TESIS DEFENDIDA POR

Rodrigo Abraján Guerrero

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Luis Alejandro Márquez Martínez

Director del Comité

Dr. Joaquín Álvarez Gallegos

Miembro del Comité

Dr. Carlos Alberto Brizuela Rodríguez

Miembro del Comité

M. C. José Ricardo Cuesta García

Miembro del Comité

Dr. Roberto Conte Galván

Coordinador del programa de posgrado en Electrónica y Telecomunicaciones

Dr. David Hilario Covarrubias Rosales

Director de Estudios de Posgrado

14 de julio de 2011

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS EN ELECTRÓNICA Y TELECOMUNICACIONES

SINTONIZACIÓN ÓPTIMA DE CONTROLADORES, APLICACIÓN A UN PÉNDULO INVERTIDO

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

RODRIGO ABRAJÁN GUERRERO

Ensenada, Baja California, México, julio de 2011

RESUMEN de la tesis de **RODRIGO ABRAJÁN GUERRERO**, presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en ELECTRÓNICA Y TELECOMUNICACIONES con orientación en INSTRUMENTACIÓN Y CONTROL. Ensenada, Baja California, julio de 2011.

SINTONIZACIÓN ÓPTIMA DE CONTROLADORES, APLICACIÓN A UN PÉNDULO INVERTIDO

Resumen aprobado por:

Dr. Luis Alejandro Márquez Martínez

Director de Tesis

Este trabajo de tesis presenta una metodología para sintonizar parámetros de controladores, con el fin de mejorar su desempeño. Se plantea el problema de sintonización como un problema de optimización, para lo cual se emplean y comparan tres algoritmos heurísticos de optimización. Como funciones objetivo a minimizar, se proponen algunos índices de desempeño y el tiempo de establecimiento.

Se aplicó esta metodología para sintonizar tres controladores distintos − control PID, control por ubicación de polos y control *backstepping* − aplicados a un sistema mecánico subactuado: un péndulo invertido. Se presentan los resultados numéricos obtenidos en simulación y los obtenidos experimentalmente en laboratorio.

Para implementar esta metodología de manera experimental en laboratorio, se desarrolló una plataforma en Matlab, la cual permite realizar una sintonización tanto en simulación como de manera experimental de manera automática (sin requerir intervención permanente del usuario).

Palabras clave: sintonización de controladores, algoritmos heurísticos, índices de desempeño, tiempo de establecimiento, péndulo invertido.

ABSTRACT of the thesis presented by **RODRIGO ABRAJÁN GUERRERO**, in partial fulfillment of the requirements of the degree of MASTER OF SCIENCE in ELECTRONICS AND TELECOMMUNICATIONS with orientation in INSTRUMENTATION AND CONTROL. Ensenada, Baja California, july 2011.

OPTIMAL CONTROLLER TUNING, APPLICATION TO AN INVERTED PENDULUM

This dissertation presents a methodology to tune controllers' parameters, to improve their performance. The tuning problem is stated as an optimization problem. For that, three heuristic optimization algorithms are employed and compared. Several performance indices and the settling time are used as objective functions.

This methodology was applied to tune three different controllers − PID, pole placement, and backstepping − applied to an underactuated mechanical system: an inverted pendulum. Numerical results obtained by simulation and experimentally in laboratory are shown and analysed.

To implement this methodology, a testing platform was developed in Matlab, which allows us to do automatic tuning through simulation as well as experimentally (without permanent user intervention).

Keywords: controller tuning, heuristic optimization, performance indices, settling time, inverted pendulum.

A mi familia ....

Agradecimientos

A mi querida esposa, Ivetth, por su apoyo incondicional.

A mis padres, Antonio y Carmen, quienes siempre me han apoyado en todos mis proyectos.

A mis hermanos, Antonio y Oscar, con quienes sé que siempre puedo contar.

A mi director de tesis, Dr. Alejandro Márquez Martínez, por la paciencia, apoyo y guía para realizar este trabajo.

A los miembros del comité de tesis por sus valiosos comentarios.

A los compañeros del cubo 250 (Abimael, David e Irak), por compartir esta experiencia que fue la maestría.

A los Tulícanos: Abimael, David, Irak, Lupe, Raúl, Aracelí, Hirata, Lalo, Rigo (Villavicencio), Karen, Perla, Rigo (Jáuregui), Brenda, Rebeca, Elizabeth, Aarón, Pedro, Christian, Edwin, Lily, Anita, Tampíco, Paúl, Galaviz y Cheche.

A todo el personal del departamento de Electrónica y Telecomunicaciones.

Al CICESE por permitirme estudiar en esta institución.

Al CONACyT por su apoyo económico a través de la beca de maestría y proyecto de investigación CB200662026.

Contenido

Página

Resumen en español **i**

Resumen en inglés **ii**

Dedicatoria **iii**

Agradecimientos **iv**

Contenido **v**

Lista de Figuras **viii**

Lista de Tablas **xii**

I. INTRODUCCIÓN **1**
- I.1 Antecedentes y motivación . . . 1
- I.2 Objetivos . . . 3
  - I.2.1 Objetivo general . . . 3
  - I.2.2 Objetivos específicos . . . 3
- I.3 Descripción de la tesis . . . 3

II. Modelo del péndulo invertido **5**
- II.1 Configuración . . . 5
- II.2 Ecuaciones de movimiento . . . 5
- II.3 Representación en espacio de estados . . . 8
- II.4 Puntos de equilibrio . . . 9

III. Control del péndulo invertido **12**
- III.1 Control PID . . . 12
  - III.1.1 Diseño del controlador . . . 15
- III.2 Ubicación de polos . . . 17
  - III.2.1 Procedimiento para determinar la matriz K . . . 19
  - III.2.2 Diseño del controlador . . . 20
- III.3 *Backstepping* o control en retroceso . . . 22
  - III.3.1 Diseño del controlador . . . 25
  - III.3.2 Grados de libertad extra para el control . . . 27
- III.4 Observador con estimación de perturbaciones . . . 28
  - III.4.1 Estructura del observador . . . 29
  - III.4.2 Realización para el péndulo . . . 30

Contenido (continuación)

Página

IV. Metodología **31**
IV.1 Función objetivo . . . 31
IV.1.1 Índices de desempeño . . . 31
IV.1.2 Tiempo de establecimiento, T_s . . . 33
IV.2 Algoritmos de optimización . . . 34
IV.2.1 Hooke y Jeeves . . . 34
IV.2.2 Nelder y Mead (simplex) . . . 36
IV.2.3 Recocido simulado . . . 38
IV.3 Conclusiones del capítulo . . . 40

V. Plataforma de simulación y experimental **43**
V.1 Simulación . . . 43
V.1.1 Bloque: Parámetros iniciales . . . 43
V.1.2 Bloque: Genera nuevos parámetros . . . 45
V.1.3 Bloque: Evaluar función . . . 45
V.1.4 Bloque: Paro . . . 46
V.2 Experimentación . . . 46
V.2.1 Hardware . . . 46
V.2.2 Software . . . 47

VI. Resultados en simulación **51**
VI.1 PID . . . 51
VI.1.1 Hooke y Jeeves . . . 52
VI.1.2 Nelder y Mead . . . 54
VI.1.3 Recocido simulado . . . 57
VI.2 Ubicación de polos . . . 60
VI.2.1 Hooke y Jeeves . . . 61
VI.2.2 Nelder y Mead . . . 64
VI.2.3 Recocido simulado . . . 66
VI.3 *Backstepping* caso 1 . . . 69
VI.3.1 Hooke y Jeeves . . . 70
VI.3.2 Nelder y Mead . . . 72
VI.3.3 Recocido simulado . . . 74
VI.4 *Backstepping* caso 2 . . . 77
VI.4.1 Hooke y Jeeves . . . 77
VI.4.2 Nelder y Mead . . . 79
VI.4.3 Recocido simulado . . . 81

Contenido (continuación)

Página

VII. Resultados experimentales **84**
VII.1 PID 84
VII.2 Ubicación de polos 87
VII.3 *Backstepping* caso 1 89
VII.4 *Backstepping* caso 2 91

VIII. Discusión y conclusiones **93**
VIII.1 Análisis de resultados 93
VIII.1.1 Simulación 93
VIII.1.2 Experimental 95
VIII.2 Conclusiones 96
VIII.3 Trabajo a futuro 98

REFERENCIAS **99**

Apéndice **102**

A. Congreso nacional AMCA 2010 **102**

B. Programas en Matlab y C/C++ **108**
B.1 Métodos de optimización multivariable 108
B.1.1 Hooke y Jeeves 108
B.1.2 Nelder y Mead 110
B.1.3 Recocido simulado 114
B.2 funObj 116
B.2.1 Simulación 117
B.2.2 Experimento físico 120

Lista de Figuras

Figura		Página
1	Diagrama de la planta ECP 505.	6
2	Diagrama con el mecanismo invertido, configuración a utilizar.	6
3	Esquema típico para utilizar control PID.	12
4	Acción proporcional e integral.	14
5	Predicción del error.	15
6	Caracterización de respuesta al escalón $y(t)$ en lazo abierto.	16
7	Control proporcional para obtener K_u.	16
8	Sistema en lazo abierto.	18
9	Sistema en lazo cerrado con $u = -Kx$.	18
10	Tiempo de establecimiento ante una entrada tipo escalón.	33
11	Movimiento exploratorio en $\mathbb{R}^2$.	35
12	Movimiento exploratorio en $\mathbb{R}^2$.	35
13	Optimización con método Hooke-Jeeves, curvas de nivel para $\mathbb{R}^2$.	36
14	Pseudocódigo para el algoritmo de recocido simulado	41
15	Esquema de sintonización.	44
16	Archivo fuente MEX, ejemplo.cpp	49
17	Respuesta ante un escalón de magnitud 0.1745 rad en la referencia, en lazo cerrado con un control PID con parámetros Ziegler-Nichols.	52
18	Control PID con parámetros Ziegler-Nichols.	52
19	Posición angular x_1 usando PID sintonizado con Hooke y Jeeves.	54
20	Posición de la barra x_3 y señal de control usando PID sintonizado con Hooke y Jeeves.	55
21	Posición angular x_1 usando PID sintonizado con Nelder y Mead.	56
22	Posición de la barra x_3 y señal de control usando PID sintonizado con Nelder y Mead.	57

Lista de Figuras (continuación)

