la posición angular del eje de salida del motor en tensión de salida. Para conseguir esto, se acopla su eje al eje reducido del motor. El potenciómetro se alimenta en sus extremos con un determinado voltaje (en la implementación se han empleado ±15V.). La posición que tiene el cursor del mismo dará un voltaje (V_p) proporcional a la posición del eje. El esquema es el que se muestra en la figura C.5.

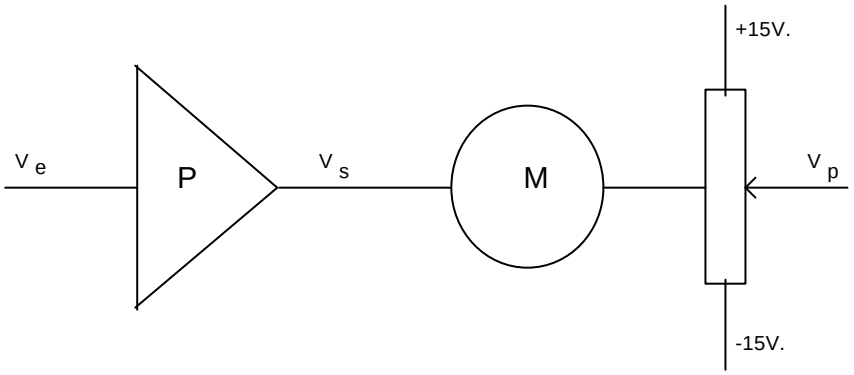


Figura C.5. Sensor de posición.

Hay que notar que con este dispositivo se tiene información de la posición dentro de una vuelta interna del eje, pero no se conoce el número de vueltas del eje.

C.1.2 SENSOR DE VELOCIDAD

El elemento empleado para medir la velocidad es una tacodinamo o taquímetro. Una tacodinamo de corriente continua es un generador que produce una señal a la salida proporcional a la velocidad de rotación del eje. Esta señal se acondiciona a través del módulo TF-100, cuya salida dará un voltaje que será el que se tome como medida de la velocidad.

C.2 ELEMENTOS DEL LAZO DE CONTROL

El lazo de control lo componen los distintos elementos que se ven en la figura C.6. En la sección anterior se describió el motor, el actuador y los sensores. A continuación se describe la función de los otros elementos:

C.2.1 CONVERSORES

La conversión, tanto digital-analógico como analógico-digital, se realizará empleando un módulo de tratamiento de datos ADDA-100. La función del módulo es la lectura y escritura de datos analógicos desde y hacia un ordenador PC. En el proceso intermedio se produce la conversión de datos analógicos a digitales para el tratamiento en el PC, y la conversión de los datos enviados desde el ordenador de digitales a analógicos.

El módulo dispone de cuatro entradas analógicas y cuatro salidas analógicas. El voltaje máximo tanto para las entradas como para las salidas es de +10V. y el mínimo -10V. Estos módulos se pueden conectar en serie, de forma que se consiguen más entradas y salidas analógicas. Para ello cada módulo debe tener asignado un número que los identifique mediante el selector de modulo de que disponen.

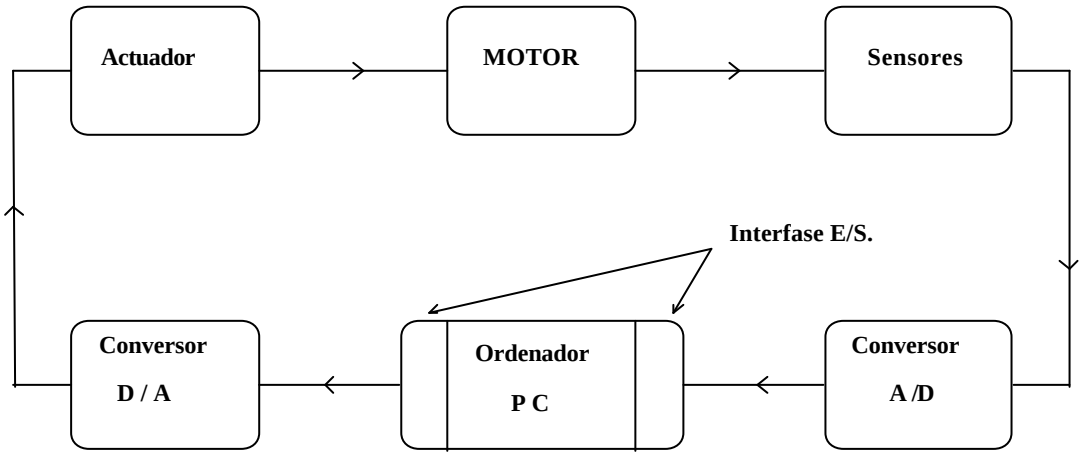


Figura C.6. Elementos del bucle de control.

En la sección anterior se describió el motor, el actuador y los sensores. A continuación se describe la función de los otros elementos:

C.2.2 INTERFASE CON EL ORDENADOR PC

La comunicación entre el ordenador y los módulos se realiza a través de una tarjeta llamada INTERFACE 100. Esta comunicación se lleva a cabo a través de puertos paralelo (PPI-8255).

El 8255 tiene tres puertos de datos (A, B y C) y uno de control. El puerto de control se utiliza para inicializar los puertos de datos como entrada-salida.

Los tres puertos de datos son enviados al exterior mediante un conector tipo D de veinticinco lineas (tres puertos de datos de ocho bits y un línea reservada para la tierra), como se muestra en la figura C.7.

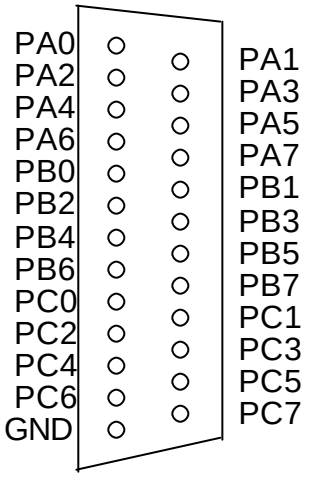


Figura C.7 .Conector de la interfase.

Los puertos están direccionados en:

Puerto A ⟶ 500 H
Puerto B ⟶ 501 H
Puerto C ⟶ 502 H
Puerto de control → 503 H

El puerto A debe estar inicializado como entrada, y los puertos B y C como salidas. Para hacer esto hay que enviar al puerto de control el byte 90 H. Según esto, la disposición de los puertos será tal como se muestra en la figura C.8.

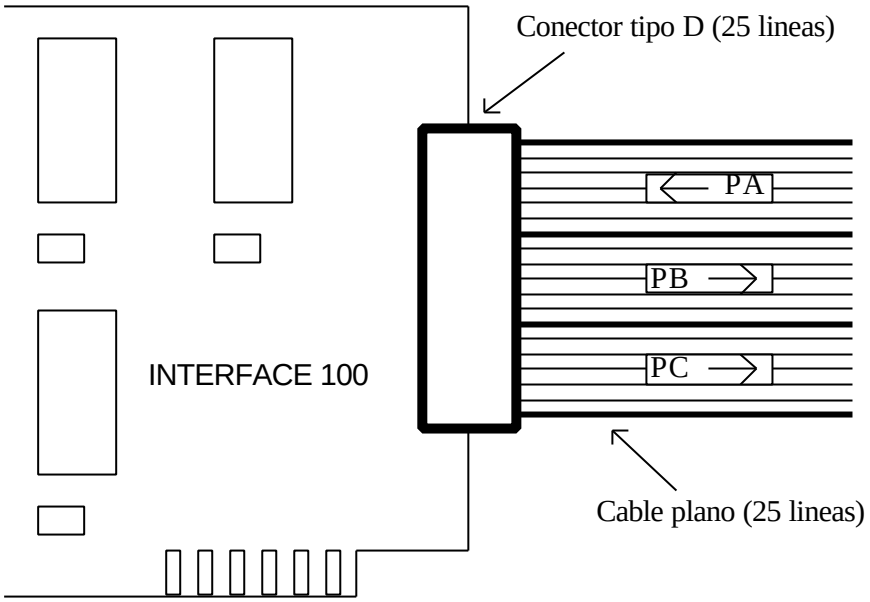


Figura C.8. Esquema de la INTERFACE 100.

De esta forma en el puerto A se hará la lectura del dato digital, por el puerto B se envía el dato digital y con el puerto C se controla el conversor. Para realizar cualquier operación de lectura o escritura se debe enviar una secuencia de bytes al módulo ADDA 100. El formato del byte de control es

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
S	**ES1**	**ES0**	**LE1**	**LE0**	**M2**	**M1**	**M0**

siendo

M2M1M0 → Módulo seleccionado.
LE1LE0 → Selección de lectura o escritura:
Lectura: 11,
Escritura: 10.
ES1ES0 → Entrada o salida a seleccionar.
S → Selección de signo o dato.

En esta interfase el dato está formado por un byte para la magnitud y un bit para el signo. Por ejemplo, la representación de -10 V sería 0FF H con el signo correspondiente almacenado en un bit de otro byte; para representar +10V se tendría la misma magnitud en el byte pero con el bit de signo invertido.

Para estimar la resolución del conversor hay que tener en cuenta que el rango 0→10V. se representa con números que van desde 0 a 0FF H, luego la resolución será

$$\text{Resolución} \rightarrow \frac{10\text{V.}}{255} = 0.039\text{V.}$$

C.2.3 SALIDAS ANALÓGICAS

Para la salida de un dato analógico el proceso a seguir es:

1.- Poner en el puerto B la magnitud del dato en hexadecimal.
2.- Enviar por el puerto C el byte de control correspondiente atendiendo al convenio establecido para el byte de control, siendo S=0 si el dato es positivo ó S=1 si el dato es negativo. Recuérdese que PC4PC3 es 10.
3.- Esperar hasta la conversión del dato. El tiempo de conversión en el ADDA 100 es de 5 μ seg.
4.- Enviar un byte al puerto C para mantener el byte enviado. Este byte será igual que el anterior salvo que PC4 PC3 será 00.

C.2.4 ENTRADAS ANALÓGICAS

El procedimiento que se debe seguir para capturar un dato analógico es:

1.- Enviar el byte de control al puerto C especificando el módulo de donde leer, el canal de entrada y con S=0 para indicar que primero se leerá el signo del dato. PC4PC3 deben estar a 11.
2.- Esperar el tiempo de conversión para el signo (≈5 μseg.).
3.- Lectura del signo. Este viene dado por el bit 0 del puerto A. Si este bit es 0 entonces el dato es positivo, y si es 1 el dato es negativo.
4.- Enviar un byte de control al puerto C para la lectura del valor absoluto del dato. Este byte será igual al del paso 1 salvo en el bit 7 que será 1.
5.- Esperar el tiempo de conversión para el dato (≈40 μseg.).
6.- Leer el valor absoluto del dato por el puerto A..

C.2.5 ORDENADOR

Una vez resuelta la adquisición de los datos seguiría la fase de procesamiento de los mismos y generación de las entradas de la planta. En el ordenador debe estar ejecutándose un programa que gestione la adquisición y salida de datos y que tenga implementado el controlador. La estrategia de control propuesta se ha intentado implementar en diferentes procesadores de la familia 80x86, desde el PC-XT al PC-486.

Apéndice D

RESUMEN DE ALGUNAS DE LAS FUNCIONES DE MATLAB UTILIZADAS

En esta apéndice se describen algunas de las funciones más importantes de las toolboxes de Control Robusto [Chi92] y de redes neuronales [Dem93b] del paquete Matlab. Estas funciones han sido utilizadas en la implementación de algunos de los programas desarrollados.

D.1 FUNCIONES DE DISEÑO DE LA TOOLBOX DE CONTROL ROBUSTO DE MATLAB

En este apéndice se describen las funciones más importantes de diseño robusto de la Toolbox de Control Robusto del paquete MATLAB [Chi92].

D.1.1 mksys

Crea una variable de MATLAB simple que contiene todas las matrices que describen a un sistema, sus dimensiones y sus nombres estándar (dependiendo del tipo de sistema). *mksys* implementa la estructura de datos para el diseño robusto usada para simplificar el trabajo del usuario con las variables que aparecen en el diseño de los controladores.

Sintaxis:

S=mksys(A,B,C,D) ó
S=mksys(V1,V2,...,VN, TY)

Descripción:

mksys compacta las matrices que describen un sistema de tipo TY en una estructura, S, usando el estándar para los nombres de las variables determinados por los valores de la cadena TY según la siguiente tabla:

TY	V1,V2,...,VN	Descripción
'ss'	'a,b,c,d,ty'	Espacio de estados estándar (defecto)
'des'	'a,b,c,d,e,ty'	Sistema descriptor
'tss'	'a,b1,b2,c1,c2,d11,d12,d21,d22,ty'	Espacio de estados de 2 puertos
'tdes'	'a,b1,b2,c1,c2,d11,d12,d21,d22,e,ty	Descriptor de 2 puertos
'gss'	'sm,dimx,dimu,dimy,ty'	Espacio de estados general
'gdes'	'e,sm,dimx,dimu,dimy,ty'	Descriptor general
gpsm'	'psm,deg,dimx,dimu,dimy,ty'	Sistema matricial polinomial
'tf'	'num,den,ty'	Función de transferencia
'tfm'	'num,den,m,n,ty'	Función de transferencia matricial
'imp'	'y,ts,nu,ny'	Respuesta impulso

augss/augtf

Estas funciones generan un modelo de estado para la planta aumentada *P*(*s*), en la forma adecuada para el diseño de sensibilidad mixta H_2 o H_∞.

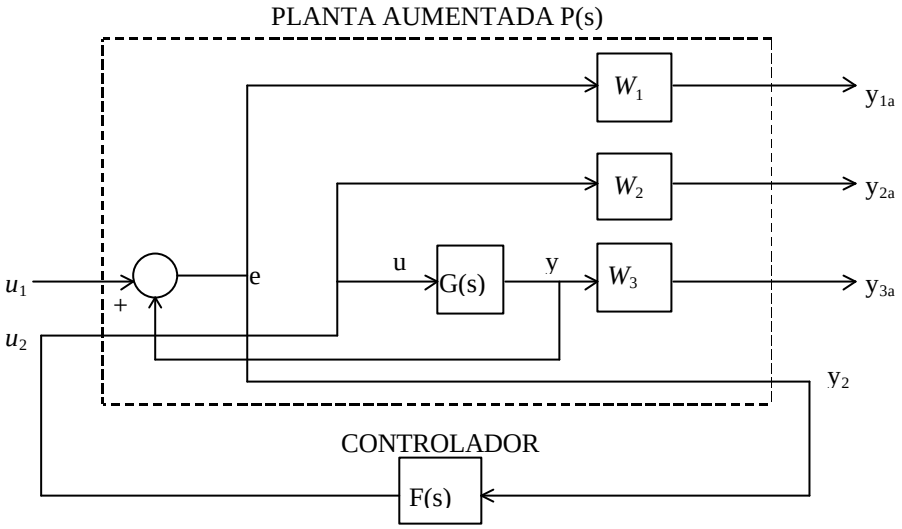


Figura C.1. Planta aumentada.

Sintaxis:

[A,B1,B2,C1,C2,D11,D12,D21,D22]= augss(AG, BG, …, AW1, BW1, …, AW2, … BW2, …, AW3, BW3, …);

[A,B1,B2,C1,C2,D11,D12,D21,D22]= augss(AG, BG, …, AW1, BW1, …, AW2, … BW2, …, AW3, BW3, …,W3POLY);

[A,B1,B2,C1,C2,D11,D12,D21,D22]=augtf(AG,BG,CD,DG,W1,W2,W3)

[TSS_]=augss(SS_G, SS_W1, SS_W2, SS_W3, W3POLY)

[TSS_]=augtf(SS_G, W1, W2, W3)

[TSS_]=augss(SS_G, SS_W1, SS_W2, SS_W3)

Descripción:

augss calcula un modelo de espacio de estados para la planta aumentada P(s) con funciones de peso W_1(s), W_2(s), W_3(s) penalizando la señal de error, la señal de control y la salida respectivamente (ver figura c.1). La planta aumentada producida por augsss es

$$P(s) = \left[\begin{array}{cc} W_1 & W_1 G \\ 0 & W_2 \\ 0 & W_3 G \\ \hline I & -G \end{array}\right]$$

que en representación de estado es

$$P(s) = \left[\begin{array}{ccc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

De esta forma la función de transferencia en lazo cerrado es

$$T_{y1u1} \equiv \begin{bmatrix} W_1 S \\ W_2 FS \\ W_3 T \end{bmatrix}$$

donde S y T son las funciones de sensibilidad y sensibilidad complementaria definidas mediante las ecuaciones (3.13) y (3.14), respectivamente.

La funciones de transferencia G(s), W_1(s) y W_3(s)G(s) deben ser propias. W_3(s), por el contrario, puede ser impropia. Los datos de entrada a la función *augss* son las matrices de estado de G(s), W_1(s) y W_2(s):

$$SS\_G \equiv \left[\begin{array}{c|c} A_g & B_g \\ \hline C_g & D_g \end{array}\right] = mksys(ag, bg, cg, dg);$$

$$SS\_W_1 \equiv \left[\begin{array}{c|c} A_{W1} & B_{W1} \\ \hline C_{W1} & D_{W1} \end{array}\right] = mksys(aw1, bw1, cw1, dw1);$$

$$SS\_W_2 \equiv \left[\begin{array}{c|c} A_{W2} & B_{W2} \\ \hline C_{W2} & D_{W2} \end{array}\right] = mksys(aw2, bw2, cw2, dw2);$$

y la matriz de transferencia W_3(s) (que puede ser impropia):

$$W_3(s) = C_{W3}(Is - AW_3)^{-1}BW_3 + DW_3 + P_n s^n + ... + P_1 s + P_0$$

que será entrada a *augss* de la siguiente forma:

$$SS\_W_3 \equiv \begin{bmatrix} A_{W3} & B_{W3} \\ C_{W3} & D_{W3} \end{bmatrix} = mksys(aw3, bw3, cw3, dw3);$$

W3POLY = [P_n, …, P_1, P_0].

Si alguna de las funciones de peso W_i está ausente, se introduce SS_Wi=[ ].

La diferencia entre *augss* y *augtf* es que las funciones de peso con *augtf* se introducen como matrices de transferencia diagonales. El numerador y el denominador de cada elemento diagonal se introducen como filas de la matriz de peso. Por ejemplo, las matrices de peso

$$W_1(s) = \begin{bmatrix} \dfrac{s+100}{100s+1} & 0 \\ 0 & \dfrac{s+100}{100s+1} \end{bmatrix};\ W_2 = [\ ];\ W_3(s) = \begin{bmatrix} \dfrac{s^2}{1000} & 0 \\ 0 & \dfrac{s^2(ts+1)}{1000} \end{bmatrix}$$

se introducirían como

$$W_1(s) = \begin{bmatrix} 1 & 100 \\ 100 & 1 \\ 1 & 100 \\ 100 & 1 \end{bmatrix};\ W_2 = [\ ];\ W_3(s) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1000 \\ t & 1 & 0 & 0 \\ 0 & 0 & 0 & 1000 \end{bmatrix}.$$

D.1.2 sigma

Calcula los valores singulares de un sistema.

Sintaxis:

[sv,w]=sigma(A, B, C, D);
[sv,w]=sigma(A, B, C, D, 'inv');
[sv,w]=sigma(A, B, C, D, w);
[sv,w]=sigma(A, B, C, D,w,'inv');
[sv,w]=sigma(SS_,…);

Descripción:
jjjj

sigma calcula los valores singulares de la matriz compleja $C(jwI-A)^{-1}B + D$ en función de la frecuencia. Para sistemas cuadrados, *sigma*(*A*, *B*, *C*, *D*, 'inv') calcula los valores singulares de la matriz compleja inversa

$$G^{-1}(jw) = D^{-1}C[(jwI-(A-BD^{-1}C)]^{-1}BD^{-1} + D^{-1}$$

D.1.3 hinf

Síntesis de control H_∞

Sintaxis:

[ACP, BCP, CCP, DCP, ACL, BCL, CCL, DCL, HINFO, AK,…, DK22] = … hinf(A,B1,B2,C1,C2,D11, … D12, D21,D22)

[ACP, …, ACL, …, HINFO, AK,…, DK22] = hinf(A,…,D22,AU, …, DU)

[SS_CP, SS_CL, HINFO, TSS_K]= hinf(TSS_P)

[SS_CP, SS_CL, HINFO, TSS_K]= hinf(TSS_P,SS_U)

Descripción:

hinf resuelve el problema de pequeña ganancia del control robusto, que consiste en encontrar un controlador estable F(s) para el sistema

$$P(s) = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

de tal forma que la función de transferencia en lazo cerrado satisfaga la desigualdad

$$\left\| T_{y_1 u_1} \right\|_\infty \leq 1$$

Las matrices de estado para una solución particular de F(s) y la correspondiente T_{y1u1}(s) se retornan como

$F(s)$ = (ACP,BCP,CCP,DCP) ó SS_CP

$T(s)$ = (ACL,BCL,CCL,DCL) ó SS_CL.

También se retorna una parametrización de todos los controladores K(s) que verifican $\left\| T_{y_1 u_1} \right\|_\infty \leq 1$, en las matrices AK, …, DK22.

D.2 FUNCIONES DE LA TOOLBOX DE REDES NEURONALES DE MATLAB

D.2.1 purelin

Cálculo de la salida de una capa de una red neuronal considerando función de activación lineal para las neuronas.

Sintaxis:

purelin(*N*)
purelin(*N*,*B*)

Descripción:

*a*1 = *purelin*(*N*) devuelve en a1 el vector de salida de la capa cuya entrada es el vector *N*, considerando que las neuronas tienen una función de activación lineal.

*a*1 = *purelin*(*N*, *B*) en este caso a la salida se le añade el vector de bias *B*.

D.2.2 logsig

Cálculo de la salida de una capa de una red neuronal considerando función de activación sigmoide para las neuronas.

Sintaxis:

logsig(*N*)
logsig(*N*,*B*)

Descripción:

*a*1 = *logsig*(*N*) devuelve en a1 el vector de salida de la capa cuya entrada es el vector *N*, considerando que las neuronas tienen una función de activación sigmoide.

*a*1 = *logsig*(*N*, *B*) en este caso a la salida se le añade el vector de bias B.

D.2.3 deltalin

Esta función calcula las derivadas del error para una capa de neuronas lineales.

Sintaxis:

deltalin(*A*, *E*)
deltalin(*A*, *D*2, *W*2)

Descripción:

deltalin(*A*, *E*) devuelve una matriz con las derivadas del error para la capa de salida de la red con función de activación lineal, donde *A* es el vector de salida de la capa y *E* es el error.

deltalin(*A*, *D*2, *W*2) devuelve una matriz con las derivadas de los errores para una capa oculta de la red con función de activación lineal. *A* es la salida de la capa, *D*2 es la matriz de derivadas del error de la siguiente capa y *W*2 es la matriz de pesos de la siguiente capa.

D.2.4 deltalog

Esta función calcula las derivadas del error para una capa de neuronas con función de activación sigmoidal.

Sintaxis:

deltalog(*A*, *E*)
deltalog(*A*, *D*2, *W*2)

Descripción:

deltalog(*A*, *E*) devuelve una matriz con las derivadas del error para la capa de salida de la red con función de activación sigmoidal, donde *A* es el vector de salida de la capa y *E* es el error.

deltalog(*A*, *D*2, *W*2) devuelve una matriz con las derivadas de los errores para una capa oculta de la red con función de activación sigmoidal. *A* es la salida de la capa, *D*2 es la matriz de derivadas del error de la siguiente capa y *W*2 es la matriz de pesos de la siguiente capa.

D.2.5 learnbpm

Implementa el algoritmo backpropagation en una red neuronal, calculando los incrementos que se deben sumar a los pesos de la red.

Sintaxis:

[*dw*1]=*learnbpm*(*p*,*d*1,*lr*,*mc*,*dW*);

[*dw*1,*db*1]=*learnbpm*(*p*,*d*1,*lr*,*mc*,*dW*,*dB*);

Descripción:

[*dw*1]=*learnbpm*(*p*,*d*1,*lr*,*mc*,*dW*) devuelve una matriz con los incrementos a aplicar a los pesos de la red según el algoritmo backpropagation. P es la matriz de vectores de entrada, D es la matriz de vectores con las derivadas del error, lr es la velocidad de aprendizaje, mc es la constante de momento y *dW* es la matriz previa de incrementos. La matriz de incrementos se calcula como

$$\Delta W(i + 1, j + 1) = mc\Delta W(i, j) + (1\text{-}mc)lrD(i)P(j)$$

[*dw*1,*db*1]=*learnbpm*(*p*,*d*1,*lr*,*mc*,*dW*,*dB*) devuelve tanto una matriz de incremento para los pesos como para los *bias*.

Apéndice E

CÓDIGOS DE LOS PROGRAMAS DE SIMULACIÓN Y CONTROL

En este apéndice se lista el código fuente de los principales programas de simulación y de los controladores implementados.