Figura		Página
23	Posición angular x_1 usando PID sintonizado con recocido simulado.	59
24	Posición de la barra x_3 y señal de control usando PID sintonizado con recocido simulado.	59
25	Respuesta ante un escalón de magnitud 0.1745 *rad* en la referencia, en lazo cerrado con un control por ubicación de polos, con parámetros iniciales.	60
26	Control por ubicación de polos con parámetros iniciales.	60
27	Posición angular x_1 usando ubicación de polos sintonizado con Hooke y Jeeves.	62
28	Posición de la barra x_3 y señal de control usando ubicación de polos sintonizado con Hooke y Jeeves.	63
29	Posición angular x_1 usando ubicación de polos sintonizado con Nelder y Mead.	65
30	Posición de la barra x_3 y señal de control usando ubicación de polos sintonizado con Nelder y Mead.	65
31	Posición angular x_1 usando ubicación de polos sintonizado con recocido simulado.	67
32	Posición de la barra x_3 y señal de control usando ubicación de polos sintonizado con recocido simulado.	68
33	Respuesta ante un escalón de magnitud 0.1745 *rad* en la referencia, en lazo cerrado con un control *backstepping*, con parámetros iniciales.	69
34	Control *backstepping* con parámetros iniciales.	69
35	Posición angular x_1 usando *backstepping* caso 1 sintonizado con Hooke y Jeeves.	71
36	Posición de la barra x_3 y señal de control usando *backstepping* caso 1 sintonizado con Hooke y Jeeves.	71
37	Posición angular x_1 usando *backstepping* caso 1 sintonizado con Nelder y Mead.	73

Lista de Figuras (continuación)

Figura		Página
38	Posición de la barra x_3 y señal de control usando *backstepping* caso 1 sintonizado con Nelder y Mead.	73
39	Posición angular x_1 usando *backstepping* caso 1 sintonizado con recocido simulado.	75
40	Posición de la barra x_3 y señal de control usando *backstepping* caso 1 sintonizado con recocido simulado.	76
41	Posición angular x_1 usando *backstepping* caso 2 sintonizado con Hooke y Jeeves.	78
42	Posición de la barra x_3 y señal de control usando *backstepping* caso 2 sintonizado con Hooke y Jeeves.	78
43	Posición angular x_1 usando *backstepping* caso 2 sintonizado con Nelder y Mead.	80
44	Posición de la barra x_3 y señal de control usando *backstepping* caso 2 sintonizado con Nelder y Mead.	80
45	Posición angular x_1 usando *backstepping* caso 2 sintonizado con recocido simulado.	82
46	Posición de la barra x_3 y señal de control usando *backstepping* caso 2 sintonizado con recocido simulado.	83
47	Posición angular x_1 usando PID - optimización experimental del ITAE.	85
48	Posición de la barra x_3 y señal de control usando PID - optimización experimental del ITAE.	85
49	Posición angular x_1 usando PID - optimización experimental del T_s.	86
50	Posición de la barra x_3 y señal de control usando PID - optimización experimental del T_s.	87
51	Posición angular x_1 usando ubicación de polos - optimización experimental del ITAE.	88
52	Posición de la barra x_3 y señal de control usando ubicación de polos - optimización experimental del ITAE.	88

Lista de Figuras (continuación)

Figura Página

53 Posición angular x_1 usando *backstepping* caso 1 - optimización experimental del ITAE. 89

54 Posición de la barra x_3 y señal de control usando *backstepping* caso 1 - optimización experimental del ITAE. 90

55 Estimación de perturbación del primer eslabón usando *backstepping* caso 1. 90

56 Posición angular x_1 usando *backstepping* caso 2 - sintonización experimental. 91

57 Posición de la barra x_3 y señal de control usando *backstepping* caso 2 - sintonización experimental. 92

58 Estimación de perturbación del primer eslabón usando *backstepping* caso 2. 92

59 Comparación de costo computacional con respecto al algoritmo de optimización. 94

Lista de Tablas

Tabla		Página
I	Parámetros del PID utilizando el primer método de Ziegler-Nichols.	16
II	Parámetros del PID utilizando el segundo método de Ziegler-Nichols.	17
III	Control PID sintonizado con Hooke y Jeeves.	53
IV	Costo computacional al sintonizar el control PID con Hooke y Jeeves.	53
V	Control PID sintonizado con Nelder y Mead.	55
VI	Costo computacional al sintonizar el control PID con Nelder y Mead.	56
VII	Control PID sintonizado con recocido simulado; promedio y desviación estándar.	58
VIII	Costo computacional al sintonizar el control PID con recocido simulado; promedio y desviación estándar.	58
IX	PID sintonizado con recocido simulado; T_s cercano al promedio.	58
X	Ubicación de polos sintonizado con Hooke y Jeeves.	61
XI	Costo computacional al sintonizar el control por ubicación de polos con Hooke y Jeeves.	61
XII	Ubicación de polos sintonizado con Nelder y Mead.	64
XIII	Costo computacional al sintonizar el control por ubicación de polos con Nelder y Mead.	64
XIV	Ubicación de polos sintonizado con recocido simulado; promedio y desviación estándar.	66
XV	Costo computacional al sintonizar el control por ubicación de polos con recocido simulado; promedio y desviación estándar.	66
XVI	Ubicación de polos sintonizado con recocido simulado; T_s cercano al promedio.	67
XVII	*Backstepping* caso 1 sintonizado con Hooke y Jeeves.	70
XVIII	Costo computacional al sintonizar *backstepping* caso 1 con Hooke y Jeeves.	70
XIX	*Backstepping* caso 1 sintonizado con Nelder y Mead.	72

Lista de Tablas (continuación)

Tabla		Página
XX	Costo computacional al sintonizar *backstepping* caso 1 con Nelder y Mead.	72
XXI	*Backstepping* caso 1 sintonizado con recocido simulado; promedio y desviación estándar.	74
XXII	Costo computacional al sintonizar *backstepping* caso 1 con recocido simulado; promedio y desviación estándar.	74
XXIII	*Backstepping* caso 1 sintonizado con recocido simulado; T_s cercano al promedio.	75
XXIV	*Backstepping* caso 2 sintonizado con Hooke y Jeeves.	77
XXV	Costo computacional al sintonizar *backstepping* caso 2 con Hooke y Jeeves.	77
XXVI	*Backstepping* caso 2 sintonizado con Nelder y Mead.	79
XXVII	Costo computacional al sintonizar *backstepping* caso 2 con Nelder y Mead.	79
XXVIII	*Backstepping* caso 2 sintonizado con recocido simulado; promedios y desviación estándar.	81
XXIX	Costo computacional al sintonizar *backstepping* caso 2 con recocido simulado; promedios y desviación estándar.	81
XXX	*Backstepping* caso 2 sintonizado con recocido simulado; T_s cercano al promedio.	82
XXXI	Tiempos de establecimiento, PID experimental.	87

Capítulo I

INTRODUCCIÓN

I.1 Antecedentes y motivación

Al diseñar un controlador por lo general se busca garantizar la estabilidad del sistema. Muchas veces la estabilidad puede depender de una buena elección de los parámetros o ganancias del controlador. Además, aunque un control nos garantice la estabilidad del sistema, es importante que tenga un buen desempeño; se debe definir una manera de medir el desempeño. El desempeño del controlador dependerá de la elección de los parámetros.

Un primer referente al hablar de sintonización de controladores para el control PID, es el trabajo de Ziegler y Nichols (1942) y otros que se han derivado de éste como (Chien *et al.*, 1952; Aström *et al.*, 1993; Vrancic *et al.*, 1998). Algunos más recientes utilizan técnicas como algoritmos genéticos (de Almeida *et al.*, 2005) y recocido simulado (Yachen y Yueming, 2008).

Para sistemas lineales, existen técnicas que permiten diseñar un control que garantice que las especificaciones de diseño impuestas sean satisfechas. Algunas de estas son técnicas de control óptimo (Kwakernaak y Sivan, 1972) como el control LQR, el cual requiere la minimización de una función objetivo. El inconveniente con estas técnicas es que en ocasiones no es fácil traducir las especificaciones de desempeño en una función objetivo o de costo adecuada dependiente de los parámetros del controlador.

Otra técnica, es la presentada por Boyd y Barratt (1991), donde se sintetiza un controlador minimizando una función objetivo sujeta a restricciones, utilizando técnicas de optimización convexa. Esta última metodología puede extenderse a una clase de sistemas no lineales que pueden ser transformados en un sistema lineal mediante retroalimentación del estado y un cambio de coordenadas (Brizuela Rodriguez, 1994).

Para sistemas no lineales, obtener un control o los parámetros que garanticen cierto desempeño deseado, puede ser un poco más complicado, dado que esto puede requerir resolver ecuaciones diferenciales no lineales. De aquí que surge la idea de buscar un

método que nos permita encontrar los parámetros óptimos para un controlador con una estructura ya definida.

En este trabajo se desarrolló una infraestructura, que implementa una serie de algoritmos que permiten automatizar la sintonización de los parámetros del controlador, independientemente de su estructura, tanto en simulación como de manera experimental con el sistema físico que se esté controlando.

Esta infraestructura fue utilizada para sintonizar tres controladores para el péndulo invertido ECP 505. El control del péndulo invertido es un problema académico clásico. La configuración más común es la del péndulo sobre un carro, donde el desplazamiento del carro permite balancear el péndulo. Este mecanismo apareció por primera vez en la literatura en (Roberge, 1960), y se encuentra en otros textos como (Dorf, 1967; Spong, 1998; Ogata, 2009). Sin embargo, la dinámica del péndulo ECP 505 equivale más al mecanismo bola y viga, donde la bola sería la parte actuada y no la viga.

El péndulo ECP 505 es un mecanismo formado por dos eslabones. El primer eslabón se encuentra unido a la base por una unión rotacional y en el extremo unido al segundo eslabón con una unión prismática. El segundo eslabón se utiliza para balancear el péndulo. El sistema es subactuado, cuenta con dos grados de libertad y sólo un actuador, el actuador se encuentra en la unión entre los dos eslabones (véase Figura 1).

El péndulo ECP 505 es utilizado comúnmente para probar distintos controladores para sistemas subactuados. Recientemente Wu *et al.* (2010) presentaron un trabajo donde se busca controlar un nuevo tipo de dirigibles no tripulados con una novedosa manera de propulsarse al variar de manera cíclica la flotabilidad y el ángulo de ataque de la aeronave (Purandare, 2007). En el trabajo de Wu y colegas, se muestra una simplificación del modelo de una aeronave de este tipo, donde el ángulo de ataque se controla moviendo un lastre, haciendo que su centro de gravedad cambie de posición. El modelo simplificado resulta ser equivalente a un mecanismo como el ECP 505, pero invertido verticalmente, como se verá en el siguiente capítulo.

Tomando esta idea de la equivalencia entre los modelos, se decidió usar el péndulo ECP 505 invertido verticalmente, pensando en una posible aplicación que ayude a implementar controles automáticos para aeronaves de este tipo.

I.2 Objetivos

I.2.1 Objetivo general

Mejorar el desempeño de sistemas de control mediante la sintonización óptima de los parámetros del controlador.

I.2.2 Objetivos específicos

- Definir un método de sintonización óptima de parámetros para controladores.
- Proponer criterios que permitan comparar el desempeño de los controladores al variar sus parámetros.
- Analizar la implementación de la sintonizacion de tres controladores en simulación y de manera experimental.

I.3 Descripción de la tesis

En el Capítulo II se presenta la obtención del modelo del péndulo invertido ECP 505 con la configuración que se utilizará, su representación en espacio de estados y análisis de los puntos de equilibrio.

En el Capítulo III se presentan los controladores que se sintonizarán: PID, ubicación de polos y *backstepping* o control en retroceso. También se describe el observador que se utilizará para la estimación de los estados que no se miden directamente en el mecanismo en la etapa experimental.

En el Capítulo IV se presenta la metodología que se propone para la sintonización de los controladores. Aquí se definen distintas funciones objetivo con las que se trabajará y los tres algoritmos de optimización que se utilizarán.

En el Capítulo V se describe como se hizo la implementación de esta metodología, tanto en simulación, como de manera experimental en el laboratorio.

En el Capítulo VI se muestran los resultados que se obtuvieron en simulación al implementar la metodología propuesta para la sintonización de parámetros para los distintos controladores.

En el Capítulo VII se muestran resultados experimentales obtenidos en el Laboratorio de Control con el péndulo físico.

Por último, en el Capítulo VIII se presenta un análisis de los resultados obtenidos y conclusiones generales de este trabajo, así como el trabajo a futuro.

Capítulo II

Modelo del péndulo invertido

En este capítulo se presenta el desarrollo para obtener las ecuaciones de movimiento del péndulo ECP 505, considerando la configuración que se utilizará; es decir, con el mecanismo invertido, con respecto a su posición normal de operación. También se obtiene la representación en espacio de estados del sistema, ya que con ésta se harán las simulaciones e implementación de los controladores en los capítulos siguientes. Por último se obtienen los puntos de equilibrio y se analiza su estabilidad.

II.1 Configuración

En la Figura 1 se muestra un diagrama de la planta ECP 505. Típicamente esta planta se utiliza como se muestra en dicha figura; sin embargo, en este trabajo se utiliza el sistema invertido verticalmente y se ajusta el mecanismo para que el centro de gravedad cg_1 coincida con el pivote. Una vez invertido podemos representar el sistema con el diagrama mostrado en la Figura 2.

II.2 Ecuaciones de movimiento

Para obtener las ecuaciones dinámicas del sistema emplearemos las ecuaciones de movimiento de Lagrange, como se presenta en (Kelly y Santibáñez, 2003).

El lagrangiano ($\mathcal{L}$) se define como la diferencia de la energía cinética ($\mathcal{K}$) y la potencial ($\mathcal{U}$):

$$\mathcal{L}(q(t), \dot{q}(t)) = \mathcal{K}(q(t), \dot{q}(t)) - \mathcal{U}(q(t)).$$

Las ecuaciones de movimiento de Lagrange están dadas por:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right] - \frac{\partial \mathcal{L}}{\partial q_i} = \mathcal{F}_i, \tag{1}$$

donde q_i es la *i-ésima* coordenada generalizada y $\mathcal{F}_i$ la fuerza generalizada asociada.

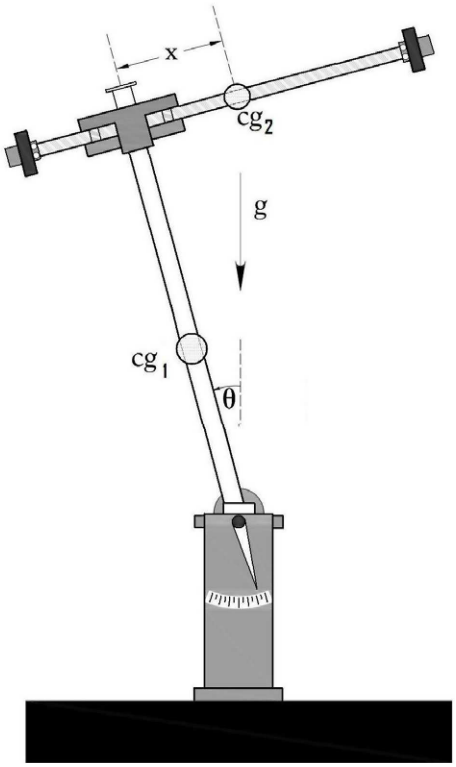


Figura 1: Diagrama de la planta ECP 505.

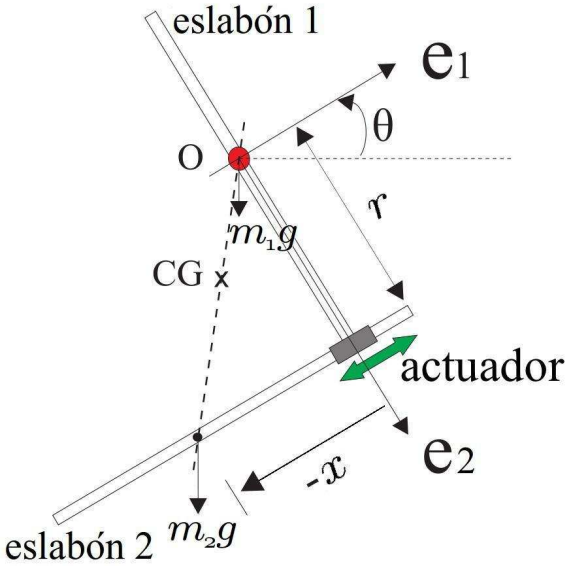


Figura 2: Diagrama con el mecanismo invertido, configuración a utilizar.

Para el péndulo invertido utilizaremos el lagrangiano como se propone en (Wu *et al.*, 2010), con el cual se representa el modelo reducido de un dirigible, equivalente al diagrama de la Figura 2.

$$\mathcal{L} = \frac{1}{2}J\dot{\theta}^2 + \frac{1}{2}mv_m^2 - mg(x\,\mathrm{sen}(\theta) - r\cos(\theta)).$$

La masa m_1 no aparece en el lagrangiano ya que su centro de gravedad coincide con el pivote. J es la suma des los momentos de inercia de los eslabones 1 y 2, cada uno respecto a su centro de gravedad; m es la masa del segundo eslabón, g es la gravedad, r es la distancia del pivote al segundo eslabón y v_m es la velocidad del centro de masa de m.

Escogemos $q_1 = \theta$ y $q_2 = x$; resultando lo siguiente:

$$\mathcal{L} = \frac{1}{2}J\dot{q}_1^2 + \frac{1}{2}mv_m^2 - mg(q_2\,\mathrm{sen}(q_1) - r\cos(q_1)), \tag{2}$$

donde

$$v_m^2 = \dot{\mathbf{x}}^2 + \dot{\mathbf{y}}^2,$$

con $\mathbf{x}$ y $\mathbf{y}$ definidas como sigue,

$$\mathbf{x} = \mathrm{sen}\,(q_1)\ r + \cos\,(q_1)\ q_2,$$

$$\mathbf{y} = \mathrm{sen}\,(q_1)\ q_2 - \cos\,(q_1)\ r.$$

Las derivadas con respecto al tiempo son:

$$\dot{\mathbf{x}} = \cos\,(q_1)\ \dot{q}_1\, r + \cos\,(q_1)\ \dot{q}_2 - \mathrm{sen}\,(q_1)\ \dot{q}_1\, q_2,$$

$$\dot{\mathbf{y}} = \mathrm{sen}\,(q_1)\ \dot{q}_1\, r + \mathrm{sen}\,(q_1)\ \dot{q}_2 + \cos\,(q_1)\ \dot{q}_1\, q_2.$$

Sumando el cuadrado de cada componente obtenemos v_m^2.

$$v_m^2 = \dot{q}_1^2\, r^2 + 2\,\dot{q}_1\,\dot{q}_2\, r + \dot{q}_2^2 + \dot{q}_1^2\, {q_2}^2. \tag{3}$$

Sustituyendo (3) en (2) obtenemos

$$\mathcal{L} = \frac{1}{2}\left(\dot{q}_1^2\, J + m\,\dot{q}_1^2\, r^2 + m\,\dot{q}_2^2 + m\,\dot{q}_1^2\, {q_2}^2\right) + m\,\dot{q}_1\,\dot{q}_2\, r + g\,m\ \cos\,(q_1)\ r - g\,m\ \mathrm{sen}\,(q_1)\ q_2,$$

de donde

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial q_1} = -g\,m\ \text{sen}\,(q_1)\ r - g\,m\ \cos\,(q_1)\ q_2,\\
&\frac{\partial \mathcal{L}}{\partial \dot{q}_1} = \dot{q}_1\,J + m\,\dot{q}_1\,r^2 + m\,\dot{q}_2\,r + m\,\dot{q}_1\,{q_2}^2,\\
&\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{q}_1}\right] = \ddot{q}_1\,J + m\,\ddot{q}_1\,r^2 + m\,\ddot{q}_2\,r + 2\,m\,\dot{q}_1\,q_2\,\dot{q}_2 + m\,\ddot{q}_1\,{q_2}^2,\\
&\frac{\partial \mathcal{L}}{\partial q_2} = m\,\dot{q}_1^2\,q_2 - g\,m\ \text{sen}\,(q_1)\,,\\
&\frac{\partial \mathcal{L}}{\partial \dot{q}_2} = m\,\dot{q}_1\,r + m\,\dot{q}_2,\\
&\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{q}_2}\right] = m\,\ddot{q}_1\,r + m\,\ddot{q}_2.
\end{aligned}$$

De esta forma y de acuerdo a (1), las dos ecuaciones de movimiento son:

$$\ddot{q}_1\,J + m\,\ddot{q}_1\,r^2 + m\,\ddot{q}_2\,r + g\,m\ \text{sen}\,(q_1)\ r + 2\,m\,\dot{q}_1\,q_2\,\dot{q}_2 + m\,\ddot{q}_1\,{q_2}^2 + g\,m\ \cos\,(q_1)\ q_2 = 0, \quad (4)$$

$$m\,\ddot{q}_1\,r + m\,\ddot{q}_2 - m\,\dot{q}_1^2\,q_2 + g\,m\ \text{sen}\,(q_1) = u(t). \quad (5)$$