E.1. SIMULACIÓN DEL CONTROLADOR GPC

E.1.1 gpcs7.m

```
% GPCS7.M script que aplica el controlador GPC a la
% planta que indicada a traves del fichero planta?.mat.
% Los calculos offline los realiza a traves de la funcion offline.m
% y los calculos online los realiza a traves de la funcion online.m
%
%
% FUNCIONES NECESARIAS: OFFLINE.M,  ONLINE.M
%
clear all;
global k nb na w DU1 u DA DB Y G2 DUant F N NU B lambda y T ypred1 A B;

file2=input(' - Fichero con los datos de la planta ?: ','s');
file=['load ',file2];
eval(file);
```

```
fig=input(' - Numero de figura?: ');
na=length(A)-1;     % Grado del polinomio A
nb=length(B)-1;     % Grado del polinomio B
N=input(' - Horizonte de salida: ');              % Horizonte salida.
frec=input(' - Frecuencia de la consigna: ');
NU=input(' NU? ');
consig=1;
nfin=input(' - Numero de etapas: ');
w=consig*ones(nfin+N,1);    % Secuencia de referencia.
t=0:T:(nfin+N-1)*T;
lambda=0.001;
umax=+10;
umin=-10;
DA=delta(A);  % Valor de DA. Emplea la funci¢n "delta.m" creada por m¡.
DB=delta(B);  % Se emplea al final de cada ciclo para obtener la salida.
DU=zeros(NU,1); % DU  contiene los incrementos en los comandos
DUant=zeros(nb+1,1); %este vector contiene DU(t), DU(t-1), ..., DU(t-nb)
u=zeros(nb+1,1);% tiene los comandos anteriores u(t-1),...,u(t-(nb+1))
y=zeros(na+1,1);  % Vector  contiene las salidas anteriores del sistema:
%y=ones(na+1,1);
coma(1)=0.1;
yy(1)= 0;
yy(2)=0;
    offline(N);

for ii=2:nfin,
   [sal,com]=online(ii,N);
   coma(ii)=com;
   yy(ii+1)=sal;
   valorn(ii)=N;

end % for

figure(fig);
hold off;
subplot(211);
k=0:T:(length(yy)-2)*T;
[kk,YY]=stairs(k,yy(2:length(yy)));
plot(k,yy(2:length(yy)),'c15');
ejeymin=min(min(YY),min(w));
ejeymax=max(max(YY),max(w));
ejeymin=ejeymin-0.1*sign(ejeymin)*min(ejeymin);
ejeymax=ejeymax+0.1*sign(ejeymax)*max(ejeymax);
axis([0,t(nfin+1),ejeymin,ejeymax]);
texto1=['GPC: Lambda=',num2str(lambda),'; T=',num2str(T)];
texto2=['; N=',num2str(N),'; NU=',num2str(NU)];
texto1=[texto1, texto2];
title(texto1);
xlabel('Time (sec.)');
ylabel('Output');
grid
hold on
plot(k,w(1:length(k)),'c15--');
hold off
subplot(212);
k=0:T:(length(coma)-1)*T;
plot(k,coma);
ylabel('Applied command');
xlabel('Time (sec.)');
texto1=[' Data file ---> ',file2];
title(texto1);
grid
```

```
subplot(111);
```

E.1.2 online.m

```
function [sal, com]=online(k,N);
%
% Función que contiene todas las operaciones que
% se realizan on-line en el gpc.
% El parametro k  es la etapa de simulación.
% Es utilizada por el programa gpcnn5.m y gpcs7.m
%
%
global nb na w DU1 u A B G2 DUant F y ypred1;

%                       CALCULO DEL VECTOR f
%
    f=zeros(N,1);
    if(nb >= 1),
           for j=1:N,
               f(j)=G2(j,:)*DUant(1:nb,1)+F(j,:)*y;
           end
       else
           for j=1:N,
               f(j)=F(j,:)*y;
           end
       end
%                       CALCULO DE LA LEY DE CONTROL
%
     ww=w(k:k+N-1);
    DU=DU1*(ww-f);
   %keyboard;
   % pause;
%       ACTUALIZAMOS LOS INCREMENTOS ANTERIORES DU(t),...,DU(t-nb)
    for i=nb+1:-1:2,
        DUant(i)=DUant(i-1);
    end
    DUant(1)=DU(1);

%                       ACTUALIZAMOS EL VECTOR U
    for i=nb+2:-1:2,
        u(i)=u(i-1);
    end
    u(1)=DU(1)+u(1);

%                       CALCULO DEL VECTOR DE SALIDA.
y2=y;
u2=u;
for j=1:N,
    y1=0;                       % Valor de y(t+1).
    for i=2:na+1,
        y1=-A(i)*y2(i-1)+y1;
    end
    for i=1:nb+1,
        y1=B(i)*u2(i)+y1;
    end
%         actuazalizacion de y2 (predicha).

    for i=na+1:-1:2,
```

```
        y2(i)=y2(i-1);
    end
%         actualizacion de u2  (predicha),
    for i=nb+2:-1:2,
        u2(i)=u2(i-1);
    end
  y2(1)=y1;
  ypred1(j)=y1;
end     %for j
    for i=na+1:-1:2,    % actualizacion de y (real)
        y(i)=y(i-1);
    end
    y(1)=ypred1(1);


    if y1==nan error(' SE HA OBTENIDO NAN PARA EL VALOR DE Y'); end
    U(k)=u(1);
  sal=y(1);
  com=u(1);
```

E.1.3 offline.m

```
function [MG2,MDU1]=offline(N);
%
% función que tiene implementada todos los cálculos
% off-line del gpc.
% Devuelve la matrix G2 y la matriz DU1.
%
global nb na w DU1 u DA DB G2 DUant F NU B lambda;

F=-DA;        % F1=q(1-DA)
F(1)=[];      %
for j=2:N,
    for i=1:na,
        F(j,i)=F(j-1,i+1)-DA(i+1)*F(j-1,1);
   end
        F(j,na+1)=-DA(na+2)*F(j-1,1);
end
%
% CALCULO DE Ej
%
%
E=1;
for j=2:N,
    E(j,1)=1;
    for i=2:j,
        E(j,i)=F(i-1,1);
    end
end

%         CALCULO DE G (MATRIZ DE POLINOMIOS Ei*B)
%
G=zeros(N,nb+N);
G(1,1:nb+1)=B;
B1=[0,B];
for j=2:N,
     GG=G(j-1,1:nb+j)+E(j,j)*B1; % La nueva fila de G sera igual a la anterior
                                 % mas el producto del nuevo elemento que aporta
                                 % el nuevo E por B.
```

```
     G(j,1:nb+j)=GG(1:nb+j);
     B1=[0,B1];
end

% CALCULO DE G1. MATRIZ QUE DA CUENTA DE LOS COMANDOS A APLICAR EN EL FUTURO
% (Triangular inferior).
G1=zeros(N,NU);
for j=1:NU,
    for i=1:j,
        G1(j,i)=G(j,(j+1)-i);
    end
end
for j=NU+1:N
    for i=1:NU,   % A PARTIR DE LA FILA NU LA MATRIZ SOLO TIENE NU COLUMNAS.
        G1(j,i)=G(j,(j+1)-i);
    end
end
%
%       CALCULO DE UNA MATRIZ ADICIONAL G2 QUE  EMPLEO  PARA  CALCULAR
%       LAS f. LA FILA J DE ESTA MATRIZ SERIA Gn-SUM(q^-i*gi,N), QUE ES
%       LO QUE HAY QUE MULTIPLICAR POR DU PARA EL CALCULO DE f. ESTA PARTE DE
%       LA MATRIZ G DA CUENTA DE LOS COMANDOS CONOCIDOS.
if nb >= 1,
   for j=1:N,
       G2(j,:)=G(j,j+1:j+nb);
   end
else
   G2=zeros(N,1);
end
DU1=inv(G1'*G1+lambda*eye(NU))*G1';
MG2=G2;
MDU1=DU1;
```

E.1.4 delta.m

```
%
% Función que halla el producto D*A.
% Siendo A un polinomio en potencias de q^-1
% y D el operador incremento definido por (1-q^-1).
% La llamada a esta funci¢n se hace indicando el
% polinomio A al que se le aplica el operador
% incremento.
%
%Ejemplo:
%            Q=delta(A)
%
function da=delta(A);
dimenA=length(A);
B=[0 A];
A=[A 0];
da=A-B;
```

E.1.5 param.m

```
% Fichero para calcular los polinomios A y B
% para el controlador GPC a partir del numerador
% y controlador de la planta continua.
```

```
% El resultado se guarda en el fichero que se
% indique.
%
clear;
num=input(' - Numerador de la planta cont¡nua: ');
den=input(' - Denominador de la planta cont¡nua: ');
T=input(' - Periodo de muestreo: ');
file=['save ',input(' - Fichero mat donde guardar los datos: ','s')];
[numd,dend]=c2dm(num,den,T,'zoh');
A=dend;
B=numd;
% Estos polinomios corresponden a la ecuacion
%            A(q-1)y(t)=B(q-1)u(t)
% nuestro modelo carima asume un retardo de una etapa. Para tener en cuenta
% este retardo se debe eliminar el primer elemento del vector B.
%  As¡ que lo hacemos ahora:
B(1)=[];
eval(file);
```

E.2 CONTROL DE UN MOTOR DC

E.2.1 gpcc1.c

```
/* Este proegrama implementa el controlador GPC pseudo-multivariable
   sobre un motor de corriente continua
   La salida de resultados es a traves de un fichero de resultados con
   formato matlab (*.mat) */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#define B BLUE
#define W YELLOW
#define UP     0x48
#define DOWN   0x50
#define RIGHT  0x4d
#define LEFT   0x4b
#define RETURN 0x0d
#define ESC    0x1b
#define posini  3
#define toppos  0
#define botpos  6
#define toppos2 0
#define botpos2 4
#define separ   3
extern _toma6(void);
extern int datpos[];
extern int datvel[];
extern int com[];
extern int muestras;
int ypos1=0,xpos1=0,posini1=0,posact=0,comando,cuenta,nummenu=0,contador,\
epsi2=0.1,epsi3=0.2,reji[4160];
char *texto1[]={" 1.- FICHERO DE COMANDOS:       ",
             " 2.- FICHERO DE SALIDA:         ",
             " 3.- PERIODO:                   ",
```

```
            " 4.- DISTANCIA ACCION P. :      ",
            " 5.- DISTANCIA MINIMA AL ORIGEN:",
            " 6.- CONSTANTE DE POSICION (KP):",
            " 7.- CONTINUAR.                 "};
char *fichcom="u1.bin",*fichsal="salida.mat",buffer[15],buffer1[15];
float param[4]={0.01,1.00,0.5,10},*datpos1,*datvel1,*com1;
void inverso(void);
void directo(void);
void mover(void);
void cambcom(void);
void cambper(void);
void cambeps2(void);
void cambeps3(void);
void cambkp(void);
void corrige1(int *pj,int *pdec);
/*---------------------------------------------------------------------*/
typedef struct{           /* Estructura que representa la cabecera */
       long type;        /* del fichero .mat */
       long mrows;
       long ncols;
       long imagf;
       long namleng;
}Fmatrix;


/*-------------------------------------------------------------MAIN --------*/
void main(void)
{
float kv,tiempo,cons1,cons2,periodo,*pr;
char *var1="posi",*var2="velo",*var3="coma";
int mn,i,dim=2000;
FILE *fp;
Fmatrix x;
void menu1(void);
kv=2.5;            //  Ganancia de la funci¢n de transferencia de velocidad
                   // Valores de los parametros de la cabecera para el fichero
                   // .mat
x.type=10;
x.ncols=1;     /* n§ de columnas de cada variable (posi,velo ycoma */
x.imagf=0;
x.namleng=5;
posact=0;
if ((datpos1=calloc(dim,sizeof(float)))==NULL){
       printf("Memoria insuficiente para crear buffer de resultados !!\n");
       exit(1);   /* Terminar */;
       } /*datpos1 */
if ((datvel1=calloc(dim,sizeof(float)))==NULL){
       printf("Memoria insuficiente para crear buffer de resultados !!\n");
       exit(1);   /* Terminar */;
       } /* datvel1 */
if ((com1=calloc(dim,sizeof(float)))==NULL){
       printf("Memoria insuficiente para crear buffer de resultados !!\n");
       exit(1);   /* Terminar */;
       } /* com1 */
menu1();
while(1) {          /* Bucle del primer menu */
/*     menu1(); */
       nummenu=0;  /* primer menu */
       mover();
       if(posact==0){
              gotoxy(38,ypos1);
              directo();
```

```
        cprintf("           ");
        gotoxy(38,ypos1);
        buffer[0]=13;             /* n§ m ximo de caracteres */
        fflush(stdin);
        fichcom=cgets(buffer);
        if ((*fichcom)=='\0'){      /* Si se pulsa return */
            gotoxy(38,ypos1);        /* de nuevo,se guarda */
            fichcom="comandos.mat"; /* comandos.mat en fichero */
            cprintf("%s",fichcom);
            } /* if */
        gotoxy(xpos1,ypos1);
        } /* posact==0  */
    if(posact==1){
        gotoxy(38,ypos1);
        directo();
        cprintf("           ");
        gotoxy(38,ypos1);
        buffer1[0]=13;            /* n§ m ximo de caracteres */
        fflush(stdin);
        fichsal=cgets(buffer1);
        if ((*fichsal)=='\0'){         /* Si se pulsa return */
              gotoxy(38,ypos1);      /* de nuevo,se guarda */
              fichsal="salida.mat";  /* salida.mat en fichero */
              cprintf("%s",fichsal);
          } /* if */
        gotoxy(xpos1,ypos1);
        } /* posact==1 */
    if(posact==2){
        gotoxy(38,ypos1);
        directo();
        cprintf("       ");
        gotoxy(38,ypos1);
        cambper();
        }  /* posact==2 */
    if(posact==3){
        gotoxy(38,ypos1);
        directo();
        cprintf("       ");
        gotoxy(38,ypos1);
        cambeps2();
        }  /* posact==3 */
    if(posact==4){
        gotoxy(38,ypos1);
        directo();
        cprintf("       ");
        gotoxy(38,ypos1);
        cambeps3();
        }  /* posact==4 */
    if(posact==5){
        gotoxy(38,ypos1);
        directo();
        cprintf("       ");
        gotoxy(38,ypos1);
        cambkp();
        }
    if(posact==6){
        cuenta=1193180*param[0];
        epsi2=param[1]*255/10;
        epsi3=param[2]*255/10;
        window(1,1,80,25);
        directo();
        clrscr();
```

```
              gotoxy(5,12);
              cprintf("---> En el momento que pulse una tecla\
 comenzar  la accion de control... ");
              getch();
              toma6();     // RUTINA ENSAMBLADOR QUE IMPLEMENTA EL CONTROLADOR
              x.mrows=muestras/2;
              for (i=0 ; i<x.mrows ; i++){
                     datpos1[i]=datpos[i]*10.*kv/(param[3]*255.);
                     datvel1[i]=datvel[i]*10./255.;
                     com1[i]=com[i]*10./255.;
                     } /* for i=0 */
              if((fp=fopen(fichsal,"wb"))==NULL){
                     system("cls");
                     printf("Error al abrir el fichero de resultados.");
                     exit(0);
                     }
              pr=&datpos1[0];
              fwrite(&x,sizeof(Fmatrix),1,fp);
              fwrite(var1,sizeof(char),x.namleng,fp);
              mn=x.mrows*x.ncols;
              fwrite(pr,sizeof(float),mn,fp);

              pr=&datvel1[0];
              fwrite(&x,sizeof(Fmatrix),1,fp);
              fwrite(var2,sizeof(char),x.namleng,fp);
              fwrite(pr,sizeof(float),mn,fp);

              pr=&com1[0];
              fwrite(&x,sizeof(Fmatrix),1,fp);
              fwrite(var3,sizeof(char),x.namleng,fp);
              fwrite(pr,sizeof(float),mn,fp);

              fclose(fp);
              printf("\r\n\n -- Pulse una tecla --");
              getch();
              clrscr();
              textcolor(W);
              textbackground(B);
              clrscr();
              menu1();
              continue;
              }  /* posact==5 */
       }  /* while del primer menu */
}  /*main*/


/* ---------------------------------------------------------------------*/
void corrige1(int *pj,int *pdec)
{
        gotoxy(38,ypos1);
        cprintf("     ");
        gotoxy(38,ypos1);
        printf("\a");
        *pj=0;
        *pdec=0;
}


/* ------------------------------------------------------------------- */
void menu1(void)
{
int j=1,i;
```

```
window(1,1,80,25);
directo();
clrscr();
window(17,2,67,25);
directo();
cputs("ÉÍÍÍÍÍÍÍÍÍÍÍÍ  CONTROL GPC ÍÍÍÍÍÍÍÍÍÍÍÍ ");
inverso();
cputs("PDC8");
directo();
cputs(" ÍÍÍ»");
for (j=2;j<23;j++){
       gotoxy(1,j);
       cputs("º");
       gotoxy(50,j);
       cputs("º");
       }
gotoxy(1,23);
cputs("ÈÍÍÍ Salir/Esc ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼");
xpos1=4; /* comienzo de la primera opcion*/
ypos1=posini;
inverso();
gotoxy(xpos1,ypos1);
cprintf("%s",texto1[0]);
directo();
cprintf("  %s",fichcom);
ypos1+=separ;
gotoxy(xpos1,ypos1);
cprintf("%s  %s",texto1[1],fichsal);
ypos1+=separ;
gotoxy(xpos1,ypos1);
cprintf("%s   %5.3f",texto1[2],param[0]);
ypos1+=separ;
gotoxy(xpos1,ypos1);
cprintf("%s   %4.2f",texto1[3],param[1]);
ypos1+=separ;
gotoxy(xpos1,ypos1);
cprintf("%s   %4.2f",texto1[4],param[2]);
ypos1+=separ;
gotoxy(xpos1,ypos1);
cprintf("%s  %5.2f",texto1[5],param[3]);
ypos1+=separ;
gotoxy(xpos1,ypos1);
cprintf(" %s",texto1[6]);
ypos1=ypos1-(botpos)*separ;
gotoxy(xpos1,ypos1);
posact=0;
return;
}


/* ---------------------------------------------------------------------- */
void inverso(void)
{
       textcolor(B);
       textbackground(W);
}


/* ---------------------------------------------------------------------- */
void directo(void)
{
       textcolor(W);
```

```
       textbackground(B);
}


/* ------------------------------------------------------------------ */
void mover(void)
{
       unsigned eleccion;
       while(1){
              eleccion=getch();
              if (eleccion==UP){
                     gotoxy(xpos1,ypos1);
                     directo();
                     cprintf("%s",texto1[posact]);
                     if (posact!=toppos){
                            posact=posact-1;
                            }
                     else if (posact==toppos){
                            posact=botpos;}
                     ypos1=posini+posact*separ;
                     gotoxy(xpos1,ypos1);
                     inverso();
                     cprintf("%s",texto1[posact]);
                     gotoxy(xpos1,ypos1);
                     }
              if (eleccion==DOWN){
                     gotoxy(xpos1,ypos1);
                     directo();

                     cprintf("%s",texto1[posact]);
                     if (posact!=botpos){
                            posact=posact+1;
                                   }
                     else if (posact==botpos){
                            posact=toppos;}
                     ypos1=posini+posact*separ;
                     gotoxy(xpos1,ypos1);
                     inverso();
                     cprintf("%s",texto1[posact]);
                     gotoxy(xpos1,ypos1);
                     }
              if (eleccion==RETURN){
                     return;
                     } /* return */
              if (eleccion==ESC){
                     system("cls");
                     free(datpos1);
                     free(datvel1);
                     exit(1);
                     } /* esc */
       } /*while*/
} /* mover */


/* ---------------------------------------------------- PERIODO -------- */

void cambper(void)
{
int j=0,valid,ia,ia1=0,dec=0;
char a,per[7]="00000";
       while(1){
              a=getch();
```

```
            if(a==RETURN){
                  param[0]=atof(per);
                  if (param[0]==0) param[0]=0.01;
                  gotoxy(38,ypos1);
                  cprintf(" %5.3f\n\n",param[0]);
                  gotoxy(xpos1,ypos1);
                  dec=0;
                  per[0]='0';
                  per[1]='0';
                  per[2]='0';
                  per[3]='0';
                  per[4]='0';
                  break;
                  }   /* return */
            valid=isdigit(a);
            if(((valid==0) && (a!='.')) || (j==5)){
                  corrige1(&j,&dec);
                  ia1=0;
                  continue;
                  } /* valid */
            ia=a/16;      /* Convierte "a" en entero */
            ia=a-ia*16;
            if((j==0) && ((ia>0) || (a=='.'))){
                  printf("\a"); /*Evita que el primer caracter*/
                  continue;   /*sea mayor que cero o un punto*/
                  } /* j==0 */
            if ((j>1) && (a=='.')){
                  corrige1(&j,&dec);
                  ia1=0;
                  continue;
                  } /* j>1 */
            if (a=='.') dec=1;
            if((ia>0) && (dec==0)){
                  corrige1(&j,&dec);
                  ia1=0;
                  continue;
                  }/* ia>0*/
    if( ((ia>0)&&(j==2))||((ia>2)&&(j==3))||((ia>7)&&(j==4)&&(ia1==2)) ){
                        corrige1(&j,&dec); /* Hace que el periodo */
                        continue;          /* máximo sea 0.027 */
                         }
            putch(a);
            per[j]=a;
            j=j+1;
            ia1=ia;
            }  /* while */
} /* cambper */


/* ------------------------------------------ CAMBIAR CONSTANTE KP ---- */

void cambkp(void)
{
      int j=0,valid,ia,dec=0;
      char a,cons[5]="00000";
      gotoxy(41,ypos1);
      textbackground(B);
      textcolor(W);
      cprintf("       ");
      gotoxy(38,ypos1);
      while(1){
          a=getch();
```

```
           if(a==RETURN){
                    if (dec==0)  cons[j]='.';
                    param[3]=atof(cons);
                    if (param[3]==0) param[3]=10;
                    gotoxy(38,ypos1);
                    cprintf("%5.2f",param[3]);
                    gotoxy(xpos1,ypos1);
                    dec=0;
                    cons[0]='0';
                    cons[1]='0';
                    cons[2]='0';
                    cons[3]='0';
                    cons[4]='0';
                    break;
                } /* RETURN */
                valid=isdigit(a);
                if(((valid==0) && (a!='.')) || (j==5)){
                       corrige1(&j,&dec);
                       continue;
                       }    /* valid==0 */
             if((valid==0) && (dec==1)){
                       corrige1(&j,&dec);
                       continue;
                       }     /* valid ==0*/
             ia=a/16;
             ia=a-ia*16;
             if((j==0) && (a=='.')){
                       printf("\a");
                       continue;
                       } /* j==0*/
             if ((j==2) && (dec==0) && (a!='.')){
                       corrige1(&j,&dec);
                       continue;
                       } /* j=2 */
/*           if ((j>2) && (a=='.')){
                       corrige1(&j,&dec);
                       continue;
                       }    */
            if (a=='.') dec=1;
            putch(a);
            cons[j]=a;
            j=j+1;
            }  /* while */
} /* cambkp */
```

E.2.2 gpcae5.asm

```
;------------------------ CONTROLADOR GPC ---------------------------
; Rutina ensamblador que implementa el controlador GPC
; pseudo-multivariable. La estrategia se basa en multiplexar
; las acciones de control sobre la variable de posición y
; velocidad en el tiempo.
; El codigo esta escrito empleando las directivas simplificadas
; de ensamblador.
; Periodo de muestreo T=0.01
title gpcae5
    cr equ 0dh
    lf equ 0ah
.MODEL small
```

```
.STACK 100h
.DATA
       texto1  db 'ERROR AL ABRIR EL FICHERO DE COMANDOS.$'
       texto2  db 'ERROR AL LEER FICHERO DE COMANDOS.$'
       texto3  db 'Posicion negativa en la etapa 1 $'  ;QUITAR SI TODO EL PLANO.
       texto4  db 'ERROR AL MEDIR TIEMPO DE EJECUCION$'
       texto5  db cr,lf,'Tiempo hasta antes de la acci¢n proporcional: $'
       texto6  db 'Tiempo total en alcanzar consigna: $'
       texto7  db ' decimas de ms. ',cr,lf,cr,'$'
       texto8  db cr,lf,'Etapa3: buffer lleno,suspendida la accion de
control.',cr,lf,'$'
       texto9  db 'Etapa 1: buffer lleno,suspendida la accion de
control',cr,lf,'$'
       texto10 db 'Etapa 2: buffer lleno,suspendida la accion de
control',cr,lf,'$'
       texto11 db 'ERROR: LA POSICION INICIAL NO ES CORRECTA. ',cr,lf,'$'
       handle  dw 0    ; HANDLE DEL FICHERO DE COMANDOS.
       num     db 12 dup (?)
       i       dw (?)  ; Variable donde se guarda la velocidad.
       j       dw (?)  ; Variable donde se guarda la posici¢n.
       i0      db 0    ; Si in=0 i es positiva, si in=1 i es negativa.
       j0      db 0    ; Si jn=0 j es positiva, si jn=1 j es negativa.
       numpunt dw 64   ; N£mero de puntos en cada semieje.
       umax   dd 10.0
       umin   dd -10.0
    signo   db (?)  ;Signo del comando que se envia.
       consta  db 2             ;constante para el control proporcional
       cont1   dd 0             ;variables utilizadas para la medida de
       cont2   dd 0             ;tiempo
       cont3   dd 0
       npini   dw 2
       npmed  dw 1
       npfin  dw 1
       nvini   dw 1
       nvmed   dw 50
       nvfin   dw 50
       epmed   dw 125
       evfin  dw 100
       indic1  db 0
       indic2  db 0
       jabs    dw 0
       iabs    dw 0
       np            dw ?
       nv            dw ?
       na     dw 2
       nb     dw 1
       N      dw 10
       NU     dw 4
       G2p dd
0.0001,0.0004,0.0007,0.0012,0.0017,0.0024,0.0032,0.0040,0.0049,0.0060
;      DUant   dd 0.0, 0.0    ;dimension (nb+1)x1 .Aunque solo interesa
                              ;DUant(1),DUant(2),...DUant(nb+1) pero no
                              ;DUant(0) que es DU(t)
       F   dd 2.9608,   - 2.9216,   0.9608
              dd 5.8447,   - 7.6894,   2.8447
              dd 9.6155,   -14.2310,   5.6155
              dd 14.2385,  -22.4770,   9.2385
              dd 19.6802,  -32.3604,  13.6802
              dd 25.9085,  -43.8170,  18.9085
              dd 32.8926,  -56.7853,  24.8926
              dd 40.6029,  -71.2058,  31.6029
              dd 49.0108,  -87.0217,  39.0108
```

```
              dd 58.0891, -104.1782,  47.0891
       DU1    dd
0.7240,2.4687,4.5237,6.3932,7.8858,9.0164,9.7991,10.2476,10.3749,10.1937

       y   dd 2.5
              dd 2.5
              dd 2.5
;      w   dd 0.0
;      dd 0.0
;             dd 0.0
;             dd 0.0
       fm  dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
       u   dd 0.0
       DU     dd 0
       nav    dw 2
       nbv    dw 7
       DUant   dd 0.0, 0.0, 0.0,0.0,0.0,0.0,0.0,0.0
                              ;dimension (nb+1)x1 .Aunque solo interesa
                           ;DUant(1),DUant(2),...DUant(nb+1) pero no
                            ;DUant(0) que es DU(t)
       G2v dd  0.0289, 0.0386, 0.0371, 0.0356, 0.0342, 0.0136, 0.0088
              dd     0.0611, 0.0671, 0.0645, 0.0620, 0.0403, 0.0194, 0.0069
              dd  0.0784, 0.0795, 0.0764, 0.0541, 0.0327, 0.0122, 0.0034
              dd  0.0961, 0.0984, 0.0753, 0.0531, 0.0317, 0.0112, 0.0050
              dd  0.1188, 0.1024, 0.0791, 0.0568, 0.0353, 0.0146, 0.0062
              dd  0.1196, 0.1020, 0.0788, 0.0564, 0.0349, 0.0143, 0.0052
              dd  0.1183, 0.1004, 0.0772, 0.0549, 0.0334, 0.0128, 0.0049
              dd  0.1182, 0.1009, 0.0777, 0.0554, 0.0339, 0.0133, 0.0054
              dd  0.1188, 0.1015, 0.0783, 0.0559, 0.0345, 0.0138, 0.0054
              dd  0.1187, 0.1012, 0.0779, 0.0556, 0.0341, 0.0135, 0.0052
       Fv  dd       0.7800, -0.2191, 0.4391
              dd  0.3893,  0.2682, 0.3425
              dd     0.5719,  0.2572, 0.1709
              dd     0.7033,  0.0456, 0.2511
              dd     0.5942,  0.0970, 0.3088
              dd     0.5605,  0.1786, 0.2609
              dd     0.6158,  0.1381, 0.2461
              dd     0.6184,  0.1112, 0.2704
              dd     0.5936,  0.1349, 0.2715
              dd     0.5979,  0.1415, 0.2606
       DU1v dd 0.0421, 0.0764, 0.1040, 0.1260, 0.1472, 0.1676, 0.1540, 0.1458,
0.1422, 0.1422
    yv   dd 0.0
              dd 0.0
              dd 0.0
       fmv  dd 0.0
            dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
              dd 0.0
```

```
               dd 0.0
               dd 0.0
       indfv           dw 0
       intnum     dw 128
       floatnum   dd ?
       convfactor dd 0.009803

       convf1     dd 25.5
       comando    dw 0
       EXTRN   _cuenta:word
       EXTRN   _epsi2:word
       EXTRN   _epsi3:word
       EXTRN   _reji:word
       TRI1    EQU     16384
       TRI2    EQU     549
       TRI3    EQU     8788
       wFrac           DW      0
       DWLowTickCount  DD      0
       PUBLIC _datpos
       PUBLIC _datvel
       PUBLIC _com
       PUBLIC _muestras
       _muestras dw 0
       _datpos   dw 1500 dup(0)
       _datvel   dw 1500 dup(0)
       _com      dw 1500 dup(0)
.FARDATA
       hola db 1
;      reji dw 32000 dup(?)        ;matriz de comandos
;-----------------------------------------------------------------------
mensaje macro arg
      mov dx,offset arg
      mov ah,09h
      int 21h
endm
;-----------------------------------------------------------------------
.CODE
PUBLIC _toma6
_toma6  PROC
       mov ax,@data
       mov ds,ax
       mov ax,@fardata
       mov es,ax
       ASSUME es:@fardata
       xor di,di
       xor cx,cx
       finit               ;inicializar el coprocesador

       call limpia
       xor dx,dx
       call cursor

       mov dx,503h
       mov al,90h       ;INICIALIZAR EL 8255 (INTERFACE 100)
       out dx,al

       mov al,0b6h      ;Inicializar el contador de tiempo (canal 2 del 8254).
       out 43h,al       ;Determinar el periodo (periodo=cuenta/1193180).
       mov ax,[_cuenta] ;Primero el byte bajo y luego el alto.
       out 42h,al
       mov al,ah
       out 42h,al
```

```
       in al,61h        ;Activar el contador.
       or al,1
       out 61h,al
       call leerdat
;      mov ax,[j]
;      or ax,ax
;      jns inicio
;      mensaje texto11
;      mov [comando],0
;      call envcom
;      jmp final
inicio:
       mov ax,[j]
       mov [intnum],ax
       call int2flo              ;
       fld dword ptr floatnum
       fst dword ptr y
       fst dword ptr [y+4]
       fstp dword ptr [y+8]

       push di
       call    TickCount              ;primera llamada absoluta
       pop  di
       mov     ax,word ptr [DWLowTickCount]
       mov     dx,word ptr [DWLowTickCount+2]
       mov     word ptr [cont1],ax
       mov     word ptr [cont1+2],dx
       call near ptr etapa1
       cmp ax,02h                 ;Si buffer lleno,acabar.
       je final
       push di
       push bx
       call    TickCount              ;Segunda medida de tiempo.
       pop bx
       pop di
       mov     ax,word ptr [DWLowTickCount]
       mov     dx,word ptr [DWLowTickCount+2]
       mov     word ptr [cont2],ax
       mov     word ptr [cont2+2],dx
       call near ptr etapa3
       push di
       push bx
       call    TickCount              ;Tercera llamada absoluta
       pop bx
       pop di
       mov     ax,word ptr [DWLowTickCount]
       mov     dx,word ptr [DWLowTickCount+2]
       mov     word ptr [cont3],ax
       mov     word ptr [cont3+2],dx

       mov [comando],0
       call envcom
       call near ptr tiempos
       cmp ax,1
       je final
final:
       mov [_muestras],di
       ret

_toma6   ENDP
```

```
;------------------------------------------------------------------------------
etapa1 PROC NEAR
       xor bx,bx
       xor si,si
       push bx
       mov ax,npini              ;valores iniciales de np y nv
       mov np,ax
       mov ax,nvini
       mov nv,ax
e1entry: mov cx,np
alta1:
       in al,61h
       test al,20h
       jnz alta1
baja1:  in al,61h
       test al,20h
       jz baja1
       call leerdat           ;leer datos
       mov ax,[i]
       mov [_datvel+di],ax    ;almacena la velocidad en un buffer.
ipos:   mov ax,[j]
       mov [_datpos+di],ax    ;almacena los datos de posici¢n en un buffer.
       inc di
       inc di
jpos:
       mov ax,[j]
       cmp ax,0
       jge e1seg26
       jmp fin1
e1seg26:     cwd
       xor ax,dx
       sub ax,dx
       mov bx,ax

       mov ax,[i]
       cwd
       xor ax,dx
       sub ax,dx
       add ax,bx
       cmp ax,_epsi2
       jg e1seguir1
       jmp fin1
e1seguir1:cmp di,3000
       jne e1seguir11
           mov [comando],0
           call envcom
           mensaje texto9
           mov ax,02
           jmp fin1
e1seguir11:
       call controlp

       mov ax,comando
       cwd
       xor ax,dx
       sub ax,dx
       cmp ax,255
       jle e1seg19
       mov comando,255
       fld dword ptr DU
       fsub dword ptr u
       cmp signo,80h
```

```
       jne e1seg29
       fadd dword ptr umax
       fst dword ptr DU
       fstp dword ptr DUant
       fld  dword ptr umax
       fstp  dword ptr u
       jmp e1seg19
e1seg29:
       fsub dword ptr umax
       fst dword ptr DU
       fstp dword ptr DUant
       fld  dword ptr umin
       fstp  dword ptr u
e1seg19:
       pop bx
       call envcom
       push bx
       loop prealta1
       jmp e1seg31
prealta1:
       jmp alta1
e1seg31:
       mov ax,nv
       or ax,ax
       jnz e1seguir3
       jmp e1seg25
e1seguir3:
       mov cx,nv
alta2:
       in al,61h
       test al,20h
       jnz alta2
baja2:  in al,61h
       test al,20h
       jz baja2
       call leerdat
       mov ax,[i]
       mov [_datvel+di],ax
ipos2:   mov ax,[j]
       mov [_datpos+di],ax
       inc di
       inc di
jpos2:
       mov ax,[j]
       cmp ax,0
       jge e1seg27
       jmp fin1
e1seg27:
       cwd
       xor ax,dx
       sub ax,dx
       mov bx,ax
       mov jabs,ax

       mov ax,[i]
       cwd
       xor ax,dx
       sub ax,dx
       mov iabs,ax
       add ax,bx
       cmp ax,_epsi2
       jg e1seguir4
```

```
       jmp fin1
e1seguir4:cmp di,3000
       jne e1seguir12
           mov [comando],0
           call envcom
           mensaje texto9
           mov ax,02
           jmp fin1
e1seguir12:
       call controlv

       mov ax,comando
       cwd
       xor ax,dx
       sub ax,dx
       cmp ax,255
       jle e1seguir18
       mov comando,255
       fld dword ptr DU
       fsub dword ptr u
       cmp signo,80h
       jne e1seg30
       fadd dword ptr umax
       fst dword ptr DU
       fstp dword ptr DUant
       fld  dword ptr umax
       fstp  dword ptr u
       jmp e1seguir18
e1seg30:
       fsub dword ptr umax
       fst dword ptr DU
       fstp dword ptr DUant
       fld  dword ptr umin
       fstp  dword ptr u
e1seguir18:
       pop bx
       call envcom
       push bx

       loop prealta2
       jmp e1seg25
prealta2: jmp alta2

e1seg25:
       mov ax ,jabs
       cmp ax,epmed
       jg  e1seguir2
       cmp indic1,0
       jne e1seg21
       mov ax,npmed
       mov np,ax
       mov ax,nvmed
       mov nv,ax
       mov indic1,1
e1seg21:
       mov ax,iabs
       cmp ax,evfin
       jg e1seguir2
       cmp indic2,0
       jne e1seguir2
       mov ax,npfin
       mov np,ax
```

```
       mov ax,nvfin
       mov nv,ax
       mov indic2,1
e1seguir2:
       jmp e1entry
fin1:
       pop bx
       ret
etapa1 ENDP


;-----------------------------------------------------------------
etapa3  PROC   ; CONTROL PROPORCIONAL
alta3:
       in al,61h
       test al,20h
       jnz alta3
baja3:  in al,61h
       test al,20h
       jz baja3
       mov [signo],0h
       mov ax,[j]
       or ax,ax
       jns umas3
       mov [signo],80h
       neg ax
umas3:
       mov dl,[consta]
       mul dl
       xor ah,ah
       mov [comando],ax
       call envcom
       push bx
       call leerdat
       mov ax,[j]
       mov [_datpos+di],ax
       cwd
       xor ax,dx
       sub ax,dx
       mov bx,ax
       mov ax,[i]
       mov [_datvel+di],ax
       inc di
       inc di
       cmp di,3000
       jne e3seguir
          pop bx
          mov [comando],0
          call envcom
          mensaje texto8
            mov ax,02
          jmp fin3
e3seguir:    cwd
       xor ax,dx
       sub ax,dx
       add ax,bx
       pop bx
       cmp ax,_epsi3
       jle fin3
       jmp alta3
fin3:
       ret
```

```
       etapa3  ENDP


;---------------------------------------------------------------------
;
;   subrutina para leer los datos de posici¢n y velocidad
leerdat PROC NEAR
       push cx
LEERPOS:
       mov dx,502h       ;ENVIAR BYTE DE CONTROL PARA LECTURA SIGNO DE POSICION
       mov al,18h        ;PORTC=0.00.11.000=18H  (canal 0)
       out dx,al
       mov cx,20         ; 20 para proces. 486 DX2, 10 para XT/AT
TIEMPOP1:               ;TIEMPO DE CONVERSION (5 MICROSEG.)
       loop TIEMPOP1
       dec dx
       dec dx            ;LEER SIGNO
       in al,dx
       and ax,1          ;AISLAR EL BIT 0 DE AL
       mov bl,al         ;GUARDAR SIGNO EN BL.
       mov dx,502h       ;ENVIAR BYTE DE CONTROL PARA LECTURA DE MAGNITUD
       mov al,098h       ;PORTC=1.00.11.000=98H  (canal 0)
       out dx,al
       mov cx,100h       ; 100h para 486 DX2 ¢ 60h para XT/AT
TIEMPOP2:               ;TIEMPO DE CONVERSION (40 MICROSEG.)
       loop TIEMPOP2
       dec dx
       dec dx            ;LEER MAGNITUD
       in al,dx
       cmp bl,1
       je seguir1
       neg ax
seguir1:
       mov  [j],ax       ;ALMACENA el error (r-y) en j

LEERVEL:
       mov dx,502h
       mov al,38h       ;PORTC=0.01.11.000=38H (canal 1)
       out dx,al
       mov cx,20        ;20 para 486 DX2 ¢ 10 para XT/AT
TIEMPOV1:               ;TIEMPO DE CONVERSION (5 MICROSEG.)
       loop TIEMPOV1
       dec dx
       dec dx           ;LEER SIGNO
       in al,dx
       and ax,1         ;AISLAR EL BIT 0 DE AL
       mov bl,al        ;GUARDAR SIGNO EN bl
       mov dx,502h      ;ENVIAR BYTE DE CONTROL PARA LECTURA DE magnitud
       mov al,0b8h      ;PORTC=1.01.11.000=0b8H  (canal 1)
       out dx,al
       mov cx,100h      ;100h para 486 DX2 ¢ 60h para XT/AT
TIEMPOV2:              ;TIEMPO DE CONVERSION (40 MICROSEG.)
       loop TIEMPOV2
       dec dx
       dec dx           ;LEER MAGNITUD
       in al,dx
       cmp bl,1
        je seguir2
        neg ax
seguir2:
       mov [i],ax
       pop cx
```

```
       ret
leerdat ENDP


;-------------------------------------------------------------------
; subrutina que realiza el control gpc en posici¢n
;
controlp proc
       push cx
       mov ax,[j]                ;ACTUIALIZAR YP
       mov [intnum],ax
       call int2flo              ;convertir la posici¢n de entero a flotante
       call actualy              ;actualizacion de y
       fld dword ptr floatnum
       fstp dword ptr y
       mov ax,[i]                ;ACTUALIZAR YV
       mov [intnum],ax
       call int2flo              ;convertir la posici¢n de entero a flotante
       call actyv                ;actualizacion de y
       fld dword ptr floatnum
       fstp dword ptr yv
       call obtenf
       call obtenu
       call actDU
       call flo2int
       pop cx
       ret
controlp endp


;-------------------------------------------------------------------
; subrutina que realiza el control gpc en velocidad
;
controlv proc
       push cx
       mov ax,[i]                ; ACTUALIZAR Yv
       mov [intnum],ax
       call int2flo              ;convertir la posici¢n de entero a flotante
       call actyv                ;actualizacion de y
       fld dword ptr floatnum
       fstp dword ptr yv
       mov ax,[j]                ;ACTUIALIZAR YP
       mov [intnum],ax
       call int2flo              ;convertir la posici¢n de entero a flotante
       call actualy              ;actualizacion de y
       fld dword ptr floatnum
       fstp dword ptr y
       call obtenfv
       call obtenuv
       call actDU
       call flo2int
       pop cx
       ret
controlv endp


;--------------------------------------------------------------------
; SUBRUTINA PARA CONVERTIR EL DATO LEIDO DE POSICION O VELOCIDAD DE
; ENTERO A PUNTO FLOTANTE
; EL DATO A CONVERTIR SE DEBE PASAR EN LA VARIABLE INTNUM, EL RESULTADO
; SE RETORNA EN LA VARIABLE FLOATNUM
;
```

```
int2flo proc
       fild word ptr intnum
       fmul dword ptr convfactor
       fstp floatnum
       ret
       int2flo endp


;-------------------------------------------------------------------------
; ACTUALIZACION DEL VECTOR y
;
actualy proc
       push cx
       mov cx,na
       mov si,cx
       dec si
       shl si,1
       shl si,1
act1:   fld dword ptr y[si]
       fstp dword ptr y[si+4]
       sub si,4
       loop act1
       pop cx
       ret
actualy endp


;-------------------------------------------------------------------------
obtenf proc
       push cx
       push di
       xor si,si
       xor di,di
        xor bx,bx
       mov cx,N


obt1:   push cx
       fldz
       fld dword ptr G2p[di]
       fmul DUant[4]
       faddp
;segundo sumando
       fldz
       mov cx,na
       inc cx
obt3:   fld dword ptr F[bx]
       fld dword ptr y[si]
       fmulp
       faddp
       add bx,4
       add si,4
       loop obt3
       faddp
       fstp dword ptr fm[di]
       add di,4
       xor si,si
       pop cx
       loop obt1
       pop di
       pop cx
       ret
```

```
obtenf endp


;---------------------------------------------------------------------
; OBTENCION DE LOS RESULTADOS
;
obtenu proc
       push cx
       xor si,si
       mov cx,N
       fldz
odu1:    fld dword ptr DU1[si] ;calculo de DU
       fld dword ptr fm[si]
       fchs
       fmulp
       faddp
       add si,4
       loop odu1
       fst dword ptr DU
       fld dword ptr u
       faddp
       fstp dword ptr u
       pop cx
       ret
obtenu endp


;---------------------------------------------------------------------
; ACTUALIZACION DEL VECTOR DUant
;
actDU proc
       push cx
       mov cx,nbv
       mov si,cx
       dec si
       shl si,1
       shl si,1
actdu1:
       fld dword ptr DUant[si]
       fstp dword ptr DUant[si+4]
       sub si,4
loop actdu1
;      fld dword ptr DUant
;      fstp dword ptr DUant[4]
       fld dword ptr DU
       fstp dword ptr DUant
       pop cx
       ret
actDU endp


;---------------------------------------------------------------------
; SUBRUTINA PARA CONVERTIR EL COMANDO OBTENIDO DE PUNTO FLOTANTE A
; ENTERO.
; EL DATO A CONVERTIR SE DEBE PASAR EN LA VARIABLE FLOATNUM, EL RESULTADO
; SE RETORNA EN LA VARIABLE INTNUM
;
flo2int proc
       fld dword ptr convf1
       fld dword ptr u
       fmulp
       fistp word ptr comando
```

```
       mov signo,80h
       cmp comando,0
       jge finflo
       neg comando
       mov signo,00h
finflo:
       ret
flo2int endp


;-------------------------------------------------------------------------
; SUBRUTINA PARA ACTUALIZAR EL VECTOR YV
actyv proc
       fld dword ptr yv[4]
       fstp dword ptr yv[8]
       fld dword ptr yv
       fstp dword ptr yv[4]
       ret
actyv endp


;-------------------------------------------------------------------------
obtenfv proc
       push cx
       push di
       xor si,si
       xor di,di
;        xor bx,bx
        xor ax,ax
        mov cx,N
obtv1:   push cx
       mov cx,nbv
    fldz
       xor bx,bx
sum1:   fld dword ptr G2v[di]
       fmul DUant[bx+4]
        faddp
        add di,4
        add bx,4
       loop sum1
;segundo sumando
       fldz
       mov cx,nav
       inc cx
       mov bx,indfv
obtv3:   fld dword ptr Fv[bx]
       fld dword ptr yv[si]
       fmulp
       faddp
       add bx,4
       add si,4
       loop obtv3
        faddp
        xchg ax,di
        fstp dword ptr fm[di]
       add di,4
        xchg ax,di
       xor si,si
       mov indfv,bx
       pop cx
       loop obtv1
       mov indfv,0
```

```
       pop di
       pop cx
       ret
obtenfv endp


;-------------------------------------------------------------------------
;subrutina para obtener DU y u para velocidad
obtenuv proc
       push cx
       xor si,si
       mov cx,N
       fldz
oduv1:    fld dword ptr DU1v[si]      ;calculo de DU
       fld dword ptr fm[si]
       fchs
       fmulp
       faddp
       add si,4
       loop oduv1
       fst dword ptr DU
       fld dword ptr u
       faddp
       fstp dword ptr u
       pop cx
       ret
obtenuv endp


;-------------------------------------------------------------------------
;                   SUBRUTINA PARA ENVIAR EL COMANDO AL SISTEMA
envcom PROC NEAR
       push cx
       xor ax,ax
       mov al,byte ptr [comando]
       cmp [signo],80h
       jne com1
          neg ax
com1:  mov [_com+bx],ax
       inc bx
       inc bx
       mov al,byte ptr [comando]  ;envia valor absoluto del comando
       mov dx,501h
       out dx,al
       mov al,10h            ;byte de control para enviar el dato (0 00 10 000)
       add al,[signo]        ;define el signo
       inc dx
       out dx,al
       mov cx,30
repet:    loop repet         ;tiempo de conversion del dato

       cmp al,90h
       sub al,10h            ;envia a port c byte de control (s 00 00 000) para
       out dx,al             ;mantener el valor enviado en la salida
       pop cx
       ret
envcom ENDP


;-----------------------------------------------------------------
; Subrutina que halla el tiempo transcurrido a partir de las lecturas
; del contador de tiempo almacenadas en cont1,cont2,cont3.
```

```
tiempos proc near
tinicio:
  mov bx,word ptr [cont1]
  mov cx,word ptr [cont1+2]
  mov ax,word ptr [cont2]
  mov dx,word ptr [cont2+2]
  sub ax,bx
  sbb dx,cx
  mov cx,10
  mov si,10
  mov byte ptr [num+si],0h
  dec si
  call bin_a_asc
  mensaje texto5
  call sacanum
  mensaje texto7
  mov bx,word ptr [cont1]
  mov cx,word ptr [cont1+2]
  mov ax,word ptr [cont3]
  mov dx,word ptr [cont3+2]
  sub ax,bx
  sbb dx,cx
  mov cx,10
  mov si,10
  mov byte ptr [num+si],0h
  dec si
  call bin_a_asc
  mensaje texto6
  call sacanum
  mensaje texto7
tfinal:
  ret
tiempos endp


;---------------------------------------------------------------------
TickCount   PROC   NEAR
       push es
       mov     al,0C2h         ;orden de lectura del contador 0
;(antes C2 = 11.00.001.0)
;
       cli
       out     43h,al          ;se env¡a a la palabra de estado
       jmp     short $+2       ;espera
       in      al,40h          ;obtiene el estado de la salida
        jmp     short $+2
       shl     al,1            ;acarreo = estado de la salida

       in      al,40h          ;lectura contador bajo
       jmp     short $+2
       mov     cl,al           ;se salva el contador bajo
       in      al,40h          ;lectura contador alto
       jmp     short $+2
       mov     ch,al           ;se salva el contador alto

       jcxz    restart         ;recomenzar si el contador es cero
       rcr     cx,1            ;combinar el contador con el estado
                           ;de la salida
;      not     cx              ;cambiar alto -> bajo por bajo -> alto
       neg     cx

       mov     bx,0040h        ; rea de datos de la ROM BIOS
```

```
       mov     es,bx
       mov     bx,006Ch        ;ES:[BX] = informaci¢n de timer

       mov     ax,TRI1
       mul     word ptr es:[bx]
       mov     wFrac,ax        ;fracci¢n
       mov     si,dx
       mov     ax,TRI1
       mul     word ptr es:[bx+2]
       mov     di,dx
       add     si,ax
       adc     di,0
       mov     ax,TRI2
       mul     word ptr es:[bx]
       add     si,ax
       adc     di,dx
       mov     ax,TRI2
       mul     word ptr es:[bx+2]
       add     di,ax
       mov     ax,TRI3
       shr     cx,1
       shr     cx,1
       shr     cx,1
       shr     cx,1
       mul     cx
       add     ax,word ptr wFrac
       mov     ax,si
       adc     ax,dx
       mov     dx,di
       adc     dx,0

       cmp     dx,word ptr [DWLowTickCount+2]
       ja      done
       cmp     ax,word ptr [DWLowTickCount]
       jb      restart

done:
       mov     word ptr [DWLowTickCount],ax
       mov     word ptr [DWLowTickCount+2],dx
       sti
       pop es               ;restablece segmento.
       ret

restart:
       sti
       jmp     TickCount
TickCount   ENDP


;---------------------------------------------------------------------------
; Subrutina de conversion de un n§ binario a ascii hexadecimal
; Llamar con DX:AX = valor con signo de 32 bits.
;            CX    = base
;            SI    = ultimo byte del area donde almacenar el resultado
;(asegurese que hay suficiente espacio disponible
;para guardar la cadena en la base elegida)
;Devuelve    SI = apunta al comienzo de la cadena ASCII.
;
; Destruye AX, BX, CX, DX, y SI.
;

bin_a_asc proc      near   ;Convierte DX:AX a ASCII.
```

```
                        ;Obliga a guardar al menos un digito.
       mov    byte ptr [num+si],'0'
       or     dx,dx         ;Comprueba el signo del valor de 32 bits
       pushf                ;y salva dicho signo en la pila.
       jns    bin1          ;Salta si era positivo
       not    dx            ;si era negativo, halla el complemento a 2
       not    ax            ;del valor.
       add    ax,1
       adc    dx,0
bin1:                       ;Divide el valor de 32 bits por la raiz
                            ;para extraer el siguiente digito de la
                            ;cadena.
       mov    bx,ax         ;Es el valor cero ya?
       or     bx,dx
       jz     bin3          ;Si, hemos hecho la conversion.
       call   divide        ;No, divide por la raiz
       add    bl,'0'        ;convierte el resto a un digito ASCII.
       cmp    bl,'9'        ;Podriamos estar convirtiendo a ASCII Hex.
       jle    bin2          ;salta si esta en el rango 0-9,
       add    bl,'A'-'9'-1  ;corrige si esta en el rango A-F.
bin2:  mov    [num+si],bl         ;Almacena este caracter en la cadena.
       dec    si            ;Retrocede a traves de la cadena,
       jmp    bin1          ;y lo hace de nuevo.
bin3:                       ;Recupera el flag de signo
       inc si                   ;Apunta al primer valor de la cadena.
       popf                 ;Era negativo el valor original?
       jns    bin4          ;No, salta.
                            ;Si, almacena el signo en la cadena devuelta.
       mov    byte ptr [num+si],'-'
bin4:  ret                  ;Vuelve a la rutina de llamada.
bin_a_asc endp


;-------------------------------------------------------------------------------
; Subrutina para dividir un n§ de 32 bits. Utilizada en bin_a_asc.
; Llama con DX:AX = dividendo de 32 bits.
;           CX    = divisor
;
; Devuelve  DX:AX = cociente
;           BX    = resto
;          CX    = divisor (sin modificar)
;
divide proc   near          ; Divide DX:AX por CX
       jcxz   div1          ; termina si es division por cero
       push   ax            ; 0:dividendo_superior/divisor
       mov    ax,dx
       xor    dx,dx
       div    cx
       mov    bx,ax         ; BX = cociente1
       pop    ax            ; resto1:dividendo_inferior/divisor
       div    cx
       xchg   bx,dx         ; DX:AX = cociente:cociente2
div1:  ret                  ; BX = resto2
divide endp


;-----------------------------------------------------------------------
; Subrutina que saca por pantalla el tiempo invertido
sacanum proc near
       mov dl,[num+si]
       cmp dl,0h
         je finsaca
```

```
       mov ah,02h
       int 21h
       inc si
       jmp sacanum
finsaca:
       ret
sacanum endp


;-------------------------------------------------------------------------
limpia proc near
       mov ax,0600h    ;scroll
       mov bh,07
       xor cx,cx
       mov dx,184fh
       int 10h
limpia endp


;-------------------------------------------------------------------------
cursor proc near
       mov ah,02h
       xor bh,bh
       int 10h
       ret
cursor endp


;-------------------------------------------------------------------------

       END _toma6
```