II.3 Representación en espacio de estados

Para encontrar una representación en espacio de estados primero llevamos las ecuaciones de movimiento (4) y (5) a la forma:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = u,$$

quedando de la siguiente manera:

$$\begin{bmatrix} J + m\,r^2 + m\,{q_2}^2 & m\,r \\ m\,r & m \end{bmatrix}\begin{bmatrix}\ddot{q}_1\\ \ddot{q}_2\end{bmatrix} + \begin{bmatrix} 2\,m\,q_2\,\dot{q}_2 & 0 \\ -m\,\dot{q}_1\,q_2 & 0 \end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix} + \begin{bmatrix} g\,m\ \text{sen}\,(q_1)\ r + g\,m\ \cos\,(q_1)\ q_2 \\ g\,m\ \text{sen}\,(q_1) \end{bmatrix} = \begin{bmatrix} 0 \\ u(t) \end{bmatrix}. \quad (6)$$

De (6) obtenemos $\ddot{q}$.

$$\ddot{q} = M^{-1}(q)\big[u - C(q,\dot{q})\dot{q} - G(q)\big]. \quad (7)$$

Calculamos la inversa de M,

$$M^{-1} = \begin{bmatrix} \frac{1}{J+m\,{q_2}^2} & -\frac{r}{J+m\,{q_2}^2} \\ -\frac{r}{J+m\,{q_2}^2} & \frac{J+m\,r^2+m\,{q_2}^2}{m\,J+m^2\,{q_2}^2} \end{bmatrix},$$

y la utilizamos en (7), para obtener

$$\begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} -\frac{m\,r\,q_2\,\dot{q}_1^2+2\,m\,q_2\,\dot{q}_2\,\dot{q}_1+m\,g\,q_2\,\cos(q_1)+r\,u}{J+m\,q_2^2} \\ \frac{\left(J\,q_2+m\,q_2^3+m\,r^2\,q_2\right)\dot{q}_1^2+2\,m\,r\,q_2\,\dot{q}_2\,\dot{q}_1-\left(J+m\,q_2^2\right)g\,\operatorname{sen}(q_1)+m\,g\,r\,q_2\,\cos(q_1)+\left(\frac{J}{m}+q_2^2+r^2\right)u}{J+m\,q_2^2} \end{bmatrix}. \quad (8)$$

Con un cambio de variable definimos los estados de la siguiente manera:

$$x_1 = q_1,\ x_2 = \dot{q}_1,\ x_3 = q_2,\ x_4 = \dot{q}_2.$$

Utilizando las nuevas variables en la ecuación (8), obtenemos la representación en espacio de estados de la forma $\dot{x} = f(x, u)$:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{mrx_3x_2^2+2mx_3x_4x_2+mgx_3\cos x_1+ru}{J+mx_3^2} \\ x_4 \\ \frac{(Jx_3+mx_3^3+mr^2x_3)x_2^2+2mrx_3x_4x_2-(J+mx_3^2)g\,\operatorname{sen} x_1+mgrx_3\cos x_1+(\frac{J}{m}+x_3^2+r^2)u}{J+mx_3^2} \end{bmatrix}. \quad (9)$$

II.4 Puntos de equilibrio

Para obtener los puntos de equilibrio hacemos $u = 0$ y $\dot{x} = 0$ (o $f(x) = 0$) en la ecuación (9). De inmediato se observa que $x_2 = 0$ y $x_4 = 0$, con lo cual se obtienen las siguientes ecuaciones:

$$mgx_3\cos x_1 = 0, \quad (10)$$

$$-(J + mx_3^2)g\,\operatorname{sen} x_1 + mgrx_3\cos x_1 = 0. \quad (11)$$

Para que la ecuación (10) se satisfaga hay dos opciones: $x_3 = 0$ o $x_1 = \frac{n\pi}{2}$, con n impar.

Si tomamos $x_3 = 0$, entonces la ecuación (11) queda de la siguiente manera:

$$-Jg\,\operatorname{sen} x_1 = 0,$$

en la cual x_1 debe ser igual a $n\pi$ con n entero, para que se satisfaga.

Por otro lado si tomamos $x_1 = \frac{n\pi}{2}$, con n impar, la ecuación (11) quedaría:

$$J + mgx_3^2 = 0,$$

donde debería cumplirse que

$$x_3^2 = -\frac{J}{mg},$$

lo cual no es posible, dado que $\frac{J}{mg} > 0$.

Por lo tanto los puntos de equilibrio quedan dados por:

$$x^* = [n\pi, 0, 0, 0]^{\mathrm{T}}, \text{ para } n \text{ entero},$$

lo que físicamente corresponde a tener el eslabón 1 del péndulo paralelo a la vertical y el centro de gravedad del eslabón 2 (m) en el extremo del eslabón 1. Además si se gira esta configuración π radianes en cualquier sentido obtenemos otro equilibrio.

En particular para la configuración que se trabajará, consideraremos sólo un punto de equilibrio, para $n = 0$. Cualquier otra n par representaría el mismo punto, sólo algunas revoluciones adelante o atrás; y para n impar implicaría un giro de π radianes, lo cual no se puede alcanzar por restricciones físicas del aparato.

Estabilidad

Para analizar la estabilidad del punto de equilibrio utilizaremos el método indirecto de Lyapunov (Khalil, 2002).

Primero necesitamos linealizar el sistema (9) alrededor del punto de equilibrio $x^* = [0, 0, 0, 0]^{\mathrm{T}}$, para obtener la representación de la forma $\dot{x} = Ax + Bu$.

El sistema linealizado queda:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{J} & 0 \\ 0 & 0 & 0 & 1 \\ -g & 0 & \frac{mgr}{J} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{r}{J} \\ 0 \\ \frac{1}{m} + \frac{r^2}{J} \end{bmatrix} u. \tag{12}$$

Los valores propios de A están dados por:

$$\lambda_{1,2} = \pm\sqrt{\frac{gmr + g(m^2r^2 + 4Jm)^{1/2}}{2J}} \quad \lambda_{3,4} = \pm\sqrt{\frac{gmr - g(m^2r^2 + 4Jm)^{1/2}}{2J}}.$$

Dado que al menos λ_1 será real positivo, tenemos que el equilibrio es inestable.

Capítulo III

Control del péndulo invertido

En este capítulo se hace una descripción de los controladores utilizados: PID, ubicación de polos y *backstepping*. Para el caso de ubicación de polos y *backstepping* se detalla un poco sobre el diseño del control. Por último se agrega una sección donde se describe el observador utilizado para estimar las velocidades del sistema, ya que éstas son utilizadas en el control.

III.1 Control PID

El control PID o control proporcional-integral-derivativo es conocido de esta manera, ya que su acción de control incluye una parte proporcional, una integral y una derivativa del error. Con el error e definido como la diferencia entre una referencia y_{ref} y la salida y de la planta:

$$e = y_{ref} - y.$$

El esquema que comúnmente utiliza este control se muestra en la Figura 3.

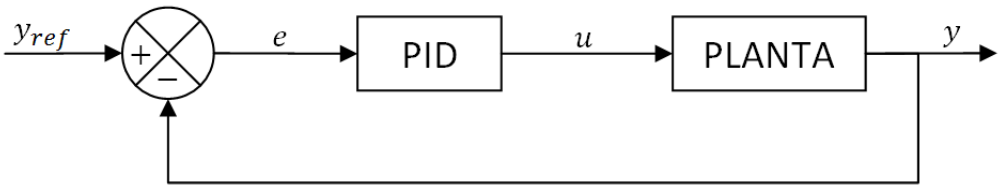


Figura 3: Esquema típico para utilizar control PID.

El control en el tiempo $u(t)$ se puede expresar de la siguiente manera:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(\tau)d\tau + K_p T_d \frac{d}{dt}e(t), \tag{13}$$

donde K_p es conocida como la ganancia proporcional, T_i el tiempo integral y T_d el tiempo derivativo.

La representación en el dominio de Laplace sería de la siguiente manera:

$$G(s) = \frac{U(s)}{E(s)} = K_p \left[1 + \frac{1}{T_i s} + T_d s\right],$$

donde $U(s)$ y $E(s)$ representan las transformadas de Laplace de $u(t)$ y $e(t)$, respectivamente.

Acción proporcional

La acción proporcional del control PID está representada por el primer término de (13). Un control con pura acción proporcional sería de la forma:

$$u_p(t) = K_p e(t),$$

donde el control sería un valor proporcional al error. Este tipo de control puede resultar en un error en estado estacionario distinto a cero, ya que en algún momento el error puede ser relativamente pequeño, de tal manera que la acción del control no sea suficiente para tener algún efecto en la planta. Este tipo de caso se puede dar en una planta mecánica con un coeficiente de fricción seca alto o con un vector de pares gravitacionales no nulo.

Acción integral

La acción integral se obtiene del segundo término de (13).

$$u_i(t) = \frac{K_p}{T_i} \int_0^t e(\tau) d\tau.$$

Esta componente del control ayuda a asegurar que en estado estacionario el error sea igual a cero. Ya que aunque el error en la salida sea muy pequeño, con el paso del tiempo este error se acumulará y la acción de control se incrementará de igual manera, hasta que la magnitud del control tenga efecto en la planta. En caso de ser un error negativo, la acción de control comenzará a decrecer hasta que el error se haga cero o cambie de signo. La componente integral ayuda a desatascar el sistema, cuando la

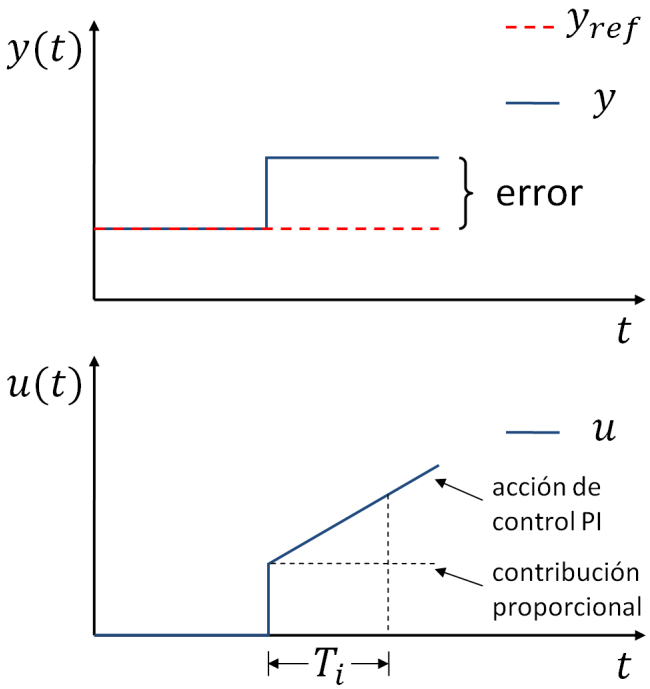


Figura 4: Acción proporcional e integral.

componente proporcional no es suficiente para modificar la salida. En la Figura 4 se puede observar el efecto de la parte integral en el control.