E.2.3 gpcd5.m

```
% SIMULACION DETERMINISTA/ESTOCASTICA DEL CONTROLADOR GPC
% PSEUDO-MULTIVARIABLE ACTUANDO SOBRE EL MOTOR.
%
clear;
% La relación con funciones de transferencia es:
% den <--> A ; num <--> B
file=['load ',input(' Fichero de datos (entre comillas simples)?: ')];
eval(file);
npini=input(' Numero de controles en posicion inicialmente?: ');
              % np es el numero de vece que se actuara sobre la posicion.
npmed=input(' Numero de controles en posicion a la mitad?: ');
npfin=input(' Numero de controles en posicion al final?: ');
nvini=input(' Numero de controles en velocidad inicialmente?: ');
              % nv es el numero de veces que se actuara sobre la velocidad.
nvmed=input(' Numero de controles en velocidad a la mitad?: ');
nvfin=input(' Numero de controles en velocidad al final?: ');
Ap=A;
Bp=B;
nap=length(A)-1;         % Grado del polinomio A
nbp=length(B)-1;         % Grado del polinomio B
N=input(' Horizonte de salida?: ');
indic1=0;indic2=0;
nfin=20;
lambdap=input(' Lambdap?: ');
```

```
lambdav=input(' Lambdav?: ');
evfin=1.0;

if (lambdav<=0.1),
       epmed=0.5;

elseif lambdav>1,
       epmed=1.5;
else
       epmed=1;
end
%nfin=input('nfin? ');
wp=zeros(nfin+N,1);
wv=zeros(nfin+N,1);
%w=5*ones(nfin+N,1);
%NU=input(' Horizonte de control?: ');
NU=4;
umax=+10;
umin=-10;
DAp=delta(A);
DBp=delta(B);
DUp=zeros(NU,1);
DUantp=zeros(nbp+1,1);
up=zeros(nbp,1);        %Vector con comandos anteriores u(t-1),...,u(t-(nb+1))
yp=-2.5*ones(nap+1,1);  %Vector con salidas anteriores del sistema
Yp(1)=yp(1);

%  CALCULO DE LOS POLINOMIOS Fj. Se emplea para el calculo de Ej y para
%  los fp. ESTOS POLINOMIOS SE ALMACENARAN EN UNA MATRIZ DONDE LA FILA J-ESIMA
%  REPRESENTA EL POLINOMIO Fj (i --> columnaps j --> filas ). Dim: (N,nap+1)
%
Fp=-DAp;        % F1=q(1-DAp)
Fp(1)=[];      %
for j=2:N,
    for i=1:nap,
        Fp(j,i)=Fp(j-1,i+1)-DAp(i+1)*Fp(j-1,1);
    end
    Fp(j,nap+1)=-DAp(nap+2)*Fp(j-1,1);
end
%
% CALCULO DE Ej
%
%
Ep=1;
for j=2:N,
    Ep(j,1)=1;
    for i=2:j,
        Ep(j,i)=Fp(i-1,1);
    end
end

%         CALCULO DE Gp (MATRIZ DE POLINOMIOS Ei*B)
%
Gp=zeros(N,nbp+N);
Gp(1,1:nbp+1)=B;
B1=[0,B];
for j=2:N,
     GGp=Gp(j-1,1:nbp+j)+Ep(j,j)*B1; % La nueva fila de Gp sera igual a la
anterior
                                % mas el producto del nuevo elemento que aporta
                                % el nuevo E por B.
     Gp(j,1:nbp+j)=GGp(1:nbp+j);
```

```
     B1=[0,B1];
end

% CALCULO DE G1p. MATRIZ QUE DA CUENTA DE LOS COMANDOS A APLICAR EN EL FUTURO
% (Triangular inferior).
G1p=zeros(N,NU);
for j=1:NU,
    for i=1:j,
        G1p(j,i)=Gp(j,(j+1)-i);
    end
end
for j=NU+1:N
    for i=1:NU,   % A PARTIR DE LA FILA NU LA MATRIZ SOLO TIENE NU COLUMNAS.
        G1p(j,i)=Gp(j,(j+1)-i);
    end
end
%
%       CALCULO DE UNA MATRIZ ADICIONAL G2p QUE  EMPLEO  PARA  CALCULAR
%       LAS fp. LA FILA J DE ESTA MATRIZ SERIA Gn-SUM(q^-i*gi,N), QUE ES
%       LO QUE HAY QUE MULTIPLICAR POR DUp PARA EL CALCULO DE fp. ESTA PARTE DE
%       LA MATRIZ G DA CUENTA DE LOS COMANDOS CONOCIDOS.
if nbp >= 1,
   for j=1:N,
       G2p(j,:)=Gp(j,j+1:j+nbp);
   end
else
   G2p=zeros(N,1);
end
DU1p=inv(G1p'*G1p+lambdap*eye(NU))*G1p';
%*************************************************************************
%********************** PARAMETROS PARA LA VELOCIDAD **********************
%*************************************************************************
load planta5;
%AAv=A;
%BBv=B;
Av=A;
Bv=B;
CCv=[1 -1.12 -0.1027 0.08 0.33];
DDv=[1 -1.90 0.99];
nav=length(Av)-1;         % Grado del polinomio A (ruido)
nbv=length(Bv)-1;         % Grado del polinomio B (ruido)
%naav=length(AAv)-1;       % Grado del polinomio A (no ruido)
%nbbv=length(BBv)-1;       % Grado del polinomio B    (no ruido)
rand('normal');
for i=1:nfin,
      vg(i)=rand;
end
var=0.002;
desv=sqrt(var);
vg=vg*desv;
vm=filter(CCv,DDv,vg);
             load ruivm
wv=zeros(nfin+N,1);
DAv=delta(Av);
DBv=delta(Bv);
DUv=zeros(NU,1);
DUantv=zeros(nbv+1,1);
yv=zeros(nav+1,1);
Yv(1)=yv(1);
%  CALCULO DE LOS POLINOMIOS Fj.
Fv=-DAv;        % F1=q(1-DAv)
Fv(1)=[];
```

```
for j=2:N,
    for i=1:nav,
        Fv(j,i)=Fv(j-1,i+1)-DAv(i+1)*Fv(j-1,1);
    end
    Fv(j,nav+1)=-DAv(nav+2)*Fv(j-1,1);
end
% CALCULO DE Ej
Ev=1;
for j=2:N,
    Ev(j,1)=1;
    for i=2:j,
        Ev(j,i)=Fv(i-1,1);
    end
end
%         CALCULO DE Gv (MATRIZ DE POLINOMIOS Ei*B)
Gv=zeros(N,nbv+N);
Gv(1,1:nbv+1)=Bv;
B1=[0,Bv];
for j=2:N,
     GGv=Gv(j-1,1:nbv+j)+Ev(j,j)*B1;
     Gv(j,1:nbv+j)=GGv(1:nbv+j);
     B1=[0,B1];
end
G1v=zeros(N,NU);
for j=1:NU,
    for i=1:j,
        G1v(j,i)=Gv(j,(j+1)-i);
    end
end
for j=NU+1:N
    for i=1:NU,
        G1v(j,i)=Gv(j,(j+1)-i);
    end
end
if nbv >= 1,
   for j=1:N,
       G2v(j,:)=Gv(j,j+1:j+nbv);
   end
else
   G2v=zeros(N,1);
end
DU1v=inv(G1v'*G1v+lambdav*eye(NU))*G1v';
%
%°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
%
%         COMIENZO DEL BUCLE QUE SE REPITE N VECES Y QUE VA OBTENIENDO
%         EN CADA MOMENTO EL COMANDO Y LA SALIDA DEL SISTEMA SEGUN PROPONE
%         EL CONTROLADOR GPC.
%
np=npini;
nv=nvini;
global N nap nbp nbv nav wp wv Fp Fv G2p G2v DU1p DU1v DUantp DUantv k;
k=1;
error=2.5;
precis=0.8;
           return;
           hola=1
while ( k<nfin & error>precis),   %BUCLE PRINCIPAL
%****************** CONTROL EN POSICION **********************************
%
for i=1:np,
    if k < nfin,
```

```
    [up DUp]=cpd5(DUp,yp,up);
    DUv=DUp;
%------------------------ SALIDA DE POSICION -----------------------------
    y1p=0;                  % Valor de yp(t+1).
    for i=2:nap+1,               % nap=2
        y1p=-Ap(i)*yp(i-1)+y1p;
    end
    for i=1:nbp+1,
        y1p=Bp(i)*up(i)+y1p;
    end
%         ACTUALIZACION DE yp.
%      y2p=-Ap(2)*yp(1)-Ap(3)*yp(2)+Bp(1)*up(1)+Bp(2)*up(2)
    for i=nap+1:-1:2,
        yp(i)=yp(i-1);
    end
    yp(1)=y1p;
    Yp(k+1)=y1p;
    Up(k)=up(1);
%------------------------------ SALIDA EN VELOCIDAD -------------------------
    y1v=0;                  % Valor de yv(t+1).
    for i=2:nav+1,
        y1v=-Av(i)*yv(i-1)+y1v;
    end
    for i=1:nbv+1,
        y1v=Bv(i)*up(i,1)+y1v;
    end
    y1v=y1v+vm(k);
%         ACTUALIZACION DE yv.
%      y2v=-Av(2)*yv(1)+Bv(1)*up(1)
    for i=nav+1:-1:2,
        yv(i)=yv(i-1);
    end
    yv(1)=y1v;
    Yv(k+1)=y1v;
    Uv(k)=up(1);
%    dist=abs(yv(1))+abs(yp(1));       % norma 1
    dist=abs(yp(1));
    if (dist>5 | abs(yv(1))>10 | abs(yp(1))>10),
         plot(-Yp,-Yv);
         error(' ERROR: OBJETIVO DE CONTROL NO ALCANZADO.');
    end
    k=k+1;
       error=abs(yp(1))+abs(yv(1));
   if (error<precis),
      break;
   end              % if error

    end  % if k< nfin+1
end % for i=1:np
%***************************** CONTROL EN VELOCIDAD **********************
for i=1:nv,
  if k < nfin,
  [up DUv]=cvd5(DUv,yv,up);
  DUp=DUv;
%--------------------------- SALIDA EN POSICION --------------------------
    y1p=0;                  % Valor de yp(t+1).
    for i=2:nap+1,
        y1p=-Ap(i)*yp(i-1)+y1p;
    end
    for i=1:nbp+1,
        y1p=Bp(i)*up(i)+y1p;
    end
```

```
%         ACTUALIZACION DE yp.
%      y2p=-Ap(2)*yp(1)-Ap(3)*yp(2)+Bp(1)*up(1)+Bp(2)*up(2)
    for i=nap+1:-1:2,
        yp(i)=yp(i-1);
    end
    yp(1)=y1p;
    Yp(k+1)=y1p;
    Up(k)=up(1);
%----------------  CALCULO DE LA SALIDA EN VELOCIDAD -----------------------
    y1v=0;                   % Valor de yv(t+1).
    for i=2:nav+1,
        y1v=-Av(i)*yv(i-1)+y1v;
    end
    for i=1:nbv+1,
        y1v=Bv(i)*up(i)+y1v;
    end
    y1v=y1v+vm(k);        % Se¤al afectada por el ruido

%         ACTUALIZACION DE yv.
%      y2v=-Av(2)*yv(1)+Bv(1)*up(1)
    for i=nav+1:-1:2,
        yv(i)=yv(i-1);
    end
    yv(1)=y1v;
    Yv(k+1)=y1v;
    Uv(k)=up(1);

    dist=abs(yp(1));
    if (dist>5 | abs(yv(1))>10 | abs(yp(1))>10),
         plot(-Yp,-Yv);
         error(' ERROR: OBJETIVO DE CONTROL NO ALCANZADO.');
    end

    k=k+1;
       error=abs(yp(1))+abs(yv(1));
    if (error<precis),     % Objetivo de control alcanzado.
       break;
    end             % if

    end      % if k< (nfin)

end          % for i=1:nv

   if ((abs(yp(1)) < epmed) & (indic1==0)),
      np=npfin;
      nv=nvfin;
   end
   if ( (abs(yv(1)) < evfin) & (indic2==0) & (indic1==1)),
      np=npfin;
      nv=nvfin;
         indic2=1;
   end

end   % while k<nfin+1 DEL BUCLE PRINCIPAL

Yp=-Yp;
Yv=-Yv;
Uv(k)=Uv(k-1);
Up(k)=Up(k-1);
hold off;
subplot(211);
plot(Yp,Yv);
```

```
texto1=['GPC Lp= 'num2str(lambdap) '; Lv= 'num2str(lambdav) '; T= 'num2str(T)];
texto2=['; N= 'num2str(N) '; NU= 'num2str(NU)];
texto1=[texto1 texto2];
title(texto1);
xlabel('Posicion');
ylabel('Velocidad');
subplot(212);
plot(Yp,Up);
texto3=['npini= ' num2str(npini) ', npfin= 'num2str(np)];
texto4=[ '; nvini= 'num2str(nvini) ', nvfin= 'num2str(nv)];
texto3=[texto3 texto4];
title(texto3)
ylabel('Comando');
xlabel('Posicion');
subplot(111);

save determ
```

E.3 CONTROL GPC ROBUSTO

E.3.1 gpcrob.m

```
%fichero que aplica un controlador gpc a laplanta estabilizada
%mediante un controlador robusto.
%
clc
clear
disp(' ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿');
disp(' ³          C O N T R O L A D O R  G P C / R O B U S T O        ³');
disp(' ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ');
disp('');
o=menu('         ENTRADA DE DATOS           ','Tomar datos de planta
estabilizada'...
,'Estabilizar planta');
if (o==1)
       file1=['load ',input('    - Fichero de datos de la planta estabilizada:
','s')];
       eval(file1);
elseif (o==2)
       clc;
       disp(' ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿');
       disp(' ³         E S T A B I L I Z A D O R  R O B U S T O         ³');
       disp(' ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ');
       disp('')
%    disp(' ------  CONSTRUCCION DE LA PLANTA -------');
%      disp('');disp('');
%      npol=input(' - Numero de polos: ');
%      ncer=input(' - Numero de ceros: ');
%      den=1;num=1;
%      disp('');
%      for i=1:ncer,
%             cero=input(' - Cero: ');
%             num=conv(num,[1 -cero]);
%      end
%      for i=1:npol,
%             polo=input(' - Polo: ');
%             den=conv(den,[1 -polo]);
```

```
%      end

%      num1=conv([1 0.5+10*j],[1 0.5-10*j]);
%      num2=[1 70];
%num1=[1 102.1 216.1 1115.5 1050];
%num2=1;
%      num1=conv(conv([1 2+j],[1 2-j]),conv([1 3+5*j],[1 3-5*j]));
%      num2=[1 1];
%      num1=input('numerador continuo?: ');
%      num2=input('numerador continuo?: ');
%      num=conv(num1,num2);
%      den1=conv(conv([1 5+10*j],[1 5-10*j]),conv([1 10+10*j],[1 10-10*j]));
 %   den1=[1 26.5 258.5 1154 2570];
%      den2=1;
%      den2=conv([1 0.5],[1 100]);
%      den1=input('den1?: ');
       %den2=input('den2?: ');
%den1=[1 17 85 164 148 80];
%den2=1;
%      den1=conv(conv([1 1+j],[1 1-j]),conv([1 2+5*j],[1 2-5*j]));
%      den2=conv([1 3+j],[1 3-j]);
%      den=conv(den1,den2);
       %num=1;
       %den=[1 -3 2];
num=input(' Numerador cont¡nuo?');
den=input(' Denominador cont¡nuo?');

       [ag,bg,cg,dg]=tf2ss(num,den)
       [nsal,nent]=size(dg);
       w=logspace(-3,5,100);
       svg=sigma2(ag,bg,cg,dg,1,w);svg=20*log10(svg);
       %semilogx(w,svg);
       %k=input('k?:');
       %numw1=2000;
       %---------------------------------------- PERTURBACIONES ----------
       dnw1i=100*conv([1/5000 1],[1/5000 1]);
       nuw1i=conv([1/100 100],[1/100 1]);
       svw1i=bode(nuw1i,dnw1i,w);svw1i=20*log10(svw1i);
       semilogx(w,svw1i);
       grid;
       hold on;
              nuw3i=[0 2000];
       nuw3i=[0 10000];    %numerador menos restrictivo
       dnw3i=[1 0];
       svw3i=bode(nuw3i,dnw3i,w);svw3i=20*log10(svw3i);
       semilogx(w,svw3i);
       title('Respuesta en frecuencia de las perturbaciones 1/w1 y 1/w3')
       hold off;
       pause;
       %------------------------------------ PLANTA AUMENTADA -------------
       Gam=1.2;
       sysg=[ag bg;cg dg];
       w1=[Gam*dnw1i;nuw1i];
       w2=[];
       w3=[dnw3i;nuw3i];
       xg=length(den)-1;
       [A,B1,B2,C1,C2,D11,D12,D21,D22]=augtf(sysg,xg,w1,w2,w3);
       D11
       [aa,bb,cc,mm,tt]=obalreal(A,[B1,B2],[C1;C2]);
       A=aa;B1=bb(:,1:nent);
       B2=bb(:,nent+1:nent+nsal);
       C1=cc(1:2*nsal,:);
```

```
       C2=cc((2*nsal+1):(2*nent+nent),:);
       %------------------------------------------ HINF -------------------
       %D11=0.1
       hinf
       pltopt
       [numf,denf]=ss2tf(acp,bcp,ccp,dcp);      % controlador puesto en fdt
       numfg=conv(num,numf);        % fdt de F*G    (F --> controlador)
       denfg=conv(den,denf);
       denlc=sumpol(numfg,denfg);  % fdt en lazo cerrado continua
       numlc=numfg;
       disp('');
       disp('');
       disp('---  F.D.T. DE LA PLANTA CONTINUA ESTABILIZADA EN LAZO CERRADO ---
');
       printsys(numlc,denlc)

end
T=input('   - Periodo de muestreo?: ');
[numd,dend]=c2dm(num,den,T,'zoh');
[numfd,denfd]=c2dm(numf,denf,T,'zoh');    %fdt discreta del controlador F
[numfgd,denfgd]=c2dm(numfg,denfg,T,'zoh');    %fdt discreta del controlador F
[numlcd,denlcd]=c2dm(numlc,denlc,T,'zoh'); % fdt en lazo cerrado discreta
B=numlcd;A=denlcd;
B(1)=[];
i=1;
lima=length(A);
while i<=lima
       if A(i)==0 A(i)=[]; lima=lima-1; end
       i=i+1;
end
i=1;
limb=length(B);
while i<=limb
       if B(i)==0 B(i)=[]; limb=limb-1; end
       i=i+1;
end
%------------------------------------------ CEROS Y POLOS IGUALES --------
disp('');
disp('---------------- ELIMINANDO CEROS Y POLOS IGUALES -----------------');
raiza=roots(A);                   % Comparamos si existen ceros y polos
raizb=roots(B);                   % que sean iguales ...
lima=length(raiza);
limb=length(raizb);
i=0;j=0;
while i<=lima-1
       i=i+1;
       while j<=limb-1
              j=j+1;
              if (abs(raiza(i)-raizb(j))) < 0.00001 % si es as¡, se eliminan
                     raiza(i)=[];
                     raizb(j)=[];
                 j=0;
                 i=0;
                 lima=lima-1;
                 limb=limb-1;
                     disp(' - Warning:  Raiz cancelada !!')
                     break;
              end
       end
    j=0;
end
%-------------------------------------- RECONSTRUCCION A y B ----------------
```

```
disp('');
disp(' ------------- RECONSTRUYENDO LOS POLINOMIOS A Y B -------------');
I=find(imag(raiza)~=0);    % primero hacemos la convolucion de las raices
A=1;                       % complejas conjugadas ...
for k=1:2:length(I)
       A=conv(A,conv([1 -raiza(I(k))],[1 -raiza(I(k+1))]));
end
I=find(imag(raiza)==0);    % y luego la de las raices reales
for k=1:length(I)
       A=conv(A,[1 -raiza(I(k))]);
end

I=find(imag(raizb)~=0);    % lo mismo para el polinomio B
B=1;
for k=1:2:length(I)
       B=conv(B,conv([1 -raizb(I(k))],[1 -raizb(I(k+1))]));
end
I=find(imag(raizb)==0);
for k=1:length(I)
       B=conv(B,[1 -raizb(I(k))]);
end
B=B/(B(1)/numlcd(2));
% guardamos los resultados
file2=['save ',input('  - Fichero mat donde guardar los datos de la planta?:…
','s'), ' A',' B',' T',' num',' den',' denlc',' numlc',' numlcd',' denlcd','…
raiza',' raizb',' numf',' denf',' numfgd',' denfgd',' numfd',' denfd','… numd','…
dend',' denfg',' numfg'];
eval(file2);

%-------------------------------- CONTROLADOR GPC ----------------------
disp('');
disp('------------------- EJECUTANDOSE EL CONTROLADOR GPC ----------------');
gpcs7;
```

E.3.2 sumpol.m

```
%
%  Funcion que suma dos polinomios de igual o diferente grado
%  Esta función es empleada por el script gpcrob.m
%
function [pol3]=sumpol(pol1,pol2);
if length(pol1)==length(pol2);
       pol3=pol1+pol2;
elseif length(pol1)<length(pol2),
       pol3=pol2+[zeros(1,length(pol2)-length(pol1)) pol1];
elseif length(pol2)<length(pol1),
       pol3=pol1+[zeros(1,length(pol1)-length(pol2)) pol2];
end
```

E.4 CONTROL NEURONAL

E.4.1 npid102.c

```
/* CONTROL NEURONAL SELF-TUNING DE UN CONTROLADORPI
 SE DEBE COMPILAR CON EL MODELO DE MEMORIA MEDIUM.
```

```
 PARA EJECUTAR ESTE PROGRAMA, DEBE PASAR COMO ARGUMENTOS
 EN LA LINEA DE COMANDO LOS VALORES Q, R Y S RESPECTIVAMENTE */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <time.h>
#include <filew.h>
#include <guarda3.h>
#define L 3                     /*nº de capas de la red. */
#define factor 0.01             /* factor de inicializacion  de pesos */
#define sysorder 1              /* orden del sistema */
#define nzeros   0              /* no de ceros del sistema */
#define dimuant  3
#define dimyant  2
#define T 0.01                  /*periodo de muestreo */
#define vel 0.3                 /*factor que regula las vel. de aprend. */
#define pesd 0.0                /* peso de la accion derivativa */
#define pesi 1                   /* peso de la accion integral*/
#define pesp 1                  /* peso de la accion proporcional*/
#define kfinal 1000             /*no de etapas (suma del vector ntran)*/
#define pi 3.141592
#define P 1                     /* n§ de patrones de entrenamiento*/
#define eps 0.000001            /* error requerido*/
#define maxiter 0               /* n§ maximo de iteraciones*/
#define factor 0.01             /* factor de inicializacion  de pesos */
#define alfa   0.001            /* factor de momento en update_weights*/
#define beta   1                /* factor de sigmoide*/
#define ntimes 10               /* nº de veces que se entrena un patron */
#define grab 1                  /*si es 1 se graban los datos a fichero*/

#define w(k,node,i)  *(w+dirw[k]+(node-1)*(nl[k-1]+1)+i)
#define wv(k,node,i)  *(wv+dirw[k]+(node-1)*(nl[k-1]+1)+i)
#define ulj(k,node)  *(ulj+diru[k]+node)
#define elj(k,node)  *(elj+dire[k]+node-1)
#define g(k,node,i)  *(g+dirw[k]+(node-1)*(nl[k-1]+1)+i)
double uant[dimuant]={0,0,0}, yant[dimyant]={0,0},eant[3]={0.1,0.1,0.1},\
  R,Q,S,td[2]={0.003,0},ki[2]={2,2},kp[2]={0.5,0.5},dduk[3],yref[kfinal],\
  ganan[kfinal], tiem[kfinal],ntran[8]={0,100,600,200,25,25,25,25},\
  consig[4]={1,1,1,1};

/*ntran es el numero de etapas en cada fase del control */
/*consig contiene el valor que toma la consigna */
/* ganan es un vector donde se almacena los valores que va tomando kp*/
/* dduk representa la derivada parcial de delta u frente a k */

void feed_forward(double *ulj, double *w, double *xp, double *salida,\
              int np, int nl[], int dirw[], int diru[]);
void compute_gradient(double *ulj,double *elj,double *dj,double *w,\
       double *g,double *xp,double *costo,int np,int nl[],int dirw[],\
       int diru[],int dire[], double uant[],double yant[],double eant[],\
       double duk[]);
void last_layer(double *ulj,double *elj,double *dj,double *g,double *costo,\
              int np,int nl[],int dirw[],int diru[],int dire[],double uant[],\
              double yant[],double eant[],double duk[]);
void update_weights(double *w,double *wv,double *g,double mu[],int nl[],int \
dirw[]);
double calcula_error(double *costo);
void inipesos(double *w, double *wv, int nl[], int dirw[]);
void iniciar(double *dj,double *xp,double *ulj,int np,int nl[],int diru[]\
```

```
              ,double uant[],double yant[]);
double sigmoide(double valor);

/*-------------------------------------------------------- MAIN --------*/
void main(int argc,char *argv[]){
/* variables:
        - pinte: suma de todos los errores hasta la etapa actual. Se multiplica
          por T para obtener la integral del error en el PID.
*/
double valor=123.6, y[kfinal],coma[kfinal],duk[3]={0,0,0};
/*,td[2]={0.3,0.3},ki[2]={0.5,0.5}\
                    ,kp[2]={2,2},duk[3]={0,0,0};*/
double sigm,argsig,*dj,*xp,*ulj,*w,*wv,*sal,*elj,*g,*esal,*costo, *salida,\
    error,mu[L]={2.25*vel,1.5*vel,1*vel},inte=0,*pinte,sum1,sum2;
int i,ii,j,np,iter=0,nl[L+1]={2,3,3,2},dirw[L+1],diru[L+1],dire[L+1],sizee=0,\
    sizeu=0, sizew=0,k,n;
int desplau(int k, int j);
int desplae(int k, int j);
int desplaw(int k, int j,int i);
void entrena(double *ulj,double *elj,double *dj,double *w,double *g,\
        double *xp,double *costo,int np,int nl[],int dirw[],int diru[],\
       int dire[] ,double *salida,double *wv, double mu[],double duk[]);
double sigmoide(double valory);
void setyref(double yref[]);
double planta(double uant[], double yant[],double eant[],double yref[],\
              int ii);
double pid(double e[], double uant[], double td[], double ti[], double \
           k[],double *pinte);
void pinta(double yant[], double yref[],int tam,int n);
Q=atof(argv[1]);                  /*peso del error en la funcion de costo*/
R=atof(argv[2]);                  /*peso del incremento de u en fc. costo*/
S=atof(argv[3]);
printf("Empezamos...\n");
for(k=0;k<=L;k++){
       diru[k] = sizeu;                     /*dir. base de elj de cada capa*/
       sizeu += nl[k]+1;
}
for(k=1;k<=L;k++){
       dire[k] = sizee;                   /*dir. base de ulj de capa capa*/
       dirw[k] = sizew;                   /*dir. base de w de capa capa*/
       sizew += nl[k]*(nl[k-1]+1);
       sizee += nl[k];                    /* tantas e por capa como nodos*/
}
w=(double *) malloc(sizew*8);         /* RESERVA DE MEMORIA DINAMICA */
wv=(double *) malloc(sizew*8);
g=(double *) malloc(sizew*8);
xp=(double *) malloc(nl[0]*P*8);          /*patrones entrenamiento */
dj=(double *) malloc(nl[L]*P*8);          /* salida deseada */
ulj=(double *) malloc(sizeu*8);
elj=(double *) malloc(sizee*8);
costo=(double *) malloc(P*8);              /* costo para cada patron */
salida=(double *) malloc(P*nl[L]*8);       /* salida para cada patron */
pinte=&inte;
np=0;
setyref(yref);
iniciar(dj,xp,ulj,np, nl, diru, uant, yant);
/*creaf(); */    /*crea un fichero con los valores de los pesos*/
inipesos(w,wv,nl,dirw);
ganan[0]=kp[1];
tiem[0]=ki[1];
/*ganan[1]=kp[0];*/
ulj(0,0)=1;
```

```
uant[0]=kp[1]*eant[0];   /* Al principio solo hay acci¢n proporcional */
coma[0]=uant[0];
y[0]=planta(uant,yant,eant,yref,0);            /*calcula la salida*/
for (ii=1;ii<kfinal;ii++){
       coma[ii]=pid(eant,uant,td,ki,kp,pinte);         /*calcula el comando*/
       y[ii]=planta(uant,yant,eant,yref,ii);            /*calcula la salida*/
       iniciar(dj,xp,ulj,np, nl, diru, uant, yant);
       duk[0]=eant[0];      /*derivadas parciales de Du con respecto a k*/
       duk[1]=duk[2]*T*eant[0];
       duk[2]=(eant[0]-eant[1])/T;
       dduk[0]=eant[0]-eant[1];      /*derivadas parciales de Du con k*/
       dduk[1]=T*eant[1];
       entrena(ulj,elj,dj,w,g,xp, costo, np,nl,dirw,diru,dire,salida,\
               wv,mu,duk);
       kp[1]=kp[0];
       kp[0]=*(salida);
       ganan[ii]=*(salida);
       printf("ii= %d\n",ii);
       ki[1]=ki[0];
       ki[0]=*(salida+1);
       if (ki[0] > 1e4) {
            printf(" La evolucion no es la correcta. Simulacion parada");
            getch();
           break;
           } /*if*/
       tiem[ii]=ki[0];
       printf("ki= %lf\n", ki[0]);
} /*for*/

/*printf("\n\t\t  ...ya est !");
getch();*/
pinta(y,yref,ii,2);
pinta(coma,yref,ii,2);
if (grab==1)
      grabar(coma,y,yref,ganan,tiem);
free(w);free(wv);free(g);free(xp);free(dj);free(ulj);free(elj);free(costo);
free(salida);
}  /*main*/
/* ------------------------------------------------------ SETYREF -----------*/
void setyref(double yref[]){
int i,k;
double incr, dete,a,b,c,d,x0,x1,y0,y1,t=0,ini=0,fin=0;

 /* dete es el determinante de la matriz que define la ecuacion de
       interpolaci¢n. a,b,c,d son los coeficientes del polinomio de
       interpolaci¢n y(t)= ax^3 +bx^2 +cx+ d */
/* consigna de entrenamiento inicial*/

yref[0]=0;
for (i=1;i<ntran[1];i++){
       /* yref[i]=consig0*sin(0.3*i)+consig0;*/
    yref[i]=consig[0];
}/*for*/
/*se considera una consigna variable con tres cambios ==> bucle 3 veces*/
for(k=1;k<=3;k++){
  ini=fin+ntran[2*k-1];
  fin=ini+ntran[2*k];
  x0=(double)ini*T;
  x1=(double)fin*T;
  y0=consig[k-1];
  y1=consig[k];
  dete=(4*x1*x1*x1*x0+4*x1*x0*x0*x0-6*x0*x0*x1*x1-x1*x1*x1*x1-x0*x0*x0*x0);
```

```
  a=(2*x0-2*x1)*(y0-y1)/dete;
  b=(3*x0*x0-3*x1*x1)*(y1-y0)/dete;
  c=(6*x0*x0*x1-6*x1*x1*x0)*(y0-y1)/dete;
  d=(y1*(4*x1*x0*x0*x0-3*x0*x0*x1*x1-x0*x0*x0*x0)+y0*(4*x1*x1*x1*x0-\
    3*x1*x1*x0*x0-x1*x1*x1*x1))/dete;
  t=ini*T;
  if (ntran != 0) {
       /*incr=(double)(consig-yref[ntran0-1])/(ntran);*/
/*    t=t+ini*T;*/
    for (i=ini;i<fin;i++){           /*cambio suave a la siguiente consigna*/
    yref[i]=a*t*t*t + b*t*t + c*t + d;
    t=t+T;
    }/*for*/
  }/*if*/
  for (i=fin;i<fin+ntran[2*k+1];i++){
    yref[i]=y1;               /* consigna constante */
  }
} /*for k*/

} /* setyref*/
/* ------------------------------------------------------ PLANTA -----------*/
double planta(double uant[], double yant[],double eant[],double yref[],int ii){
double salplan;
int i;
double k=2.5, a=0.4;
for (i=0;i<(dimyant-1);i++){
       yant[dimyant-i-1]=yant[dimyant-i-2];
} /* for */
salplan=yant[0]/(a*T+1) + k*T*uant[0]/(a*T+1);
yant[0]=salplan;
eant[2]=eant[1];
eant[1]=eant[0];
eant[0]=yref[ii]-yant[0];
return salplan;
} /*planta */

/*-------------------------------------------------------- PID ----------*/
double pid(double eant[], double uant[], double td[], double ki[], double
kp[],double *pinte){
int i;
for (i=0;i<(dimuant-1);i++){
       uant[dimuant-i-1]=uant[dimuant-i-2];
} /* for */
*pinte=*pinte+eant[0];
uant[0]=pesp*kp[0]*eant[0] + pesi*ki[0] * T * (*pinte);
return uant[0];
} /* pid */
/* ------------------------------------------------ ENTRENA ------------*/
void entrena(double *ulj,double *elj,double *dj,double *w,double *g,\
   double *xp,double *costo,int np,int nl[],int dirw[],int diru[],int dire[]\
   ,double *salida,double *wv, double *mu,double duk[]){

int iter=0,i,n;
double error;

do{
  for(np=0;np<P;np++){                    /* entrenamos cada patron */
    for(n=1;n<=nl[0];n++) ulj(0,n)= *(xp+np*nl[0]+n-1);   /* entrada */
    for(i=0;i<ntimes;i++){               /*ntimes veces por patron*/
      feed_forward(ulj,w,xp,salida,np,nl,dirw,diru);
      compute_gradient(ulj,elj,dj,w,g,xp, costo, np,nl,dirw,diru,dire,uant,\
                  yant,eant,duk);
```

```
      update_weights(w,wv,g,mu,nl,dirw);
    } /* i*/
  }/* for np*/
  iter++;
/*  printf("Iter= %d\n", iter);*/
  error=calcula_error(costo);
}while((error > eps) && (iter<maxiter));
 /*con la red entrenada calculamos cada salida */
/*printf("w= %g\n",*(w+10));*/
for(np=0;np<P;np++){
  for(n=1;n<=nl[0];n++) ulj(0,n)= *(xp+np*nl[0]+n-1);
  feed_forward(ulj,w,xp,salida,np,nl,dirw,diru);
}/* for np*/
}  /* entrena */
/*------------------------------------------------------ CALCULA_ERROR ----*/
double calcula_error(double *costo){
int i,j;
double error=0;
for(i=0;i<P;i++) error=error+(*(costo+i));
  error=error/P;
  return error;
}  /* calcula_error */

/*------------------------------------------------------ INIPESOS ---------*/
void inipesos(double *w,double *wv,int nl[], int dirw[]){
FILE *fpt;
int i,node,j,k,despla,a,b,c;
double aa,AA,valor;
fpt=fopen("pesos.dat","r");

for (k=1;k<=L;k++){
  for(node=1;node<=nl[k];node++){               /* nodo capa actual*/
    for(i=0;i<=nl[k-1];i++){
      fscanf(fpt,"%lf",&valor);
      w(k,node,i)=valor;
      wv(k,node,i)=w(k,node,i);
    } /*i*/
  } /* node */
} /* k */
close(fpt);
} /* inipesos */
/*------------------------------------------------------ SIGMOIDE -----------*/
double sigmoide(double valory){
double sigm;
if (valory>20)
   sigm =1;
else if (valory < -20)
   sigm =0;
else
   sigm =1/(1+exp(-beta*valory));
return sigm;
}


/*------------------------------------------------------ INICIAR -----------*/
void iniciar(double *dj,double *xp,double *ulj, int np,int nl[],int diru[],\
     double uant[],double yant[]){
int i;
*xp=eant[0];                      /*las entradas son el error y su derivada*/
*(xp+1)=(eant[0]-eant[1])/T;
} /*iniciar */
```

```
/*------------------------------------------------------ FEED_FORWARD ----*/
void feed_forward(double *ulj, double *w, double *xp, double *salida, int np,\
int nl[], int dirw [], int diru []){
int layer,node,input,i,j,despla;
double argsig=0,peso,entrada,sum1,sum2;

for(node=1;node<=nl[1];node++,argsig=0){                /*PRIMERA CAPA*/
  for(i=0;i<=nl[0];i++){
    argsig=argsig+w(1,node,i)*ulj(0,i);
  } /*i*/
  ulj(1,node)=argsig;
/*  printf("ulj(%d,%d)= %g\n",layer,node,ulj(layer,node));*/
} /*node */
ulj(1,0)=1;
for(layer=2;layer<L;layer++){                          /*SIGUIENTES CAPAS*/
  for(node=1;node<=nl[layer];node++,argsig=0){
    for(i=0;i<=nl[layer-1];i++){
      argsig=argsig+w(layer,node,i)*ulj(layer-1,i);
    } /*i*/
    ulj(layer,node)=sigmoide(argsig);
/*      printf("ulj(%d,%d)= %g\n",layer,node,ulj(layer,node));*/
  } /*node */
  ulj(layer,0)=1;                        /*entrada bias igual a uno*/
} /*layer*/

for(node=1;node<=nl[L];node++,argsig=0){                 /*ULTIMA CAPA*/
  for(i=0;i<=nl[L-1];i++){
    sum1=w(L,node,i);
    sum2=ulj(L-1,i);
    argsig=argsig+w(L,node,i)*ulj(L-1,i);
  } /*i*/
  ulj(layer,node)=argsig;
/*  printf("ulj(%d,%d)= %g\n",layer,node,ulj(layer,node));*/
} /*node */
ulj(layer,0)=1;
for(i=0;i<nl[L];i++){           /*guardamos la salida para cada patron*/
  *(salida+np*nl[L]+i)=ulj(L,i+1);
} /*i*/
} /* feedforward*/


/*------------------------------------------------ COMPUTE_GRADIENT ------*/
void compute_gradient(double *ulj,double *elj,double *dj,double *w,double *g,\
  double *xp,double *costo,int np,int nl[],int dirw[],int diru[],int dire[],\
  double uant[],double yant[],double eant[],double duk[]){
  int layer,node,i,despla;
double argsig,peso,entrada,ecumul=0,cuadrado,sum1,sum2,sum3;
void last_layer(double *ulj,double *elj,double *dj,double *g,double *costo,\
     int np,int nl[],int dirw[],int diru[],int dire[],double uant[],\
     double yant[],double eant[],double duk[]);

last_layer(ulj,elj,dj,g,costo,np,nl,dirw,diru,dire,uant,yant,eant,duk);

for(node=1;node<=nl[L-1];node++,ecumul=0){   /*  C A P A    L-1 */
  for(i=1;i<=nl[L]; i++){                  /*nodos capa siguiente */
    ecumul=ecumul+ elj(L,i) * w(L,i,node);
  }   /*i*/
  elj(L-1,node)=ecumul;
  for(i=0;i<=nl[L-2];i++){
    g(L-1,node,i) = elj(L-1,node) * ulj(L-1,node)*(1-ulj(L-1,node))*(ulj\
```

```
                    (L-2,i));
  } /*i*/
} /* node */

for (layer=L-2;layer>1;layer--){        /*C A P A S   A N T E R I O R E S*/
  for(node=1;node<=nl[layer];node++,ecumul=0){
    for(i=1;i<=nl[layer+1]; i++){               /*nodos capa siguiente */
      ecumul=ecumul+ elj(layer+1,i) * ulj(layer+1,i) \
            * (1- ulj(layer+1,i)) * w(layer+1,i,node);
    }   /*i*/
    elj(layer,node)=ecumul;
    for(i=0;i<=nl[layer-1];i++){
      g(layer,node,i) = elj(layer,node) * ulj(layer,node)\
      *(1- ulj(layer,node)) * (ulj(layer-1,i));
      sum1=g(layer,node,i);
    } /*i*/
  } /* node */
} /*layer */

for(node=1;node<=nl[1];node++,ecumul=0){   /*P R I M E R A   C A P A*/
  for(i=1;i<=nl[2]; i++){                  /*nodos capa siguiente */
    ecumul=ecumul+ elj(2,i) * ulj(2,i) \
          * (1- ulj(2,i)) * w(2,i,node);
  }   /*i*/
  elj(1,node)=ecumul;
  for(i=0;i<=nl[0];i++){
    g(1,node,i) = elj(1,node) * ulj(0,i);
  } /*i*/
} /* node */
} /*compute_gradient */


/* ---------------------------------------------------- LAST_LAYER ----*/
void last_layer(double *ulj,double *elj,double *dj,double *g,double *costo,\
     int np,int nl[],int dirw[],int diru[],int dire[],double uant[],\
     double yant[],double eant[],double duk[]){
/* en esta subrutina se calcula la derivada de J con rspecto a la salida
de la red. En el caso de un PID es igual a: -e* dy/du * du/dki * dki/dw  */
int node,i;
double dyk,deltau1,deltau2;
dyk=(yant[0]-yant[1])/(uant[0]-uant[1]);          /*aprox. de la derivada*/
deltau1=uant[0]-uant[1];
deltau2=uant[1]-uant[2];

*(costo+np*nl[0])=0;                                  /* costo a cero */
for(node=1;node<=nl[L];node++){
  elj(L,node)=(-Q*eant[0]*dyk+S*uant[0])*duk[node-1] + R*deltau1*dduk[node-1];
  for(i=0;i<=nl[L-1];i++){                          /*i -->todos los pesos*/
    g(L,node,i)= elj(L,node) * ulj(L-1,i);
  } /*i*/
} /* node */
  *(costo+np*nl[0]) += 1;

}   /* last_layer */


/*------------------------------------------------------ UPDATE_WEIGHTS ---*/
void update_weights(double *w,double *wv,double *g, double mu[],int nl[], int
dirw []){
double r,pru;
int i,node,k;
for (k=1;k<=L;k++){
```

```
  for(node=1;node<=nl[k];node++){    /*nodo de la capa actual*/
    for(i=0;i<=nl[k-1];i++){
      r=w(k,node,i) - wv(k,node,i);
      wv(k,node,i) = w(k,node,i);
      w(k,node,i)=w(k,node,i)-mu[k-1]*g(k,node,i)+alfa*r;
}/*i*/
  } /* node */
} /* k */
} /*update_weigths*/


/*-------------------------------------------------------- PINTA ----------*/
void pinta(double *vectory, double *vectory1,int tam,int n){
/* si la variable n vale 1 se pintara la grafica definida por el vector
vectory, si n vale 2 se pintara tanto vectory como vectory1
(esto se utiliza para pintar la grafica con la salida y la referencia o
la grafica con el comando solamente)*/
int redondea(double valor);
int gdriver = DETECT, gmode, errorcode;
int xmax, ymax,x,y,i,j,xant,yant;
double valymax,valymin,paso,nx=0;
char buffer[30];
/* calculo valor maximo del eje "y" de entre todas las variables a pintar */
valymax=*vectory;
valymin=*vectory;
for(i=0;i<tam;i++){
  if (*(vectory+i)<valymin) valymin=*(vectory+i);
  if (*(vectory+i)>valymax) valymax=*(vectory+i);
} /*for*/
valymax=valymax+0.2*valymax;
initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode != grOk){
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1);
} /* if */
setcolor(getmaxcolor());
xmax = getmaxx();
ymax = getmaxy();
paso=(double)xmax/(tam-1);
line(0,0,0,ymax);
line(0,0,xmax,0);                       /*ejes*/
line(xmax,ymax,xmax,0);
line(xmax,ymax,0,ymax);
sprintf(buffer,"%g",valymax);           /*valores de parametros */
outtextxy(10,10,buffer);
sprintf(buffer,"%g",valymin);
outtextxy(10,ymax-10,buffer);
sprintf(buffer,"%g",tam*T);
outtextxy(xmax-35,ymax-10,buffer);

yant= *(vectory)*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
xant=0;
setlinestyle(SOLID_LINE,1,1);
setcolor(10);

for (i=1;i<tam;i++){                    /*SALIDA DE LA RED*/
  y= *(vectory+i)*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
  nx=nx+paso;
  x=redondea(nx);
```

```
  line(xant,yant,x,y);
/*    circle(x, y, 3);   */
  xant=x;
  yant=y;
} /*for */

if (n==2) {
  yant= *(vectory1)*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
  xant=0;
  setlinestyle(SOLID_LINE,1,1);
  setcolor(11);
  nx=0;

  for (i=1;i<tam;i++){                    /*referencia*/
    y= *(vectory1+i)*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
    nx=nx+paso;
    x=redondea(nx);
    line(xant,yant,x,y);
/*    circle(x, y, 3);   */
    xant=x;
    yant=y;
  } /*for */
} /*if n*/
             /* clean up */
getch();
closegraph();
/*  return 0;*/
}  /*pinta */

/*------------------------------------------------------- REDONDEA -------*/
int redondea(double valor)
{
double a,b;
a=floor(valor);
b=valor-a;
if (valor<0){
  a=ceil(valor);
  b=a-valor;
}
if (b<=0.5)
  valor=a;
  else if(valor>0)
    valor=a+1;
  else
    valor=a-1;
return valor;
}  /*redondea */
```

E.4.2 filew.h

```
void creaf(void);
void creaf(void){
FILE *fpt,*fpt1;
int i,node,k,hola;
float valor1;
double aa,AA,nl[L+1]={2,3,3,1},valor,sum1;
fpt=fopen("pesos.dat","w");
randomize();
```

```
for (k=1;k<=L;k++){
  for(node=1;node<=nl[k];node++){
    for(i=0;i<=nl[k-1];i++){
      valor =factor*(2*(double)rand()/(double)RAND_MAX-1);
      fprintf(fpt,"%g\n",valor);
    } /*i*/
  } /* node */
} /* k */
fclose(fpt);
} /* main */
```

E.4.3 guarda3.h

```
/* Esta rutina almacena las variables del proceso.
   El formato es .mat que reconoce el MATLAB.
*/

grabar(double *coma, double *sali, double *refe, double *kp, double *ki);
grabar(double *coma, double *sali, double *refe, double *kp, double *ki)
{
typedef struct{           /* Estructura que representa la cabecera */
       long type;        /* del fichero .mat */
       long mrows;
       long ncols;
       long imagf;
       long namleng;
}Fmatrix;

char *var1="coma",*var2="sali",*var3="refe",*var4="kp",*var5="ki",datos[32];
int mn,i;
FILE *fp;
Fmatrix x;
x.type=0;            /* Valores de los parametros de la cabecera  */
x.ncols=1;            /* para el fichero .mat */
x.imagf=0;
x.namleng=5;
x.mrows=kfinal;
printf("fichero de resultados, plis: ");
scanf("%s",datos);
if((fp=fopen(datos,"wb"))==NULL){
  system("cls");
  printf("Error al abrir el fichero de resultados.");
  exit(0);
}
fwrite(&x,sizeof(Fmatrix),1,fp);
fwrite(var1,sizeof(char),x.namleng,fp);
mn=x.mrows*x.ncols;
fwrite(coma,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);
fwrite(var2,sizeof(char),x.namleng,fp);
fwrite(sali,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);
fwrite(var3,sizeof(char),x.namleng,fp);
fwrite(refe,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);
fwrite(var4,sizeof(char),x.namleng,fp);
```

```
fwrite(kp,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);
fwrite(var5,sizeof(char),x.namleng,fp);
fwrite(ki,sizeof(double),mn,fp);

fclose(fp);
return(0);
}  /*main*/
```