Acción derivativa

La acción derivativa es generada por el tercer término de (13).

$$u_d(t) = K_p T_d \frac{d}{dt} e(t).$$

Si combinamos la acción derivativa con la proporcional, podemos interpretar este control, como uno proporcional al error predicho T_d segundos adelante. Considerando una extrapolación del error con la tangente de la curva del error.

$$e(t + T_d) = e(t) + T_d \frac{d}{dt} e(t).$$

La predicción del error se muestra en la Figura 5.

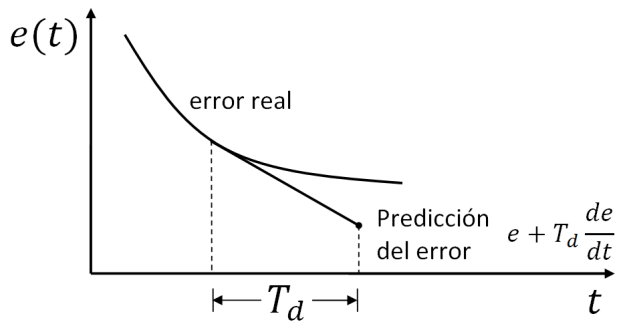


Figura 5: Predicción del error.

III.1.1 Diseño del controlador

Los métodos clásicos para determinar los parámetros del control PID fueron presentados por Ziegler y Nichols (1942). Estos métodos caracterizan la dinámica del sistema por dos parámetros y a partir de ellos obtienen los parámetros del PID. Después de estos métodos clásicos han surgido otros, algunos inspirados en ellos y otros extensiones de los mismos; algunos de los métodos se presentan en (Aström y Hägglund, 1988). A continuación se detalla un poco sobre los métodos de (Ziegler y Nichols, 1942).

Primer método de Ziegler-Nichols

En este primer método, lo que se hace es aplicar una entrada escalón al sistema en lazo abierto. La respuesta es caracterizada por dos parámetros: a y L, como se muestra en la Figura 6. Para obtener los parámetros a y L primero se determina el punto donde la pendiente de la respuesta transitoria es mayor, y se traza la recta tangente a ese punto. Los parámetros son determinados por las intersecciones con los ejes. Para poder utilizar este método la respuesta transitoria debe tener forma de "S", tal como se observa en la Figura 6. El sistema no debe tener integradores o polos dominantes complejos conjugados; de tenerlos, este método no puede utilizarse.

Una vez que se conocen a y L los parámetros del PID se obtienen como se indica en la Tabla I.

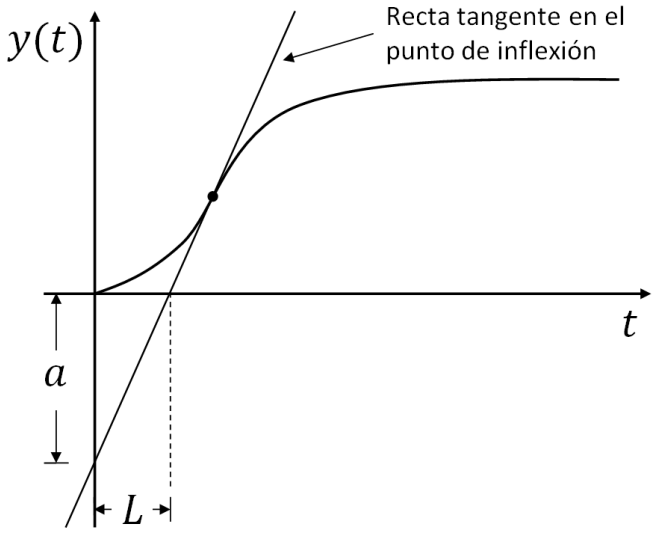


Figura 6: Caracterización de respuesta al escalón $y(t)$ en lazo abierto.

Tabla I: Parámetros del PID utilizando el primer método de Ziegler-Nichols.

$Controlador$	K_p	T_i	T_d
P	$\frac{1}{a}$		
PI	$\frac{0.9}{a}$	$3L$	
PID	$\frac{1.2}{a}$	$2L$	$\frac{L}{2}$

Segundo método de Ziegler-Nichols

Para el segundo método, se debe poner el sistema en lazo cerrado como si se utilizara el PID, pero sólo se utiliza la acción proporcional; es decir, se hace $T_i \to \infty$ y $T_d = 0$. Una vez que se tiene el control proporcional, como se muestra en la Figura 7, se debe incrementar la ganancia K_p, empezando de cero, hasta que el sistema presente oscilaciones sostenidas en la salida.

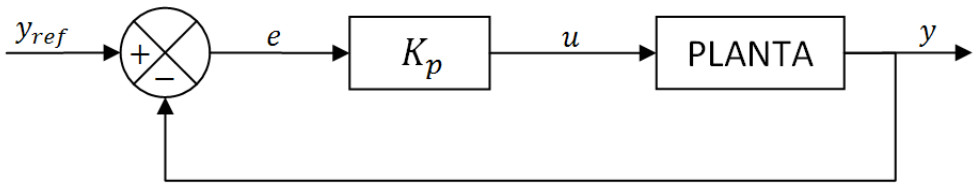


Figura 7: Control proporcional para obtener K_u.

La dinámica del sistema se caracterizará por K_u y T_u. Donde K_u es la ganancia proporcional que hace oscilar el sistema de manera sostenida, con periodo de oscilación T_u. Este método requiere llevar el sistema al límite de su estabilidad.

Una vez determinadas K_u y T_u, los parámetros del PID se obtienen como se indica en la Tabla II.

Tabla II: Parámetros del PID utilizando el segundo método de Ziegler-Nichols.

Controlador	K_p	T_i	T_d
P	$0.5K_u$		
PI	$0.4K_u$	$0.8T_u$	
PID	$0.6K_u$	$0.5T_u$	$0.12T_u$

III.2 Ubicación de polos

La técnica de ubicación de polos es conocida de esta manera ya que lo que se hace es reubicar en lazo cerrado los polos del sistema, gracias a la retroalimentación del estado con una matriz de ganancias. Para utilizar esta técnica se supone que se tiene acceso al estado completo del sistema.

Consideremos un sistema lineal

$$\dot{x} = Ax + Bu,$$

donde $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, A y B son matrices reales de dimensiones adecuadas, con una ley de control de la forma

$$u = -Kx.$$

En las figuras 8 y 9 se muestra el sistema en lazo abierto y cerrado, respectivamente.

La matriz de ganancia K debe tener dimensión $m \times n$, ya que x es un vector $n \times 1$. Con este control lo que estamos haciendo es una retroalimentación del estado. En lazo cerrado, el sistema queda

$$\dot{x} = (A - BK)x.$$

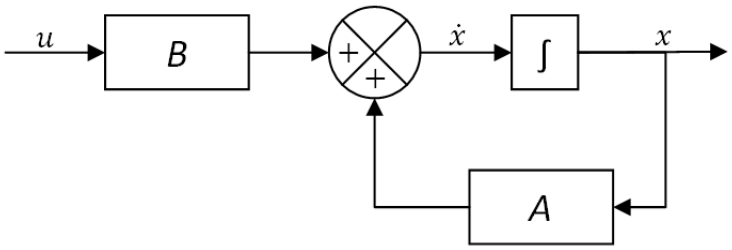


Figura 8: Sistema en lazo abierto.

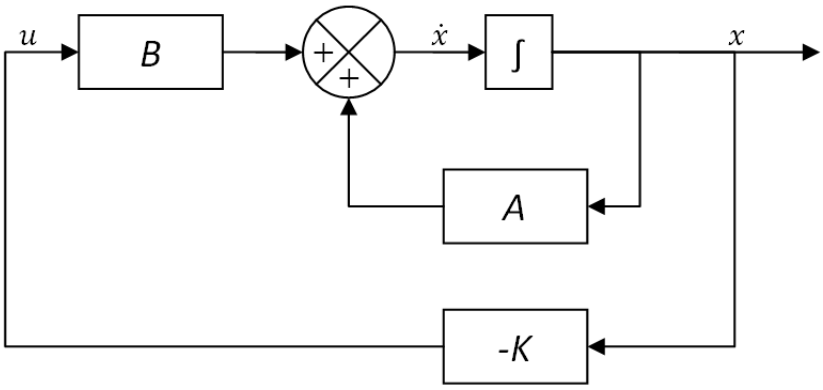


Figura 9: Sistema en lazo cerrado con $u = -Kx$.

La solución de esta ecuación está dada por:

$$x(t) = e^{(A-BK)t}x(0),$$

donde $x(0)$ es la condición inicial. La estabilidad y respuesta transitoria estan determinadas por los valores propios de la matriz $A - BK$, por lo que si encontramos una matriz K que haga que todos los valores propios de $A - BK$ tengan parte real negativa tendremos un sistema asintóticamente estable. Las trayectorias de $x(t)$ tenderán a 0 conforme t tienda a infinito. Si queremos que los polos del sistema sean: $\mu_1, \mu_2, \cdots, \mu_n$, debemos hacer que K genere el polinomio característico adecuado:

$$\left|sI - (A - BK)\right| = (s - \mu_1)(s - \mu_2)\cdots(s - \mu_n).$$

Una condición necesaria y suficiente para poder colocar los polos del sistema en lazo cerrado en cualquier posición arbitraria, es que el sistema debe ser completamente

controlable. Es decir, la matriz de controlabilidad debe ser de rango pleno, esto es:

$$\texttt{rango}\left[B|AB|\cdots|A^{n-1}B\right] = n.$$

III.2.1 Procedimiento para determinar la matriz K

El siguiente procedimiento fue tomado de (Ogata, 2009), donde se amplía un poco más sobre el uso de esta técnica. Partiendo de que se tienen definidas las matrices A y B, los pasos para ubicar los polos en los valores deseados μ_1, μ_2, $\cdots$, μ_n son los siguientes.