E.5 CONTROL DE LA GRÚA

E.5.1 grua522.c

```
/* CONTROL NEURONAL SELF-TUNIG PARA LA GRUA*/
/* PARA EJECUTAR ESTE PROGRAMA, DEBE PASAR COMO ARGUMENTOS EN LA LINEA
   DE COMANDO LOS VALORES Q, R Y S RESPECTIVAMENTE */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <time.h>
#include <filewg5.h>
#include <guardg52.h>

#define L 3                      /*n§ de capas. Hay que cambiar tambi,n nl*/
#define factor 0.01               /* factor de inicializacion  de pesos */
#define sysorder 1               /* orden del sistema */
#define nzeros   0               /* no de ceros del sistema */
#define dimuant  3
#define dimyant  2
#define T 0.03                   /*periodo de muestreo */
#define vel1 1.9                 /*factor que regula las vel. de aprend. */
#define vel3 0.5
#define pesd 1                /* peso de la accion derivativa */
#define pesi 1               /* peso de la accion integral*/
#define pesp 1                  /* peso de la accion proporcional*/
#define kfinal 800             /*no de etapas (suma del vector ntran)*/
#define pi 3.141592
#define P 1                      /* n§ de patrones de entrenamiento*/
#define eps 0.000001               /* error requerido*/
#define maxiter 0                  /* n§ maximo de iteraciones*/
#define factor 0.01               /* factor de inicializacion  de pesos */
#define alfa   0.001             /* factor de momento en update_weights*/
#define beta   1                 /* factor de sigmoide*/
#define ntimes 10                 /* n§ de veces que se entrena un patron */
#define grab 1                  /*si est  a 1 se graban los datos a fichero*/
#define w(k,node,i)  *(w+dirw[k]+(node-1)*(nl[k-1]+1)+i)
#define w3(k,node,i)  *(w3+dirw3[k]+(node-1)*(nl3[k-1]+1)+i)
#define wv(k,node,i)  *(wv+dirw[k]+(node-1)*(nl[k-1]+1)+i)
#define wv3(k,node,i)  *(wv3+dirw3[k]+(node-1)*(nl3[k-1]+1)+i)
#define ulj(k,node)  *(ulj+diru[k]+node)
#define elj(k,node)  *(elj+dire[k]+node-1)
#define g(k,node,i)  *(g+dirw[k]+(node-1)*(nl[k-1]+1)+i)

int ii;
```

```
double uant[dimuant]={0,0,0},eant1[3]={1,1,1},eant3[3]={0,0,0},\
  R,Q,S,td[2]={0.1,0.1},k3[2]={-2,-
2},ki[2]={2,2},kp[2]={0.5,0.5},dduk[3],y[kfinal],\
  coma[kfinal],ganan[kfinal],
tiemd[kfinal],ganan3[kfinal],ntran[8]={0,0,700,40,50,6,2,2},\
  consig[4]={1,1,1,1},yref[kfinal][4],*xx1,*xx2,*xx3,*xx4;

/*ntran es el numero de etapas en cada fase del control (ver pag. 40)*/
/*consig contiene el valor que toma la consigna */
/* ganan es un vector donde se almacena los valores que va tomando kp
   tiem guarda el tiempo integral del controlador pi */
/* duk es la derivada parcial de u con respecto a k */

/* dduk representa la derivada parcial de delta u frente a k */
/* p es una matriz que contiene los pesos a lo largo de la trayectoria
   para cada uno de las variables de estado*/
/* x contiene las 4 variables de estado*/

void feed_forward(double *ulj, double *w, double *xp, double *salida,\
     int np, int nl[], int dirw[], int diru[]);
void compute_gradient(double *ulj,double *elj,double *dj,double *w,\
     double *g,double *xp,double *costo,int np,int nl[],int dirw[],int\
     diru[], int dire[], double uant[],double eant[],double duk[]);
void last_layer(double *ulj,double *elj,double *dj,double *g,double *costo,\
     int np,int nl[],int dirw[],int diru[],int dire[],double uant[],\
     double eant[],double duk[]);
void update_weights(double *w,double *wv,double *g,double mu[],int nl[],int \
     dirw[]);
double calcula_error(double *costo);
void inipesos(double *w, double *wv, int nl[], int dirw[],double *w3,\
      double *wv3, int nl3[], int dirw3[]);
void iniciar(double *dj,double *xp1,double *xp3,double *ulj,int np,int\
     nl[],int diru[],double uant[]);
double sigmoide(double valor);


/*------------------------------------------------------------ MAIN --------*/
void main(int argc,char *argv[]){

double duk1[3]={0,0,0},duk3[3]={0,0,0};

double sigm,argsig,*dj1,*xp1,*ulj1,*w1,*w3,*wv1,*elj1,*g1,*costo, *salida,\

error,mu1[L]={2.25*vel1,1.5*vel1,1*vel1},mu3[L]={2.25*vel3,1.5*vel3,1*vel3}\
       ,*dj3,*xp3,*ulj3,*wv3,*sal3,*elj3,*g3,inte=0,*pinte,sum1,sum2;
/*la variable costo en este programa realmente no es utilizada*/
unsigned char *buffer1;
int i,j,np,iter=0,nl1[L+1]={2,3,3,2},dirw1[L+1],diru1[L+1],dire1[L+1],sizee1=0,\
    sizeu1=0, sizew1=0,k,n,nl3[L+1]={2,3,3,1},dirw3[L+1],diru3[L+1],dire3[L+1]\
    ,sizee3=0,sizeu3=0, sizew3=0;
void entrena(double *ulj,double *elj,double *dj,double *w,double *g,\
     double *xp,double *costo,int np,int nl[],int dirw[],\
     int diru[],int dire[],double *salida,double *wv, double mu[],\
     double duk[],double eant[]);
double sigmoide(double valory);
void setyref(void);
double planta(double uant[],double eant[],int ii);
double pid(double e[], double uant[], double td[], double ti[]\
       ,double k[],double k3[],double *pinte);
void pinta(double vectory[],int tam,int n,char *buffer1);
Q=atof(argv[1]);                  /*peso del error en la funcion de costo*/
R=atof(argv[2]);                  /*peso del incremento de u en fc. costo*/
```

```
S=atof(argv[3]);
printf("Empezamos...\n");
if (ntran[1]+ntran[2]+ntran[3]+ntran[4]+ntran[5]+ntran[6]+ntran[7] != kfinal) {
   printf("el vector ntran es inconsistente con el numero de etapas\n");
   return;
}
/* inicializacion dimensiones red x1*/
for(k=0;k<=L;k++){
       diru1[k] = sizeu1;                    /*dir. base de elj de cada capa*/
       sizeu1 += nl1[k]+1;
}
for(k=1;k<=L;k++){
       dire1[k] = sizee1;                  /*dir. base de ulj de capa capa*/
       dirw1[k] = sizew1;                  /*dir. base de w de capa capa*/
       sizew1 += nl1[k]*(nl1[k-1]+1);      /*n§ de elementos del vector w*/
       sizee1 += nl1[k];                   /* nº de elementos del vactor elj
                                              tantas e por capa como nodos*/
}
/* inicializacion dimensiones red x3*/
for(k=0;k<=L;k++){
       diru3[k] = sizeu3;                    /*dir. base de elj de cada capa*/
       sizeu3 += nl3[k]+1;
}
for(k=1;k<=L;k++){
       dire3[k] = sizee3;                  /*dir. base de ulj de capa capa*/
       dirw3[k] = sizew3;                  /*dir. base de w de capa capa*/
       sizew3 += nl3[k]*(nl3[k-1]+1);      /*n§ de elementos del vector w*/
       sizee3 += nl3[k];                   /*nº de elementos del vactor elj
                                             tantas e por capa como nodos*/
}

w1=(double *) malloc(sizew1*8);         /* RESERVA DE MEMORIA DINAMICA */
w3=(double *) malloc(sizew3*8);         /* RESERVA DE MEMORIA DINAMICA */
wv1=(double *) malloc(sizew1*8);
wv3=(double *) malloc(sizew3*8);
g1=(double *) malloc(sizew1*8);
g3=(double *) malloc(sizew3*8);
xp1=(double *) malloc(nl1[0]*P*8);          /*patrones entrenamiento */
xp3=(double *) malloc(nl3[0]*P*8);          /*patrones entrenamiento var x3*/
dj1=(double *) malloc(nl1[L]*P*8);          /* salida deseada */
dj3=(double *) malloc(nl3[L]*P*8);          /* salida deseada */
ulj1=(double *) malloc(sizeu1*8);
ulj3=(double *) malloc(sizeu3*8);
elj1=(double *) malloc(sizee1*8);
elj3=(double *) malloc(sizee3*8);
costo=(double *) malloc(P*8);               /* costo para cada patron */
salida=(double *) malloc(P*nl1[L]*8);       /* salida para cada patron */
xx1=(double *) malloc(kfinal*8);
xx2=(double *) malloc(kfinal*8);
xx3=(double *) malloc(kfinal*8);
xx4=(double *) malloc(kfinal*8);
pinte=&inte;
np=0;
*xx1=0;*xx2=0;*xx3=0;*xx4=0;
setyref();
iniciar(dj1,xp1,xp3,ulj1,np, nl1, diru1, uant);
/*creaf(); */    /*crea un fichero con los valores de los pesos*/
inipesos(w1,wv1,nl1,dirw1,w3,wv3,nl3,dirw3);
ganan[0]=kp[1];
/*ganan[1]=kp[0];*/
*ulj1=1;
*ulj3=1;
```

```
uant[0]=0.2;
/*kp[1]*eant[0];
coma[0]=uant[0];
y[0]=planta(uant,eant1,0);
for (ii=1;ii<kfinal;ii++){
 coma[ii]=pid(eant1,uant,td,ki,kp,k3,pinte);  /*calcula el comando*/
 y[ii]=planta(uant,eant1,ii);                 /*calcula la salida en ii+1*/
 iniciar(dj1,xp1,xp3,ulj1,np, nl1, diru1, uant);
 duk1[0]=eant1[0];                            /*derivadas parciales du/dk*/
 duk1[1]=(eant1[0]-eant1[1])/T;
 dduk[0]=eant1[0]-eant1[1];
 dduk[1]=T*eant1[1];
 entrena(ulj1,elj1,dj1,w1,g1,xp1,costo,np,nl1,dirw1,diru1,dire1,salida\
       ,wv1,mu1,duk1,eant1);
 kp[1]=kp[0];
 kp[0]=*(salida);
 ganan[ii]=kp[0];
 td[1]=td[0];
 td[0]=*(salida+1);
 tiemd[ii]=td[0];
 eant3[1]= eant3[0];
 eant3[0]= - *(xx3+ii+1);
 duk3[0]= *(xx3+ii+1);
 entrena(ulj3,elj3,dj3,w3,g3,xp3,costo,np,nl3,dirw3,diru3,dire3,salida\
       ,wv3,mu3,duk3,eant3);
 k3[1]=k3[0];
 k3[0]=*(salida);
 printf("ii= %d\n",ii);
 printf("kp= %lf\n",kp[0]);
 printf("td= %lf\n", td[0]);
 printf("k3= %lf\n", k3[0]);
 ganan3[ii]=k3[0];
} /*for*/

/*printf("\n\t\t  ...ya est !");
getch();*/
buffer1="x1";
pinta(y,kfinal,2,buffer1);
buffer1="x2";
pinta(xx2,kfinal,2,buffer1);
buffer1="x3";
pinta(xx3,kfinal,2,buffer1);
buffer1="x4";
pinta(xx4,kfinal,2,buffer1);
buffer1="Comando";
pinta(coma,kfinal,2,buffer1);
if (grab==1)
      grabar(coma,xx1,xx2,xx3,xx4,ganan,tiemd,ganan3);
free(w1);free(wv1);free(g1);free(xp1);free(dj1);free(ulj1);free(elj1);\
free(w3);free(wv3);free(g3);free(xp3);free(dj3);free(ulj3);free(elj3);\
free(costo);free(salida);free(xx1);free(xx2);free(xx3);free(xx4);
}  /*main*/


/* ------------------------------------------------- SETYREF -----------*/
void setyref(void){
int i,k;
double incr, dete,a,b,c,d,x0,x1,y0,y1,t=0,ini=0,fin=0,val;
 /* dete es el determinante de la matriz que define la ecuacion de
       interpolaci¢n. a,b,c,d son los coeficientes del polinomio de
       interpolaci¢n y(t)= ax^3 +bx^2 +cx+ d */
/* consigna de entrenamiento inicial*/
```

```
yref[0][0]=0;
for (i=0;i<kfinal;i++){
       yref[i][1]=0;
}/*for*/
for (i=0;i<kfinal;i++){
       yref[i][2]=0;
}/*for*/
for (i=0;i<kfinal;i++){
       yref[i][3]=0;
}/*for*/

for (i=1;i<ntran[1];i++){
       val=consig[0];
       yref[i][0]=val;
}/*for*/
/*se considera una consigna variable con tres cambios ==> bucle 3 veces*/
for(k=1;k<=3;k++){
  ini=fin+ntran[2*k-1];
  fin=ini+ntran[2*k];
  if (ini==fin) continue;
  x0=(double)ini*T;
  x1=(double)fin*T;
  y0=consig[k-1];
  y1=consig[k];
  dete=(4*x1*x1*x1*x0+4*x1*x0*x0*x0-6*x0*x0*x1*x1-x1*x1*x1*x1-x0*x0*x0*x0);
  a=(2*x0-2*x1)*(y0-y1)/dete;
  b=(3*x0*x0-3*x1*x1)*(y1-y0)/dete;
  c=(6*x0*x0*x1-6*x1*x1*x0)*(y0-y1)/dete;
  d=(y1*(4*x1*x0*x0*x0-3*x0*x0*x1*x1-x0*x0*x0*x0)+y0*(4*x1*x1*x1*x0-\
    3*x1*x1*x0*x0-x1*x1*x1*x1))/dete;
  t=ini*T;
  if (ntran != 0) {
    for (i=ini;i<fin;i++){
       yref[i][0]=a*t*t*t + b*t*t + c*t + d;
       t=t+T;
    }/*for*/
  }/*if*/
  for (i=fin;i<fin+ntran[2*k+1];i++){
    yref[i][0]=y1;              /* consigna constante */
  }
} /*for k*/

} /* setyref*/


/* ------------------------------------------------------ PLANTA -----------*/
double planta(double uant[],double eant1[],int ii){
double salplan;
int i;
double a=4,k=2.5,den, num,g=10, A,B,C,x1,x2,x3,x4;
int ml=5,mc=10,M,l=5;
M=ml+mc;
den=ml*cos(*(xx3+ii))*cos(*(xx3+ii))-M;
num=uant[0]*cos(*(xx3+ii))+ M*g*sin(*(xx3+ii))+l*ml* *(xx4+ii)*
*(xx4+ii)*sin(*(xx3+ii))*cos(*(xx3+ii));
A= -(double)g*sin(*(xx3+ii))/cos(*(xx3+ii));
B= -(double)num/(den*cos(*(xx3+ii)));
C= (double)num/(l*den);
*(xx1+ii+1)= (double)T**(xx2+ii) + *(xx1+ii);
*(xx2+ii+1)= (double)(A+B)*T + *(xx2+ii);
*(xx3+ii+1)= (double)T**(xx4+ii) + *(xx3+ii);
```

```
*(xx4+ii+1)= (double)T*C + *(xx4+ii);
x1=*(xx1+ii+1);x2=*(xx2+ii+1);x3=*(xx3+ii+1);x4=*(xx4+ii+1);
salplan= *(xx1+ii+1);
eant1[2]=eant1[1];
eant1[1]=eant1[0];
eant1[0]=yref[ii+1][0]- *(xx1+ii+1);
return salplan;
} /*planta */


/*-------------------------------------------------------- PID ----------*/
double pid(double eant[], double uant[], double td[], double ki[]\
     , double kp[],double k3[], double *pinte){
int i;
for (i=0;i<(dimuant-1);i++){
       uant[dimuant-i-1]=uant[dimuant-i-2];
} /* for */
 *pinte=*pinte+eant[0];
 uant[0]=pesp*kp[0]*eant[0] + pesd*td[0]*(eant[0]-eant[1])/T + k3[0]* *(xx3+ii);
return uant[0];
} /* pid */


/* ------------------------------------------------ ENTRENA ------------*/
void entrena(double *ulj,double *elj,double *dj,double *w,double *g,\
   double *xp,double *costo,int np,int nl[],int dirw[],int diru[],int dire[]\
   ,double *salida,double *wv, double *mu,double duk[], double eant[]){

int iter=0,i,n;
double error;

do{
  for(np=0;np<P;np++){                    /* entrenamos cada patron */
    for(n=1;n<=nl[0];n++) ulj(0,n)= *(xp+np*nl[0]+n-1);   /* entrada */
    for(i=0;i<ntimes;i++){               /*ntimes veces por patron*/
      feed_forward(ulj,w,xp,salida,np,nl,dirw,diru);
      compute_gradient(ulj,elj,dj,w,g,xp, costo, np,nl,dirw,diru,dire,uant,\
                    eant,duk);
      update_weights(w,wv,g,mu,nl,dirw);
    } /* i*/
  }/* for np*/
  iter++;
  error=calcula_error(costo);
}while((error > eps) && (iter<maxiter));
 /*con la red entrenada calculamos cada salida */
for(np=0;np<P;np++){
  for(n=1;n<=nl[0];n++) ulj(0,n)= *(xp+np*nl[0]+n-1);
  feed_forward(ulj,w,xp,salida,np,nl,dirw,diru);
}/* for np*/
}  /* entrena */


/*-------------------------------------------------------- CALCULA_ERROR ----*/
double calcula_error(double *costo){
int i,j;
double error=0;
for(i=0;i<P;i++) error=error+(*(costo+i));
  error=error/P;
  return error;
}  /* calcula_error */
```

```
/*------------------------------------------------------ INIPESOS ---------*/
void inipesos(double *w,double *wv,int nl[], int dirw[],double *w3, \
            double *wv3, int nl3[], int dirw3[]){
FILE *fpt,*fpt1;
int i,node,j,k,despla,a,b,c;
double aa,AA,valor;
if((fpt=fopen("pesos2.dat","r"))==NULL){
  system("cls");
  printf("Error al abrir el fichero pesos2.dat.");
  getch();
  exit(0);
}
if((fpt1=fopen("pesos1.dat","r"))==NULL){
  system("cls");
  printf("Error al abrir el fichero pesos1.dat.");
  getch();
  exit(0);
}

for (k=1;k<=L;k++){
  for(node=1;node<=nl[k];node++){                /* nodo capa actual*/
    for(i=0;i<=nl[k-1];i++){
      fscanf(fpt,"%lf",&valor);
      w(k,node,i) = valor;
      wv(k,node,i)=w(k,node,i);
    } /*i*/
  } /* node */
} /* k */

for (k=1;k<=L;k++){
  for(node=1;node<=nl3[k];node++){                /* nodo capa actual*/
    for(i=0;i<=nl3[k-1];i++){
      fscanf(fpt1,"%lf",&valor);
      w3(k,node,i)=valor;
      wv3(k,node,i)=w3(k,node,i);
    } /*i*/
  } /* node */
} /* k */

close(fpt1);
close(fpt);
} /* inipesos */


/*------------------------------------------------------ SIGMOIDE ----------*/
double sigmoide(double valory){
double sigm;
if (valory>20)
   sigm =1;
else if (valory < -20)
   sigm =0;
else
   sigm =1/(1+exp(-beta*valory));
return sigm;
}


/*------------------------------------------------------ INICIAR ----------*/
void iniciar(double *dj,double *xp1,double *xp3,double *ulj, int np,int nl[],int
diru[],\
     double uant[]){
int i;
```

```
      /*las entradas son el error y su derivada*/
*xp1=eant1[0];                    /*entradas para la red NN1*/
*(xp1+1)=(eant1[0]-eant1[1])/T;
*xp3=eant3[0];                  /*entradas para la red NN2*/
*(xp3+1)=(eant3[0]-eant3[1])/T;
} /*iniciar */


/*-------------------------------------------------------- FEED_FORWARD ----*/
void feed_forward(double *ulj, double *w, double *xp, double *salida, int np,\
int nl[], int dirw [], int diru []){
int layer,node,input,i,j,despla;
double argsig=0,peso,entrada,sum1,sum2;

for(node=1;node<=nl[1];node++,argsig=0){               /*PRIMERA CAPA*/
  for(i=0;i<=nl[0];i++){
    argsig=argsig+w(1,node,i)*ulj(0,i);
  } /*i*/
  ulj(1,node)=argsig;
/*  printf("ulj(%d,%d)= %g\n",layer,node,ulj(layer,node));*/
} /*node */
ulj(1,0)=1;
for(layer=2;layer<L;layer++){                         /*SIGUIENTES CAPAS*/
  for(node=1;node<=nl[layer];node++,argsig=0){
    for(i=0;i<=nl[layer-1];i++){
      argsig=argsig+w(layer,node,i)*ulj(layer-1,i);
    } /*i*/
    ulj(layer,node)=sigmoide(argsig);
/*      printf("ulj(%d,%d)= %g\n",layer,node,ulj(layer,node));*/
  } /*node */
  ulj(layer,0)=1;                        /*entrada bias igual a uno*/
} /*layer*/

for(node=1;node<=nl[L];node++,argsig=0){                 /*ULTIMA CAPA*/
  for(i=0;i<=nl[L-1];i++){
    sum1=w(L,node,i);
    sum2=ulj(L-1,i);
    argsig=argsig+w(L,node,i)*ulj(L-1,i);
  } /*i*/
  ulj(layer,node)=argsig;
/*  printf("ulj(%d,%d)= %g\n",layer,node,ulj(layer,node));*/
} /*node */
ulj(layer,0)=1;
for(i=0;i<nl[L];i++){           /*guardamos la salida para cada patron*/
  *(salida+np*nl[L]+i)=ulj(L,i+1);
} /*i*/
} /* feedforward*/


/*------------------------------------------------ COMPUTE_GRADIENT ------*/
void compute_gradient(double *ulj,double *elj,double *dj,double *w,double *g,\
  double *xp,double *costo,int np,int nl[],int dirw[],int diru[],int dire[],\
  double uant[],double eant[],double duk[]){
int layer,node,i,despla;
double argsig,peso,entrada,ecumul=0,cuadrado,sum1,sum2,sum3;
void last_layer(double *ulj,double *elj,double *dj,double *g,double *costo,\
     int np,int nl[],int dirw[],int diru[],int dire[],double uant[],\
     double eant[],double duk[]);

last_layer(ulj,elj,dj,g,costo,np,nl,dirw,diru,dire,uant,eant,duk);

for(node=1;node<=nl[L-1];node++,ecumul=0){   /*  C A P A    L-1 */
```

```
  for(i=1;i<=nl[L]; i++){                  /*nodos capa siguiente */
    ecumul=ecumul+ elj(L,i) * w(L,i,node);
  }   /*i*/
  elj(L-1,node)=ecumul;
  for(i=0;i<=nl[L-2];i++){
    g(L-1,node,i) = elj(L-1,node) * ulj(L-1,node)*(1-ulj(L-1,node))* \
                    (ulj(L-2,i));
  } /*i*/
} /* node */

for (layer=L-2;layer>1;layer--){        /*C A P A S   A N T E R I O R E S*/
  for(node=1;node<=nl[layer];node++,ecumul=0){
    for(i=1;i<=nl[layer+1]; i++){               /*nodos capa siguiente */
      ecumul=ecumul+ elj(layer+1,i) * ulj(layer+1,i) \
            * (1- ulj(layer+1,i)) * w(layer+1,i,node);
    }   /*i*/
    elj(layer,node)=ecumul;
    for(i=0;i<=nl[layer-1];i++){
      g(layer,node,i) = elj(layer,node) * ulj(layer,node)\
      *(1- ulj(layer,node)) * (ulj(layer-1,i));
      sum1=g(layer,node,i);
    } /*i*/
  } /* node */
} /*layer */

for(node=1;node<=nl[1];node++,ecumul=0){   /*P R I M E R A   C A P A*/
  for(i=1;i<=nl[2]; i++){                  /*nodos capa siguiente */
    ecumul=ecumul+ elj(2,i) * ulj(2,i) \
          * (1- ulj(2,i)) * w(2,i,node);
  }   /*i*/
  elj(1,node)=ecumul;
  for(i=0;i<=nl[0];i++){
    g(1,node,i) = elj(1,node) * ulj(0,i);
  } /*i*/
} /* node */
} /*compute_gradient */


/* ------------------------------------------------------ LAST_LAYER ----*/
void last_layer(double *ulj,double *elj,double *dj,double *g,double *costo,\
     int np,int nl[],int dirw[],int diru[],int dire[],double uant[],\
     double eant[],double duk[]){
/* en esta subrutina se calcula la derivada de J con respecto a la salida
de la red. En el caso de un PID es igual a: -e* dy/du * du/dki * dki/dw  */
int node,i;
double deltau1,deltau2,dx1,dx2,dx3,dx4,p1=1,p2=0,p3=0,p4=0,e1,e2,e3,e4;
  /*ei realmente es el error con signo contrario */
e1= *(xx1+ii+1) - yref[ii+1][0];
e2= *(xx2+ii+1) - yref[ii+1][1];
e3= *(xx3+ii+1) - yref[ii+1][2];
e4= *(xx4+ii+1) - yref[ii+1][3];
dx1=(*(xx1+ii+1)- *(xx1+ii))/(uant[0]-uant[1]);
dx2=(*(xx2+ii+1)- *(xx2+ii))/(uant[0]-uant[1]);
dx3=(*(xx3+ii+1)- *(xx3+ii))/(uant[0]-uant[1]);
dx4=(*(xx4+ii+1)- *(xx4+ii))/(uant[0]-uant[1]);

*(costo+np*nl[0])=0;                                  /* costo a cero */
/*if *(xx1+ii+1)*/
if(ii>=75){
/*  p1=1;*/
   p1=0.1;
   p2=0;
```

```
   p3=10;
   p4=0.1;
}
for(node=1;node<=nl[L];node++){
  elj(L,node)=(p1*e1*dx1 + p2*e2*dx2 + p3*e3*dx3 + p4*e4*dx4)*duk[node-1];
  for(i=0;i<=nl[L-1];i++){                          /*i -->todos los pesos*/
    g(L,node,i)= elj(L,node) * ulj(L-1,i);
  } /*i*/
} /* node */
  *(costo+np*nl[0]) += 1;
}   /* last_layer */


/*------------------------------------------------------- UPDATE_WEIGHTS ---*/
void update_weights(double *w,double *wv,double *g, double mu[],int nl[], int
dirw []){
double r,pru;
int i,node,k;
for (k=1;k<=L;k++){
  for(node=1;node<=nl[k];node++){    /*nodo de la capa actual*/
    for(i=0;i<=nl[k-1];i++){
      r=w(k,node,i) - wv(k,node,i);
      wv(k,node,i) = w(k,node,i);
      w(k,node,i)=w(k,node,i)-mu[k-1]*g(k,node,i)+alfa*r;
}/*i*/
  } /* node */
} /* k */
} /*update_weigths*/


/*------------------------------------------------------- PINTA ----------*/
void pinta(double *vectory,int tam,int n,char *buffer1){
int redondea(double valor);
int gdriver = DETECT, gmode, errorcode;
int xmax, ymax,x,y,i,j,xant,yant;
double valymax,valymin,paso,nx=0;
char buffer[30];
valymax=*vectory;
valymin=*vectory;
for(i=0;i<tam;i++){
  if (*(vectory+i)<valymin) valymin=*(vectory+i);
  if (*(vectory+i)>valymax) valymax=*(vectory+i);
} /*for*/
valymax=valymax+0.2*valymax;
initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode != grOk){
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1);
} /* if */
setcolor(getmaxcolor());
xmax = getmaxx();
ymax = getmaxy();
paso=(double)xmax/(tam-1);
line(0,0,0,ymax);
line(0,0,xmax,0);                       /*ejes*/
line(xmax,ymax,xmax,0);
line(xmax,ymax,0,ymax);
sprintf(buffer,"%g",valymax);           /*valores de parametros */
outtextxy(10,10,buffer);
```

```
sprintf(buffer,"%g",valymin);
outtextxy(10,ymax-10,buffer);
sprintf(buffer,"%g",kfinal*T);
outtextxy(xmax-35,ymax-10,buffer);
outtextxy(xmax/2,10,buffer1);

yant= *(vectory)*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
xant=0;
setlinestyle(SOLID_LINE,1,1);
setcolor(10);

for (i=1;i<tam;i++){                    /*SALIDA DE LA RED*/
  y= *(vectory+i)*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
  nx=nx+paso;
  x=redondea(nx);
  line(xant,yant,x,y);
/*    circle(x, y, 3);   */
  xant=x;
  yant=y;
} /*for */

if (n==2) {
  yant= (double)yref[0][0]*ymax/(valymin-valymax) -
(double)ymax*valymax/(valymin-valymax);
  xant=0;
  setlinestyle(SOLID_LINE,1,1);
  setcolor(11);
  nx=0;

  for (i=1;i<tam;i++){
    y= yref[i][0]*ymax/(valymin-valymax) - ymax*valymax/(valymin-valymax);
    nx=nx+paso;
    x=redondea(nx);
    line(xant,yant,x,y);
/*    circle(x, y, 3);   */
    xant=x;
    yant=y;
  } /*for */
} /*if n*/
             /* clean up */
getch();
closegraph();
/*  return 0;*/
}  /*pinta */


/*------------------------------------------------------------ REDONDEA -------*/
int redondea(double valor)
{
double a,b;
a=floor(valor);
b=valor-a;
if (valor<0){
  a=ceil(valor);
  b=a-valor;
}
if (b<=0.5)
  valor=a;
  else if(valor>0)
    valor=a+1;
  else
    valor=a-1;
```

```
return valor;
}  /*redondea */
```

E.5.2 filewg5.h

```
/* Procedimiento para guardar en un fichero el conjunto de pesos
   de la red */
void creaf(void);
void creaf(void){
FILE *fpt,*fpt1;
int i,node,k,hola;
float valor1;
double aa,AA,nl[L+1]={2,3,3,1},valor,sum1;
fpt=fopen("pesos.dat","w");
randomize();
for (k=1;k<=L;k++){
  for(node=1;node<=nl[k];node++){
    for(i=0;i<=nl[k-1];i++){
      valor =factor*(2*(double)rand()/(double)RAND_MAX-1);
      fprintf(fpt,"%g\n",valor);
    } /*i*/
  } /* node */
} /* k */
fclose(fpt);
} /* main */
```

E.5.3 guardg52.h

```
/* Esta rutina es para almacenar las variables de de la
simulación. El formato del fichero es .mat.
*/
/* diferencia con respecto a guarda2.h : se guarda la ki */

grabar(double *coma, double *est1,double *est2,double *est3,double *est4, double
*kp, double *td,double *k3);
grabar(double *coma, double *est1,double *est2,double *est3,double *est4, double
*kp, double *td,double *k3)
{
typedef struct{           /* Estructura que representa la cabecera */
       long type;        /* del fichero .mat */
       long mrows;
       long ncols;
       long imagf;
       long namleng;
}Fmatrix;

char
*var1="coma",*var2="est1",*var3="est2",*var4="est3",*var5="est4",*var6="kp",*var7
="td",*var8="k3",datos[32];
int mn,i;
FILE *fp;
Fmatrix x;
x.type=0;            /* Valores de los parametros de la cabecera  */
x.ncols=1;            /* para el fichero .mat */
x.imagf=0;
x.namleng=5;
x.mrows=kfinal;
```

```
printf("fichero de resultados, plis: ");
scanf("%s",datos);
if((fp=fopen(datos,"wb"))==NULL){
  system("cls");
  printf("Error al abrir el fichero de resultados.");
  exit(0);
}
fwrite(&x,sizeof(Fmatrix),1,fp);

fwrite(var1,sizeof(char),x.namleng,fp);   /*comando*/
mn=x.mrows*x.ncols;
fwrite(coma,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /*variable x1 */
fwrite(var2,sizeof(char),x.namleng,fp);
fwrite(est1,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /*variable x2 */
fwrite(var3,sizeof(char),x.namleng,fp);
fwrite(est2,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /*variable x3 */
fwrite(var4,sizeof(char),x.namleng,fp);
fwrite(est3,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /*variable x4 */
fwrite(var5,sizeof(char),x.namleng,fp);
fwrite(est4,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /* kp   */
fwrite(var6,sizeof(char),x.namleng,fp);
fwrite(kp,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /*  td */
fwrite(var7,sizeof(char),x.namleng,fp);
fwrite(td,sizeof(double),mn,fp);

fwrite(&x,sizeof(Fmatrix),1,fp);           /* k3 */
fwrite(var8,sizeof(char),x.namleng,fp);
fwrite(k3,sizeof(double),mn,fp);


fclose(fp);
return(0);
}  /*main*/
```