1. Verificar la controlabilidad del sistema, si la matriz de controlabilidad es de rango pleno continuar con los siguientes pasos.

2. Del polinomio característico de la matriz A:

$$\left|sI - A\right| = s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n,$$

   determinar los valores de a_1, a_2, $\cdots$, a_n.

3. Determinar la matriz de transformación T, que lleva el sistema a su forma canónica controlable. Si el sistema ya está dado en forma canónica controlable, entonces $T = I_{n\times n}$. La matriz de transformación T está dada por:

$$T = MW,$$

   donde M es la matriz de controlabilidad y

$$W = \begin{bmatrix} a_{n-1} & a_{n-2} & \cdots & a_1 & 1 \\ a_{n-2} & a_{n-3} & \cdots & 1 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ a_1 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

4. Utilizando los valores propios deseados (polos), escribir el polinomio característico deseado

$$(s-\mu_1)(s-\mu_2)\cdots(s-\mu_n) = s^n + \alpha_1 s^{n-1} + \cdots + \alpha_{n-1}s + \alpha_n,$$

y determinar los valores de α_1, α_2, ..., α_n.

5. La matriz de ganancias de retroalimentación K estará dada por:

$$K = \big[\alpha_n - a_n|\alpha_{n-1} - a_{n-1}|\cdots|\alpha_2 - a_2|\alpha_1 - a_1\big]T^{-1}.$$

III.2.2 Diseño del controlador

Ahora consideremos el modelo lineal del péndulo invertido dado por la ecuación (12), para el cual

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{J} & 0 \\ 0 & 0 & 0 & 1 \\ -g & 0 & \frac{mgr}{J} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ -\frac{r}{J} \\ 0 \\ \frac{1}{m} + \frac{r^2}{J} \end{bmatrix}.$$

Siguiendo los pasos descritos en la sección anterior, mostraremos el procedimiento para obtener la matriz de ganancias K que estabilice el sistema con polos μ_1, μ_2, μ_3 y μ_4. Hay que recordar que si uno de los polos deseados es complejo, su conjugado también debe estar presente en el diseño.

Empezamos por verificar la controlabilidad del sistema. Tomando las matrices A y B, la matriz de controlabilidad M queda de la siguiente manera:

$$M = \begin{bmatrix} 0 & -\frac{r}{J} & 0 & -\frac{g*(mr^2+J)}{J^2} \\ -\frac{r}{J} & 0 & -\frac{g*(mr^2+J)}{J^2} & 0 \\ 0 & \frac{1}{m} + \frac{r^2}{J} & 0 & \frac{gr(mr^2+2J)}{J^2} \\ \frac{1}{m} + \frac{r^2}{J} & 0 & \frac{gr(mr^2+2J)}{J^2} & 0 \end{bmatrix}.$$

Es verificable que: $\texttt{rango}\,[M] = 4$, por lo que el sistema es completamente controlable.

El polinomio característico de la matriz A es:

$$\big|sI - A\big| = s^4 - \frac{gmr}{J}s^2 - \frac{g^2m}{J},$$

de donde obtenemos los coeficientes

$$\begin{aligned} a_1 &= 0, \\ a_2 &= -\frac{gmr}{J}, \\ a_3 &= 0, \\ a_4 &= -\frac{g^2 m}{J}, \end{aligned}$$

para generar la matriz W.

$$W = \begin{bmatrix} 0 & -\frac{gmr}{J} & 0 & 1 \\ -\frac{gmr}{J} & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Ahora podemos obtener la matriz de transformación $T = MW$.

$$T = \begin{bmatrix} -\frac{g}{J} & 0 & -\frac{r}{J} & 0 \\ 0 & -\frac{g}{J} & 0 & -\frac{r}{J} \\ \frac{gr}{J} & 0 & \frac{1}{m} + \frac{r^2}{J} & 0 \\ 0 & \frac{gr}{J} & 0\frac{1}{m} + \frac{r^2}{J} & \end{bmatrix}.$$

La inversa de T queda:

$$T^{-1} = \begin{bmatrix} -\frac{mr^2+J}{g} & 0 & -\frac{mr}{g} & 0 \\ 0 & -\frac{mr^2+J}{g} & 0 & -\frac{mr}{g} \\ mr & 0 & m & 0 \\ 0 & mr & 0 & m \end{bmatrix}.$$

Escribimos el polinomio característico deseado, de donde obtenemos los coeficientes α_i:

$$(s-\mu_1)(s-\mu_2)(s-\mu_3)(s-\mu_4) = \ s^4 + \alpha_1 s^3 + \alpha_2 s^2 + \alpha_3 s + \alpha_4.$$

Con esto tenemos todo para poder calcular la matriz de ganancias K para cualesquier

polos deseados: μ_1, μ_2, μ_3 y μ_4.

$$K = \left[\alpha_n - a_n|\alpha_{n-1} - a_{n-1}|\cdots|\alpha_2 - a_2|\alpha_1 - a_1\right]T^{-1}.$$

III.3 *Backstepping* o control en retroceso

Backstepping o control en retroceso es un método de diseño recursivo no lineal, basado en la construcción sistemática de funciones de Lyapunov, al considerar una o algunas de las variables de estado como entradas de control virtual.

Consideremos el siguiente sistema con una forma adecuada para utilizar *backstepping*, como se muestra en (Khalil, 2002).

$$\dot{\eta} = f(\eta) + g(\eta)\xi, \tag{14}$$

$$\dot{\xi} = u, \tag{15}$$

donde $[\eta^T, \xi]^T \in \mathbb{R}^{n+1}$ es el estado, $u \in \mathbb{R}$ es la entrada de control y las funciones $f : \mathbb{D} \to \mathbb{R}^n$ y $g : \mathbb{D} \to \mathbb{R}^n$ son diferenciables un número suficiente de veces en un dominio $\mathbb{D} \subset \mathbb{R}^n$.

Consideremos una función $\phi : \mathbb{D} \to \mathbb{R}$, de manera que $\xi = \phi(\eta)$ sea un control estabilizante con $\phi(0) = 0$, es decir, el origen de

$$\dot{\eta} = f(\eta) + g(\eta)\phi(\eta)$$

es asintóticamente estable. Se supone se conoce una función de Lyapunov $V : \mathbb{D} \to \mathbb{R}$, de manera que $V(\eta)$ satisface la desigualdad

$$\frac{\partial V}{\partial \eta}\left[f(\eta) + g(\eta)\phi(\eta)\right] \leq -W(\eta), \tag{16}$$

donde $W(\eta)$ es una función definida positiva. Sumando y restando $g(\eta)\phi(\eta)$ en la parte derecha de (14), obtenemos la representación

$$\begin{aligned}\dot{\eta} &= \left[f(\eta) + g(\eta)\phi(\eta)\right] + g(\eta)\left[\xi - \phi(\eta)\right],\\ \dot{\xi} &= u.\end{aligned}$$

Haciendo el cambio de variable

$$z = \xi - \phi(\eta),$$

resulta

$$\begin{aligned}\dot{\eta} &= [f(\eta) + g(\eta)\phi(\eta)] + g(\eta)z,\\ \dot{z} &= u - \dot{\phi},\end{aligned}$$

como se supone conocemos f, g y ϕ, podemos encontrar

$$\dot{\phi} = \frac{\partial \phi}{\partial \eta}\left[f(\eta) + g(\eta)\xi\right].$$

Haciendo $v = u - \dot{\phi}$ se obtiene el sistema en cascada:

$$\begin{aligned}\dot{\eta} &= [f(\eta) + g(\eta)\phi(\eta)] + g(\eta)z,\\ \dot{z} &= v,\end{aligned}$$

donde la primer ecuación presenta un origen asintóticamente estable cuando el estado z es cero. Utilizando

$$V_c(\eta, \xi) = V(\eta) + \frac{1}{2}z^2,$$

como función candidata de Lyapunov, obtenemos

$$\dot{V}_c = \frac{\partial V}{\partial \eta}\left[f(\eta) + g(\eta)\phi(\eta)\right] + \frac{\partial V}{\partial \eta}g(\eta)z + zv \;\leq\; -W(\eta) + \frac{\partial V}{\partial \eta}g(\eta)z + zv,$$

si escogemos

$$v = -\frac{\partial V}{\partial \eta}g(\eta) - k\,z, \quad k > 0,$$

obtenemos

$$\dot{V}_c \leq -W(\eta) - k\,z^2,$$

lo cual muestra que el origen es asintóticamente estable. Sustituyendo v, z y $\dot{\phi}$ obtenemos el control:

$$u = \frac{\partial \phi}{\partial \eta}\left[f(\eta) + g(\eta)\xi\right] - \frac{\partial V}{\partial \eta}g(\eta) - k\left[\xi - \phi(\eta)\right]. \tag{17}$$

Este análisis se puede resumir en el siguiente lema.

Lema 1 *Considere el sistema* (14)-(15). *Sea* $\phi(\eta)$ *una ley de control por retroalimentación estabilizante de* (14) *con* $\phi(\eta) = 0$, *y* $V(\eta)$ *sea una función de Lyapunov que satisfaga* (16) *con alguna función definida positiva* $W(\eta)$. *Entonces, el control por retroalimentación de estados* (17) *estabiliza el origen de* (14)-(15), *con* $V(\eta) + \frac{[\xi-\phi(\eta)]^2}{2}$ *como función de Lyapunov. Si además, todas las suposiciones son válidas de manera global y* $V(\eta)$ *es radialmente desacotada, el origen será estable asintóticamente de manera global.*

Consideremos un sistema más general para utilizar la técnica de *backstepping*:

$$\dot{\eta} = f(\eta) + g(\eta)\xi, \tag{18}$$

$$\dot{\xi} = f_a(\eta, \xi) + g_a(\eta, \xi)u. \tag{19}$$

En este caso podemos utilizar la siguiente entrada:

$$u = \frac{1}{g_a(\eta, \xi)} \left[u_a - f_a(\eta, \xi)\right], \tag{20}$$

para transformar (19) y reducir la ecuación al integrador $\dot{\xi} = u_a$. De esta manera obtenemos la forma de (14)-(15). Por lo tanto, si existe una ley de control por retroalimentación de estado $\phi(\eta)$ y una función de Lyapunov $V(\eta)$ tales que las condiciones del Lema 1 se satisfacen para (18), de lo cual, junto con (20) obtenemos la ley de control:

$$\begin{aligned} u &= \phi_c(\eta, \xi) \\ &= \frac{1}{g_a(\eta, \xi)} \left\{ \frac{\partial \phi}{\partial \eta}[f(\eta) + g(\eta)\xi] - \frac{\partial V}{\partial \eta} g(\eta) - k[\xi - \phi(\eta)] - f_a(\eta, \xi) \right\}, \end{aligned} \tag{21}$$

para alguna $k > 0$, con una función de Lyapunov

$$V_c(\eta, \xi) = V(\eta) + \tfrac{1}{2}[\xi - \phi(\eta)]^2, \tag{22}$$

para el sistema (18)-(19).

III.3.1 Diseño del controlador

Ahora utilizaremos el control por *backstepping* en las primeras dos ecuaciones de nuestro sistema (9), esto es:

$$\dot{x}_1 = x_2, \tag{23}$$

$$\dot{x}_2 = -\frac{1}{J + mx_3^2}\left[mrx_3x_2^2 + 2mx_3x_4x_2 + mgx_3\cos x_1 + ru\right]. \tag{24}$$

En este caso tomamos $\eta = x_1$ y $\xi = x_2$. Los estados x_3 y x_4 sólo nos interesará ver si permanecen cerca del equilibrio.

Primero hacemos una transformación en la entrada:

$$u = -\frac{1}{r}\left[mrx_3x_2^2 + 2mx_3x_4x_2 + mgx_3cosx_1 + (J + mx_3^2)u_a\right],$$

y obtenemos un sistema de la forma (14)-(15):

$$\dot{x}_1 = x_2, \tag{25}$$

$$\dot{x}_2 = u_a. \tag{26}$$

Por inspección, comparando (23)-(24) con (14)-(15), obtenemos:

$$f(\eta) = 0,$$
$$g(\eta) = 1.$$

Para establizar (25), utilizamos la ley de control

$$x_2 = \phi(x_1) = -x_1.$$

Con función de Lyapunov

$$V(x_1) = \frac{1}{2}x_1^2,$$

y derivada

$$\dot{V}(x_1) = -x_1^2.$$

De (21) obtenemos la ley de control

$$u = -\frac{1}{r}\left[mrx_3x_2^2 + 2mx_3x_4x_2 + mgx_3cosx_1 + (J + mx_3^2)u_a\right], \tag{27}$$

donde

$$u_a = -x_1 - x_2 - k[x_1 + x_2].$$

Con (22) obtenemos la función de Lyapunov

$$V_c = \frac{1}{2}x_1^2 + \frac{1}{2}[x_1 + x_2]^2,$$

cuya derivada es

$$\dot{V}_c = -x_1^2 - k[x_1 + x_2]^2 \leq 0.$$

El sistema en lazo cerrado queda de la siguiente manera:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ u_a \\ x_4 \\ f_4(x) \end{bmatrix}, \tag{28}$$

donde

$$f_4(x) = \frac{(Jx_3 + mx_3^3 + mr^2x_3)x_2^2 + 2mrx_3x_4x_2 - (J + mx_3^2)g\,\mathrm{sen}\,x_1 + mgrx_3\cos x_1}{J + mx_3^2} + \frac{(c + x_3^2)\big(-\frac{1}{r}\left[mrx_3x_2^2 + 2mx_3x_4x_2 + mgx_3cosx_1 + (J + mx_3^2)u_a\right]\big)}{J + mx_3^2},$$

con

$$c = \frac{J}{m} + r^2.$$

Para analizar la dinámica interna de (28), y saber que sucederá con x_3 y x_4, se propone el siguiente cambio de variable:

$$\begin{aligned} z_1 &= x_1, & x_1 &= z_1, \\ z_2 &= x_2, & x_2 &= z_2, \\ z_3 &= x_3, & x_3 &= z_3, \\ z_4 &= c\,x_2 + x_3^2\,x_2 + r\,x_4, & x_4 &= \tfrac{1}{r}(z_4 - c\,z_2 - z_3^2\,z_2). \end{aligned}$$

La dinámica de z queda definida por:

$$\begin{aligned}
\dot{z_1} &= z_2, \\
\dot{z_2} &= u_a, \\
\dot{z_3} &= \tfrac{1}{r}\big(z_4 - c\, z_2 - z_3^2\, z^2\big), \\
\dot{z_4} &= H(z),
\end{aligned}$$

donde

$$\begin{aligned}
H(z) =& 2z_3 \left(\frac{1}{r}\right)(z_4 - c\, z_2 - z_3^2\, z_2)z_2 + \\
& \frac{r\big((J\, z_3 + m\, z_3^3 + m\, r^2\, z_3)z_2^2 + 2m\, r\, z_3(\frac{1}{r})(z_4 - c\, z_2 - z_3^2\, z_2)z2\big)}{J + m\, z_3^2} + \\
& \frac{r\,(-(J + m\, z_3^2)g\,\mathrm{sen}\, z_1 + m\, g\, r\, z_3 \cos z_1)}{J + m\, z_3^2} + \\
& \frac{r\big((c + z_3^2)\left(-\frac{1}{r}\right)\left(m\, r\, z_3\, z_2^2 + 2m\, z_3\left(\frac{1}{r}\right)(z_4 - c\, z_2 - z_3^2\, z_2)z_2 + m\, g\, z_3 \cos z_1\right)\big)}{J + m\, z_3^2}.
\end{aligned}$$

Haciendo $z_1 = z_2 = 0$ obtenemos la dinámica interna:

$$\begin{aligned}
\dot{z_3} =& \frac{1}{r} z_4, \\
\dot{z_4} =& -\, g\, z_3.
\end{aligned}$$

La dinámica interna tiene el comportamiento de un oscilador lineal, el cual es estable en el sentido de Lyapunov.

III.3.2 Grados de libertad extra para el control

Una vez diseñado el controlador se observa que éste sólo depende del parámetro k. Para poder controlar la velocidad con la que el estado x_1 converge a cero, se decide agregar una ganancia k_1 a la función $\phi(\eta)$. De manera que considerando el sistema (25)-(26), el nuevo control que estabilice (25) será:

$$\phi(x_1) = -k_1 x_1.$$

Su solución sería:

$$x_1(t) = e^{-k_1 t} x(0).$$

A esta propuesta se añade otro término, para intentar contrarrestar la fricción de Coulomb (Alvarez *et al.*, 2000) observada en el péndulo, misma que no ha sido modelada, quedando:

$$\phi(x_1) = -k_1 x_1 - k_2 \text{sign}(x_1).$$

Para evitar problemas asociados a la no diferenciabilidad de la función signo, utilizamos una aproximación con la tangente hiperbólica, quedando de la siguiente manera:

$$\phi(x_1) = -k_1 x_1 - k_2 \tanh(\lambda x_1), \quad \lambda \gg 1,$$

con función de Lyapunov

$$V(x_1) = \frac{1}{2} x_1^2,$$

y derivada

$$\dot{V}(x_1) = -k_1 x_1^2 - k_2 x_1 \tanh(\lambda x_1) \leq 0.$$

Volvemos a obtener una ley de control como la ecuación (27):

$$u = -\frac{1}{r}\left[mrx_3x_2^2 + 2mx_3x_4x_2 + mgx_3 cos x_1 + (J + mx_3^2)u_a\right],$$

donde, en este caso:

$$u_a = -x_1 - \left[k_1 + \lambda k_2 \text{sech}^2(\lambda x_1)\right]x_2 - k_3\left[k_1 x_1 + x_2 + k_2 \tanh(\lambda x_1)\right].$$

III.4 Observador con estimación de perturbaciones

En las secciones anteriores se han presentado los tres controladores que se utilizarán con el péndulo invertido. Para los métodos de ubicación de polos y *backstepping* se requiere de manera explícita conocer el estado completo, mientras que para el PID sólo se requiere conocer el estado que se quiere controlar x_1 y eventualmente x_2 para contar con la derivada del error.

El péndulo cuenta con dos *encoders* o sensores de posición, con los cuales podemos medir el ángulo θ y el desplazamiento x (ver Figura 2), pero no se cuenta con tacómetros que nos puedan proporcionar las velocidades. Es decir, sólo se tienen mediciones de x_1 y x_3. Es por esto que se requiere utilizar un observador para obtener los estados faltantes x_2 y x_4.

En (Estrada García, 2008), se presenta una comparación entre dos observadores

robustos (Rosas Almeida *et al.*, 2007) y un derivador robusto de segundo orden (Levant, 2003) aplicados al péndulo invertido.

En esta tesis se ha decidido utilizar una variación del observador robusto de Rosas Almeida *et al.* (2007). El observador está basado en la generación de un modo deslizante de orden dos con convergencia exponencial, dando robustez a la estimación del estado. Este observador además nos permite estimar las perturbaciones generadas por variaciones paramétricas o por disturbios externos. Esta propiedad se utilizará más adelante con el control *backstepping*, para compensar dinámicas no modeladas como pudiera ser la fricción.

III.4.1 Estructura del observador

El observador es propuesto para sistemas lagrangianos en forma normal,

$$\begin{aligned} \dot{x}_i &= x_{i+1} \qquad i = 1, \ldots, n-1 \\ \dot{x}_n &= f(x) + g(x)u + \gamma(t, x) \\ y &= x_1 \end{aligned}$$

donde $\gamma(t, x)$ es una perturbación. Suponiendo que la perturbación $\gamma(t, x)$ es derivable y que su derivada está acotada, se propone el siguiente observador.

$$\begin{aligned} \dot{\hat{x}}_1 &= c_{1,1}(y - \hat{y}_1) + w_1 \\ \dot{w}_1 &= c_{2,1}(y - \hat{y}_1) + c_{3,1}\,\mathrm{sign}(y - \hat{y}_1) \\ \hat{y}_1 &= \hat{x}_1 \end{aligned}$$

$$\begin{aligned} \dot{\hat{x}}_i &= c_{1,i}(w_{i-1} - \hat{y}_i) + w_i \\ \dot{w}_i &= c_{2,i}(w_{i-1} - \hat{y}_i) + c_{3,i}\,\mathrm{sign}(w_{i-1} - \hat{y}_i) \\ \hat{y}_i &= \hat{x}_i \qquad i = 2, \ldots, n-1 \end{aligned}$$

$$\begin{aligned} \dot{\hat{x}}_i &= f(y, \hat{x}) + g(y, \hat{x})u + c_{1,n}(w_{n-1} - \hat{y}_n) + w_n \\ \dot{w}_n &= c_{2,n}(w_{n-1} - \hat{y}_n) + c_{3,n}\,\mathrm{sign}(w_{n-1} - \hat{y}_n) \\ \hat{y}_n &= \hat{x}_n \end{aligned}$$

En (Rosas Almeida *et al.*, 2009) se muestra que se pueden encontrar constantes $c_{1,i}$, $c_{2,i}$ y $c_{3,i}$ tales que el sistema $\hat{x}_i$ converge a x_i, w_i a x_{i+1} y w_n a $\gamma(t,x)$.