E.5.4 gruasim.m

```
% Este script implementa la simulación de la grúa en lazo
% abierto empleando la función ode23.
%
clear;
global F ml mc l g M
F=1;
mc=10;
ml=5;
l=5;
g=10;
M=mc+ml;
```

```
t0=0;
tf=20;
x0=[-1,0,0,0];
tic;
[t,x]=ode23('grua',t0,tf,x0,0.000000001);
subplot(2,1,1);
plot(t,x(:,1));
subplot(2,1,2);
plot(t,x(:,3));
subplot 111
```

E.5.5 grua.m

```
function xder=grua(t,x)
%Funcion que devuelve las derivadas de primer orden
%del sistema de la grua.
  global F ml mc l M g

  xder(1)=x(2);
  xder(2)= -g*tan(x(3)) - (F+M*g*tan(x(3))+l*ml*sin(x(3))*x(4)*x(4))/
(ml*cos(x(3))^2-M);
  xder(3)=x(4);
  xder(4)= (F*cos(x(3)) + M*g*sin(x(3)) +l*ml*sin(x(3))*cos(x(3))*x(4)^2) /
(l*ml*cos(x(3))^2 - l*M);
```

E.5.6 gsl6.m

```
function [ret,x0,str,ts,xts]=gsl6(t,x,u,flag);
%
%
% Script SIMULINK que implementa la grúa controlada con el esquema de PID
% y una realimentación mediante variables de estado.
%GSL6  is the M-file description of the SIMULINK system named GSL6.
%      The block-diagram can be displayed by typing: GSL6.
%
%      SYS=GSL6(T,X,U,FLAG) returns depending on FLAG certain
%      system values given time point, T, current state vector, X,
%      and input vector, U.
%      FLAG is used to indicate the type of output to be returned in SYS.
%
%      Setting FLAG=1 causes GSL6 to return state derivatives, FLAG=2
%      discrete states, FLAG=3 system outputs and FLAG=4 next sample
%      time. For more information and other options see SFUNC.
%
%      Calling GSL6 with a FLAG of zero:
%      [SIZES]=GSL6([],[],[],0),  returns a vector, SIZES, which
%      contains the sizes of the state vector and other parameters.
%             SIZES(1) number of states
%             SIZES(2) number of discrete states
%             SIZES(3) number of outputs
%             SIZES(4) number of inputs
%             SIZES(5) number of roots (currently unsupported)
%             SIZES(6) direct feedthrough flag
%             SIZES(7) number of sample times
%
```

```
%      For the definition of other parameters in SIZES, see SFUNC.
%      See also, TRIM, LINMOD, LINSIM, EULER, RK23, RK45, ADAMS, GEAR.

% Note: This M-file is only used for saving graphical information;
%       after the model is loaded into memory an internal model
%       representation is used.

% the system will take on the name of this mfile:
sys = mfilename;
new_system(sys)
simver(1.3)
if (0 == (nargin + nargout))
     set_param(sys,'Location',[336,113,1013,598])
     open_system(sys)
end;
set_param(sys,'algorithm',     'RK-45')
set_param(sys,'Start time',    '0.0')
set_param(sys,'Stop time',     '30')
set_param(sys,'Min step size', '0.0001')
set_param(sys,'Max step size', '0.1')
set_param(sys,'Relative error','1e-3')
set_param(sys,'Return vars',   '')
set_param(sys,'AssignWideVectorLines','on');

add_block('built-in/Sum',[sys,'/','S1'])
set_param([sys,'/','S1'],...
             'inputs','+-',...
             'position',[235,45,260,125])

add_block('built-in/Sum',[sys,'/','S'])
set_param([sys,'/','S'],...
             'inputs','+-',...
             'position',[115,50,135,70])

add_block('built-in/Constant',[sys,'/','referencia'])
set_param([sys,'/','referencia'],...
             'position',[50,45,70,65])


%     Subsystem  'PD'.

new_system([sys,'/','PD'])
set_param([sys,'/','PD'],'Location',[4,85,366,329])

add_block('built-in/Sum',[sys,'/','PD/Sum'])
set_param([sys,'/','PD/Sum'],...
             'inputs','+++',...
             'position',[245,57,265,93])

add_block('built-in/Gain',[sys,'/','PD/D'])
set_param([sys,'/','PD/D'],...
             'Gain','D',...
             'position',[95,129,115,151])

add_block('built-in/Gain',[sys,'/','PD/Proportional'])
set_param([sys,'/','PD/Proportional'],...
             'Gain','P',...
             'position',[120,13,140,37])

add_block('built-in/Transfer Fcn',[sys,'/','PD/Integral'])
set_param([sys,'/','PD/Integral'],...
             'Numerator','[I]',...
```

```
             'Denominator','[1 0]',...
             'position',[110,57,145,93])

add_block('built-in/Derivative',[sys,'/','PD/Derivative'])
set_param([sys,'/','PD/Derivative'],...
             'position',[150,128,190,152])

add_block('built-in/Outport',[sys,'/','PD/Out_1'])
set_param([sys,'/','PD/Out_1'],...
             'position',[290,65,310,85])

add_block('built-in/Inport',[sys,'/','PD/In_1'])
set_param([sys,'/','PD/In_1'],...
             'position',[25,65,45,85])
add_line([sys,'/','PD'],[120,140;145,140])
add_line([sys,'/','PD'],[270,75;285,75])
add_line([sys,'/','PD'],[50,75;105,75])
add_line([sys,'/','PD'],[65,75;65,140;90,140])
add_line([sys,'/','PD'],[80,75;80,25;115,25])
add_line([sys,'/','PD'],[195,140;215,140;215,85;240,85])
add_line([sys,'/','PD'],[150,75;240,75])
add_line([sys,'/','PD'],[145,25;210,25;210,65;240,65])
set_param([sys,'/','PD'],...
             'Mask Display','PID',...
             'Mask Type','PID Controller',...
             'Mask Dialogue','Enter expressions for proportional, integral, and
derivative terms.\nP+I/s+Ds|Proportional:|Integral|Derivative:')
set_param([sys,'/','PD'],...
             'Mask Translate','P=@1; I=@2; D=@3;')
set_param([sys,'/','PD'],...
             'Mask Help','This block implements a PID controller where
parameters are entered for the Proportional, Integral and Derivative terms.
Unmask this block to see how it works. The derivative term is implemented using a
true derivative block.')
set_param([sys,'/','PD'],...
             'Mask Entries','10\/0\/10\/')


%     Finished composite block 'PD'.

set_param([sys,'/','PD'],...
             'position',[155,44,195,76])

add_block('built-in/State-Space',[sys,'/',['Matrix',13,'Gain']])
set_param([sys,'/',['Matrix',13,'Gain']],...
             'A','[]',...
             'B','[]',...
             'C','[]',...
             'D','K',...
             'Mask Display','K',...
             'Mask Type','Matrix Gain',...
             'Mask Dialogue','Matrix Gain.|Gain matrix:',...
             'Mask Translate','K = @1;')
set_param([sys,'/',['Matrix',13,'Gain']],...
             'Mask Help','Multiplies input vector by entered matrix to produce
output vector (y=Au).',...
             'Mask Entries','[0 -96 0]\/',...
             'position',[545,156,570,184])

add_block('built-in/Mux',[sys,'/','Mux'])
set_param([sys,'/','Mux'],...
             'inputs','3',...
```

```
             'position',[485,154,515,186])


%     Subsystem  'x3'.

new_system([sys,'/','x3'])
set_param([sys,'/','x3'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','x3/x'])
set_param([sys,'/','x3/x'],...
             'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['x3/S-function',13,'M-file which
plots',13,'lines',13,'']])
set_param([sys,'/',['x3/S-function',13,'M-file which
plots',13,'lines',13,'']],...
             'function name','sfuny',...
             'parameters','ax, color,dt',...
             'position',[130,55,180,75])
add_line([sys,'/','x3'],[90,65;125,65])
set_param([sys,'/','x3'],...
             'Mask
Display','plot(0,0,100,100,[90,10,10,10,90,90,10],[65,65,90,40,40,90,90],[90,78,6
9,54,40,31,25,10],[77,60,48,46,56,75,81,84])',...
             'Mask Type','Graph scope.')
set_param([sys,'/','x3'],...
             'Mask Dialogue','Graph scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Time range:|y-min:|y-max:|Line type (rgbw-:*).
Seperate each plot by ''/'':')
set_param([sys,'/','x3'],...
             'Mask Translate','color = @4; ax = [0, @1, @2, @3]; dt = -1;')
set_param([sys,'/','x3'],...
             'Mask Help','This block plots to the MATLAB graph window and can be
used as an improved version of the Scope block. Look at the m-file sfuny.m to see
how it works. This block can take scalar or vector input signal.')
set_param([sys,'/','x3'],...
             'Mask Entries','30\/-0.07\/0.07\/''y-/g--/c-./w:/m*/ro/b+''\/')


%     Finished composite block 'x3'.

set_param([sys,'/','x3'],...
             'position',[555,71,585,109])


%     Subsystem  'grua1'.

new_system([sys,'/','grua1'])
set_param([sys,'/','grua1'],'Location',[371,133,936,458])

add_block('built-in/Integrator',[sys,'/','grua1/Integrator3'])
set_param([sys,'/','grua1/Integrator3'],...
             'Initial','x0(3)',...
             'position',[420,190,440,210])

add_block('built-in/Integrator',[sys,'/','grua1/Integrator1'])
set_param([sys,'/','grua1/Integrator1'],...
             'Initial','x0(4)',...
             'position',[325,190,345,210])

add_block('built-in/Fcn',[sys,'/','grua1/Fcn1'])
set_param([sys,'/','grua1/Fcn1'],...
```

```
       'Expr','(u[1]*cos(u[2])+(ml+mc)*10*sin(u[2])+l*ml*u[3]*u[3]*sin(u[2])*cos(
u[2]))/(l*ml*cos(u[2])*cos(u[2])-l*(ml+mc))',...
              'position',[195,190,235,210])

add_block('built-in/Inport',[sys,'/','grua1/in_1'])
set_param([sys,'/','grua1/in_1'],...
              'position',[50,175,70,195])

add_block('built-in/Fcn',[sys,'/','grua1/Fcn'])
set_param([sys,'/','grua1/Fcn'],...
              'Expr','-10*sin(u[1])/cos(u[1]) - l*u[2]/cos(u[1])',...
              'position',[210,65,250,85])

add_block('built-in/Integrator',[sys,'/','grua1/Integrator'])
set_param([sys,'/','grua1/Integrator'],...
              'Initial','x0(2)',...
              'position',[305,65,325,85])

add_block('built-in/Integrator',[sys,'/','grua1/Integrator2'])
set_param([sys,'/','grua1/Integrator2'],...
              'Initial','x0(1)',...
              'position',[395,65,415,85])

add_block('built-in/Mux',[sys,'/','grua1/Mux'])
set_param([sys,'/','grua1/Mux'],...
              'inputs','2',...
              'position',[135,56,165,89])

add_block('built-in/Mux',[sys,'/','grua1/Mux1'])
set_param([sys,'/','grua1/Mux1'],...
              'inputs','3',...
              'position',[135,176,165,224])

add_block('built-in/Outport',[sys,'/','grua1/out_2_'])
set_param([sys,'/','grua1/out_2_'],...
              'Port','2',...
              'position',[550,125,570,145])

add_block('built-in/Outport',[sys,'/','grua1/out_1'])
set_param([sys,'/','grua1/out_1'],...
              'position',[545,65,565,85])

add_block('built-in/Outport',[sys,'/','grua1/out_3'])
set_param([sys,'/','grua1/out_3'],...
              'Port','3',...
              'position',[550,190,570,210])

add_block('built-in/Outport',[sys,'/','grua1/out_4'])
set_param([sys,'/','grua1/out_4'],...
              'Port','4',...
              'position',[550,265,570,285])
add_line([sys,'/','grua1'],[445,200;545,200])
add_line([sys,'/','grua1'],[460,200;460,120;515,120;515,20;105,20;105,65;130,65])
add_line([sys,'/','grua1'],[240,200;320,200])
add_line([sys,'/','grua1'],[295,200;295,140;110,140;110,80;130,80])
add_line([sys,'/','grua1'],[75,185;130,185])
add_line([sys,'/','grua1'],[170,75;205,75])
add_line([sys,'/','grua1'],[255,75;300,75])
add_line([sys,'/','grua1'],[330,75;390,75])
add_line([sys,'/','grua1'],[170,200;190,200])
add_line([sys,'/','grua1'],[420,75;540,75])
```

```
add_line([sys,'/','grua1'],[350,200;355,200;355,250;105,250;105,215;130,215])
add_line([sys,'/','grua1'],[350,200;415,200])
add_line([sys,'/','grua1'],[445,200;450,200;450,265;90,265;90,200;130,200])
add_line([sys,'/','grua1'],[355,75;355,135;545,135])
add_line([sys,'/','grua1'],[385,200;385,275;545,275])
set_param([sys,'/','grua1'],...
             'Mask Display','GRUA1',...
             'Mask Type','gruapru',...
             'Mask Dialogue','Introuduzca las constantes del sistema.|Masa de la
grua:|Masa de la carga:|Longitud de la cuerda:|Estado inicial (x0=[x10 x20 x30
x40]):')
set_param([sys,'/','grua1'],...
             'Mask Translate','mc=@1;ml=@2;l=@3;x0=@4;',...
             'Mask Help','La diferencia de este sitema es que no esta hecho a
partir de una función s.',...
             'Mask Entries','10\/5\/5\/[0 0 0 0]\/')


%     Finished composite block 'grua1'.

set_param([sys,'/','grua1'],...
             'position',[315,60,375,110])


%     Subsystem  'x1'.

new_system([sys,'/','x1'])
set_param([sys,'/','x1'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','x1/x'])
set_param([sys,'/','x1/x'],...
             'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['x1/S-function',13,'M-file which
plots',13,'lines',13,'']])
set_param([sys,'/',['x1/S-function',13,'M-file which
plots',13,'lines',13,'']],...
             'function name','sfuny',...
             'parameters','ax, color,dt',...
             'position',[130,55,180,75])
add_line([sys,'/','x1'],[90,65;125,65])
set_param([sys,'/','x1'],...
             'Mask
Display','plot(0,0,100,100,[90,10,10,10,90,90,10],[65,65,90,40,40,90,90],[90,78,6
9,54,40,31,25,10],[77,60,48,46,56,75,81,84])',...
             'Mask Type','Graph scope.')
set_param([sys,'/','x1'],...
             'Mask Dialogue','Graph scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Time range:|y-min:|y-max:|Line type (rgbw-:*).
Seperate each plot by ''/'':')
set_param([sys,'/','x1'],...
             'Mask Translate','color = @4; ax = [0, @1, @2, @3]; dt = -1;')
set_param([sys,'/','x1'],...
             'Mask Help','This block plots to the MATLAB graph window and can be
used as an improved version of the Scope block. Look at the m-file sfuny.m to see
how it works. This block can take scalar or vector input signal.')
set_param([sys,'/','x1'],...
             'Mask Entries','30\/0\/2\/''y-/g--/c-./w:/m*/ro/b+''\/')


%     Finished composite block 'x1'.
```

```
set_param([sys,'/','x1'],...
             'position',[495,51,525,89])

add_block('built-in/To Workspace',[sys,'/','To Workspace'])
set_param([sys,'/','To Workspace'],...
             'orientation',1,...
             'mat-name','x',...
             'position',[547,365,593,395])

add_block('built-in/Mux',[sys,'/','Mux1'])
set_param([sys,'/','Mux1'],...
             'inputs','2',...
             'position',[515,286,545,319])

add_block('built-in/To Workspace',[sys,'/','To Workspace1'])
set_param([sys,'/','To Workspace1'],...
             'mat-name','t',...
             'position',[120,247,170,263])

add_block('built-in/Clock',[sys,'/','Clock'])
set_param([sys,'/','Clock'],...
             'position',[30,245,50,265])


%     Subsystem  'Comando'.

new_system([sys,'/','Comando'])
set_param([sys,'/','Comando'],'Location',[0,59,274,252])

add_block('built-in/S-Function',[sys,'/',['Comando/S-function',13,'M-file which
plots',13,'lines',13,'']])
set_param([sys,'/',['Comando/S-function',13,'M-file which
plots',13,'lines',13,'']],...
             'function name','sfuny',...
             'parameters','ax, color,dt',...
             'position',[130,55,180,75])

add_block('built-in/Inport',[sys,'/','Comando/x'])
set_param([sys,'/','Comando/x'],...
             'position',[65,55,85,75])
add_line([sys,'/','Comando'],[90,65;125,65])
set_param([sys,'/','Comando'],...
             'Mask
Display','plot(0,0,100,100,[90,10,10,10,90,90,10],[65,65,90,40,40,90,90],[90,78,6
9,54,40,31,25,10],[77,60,48,46,56,75,81,84])',...
             'Mask Type','Graph scope.')
set_param([sys,'/','Comando'],...
             'Mask Dialogue','Graph scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Time range:|y-min:|y-max:|Line type (rgbw-:*).
Seperate each plot by ''/'':')
set_param([sys,'/','Comando'],...
             'Mask Translate','color = @4; ax = [0, @1, @2, @3]; dt = -1;')
set_param([sys,'/','Comando'],...
             'Mask Help','This block plots to the MATLAB graph window and can be
used as an improved version of the Scope block. Look at the m-file sfuny.m to see
how it works. This block can take scalar or vector input signal.')
set_param([sys,'/','Comando'],...
             'Mask Entries','30\/-25\/25\/''y-/g--/c-./w:/m*/ro/b+''\/')


%     Finished composite block 'Comando'.
```

```
set_param([sys,'/','Comando'],...
             'position',[320,326,350,364])

add_block('built-in/To Workspace',[sys,'/','To Workspace2'])
set_param([sys,'/','To Workspace2'],...
             'mat-name','F',...
             'position',[320,275,370,305])

add_block('built-in/Note',[sys,'/',['CONTROL DE LA GRUA: Control LQR para  x2 x3
y x4 y control PID para',13,' x1. Es igual añadir accion der. en el PID que poner
una ganancia sobre x2 en el LQR. ',13,' Por lo tanto es como si se tuviera un
control LQR para x1 x2 y x3 y un control P para x1. ']])
set_param([sys,'/',['CONTROL DE LA GRUA: Control LQR para  x2 x3 y x4 y control
PID para',13,' x1. Es igual añadir accion der. en el PID que poner una ganancia
sobre x2 en el LQR. ',13,' Por lo tanto es como si se tuviera un control LQR para
x1 x2 y x3 y un control P para x1. ']],...
             'position',[351,405,365,411])
add_line(sys,[75,55;110,55])
add_line(sys,[140,60;150,60])
add_line(sys,[380,70;490,70])
add_line(sys,[395,70;395,185;100,175;110,65])
add_line(sys,[380,90;550,90])
add_line(sys,[380,80;445,80;445,160;480,160])
add_line(sys,[380,100;410,100;410,180;480,180])
add_line(sys,[425,90;425,170;480,170])
add_line(sys,[520,170;540,170])
add_line(sys,[575,170;605,170;605,230;200,230;200,105;230,105])
add_line(sys,[200,60;230,65])
add_line(sys,[265,85;310,85])
add_line(sys,[520,170;520,280;480,280;480,310;510,310])
add_line(sys,[395,185;395,295;510,295])
add_line(sys,[550,305;570,305;570,360])
add_line(sys,[55,255;115,255])
add_line(sys,[280,85;280,345;315,345])
add_line(sys,[280,290;315,290])

drawnow

% Return any arguments.
if (nargin | nargout)
       % Must use feval here to access system in memory
       if (nargin > 3)
              if (flag == 0)
                     eval(['[ret,x0,str,ts,xts]=',sys,'(t,x,u,flag);'])
              else
                     eval(['ret =', sys,'(t,x,u,flag);'])
              end
       else
              [ret,x0,str,ts,xts] = feval(sys);
       end
else
       drawnow % Flash up the model and execute load callback
end
```

E.5.7 adapt.m

```
% fichero para realizar un control adptativo de la grua
% utiliza un algoritmo basado en la linealización de la planta
%
```

```
% Necesita la funcion grua1.m
clc;

disp('-----------------------------------------------------------------');
disp('------- ADAPTIVE PERTURBANCE CONTROL FOR THE CRANE ------------')
disp('-----------------------------------------------------------------');
disp(' ');disp(' ');disp(' ');
disp('             wait ...');
clear all;

lineal1
[Ad,Bd]=c2d(A,B,0.1);

global u ml mc l g M
% u entrada a la grua
ml=5;mc=10;M=ml+mc;l=5;g=10;  %INICIALIZACION de variables de simulacion
paso=0.03;

x1ini=-1.5;
alfa=100;                     %INICIALIZACION de parametros y variables
ro=0.9;                       % del algoritmo de identificacion
P=alfa*eye(5);
x=[x1ini,0,0,0]';
est=[x1ini 0 0 0];
k1=10;k2=10;k3=-96;k4=0;
u=1;
z=[x;u];
load FyG3
%F=abs(rand(4));
%G=abs(rand(4,1));
%F=0.0001*rand(4);
%G=0.0001*rand(4,1);
%save FyG3 F G
th=[F';G'];
                              % PESOS FUNCION DE COSTO
Q=[190 0 0 0;0 30.0 0 0;0 0 16000 0;0 0 0 50];
R=0.0001;
                               % B U C L E   DE   C O N T R O L
kfin=200;
t0=0;tf=paso;
for k=1:kfin,
  gam=1/(z'*P*z+ro);
  P=P-gam*P*z*z'*P;
                                     % CALCULAR VARIABLES EN K+1
%F=Ad;
%G=Bd;
   u= -(inv(R+G'*Q*G))*G'*Q*F*x;
%   K=dlqr(F,G,Q,R);
%   u=-K*x;

  [t,x]=ode23('grua1',t0,tf,est(k,:));
  est(k+1,:)= x(length(t),:);          % actualizamos variables
%  if (max(max(abs(est(:,1)))) > 5)
%      figure(1);
%      plot(est(:,1));
%      error(' ¡¡ El sistema no converge. Abortado control!!');
%      figure(1);
%  end % if
  t0=tf;tf=tf+paso;                    % de simulacion
  x=x(length(t),:)';                   % vector x del algoritmo

                                  % Actualizar vector de parametros
```

```
  for i=1:4,
    th(:,i)=th(:,i)+ gam*P*z*(x(i)-z'*th(:,i));
    e(k,i)=x(i)-z'*th(:,i);
  end
  z=[x;u];                  % Actualiza z F y G
  F=th(1:4,1:4)';
  G=th(5,:)';
end
t1=0:paso:paso*kfin;
%F=th(1:4,1:4);
%F=F';
%G=th(5,:)';
C=eye(4);
D=zeros(4,1);
figure(4)
plot(e(:,1));
hold on;
plot(e(:,2),'--');
plot(e(:,3),'-.');
plot(e(:,4),'.');
ylabel('Modeling error');
xlabel('stage');
title(' Modeling error on x1 (-), x2 (--), x3 (-.) and x4 (..)');
hold off;
figure(5);
plot(t1,est(:,1));
title(' x1 evolution');
xlabel(' Time (sec.)');
ylabel(' x1 (m.)');
grid;
figure(6);
plot(t1,est(:,3));
xlabel(' Time (sec.)');
ylabel(' x3 (rad.)');
grid;
title(' x3 evolution');
figure(1);
figure(2);
figure(3);
```

E.5.8 grua1.m

```
function xder=grua1(t,x)
%
%Funcion que devuelve las derivadas de primer orden
%del sistema de la grua.
%
  global u ml mc l M g

  xder(1)=x(2);
  xder(2)= -g*tan(x(3)) - (u+M*g*tan(x(3))+l*ml*sin(x(3))*x(4)*x(4))/
(ml*cos(x(3))^2-M);
  xder(3)=x(4);
  xder(4)= (u*cos(x(3)) + M*g*sin(x(3)) +l*ml*sin(x(3))*cos(x(3))*x(4)^2) /
(l*ml*cos(x(3))^2 - l*M);
```

E.6 CONTROL NAGPC

E.6.1 gpcnn5.m

```
% En este script se combina el gpc con redes neuronales para
% la sintonia del horizonte de prediccion.
% Es necesario el fichero mpesos.mat con los pesos iniciales
% de la red y un fichero con los datos de la planta.
%
%
% FUNCIONES NECESARIAS: OFFLINE.M, ONLINE.M

clear all;
global k nb na w DU1 u DA DB Y G2 DUant F N NU B lambda y T A B;

%PROGRAMA :GPCNN5.M
% DATOS:
file2='planta6';    % D A T O S   P A R A   E L   G P C
load planta6;
na=length(A)-1;     % Grado del polinomio A
nb=length(B)-1;     % Grado del polinomio B
               % Horizonte salida.
NU=1;
Nmax=10;
N=Nmax;
Nup=0;
Ndown=10;
consig=1;
nfin=300;
w=consig*ones(nfin+Nmax,1);    % Secuencia de referencia.
t=0:T:(nfin+Nmax-1)*T;
input(' lambda?: ');
vel=input(' Vel ?:');
P1=1;       % peso del error en la salida para la funcion de costo.
P2=0.0;       % peso del parámetro N en la función de costo.
lr=[2.25*vel,1.5*vel,vel];
coma(1)=0.2;

DA=delta(A);  % Valor de DA. Emplea la funci¢n "delta.m" creada por m¡.
DB=delta(B);  % Se emplea al final de cada ciclo para obtener la salida.
DU=zeros(NU,1); % DU  contiene los incrementos en los comandos
DUant=zeros(nb+1,1); %este vector contiene DU(t), DU(t-1), ..., DU(t-nb)
u=zeros(nb+1,1);% tiene los comandos anteriores u(t-1),...,u(t-(nb+1))
y=zeros(na+1,1);  % Vector  contiene las salidas anteriores del sistema:
%y=ones(na+1,1);  % Vector  contiene las salidas anteriores del sistema:

load mpesos

yy(1)= 0;
yy(2)=0;
eant(1)=w(1)-y(1);
kp(1)=0.2;
dw1=zeros(size(w1));
dw2=zeros(size(w2));
dw3=zeros(size(w3));
db1=0*b1;db2=0*b2;db3=0*b3;
tic
for ii=2:nfin,
```

```
    offline(N);
    [sal,com]=online2(ii,N);
    coma(ii)=com;
    yy(ii+1)=sal;
    eant(2)=eant(1); eant(1)=w(ii+1)-yy(ii+1);
    if (abs(yy(ii+1)) > 1e4),
       disp('La evolucion no es la correcta');
       return;
    end;
    p=[eant(1) ;                  % INICIAR
    (eant(1)-eant(2))/T];
%calculo de du/dn. La aproximación que se hace es que du/dn en N=Nj,
%es igual a (u(N=Nj+1)-u(N=Nj-1))/N
    DU1bis=DU1;ubis=u;G2bis=G2;DUantbis=DUant;Fbis=F;ybis=y;
    offline(N+1);
    [sal1,com1]=online2(ii,N+1);
     offline(N-1);
    [sal2,com2]=online2(ii,N-1);
    dun=(com1-com2)/2;
    DU1=DU1bis;u=ubis;G2=G2bis;DUant=DUantbis;F=Fbis;y=ybis;
    e=-P1*eant(1)*((yy(ii+1)-yy(ii))/(coma(ii)-coma(ii-1))) * dun;
    for epoca=1:2,
      a1=purelin(w1*p,b1);
      a2=logsig(w2*a1,b2);
      a3=purelin(w3*a2,b3);
      d3=deltalin(a3,e);
      d2=deltalog(a2,d3,w3);
      d1=deltalog(a1,d2,w2);
      [dw1,db1]=learnbpm(p,d1,lr(1),0.001,dw1,db1);
      w1=w1-dw1;b1=b1-db1;
      [dw2,db2]=learnbpm(a1,d2,lr(2),0.001,dw2,db2);
      w2=w2-dw2;b2=b2-db2;
      [dw3,db3]=learnbpm(a2,d3,lr(3),0.001,dw3,db3);
      w3=w3-dw3;b3=b3-db3;
    end
    Nreal(ii)=a3(1);
    N=a3(1);

    if N>Nup, Nup=N; end;
    if N<Ndown, Ndown=N; end;
    coef=inv([Ndown 1;Nup 1])*[2;Nmax];  %coefic. de la transf. lineal en N
    N=coef(1)*N+coef(2);
    N= ceil(N);
    if N > Nmax,
      N=Nmax;
    elseif  N <= 1,
      N=2;
    end
    valorn(ii)=N;
end % for
toc
hold off;
figure(1);
subplot(2,1,1);
ejeymin=min(min(yy),min(w(1:nfin)));
ejeymax=max(max(yy),max(w(1:nfin)));
ejeymin=ejeymin-0.1*sign(ejeymin)*min(ejeymin);
ejeymax=ejeymax+0.1*sign(ejeymax)*max(ejeymax);
plot(t(1:nfin),yy(2:nfin+1),'r');
axis([0 t(nfin+1) ejeymin ejeymax]);
hold on
plot(t(1:nfin), w(1:nfin));
```

```
title(' ROJO --> SALIDA, AMARILLO --> CONSIGNA');
xlabel('Time (sec.)');
hold off;
subplot(2,1,2);
plot(t(1:nfin),valorn);
axis([0 t(nfin+1) 0 Nmax]);
xlabel('Time (sec.)');
subplot(111);
figure(2);
plot(t(1:nfin),Nreal,'*');
xlabel('Time (sec.)');
```

E.6.2 trayect.m

```
% Fichero para generar trayectorias aleatorias con el tiempo.
%
% Se debe introducir cuantos segundos de trayectoria se desean
% generar. Este intervalo se subdividira en varias porciones. En cada
% uno de estos puntos se genera una variable "y" aleatoria. El script
% esta preparado para unir estos puntos generados de forma aleatoria
% mediante interpolacion cubica suave.
%
%
% Los datos de entrada son:
%     - Numero de segundos que se desea que tenga la trayectoria.
%     - Periodo de muestreo de la trayectoria (la inversa da el
%       numero de puntos del vector resultante).
%     - Numero de cambios de direccion que se desean. Este numero
%       sera la cantidad de puntos aleatorios que se generan para
%       luego unirlos. Estos puntos seran equidistantes a lo largo
%       de la trayectoria.
%  Por ejemplo es posible generar 20 segundos de trayectoria,
%  con 4 cambios en la direccion y con un espaciado entre los
%  puntos del vector de salida de 0.1. El resultado sera un
%  vector con 200 puntos y que presentara 4 cambios de direccion
%  suaves.
%  La salida es un vector w con los valores del eje de abcisas, y
%  un vector t con los valores del eje de coordenadas.
%
disp('========== GENERADOR DE TRAYECTORIAS ====================');
clear
%tfinal=20;
%period=0.1;
%ncambios=5;
tfinal=input(' - Tiempo final: ');
ncambios=input(' - Cuantos cambios desea hacer en la direccion: ');
period=input(' - Separacion entre los puntos del vector: ');
distc=tfinal/ncambios;  %distancia en tiempo entre cada cambio
distc=(fix(distc*100))/100;
distc=fix(distc/period)*period;
pporc=distc/period;          %numero de puntos en cada tramo;
pporc=pporc*ones(ncambios,1);
pporc(ncambios)=tfinal/period-sum(pporc(1:ncambios-1));
                    %si no pueden tener todos losmismos puntos, el
                    % ultimo tramo toma los que falten para
                    % completar el tiempo deseado.
xc(ncambios+1)=tfinal;
xc=[0:distc:tfinal];
x0=0;y0=0;
```

```
j=1;
for i=1:(length(xc)-1),
   x1=xc(i+1);
   y1=rand;
   dete=(4*x1*x1*x1*x0+4*x1*x0*x0*x0-6*x0*x0*x1*x1-x1*x1*x1*x1- ...
          x0*x0*x0*x0);
  a=(2*x0-2*x1)*(y0-y1)/dete;
  b=(3*x0*x0-3*x1*x1)*(y1-y0)/dete;
  c=(6*x0*x0*x1-6*x1*x1*x0)*(y0-y1)/dete;
  d=(y1*(4*x1*x0*x0*x0-3*x0*x0*x1*x1-x0*x0*x0*x0)+y0*(4*x1*x1*x1*x0- ...
    3*x1*x1*x0*x0-x1*x1*x1*x1))/dete;
   x=x0;
   for j=j:j+pporc(i),
     x=x+period;
     w(j)=a*x*x*x + b*x*x + c*x + d;
   end
   x0=x1;
   y0=y1;
end
w(length(w))=[];
t=0:period:(length(w)-1)*period;
t=t';
w=w';
figure(1);
plot(t,w);
hold off;
```

REFERENCIAS

[Aco89] L. Acosta , Memoria de Licenciatura : *Diseño y Realización de una Herramienta de CAD Basada en Programación Dinámica, Para Procesos de Control Deterministas y Estocásticos*, Facultad de química, Sección de Físicas, Univ, de La Laguna, 1989.

[Aco91] L. Acosta, Tesis doctoral: *Concepción y Desarrollo de Métodos, Basados en Lógica Heurística, para la Reducción Espacial/Temporal de la Programación Dinámica en Procesos de Control Óptimo Continuos/Discretos Deterministas y Estocásticos*, Dpto de Física Fundamental y Experimental, Facultad de Fisica, Univ. de La Laguna, 1991.

[Ale86] ALECOP, Servosistemas SAD-100, *Manuales de prácticas y del profesor*, 1986.

[Ast97] K. J. Aström y B. Wittenmark. *Computer-Controlled Systems.* Prentice Hall, 1997.

[And90] C.W. Anderson y W. Thomas Miller. *A Challenging Set of Control Problems.* En *Neural Networks for Control*, pp. 403-426, W.T. Miller, III, R. S. Sutton and P.J. Werbos, editors. Cambridge, MIT Press, 1990.

[Bar90] A.G. Barto. *Connectionist Learning for Control*. En Neural Networks for Control, pp. 5-58, W.T. Miller, III, R. S. Sutton and P.J. Werbos, editors. Cambridge, MIT Press, 1990.

[Bit90] R.R. Bitmead, M. Gevers y V. Wertz. *Adaptive Optimal Control. The Thinking Man's GPC*. Prentice-Hall, 1990.

[Bou92] F. Boustany y B. D'Andre'a-Novel. *Adaptive Control of non-completely controlled mechanical system using dynamic feedback linearization and estimation design*. International Journal of Adaptive Control and Signal Processing, Vol. 6, pp. 589-610, 1992.

[Bru80] P.M. Bruijn, L.J. Bootsma y H.B. Verbruggen. *Predictive Control using Impulse Response Models*. IFAC Symposium on Digital Computer Applications to Process Control, Dusseldorf , 1980.

[Cam93] E.F. Camacho. *Constrained Generalized Predictive Control*. IEEE Trans. on Automatic Control, Vol. 38, No. 2, pp. 1512-1516, 1993.

[Cam94] E. F. Camacho y M. Berenguel. *Application of Generalized Predictive Control to a Solar Power Plant*. En Advances in Model-Based Predictive Control, pp. 483-497. D.W. Clarke, editor. Oxford University Press, 1994.

[Cam95] E.F. Camacho y C. Bordons. *Model Predictive Control in the Process Industry*. Springer-Verlag. 1995.

[Che96] C. Cheng, y C. Chen. *Controller Design for an Overhead Crane System with Uncertainty,* Control Engineering Practice, Vol. 4, No 5, pp. 645-653, 1996.

[Chi92] R.Y. Chiang y M.G. Safonov. *Robust Control Toolbox User's Guide*. The Mathworks, Inc., 1992

[Cla79] D.W. Clarke y P.J. Gawthrop. *Self-tuning Control*. Proc. IEE, Vol 126, pp. 633-640, 1979.

[Cla84] D. W. Clarke. *Self-tuning Control of Non-minimum Phase Systems.* Automatica, Vol 20, pp. 501-517, 1984.

[Cla85] D. Clarke. *Introduction to Self-tuning Controllers*. En Self-tuning and Adaptive Control: Theory and Applications, pp. 36-71. C. J. Harris y S. A. Billings, editores. London: Peter Peregrinus LTD, 1985.

[Cla87a] Clarke D., C. Mohtadi C. y P. Stuffs. *Generalized Predictive Control. Part.I,* Automatica, Vol 23, No 2, pp. 137-148, 1987.

[Cla87b] Clarke D., C. Mohtadi C. y P. Stuffs. *Generalized Predictive Control. Part. II*, Automatica, Vol 23, No 2, pp. 149-160, 1987.

[Cla89] D. Clarke, and C. Mohtadi. *Properties of generalized predictive control*. Automatica, Vol. 25, No, 6, pp. 859-875, 1989.

[Cla91] D. Clarke y R. Scattolini. *Constrained Receding-horizon Predictive Control*. IEE Proceedings-D, Vol. 138, No 4, pp. 347-354, 1991.

[Cla94] D. Clarke. *Advances in Model-Based Predictive Control*. En Advances in Model-Based Predictive Control, pp. 3-21. D.W. Clarke, editor. Oxford University Press, 1994.

[Cut80] C.R. Cutler y B.C. Ramaker. *Dynamic Matrix Control - A Computer Control Algorithm*. Automatic Control Conference, San Francisco, USA,1980.

[Dek85] R.M.C. De Keyser y A.R. Van Cuawenberghe. *Extended Prediction Sellf-adaptive Control*. IFAC Simposium on Identification and System Parameter Estimation, York, UK, pp. 1317-1322, 1985.

[Dem91] H. Demircioglu y P.J. Gawthrop. *Continuos-time Generalized Predictive Control (CGPC).* Automatica, Vol 27, No 1, pp. 55-74, 1991.

[Dem92] H. Demircioglu y P.J. Gawthrop. *Multivariable Continuos-time Generalized Predictive Control (MCGPC).* Automatica, Vol 28, No 4, pp. 697-713, 1992.

[Dem92] H. Demircioglu y D.W. Clarke. *CGPC with guaranteed stability properties*. IEE Proceedings-D, Vol. 139, No 4, pp. 371-380, 1992.

[Dem93a] H. Demircioglu y D.W. Clarke. *Generalised Predictive Control with End-point State Weighting*. IEE Proceedings-D, Vol. 140, No 4, pp. 275-281, 1993.

[Dem93b] H. Demuth y M. Beale. *Neural Network Toolbox User's Guide*. The Mathworks, Inc., 1993.

[Doy81] J. Doyle y G. Stein. *Multivariable Feedback Design: Concepts for a Classical/Modern Synthesys*. IEEE Trans. on Automatic Control., Vol. 26, No. 1,pp. 4-16, 1981.

[Dum94] D. Dummur y P. Boucher. *Predictive Control Application in the Machine Tool Field*. En Advances in Model-Based Predictive Control, pp. 471-482. D.W. Clarke, editor. Oxford University Press, 1994.

[Flet87] R. Fletcher. *Practical Methods of Optimization*. John Wiley ans Sons, 1987.

[Fri96] B. Friedland. *Advanced Control Systems Design*. Prentice-Hall, New Jersey, 1996.

[Fuk92] T. Fukuda y T. Shibata. Theory and Applications of Neural Networks for Industrial Control Systems, IEEE Trans. on Industrial Electronics, Vol. 39, No 6, pp. 472-489, 1992.

[Goo84] G.C. Goodwin y K. S. Sin. *Adaptive Filtering Prediction and Control*. Prentice-Hall, NJ, 1984.

[Gos97] J.R. Gossner, B. Kouvaritakis y J.A. Rossiter. *Stable Generalised Predictive Control with Constraints and Bounded Disturbances*. Automatica, Vol. 33, No 4, pp. 551-568, 1997.

[Gree95] M. Green D.J.N. Limebeer. *Linear Robust Control*. Prencice Hall, New Jersey, 1995.

[Gus96] T. Gustaffson. *On the design and implementation of a rotary crane controller*. European Journal of Control, Vol. 2, No 2, pp. 166-175, 1996.

[Har89] K. Hara, T. Yamamoto, A. Kobayashi y M. Okamoto (1989). *Jib crane control to suppress load swing*. International Journal of Systems Science, Vol. 20, pp. 715-731.

[Ich91] J. L. Ichaso, Memoria de Licenciatura: *Diseño y Realización de un Herramienta de Análisis/Síntesis de Controladores Predictivos*, Dpto de Física Fundamental y Experimental, Universidad de La Laguna, 1991.

[Jac85] O.L.R. Jacobs. *Introduction to adaptive control*. En Self- tuning and Adaptive Control: theory and applications, pp. 1-31. C.J. Harris y S.A. Billings, editores. Peter Peregrinus, IEE Control Engineering Series; 15, London, 1985.

[Hus93] D.R. Hush y B.G. Horne. *Progress in Supervised Neural Networks.* IEEE Signal Processing Magazine, January, 1993.

[Kay80] T. Kaylath. *Linear Systems.* Prentice-Hall, NJ, 1980.

[Kat97] M.R. Katebi y M.A. Johnson. *Predictive Control for Large-Scale Systems.* Automatica, Vol. 33 No 3, pp. 421-525, 1997.

[Kou92] B. Kouvaritakis, J.A. Rossiter y A.O.T. Chang. *Stable Generalised Predictive Control: an algorithm with guaranteed stability.* IEE Proceedings-D, Vol. 139, No 4, pp. 349-362, 1992.

[Kou93] B. Kouvaritakis, J.A. Rossiter. *Multivariable Stable Generalised Predictive Control*. IEE Proceedings-D, Vol. 140, No 5, pp. 364-372, 1993.

[Kwa] H. Kwakernaak. H_∞ - *Optimization*. Fisrt IFAC Symposium on Design Methods of Control Systems. Vol.1, 1991.

[Law74] C.L. Lawson y R.J. Hanson. *Solving Least Squares Problems.* Prentice-Hall, 1974.

[Lee94] J.H. Lee, M. Morari y C. García. *State-space Interpretation of Model Predictive Control*. Automatica, Vol. 30, No. 4, pp. 707-717, 1994.

[Lem85] J.M. Lemos y E. Mosca. *A Multipredictor-based LQ self-tuning Controller*. Symposium on Identification and System Parameter Estimation, York, UK, pp. 137-141, 1985.

[Lin94] D.A. Linkens y M. Mahfouf. *Generalised Predictive Control (GPC) in Clinical Anaesthesia*. En Advances in Model-Based Predictive Control, pp. 429-445. D.W. Clarke, editor. Oxford University Press, 1994.

[Lin96] D. A. Linkens y H.O. Nyongesa. *Learning Systems in Intelligent Control: an Appraisal of Fuzzy, Neural and Genetic Algorithm Control Applications*. IEE Proc. Control Theory and Applications, Vol. 143, No. 4, 1996.

[Lju87a] L. Ljung. *System Identification*. Theoru for the User. Prentice-Hall, 1987

[Lju87b] L. Ljung y T. Söderström. *Theory and Practice of Recursive Identification*. MIT Press, 1987.

[Lju88] L. Ljung. *System Identification Toolbox for use with Matlab*. The Mathworks, Inc., 1988.

[Lun88] Lunze J. *Robust Multivariable Feedback Control*. Prentice-Hall, UK, 1989.

[Mah92] M. Mahfouf, D.A. Linkens, A.J. Asbury, W.M. Gray y J.E. Peacock. *Generalised Predictive Control (GPC) in the Operating Theatre*. IEE Proceedings-D, Vol. 139, No 4, pp. 404-420, 1992.

[Mat92] MATLAB. Users Guide. The Mathworks, Inc., 1992.

[Men80] G. Menga y E. Mosca. *MUSMAR: Multivariable Adaptive Regulators Based on Multistep Cost Functionals*. En Advances in Control, D.G. Lainiotis y N.S. Tzannes (eds.), pp. 334-341. Reidel, Dordrecht, Holland, 1980.

[Men93] J. A. Méndez, Memoria de Licenciatura: *Implementación en tiempo real de algoritmos de control óptimo deterministas y estocásticos para un sistema*

de posición y velocidad.. Dpto. de Física Fundamental y Experimental, Universidad de La Laguna, 1993.

[Mor89] M. Morari y E. Zafiriou. *Robust Process Control*. Prentice - Hall, NJ, 1989.

[Mor95] L. Moreno, L. Acosta, J.A. Méndez, A. Hamilton, J.D. Piñeiro, J. Merino, J. Sánchez, y R. Aguilar. *Experiments on a DC motor based system for a digital control course*. International Journal of Electrical Engineering Education, Vol. 32, No2, 1995.

[Mor96] L. Moreno, L. Acosta, J.A. Méndez, A. Hamilton, G.N. Marichal, J.D. Piñeiro and J. L. Sánchez. *Stochastic Optimal Controllers for a DC Servo Motor: Applicability and Performance*. Control Engineering Practice, Vol. 4, No. 6, pp. 757-764, 1996.

[Mos90] E. Mosca, J.M. Lemos y J. Zhang. Stabilizing *I/O Receding Horizon Control*. IEEE Conference on Decision and Control, 1990.

[Mos95] E. Mosca. Optimal, *Predicitive and Adaptive Control*. Prentice Hall. USA, (1995).

[Nar90] K.S. Narendra. *Adaptive Control Using Neural Networks.* En Neural Networks for Control, pp. 115-142, W.T. Miller, III, R. S. Sutton and P.J. Werbos, editors. Cambridge, MIT Press, 1990.

[Nar91] K.S. Narendra. *Intelligent Control*. IEEE Control Systems Magazine, pp. 39-40, January, 1991.

[Ngu90] D. H. Nguyen y B. Widrow. *Neural Networks for Self-learning Control Systems.* IEEE Control Systems Magazine, Vol. 10, No. 3, pp. 18-23.

[Nic94] G. de Nicolao y R. Scattolini. *Stability and Output Terminal Constraints in Predictive Control*. En Advances in Model-Based Predictive Control, pp. 105-121. D.W. Clarke, editor. Oxford University Press, 1994.

[Nic96] G. de Nicolao, R. Scattolini y G. Sala. *An adaptive Predictive Regulator with Input Saturation*. Automatica, Vol. 32 No 4, pp. 597-601, 1996.

[Oga93] K. Ogata. Ingeniería de Control Moderna. Prentice Hall, 1993.

[Pat82] R. V. Patel y N. Munro. *Multivariable System Theory and Design*. Pergamon Press, 1982.

[Par94] T. Parisini y R. Zoppoli. *Neural Networks for Feedback Feedforward Nonlinear Control Systems.* IEEE Trans. on Neural Networks, Vol. 5, No. 3, pp.436-449, 1994.

[Pet84] V. Peterka. *Predictor -based Self-tuning Control*. Automatica, Vol 20, No 1, pp. 39-50, 1984.

[Pro63] A. I. Propoi. *Use of LP Methods for Synthesising Sample-Data Automatic Systems.* Automn Remote Control, Vol 24, 1963.

[Raw93] J.B. Rawlings y K.R. Muske. *The stability of Constrained Receding Horizon Control*. IEEE Trans. on Automatic Control, Vol. 38, No 10, 1993.

[Ric76] J. Richalet, A. Rault, J.L. Testud y J. Papon. *Algorithmic Control of Industrial Processes*. In 4th IFAC Symposium on Identification and System Parameter Estimation. Tbilisi URSS, 1976.

[Ric78] J. Richalet, A. Rault, J.L. Testud y J. Papon. *Model Predictive Heuristic Control: Application to Industrial Processes*. Automatica, 14 (2),pp. 413-428, 1978.

[Ric87] J. Richalet, S. Abu el Ata-Doss, C. Arber, H.B. Kuntze, A. Jacubash y W. Schill. *Predictive Functional Control. Application to Fast and Accurate Robots.* Proc. 10th IFAC Congress, Munich, 1987.

[Ric93] J. Richalet. *Industrial Applications of Model Based Predictive Control*. Automatica, Vol. 29, No 5, pp. 1251-1274, 1993.

[Rob91] B.D. Robinson y D.W. Clarke. *Robustness effects of a prefilter in generalised predictive control*. IEE Proc.-D, Vol. 138, No. 1, 1991.

[Ros91] J.A. Rossiter, B. Kouvaritakis y R.M. Dunnet. *Application of Generalised Predictive Control to a boiler-turbine unit for electricity generation*. IEE Proceedings-D, Vol. 138, No 1, pp. 59-67, 1991.

[Ros93] J.A. Rossiter y B. Kouvaritakis. *Constrained Stable Generalised Predictive Control*. IEE Proceedings-D, Vol. 140, No 4, pp. 243-254, 1993.

[Ros95] J.A. Rossiter, B. Kouvaritakis y J.R. Gossner. *Feasibility and Stability Results for Constrained Stable Generalised Predictive Control*. Automatica, Vol. 31, No 6, pp. 863-877, 1995.

[Ros96] J.A. Rossiter y B. Kouvaritakis. *Guaranteeing Feasibility in Constrained Stable Generalised Predictive Control*. IEE Proceedings-D, Vol. 143, No 5, pp. 463-469, 1996.

[Saf80] M. G. Safonov. *Stability and Robustness of Multivariable Feedback Systems.* MIT Press, 1980.

[Saf82] M.G. Safonov, *Stability Margins of Diagonally Perturbed Multivariable Feedback Systems.* IEE Proc. D, Vol, 129, No 2, pp. 252-255, 1982.

[Sak85] Y. Sakawa, and A. Nakazumi. *Modeling and control of rotary crane. Journal of Dynamic Systems,* Measurement, and Control, Vol 107, pp. 200-206, 1985.

[San90] A. C. Anderson. *Applications of Neural Networks in Robotics and Automation for Manufacturing*. En Neural Networks for Control, pp. 5-58, W.T. Miller, III, R. S. Sutton and P.J. Werbos, editors. Cambridge, MIT Press, 1990.

[Sim92] *SIMULINK Users Guide*. The Mathworks, Inc. 1992.

[Smi58] OJM Smith. *Feedback Control Systems.* McGraw-Hill, NY, 1958.

[Sod89] T. Söderström y P.Stoica. *System Identification*, Prentice-Hall, 1989.

[Sod91] T. Söderström, P.Stoica y B. Friedlander. *An Indirect Prediction Error Method for System Identification*. Automatica, Vol. 27, No. 1, pp. 183-188, 1991.

[Soe92] R. Söeterboek. *Predictive Control. A unified Approach*. Prentice-Hall, 1992.

[Str88] D.R. Strip. *Swing-free transport of suspend objects: a general treatment*. IEEE Trans. on Robotics and Automation, Vol 5, pp. 234-236, 1988.

[Tan92] J. Tanomaru y S. Omatu. *Process Control by on-line trained Neural Networks*, IEEE Trans. on Industrial Electronics, Vol. 39, No 6, 1992.

[Vri94] R.A.J. de Vries y H.B. Verbruggen. *Multivariable Unified Predictive Control*. En Advances in Model-Based Predictive Control, pp. 84-102. D.W. Clarke, editor. Oxford University Press, 1994.

[Wel79] P.E. Wellstead, D. Prager y P.Zanker. *Pole assignment self-tuning regulator*. Proc. IEE, Vol. 126, pp. 781-787.

[Wel93] P.E. Wellstead y M.B. Zarrop. *Self-Tuning Systems. Control and Signal Processing*. John Wiley ,1991.

[Wer91] P.J. Werbos. *An overview of Neural Networks for Control*. IEEE Control Systems, pp. 40-41, January, 1991.

[Yds84] B.E. Ydstie. *Extended Horizon Adaptive Control*. 9th IFAC World Congress, Budapest, Hungary, 1984.

[Yoo94] T.W. Yoon y D.W. Clarke. *Towards Robust Adaptive Predictive Control*. En Advances in Model-Based Predictive Control, pp. 402-414. D.W. Clarke, editor. Oxford University Press, 1994.

[Zhe95] A. Zheng y M. Morari. *Stability of Model Predictive Control with Mixed Constraints*. IEEE Transaction on Automatic Control, Vol. 40(10), pp. 1818-1823, 1995.