III.4.2 Realización para el péndulo

Retomando la representación de estados del péndulo invertido de la ecuación (9) en el Capítulo II, podemos agrupar términos para obtener la siguiente representación,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ f_1(x_1,x_2,x_3,x_4) + g_1(x_1,x_2,x_3,x_4)u + \gamma_1(t,x) \\ x_3 \\ f_2(x_1,x_2,x_3,x_4) + g_2(x_1,x_2,x_3,x_4)u + \gamma_2(t,x) \end{bmatrix}$$

donde $\gamma_1(t,x)$ y $\gamma_2(t,x)$ pueden representar una perturbación, dinámica no modelada o variaciones paramétricas. A partir de esta representación se procede a diseñar el observador.

El observador para el péndulo invertido quedaría de la siguiente manera.

$$\begin{aligned}
\dot{\hat{x}}_1 &= c_{1,1}(x_1 - \hat{x}_1) + \omega_1 \\
\dot{\omega}_1 &= c_{2,1}(x_1 - \hat{x}_1) + c_{3,1}sign(x_1 - \hat{x}_1) \\
\dot{\hat{x}}_2 &= f_1(x_1, \hat{x}_2, x_3, \hat{x}_4) + g_1(x_1, \hat{x}_2, x_3, \hat{x}_4)u + c_{1,2}(\omega_1 - \hat{x}_2) + \omega_2 \\
\dot{\omega}_2 &= c_{2,2}(\omega_1 - \hat{x}_2) + c_{3,2}sign(\omega_1 - \hat{x}_2) \\
\dot{\hat{x}}_3 &= c_{1,3}(x_3 - \hat{x}_3) + \omega_3 \\
\dot{\omega}_3 &= c_{2,3}(x_3 - \hat{x}_3) + c_{3,3}sign(x_3 - \hat{x}_3) \\
\dot{\hat{x}}_4 &= f_2(x_1, \hat{x}_2, x_3, \hat{x}_4) + g_2(x_1, \hat{x}_2, x_3, \hat{x}_4)u + c_{1,4}(\omega_3 - \hat{x}_4) + \omega_4 \\
\dot{\omega}_4 &= c_{2,4}(\omega_3 - \hat{x}_4) + c_{3,4}sign(\omega_3 - \hat{x}_4)
\end{aligned}$$

w_2 y w_4 son la estimación de las pertubaciones del eslabón 1 y 2, respectivamente.

Capítulo IV

Metodología

En este capítulo presentamos la metodología que se ha seguido para hacer la sintonización de los controladores que se presentaron en el Capítulo III. Para hacer la sintonización de estos parámetros se le ha dado el enfoque de un problema de optimización. Como en cualquier problema de optimización debemos definir una función objetivo a minimizar y el algoritmo que se va a utilizar.

Para la función objetivo se ha decidido utilizar índices de desempeño, muy comunes en teoría de control, en conjunto con tres algoritmos heurísticos de optimización.

IV.1 Función objetivo

Para comparar la respuesta de dos sistemas, utilizaremos índices de desempeño como se presenta en (Dorf y Bishop, 1996). Un índice de desempeño es una medida cuantitativa del desempeño de un sistema y se elige de manera que se enfaticen las especificaciones importantes del sistema. Los índices de desempeño deben definirse de tal manera que siempre tomen valores positivos o igual a cero; de tal manera, que para elegir el mejor de dos sistemas, se pueda hacer simplemente eligiendo el que haya tenido el menor valor para un mismo índice.

Para el problema de optimización, utilizaremos estos índices de desempeño como función objetivo, combinándolos con el tiempo de establecimiento.

IV.1.1 Índices de desempeño

Los índices serán calculados para un intervalo de tiempo finito T. El tiempo T se escoge de manera que el sistema alcance su estado estacionario. Los índices de desempeño se obtienen en función del error, el cual está definido de la siguiente manera:

$$e(t) = y_{ref}(t) - y(t);$$

además definimos una versión modificada del error

$$\hat{e}(t) = \begin{cases} 10\, e(t) & \text{para} \quad |y_{ref}(t)| < |y(t)| \\ e(t) & \text{para} \quad |y_{ref}(t)| \geq |y(t)| \end{cases},$$

con la cual se obtienen penalizaciones para sobretiros, cuando la salida $y(t)$ supera a la referencia $y_{ref}(t)$, el error es multiplicado por 10.

A continuación describiremos los 5 índices de desempeño considerados.

ITAE

El ITAE (Graham y Lathrop, 1953), por sus siglas en inglés (Integral of Time-Multiplied Absolute-value of Error) o integral del valor absoluto del error multiplicado por el tiempo, es un índice que pondera en el tiempo el valor absoluto del error entre la referencia y la salida. De esta manera se da mayor peso al error en estado estacionario que en el transitorio, donde de manera inicial los errores son típicamente grandes. El valor absoluto nos asegura que este índice siempre sea un valor positivo.

$$ITAE = \frac{1}{T} \int_0^T t|e(t)|\, dt.$$

MITAE

El MITAE es una versión modificada del ITAE, donde lo que se cambia es el error $e(t)$ por el error modificado $\hat{e}(t)$, y de esta manera se penalizan los sobretiros, una distinción que el ITAE por si solo no hace.

$$MITAE = \frac{1}{T} \int_0^T t|\hat{e}(t)|\, dt.$$

ISE

El ISE por sus siglas en inglés (Integral of the Square of the Error) o integral del error cuadrático, resultará en un valor positivo tanto para valores positivos como para valores negativos, gracias a que el error se eleva al cuadrado. Este índice ayuda a discriminar entre sistemas subamortiguados y sobreamortiguados (Dorf y Bishop, 1996).

$$ISE = \frac{1}{T} \int_0^T e^2(t)\, dt.$$

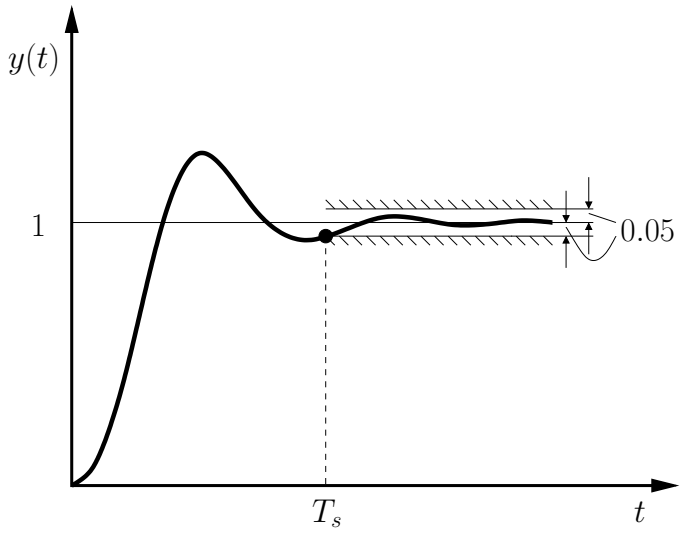


Figura 10: Tiempo de establecimiento ante una entrada tipo escalón.

MISE

El MISE es la versión modificada del ISE, como vimos con el MITAE, de tal manera que los sobretiros se penalizan al cambiar el error $e(t)$ por $\hat{e}(t)$.

$$MISE = \frac{1}{T}\int_0^T \hat{e}^2(t)\,dt.$$

IAE

El IAE, por sus siglas en inglés (Integral of the absolute value of error) o integral del valor absoluto del error, simplemente acumula de manera general el error presentado en el transitorio, sin dar más peso al error en el inicio o en el final.

$$IAE = \frac{1}{T}\int_0^T |e(t)|\,dt.$$

IV.1.2 Tiempo de establecimiento, T_s

El tiempo de establecimiento T_s, se define como el tiempo mínimo después del cual la respuesta permanece en el intervalo del $\pm 5\%$ del valor en estado estacionario (Véase Figura 10).

IV.2 Algoritmos de optimización

Para resolver el problema de optimización y hacer la sintonización de los parámetros de los controladores, se han elegido tres métodos heurísticos: Hooke y Jeeves, Nelder y Mead y recocido simulado. Se han elegido estos métodos por ser sencillos de implementar, ya que no es necesario que la función de costo posea características particulares (e.g. diferenciabilidad), sólo requieren poder calcular el valor de la función objetivo en cualquier momento.

IV.2.1 Hooke y Jeeves

Este método está basado en la propuesta hecha por Hooke y Jeeves (1961). El método realiza una búsqueda directa con el uso de dos operaciones: movimiento exploratorio y movimiento patrón. Estos movimientos se repiten hasta que se satisfaga el criterio de paro.

Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función objetivo a minimizar, $b \in \mathbb{R}^n$ un conjunto inicial de parámetros y $\rho \in \mathbb{R}$ la distancia inicial de búsqueda. Definimos los vectores unitarios e_i de manera que formen una base ortogonal para $i = 1, 2, \ldots, n$, donde n es el número de parámetros de la función objetivo. Ahora definimos los movimientos.

Movimiento exploratorio

El movimiento exploratorio se lleva a cabo de la siguiente manera. Con b como punto inicial, hacemos $b_1 \leftarrow b$ y calculamos $f(b_1 + \rho e_1)$, si el movimiento de b_1 a $b_1 + \rho e_1$ genera una mejoría ($f(b_1 + \rho e_1) < f(b_1)$), entonces hacemos $b_1 \leftarrow b_1 + \rho e_1$; si no hubo mejoría, entonces evaluamos $f(b_1 - \rho e_1)$. Si hay mejoría en $b_1 - \rho e_1$, entonces hacemos $b_1 \leftarrow b_1 - \rho e_1$, de lo contrario, b_1 permanece igual. Luego se repite este proceso, pero ahora en vez de usar e_1 usamos e_2, luego e_3, hasta terminar con e_n. En caso de que no haya habido mejoría con el movimiento exploratorio, hacemos $\rho \leftarrow \dfrac{\rho}{2}$ e intentamos de nuevo. Si hubo mejoría, hacemos $b_2 \leftarrow b_1$ e intentamos un movimiento patrón.

En la Figura 11 se muestran algunos ejemplos de movimientos exploratorios cuando el espacio de búsqueda es $\mathbb{R}^2$, la entrada se refiere al punto inicial y la salida es el punto que genera una mejoría en la función objetivo.

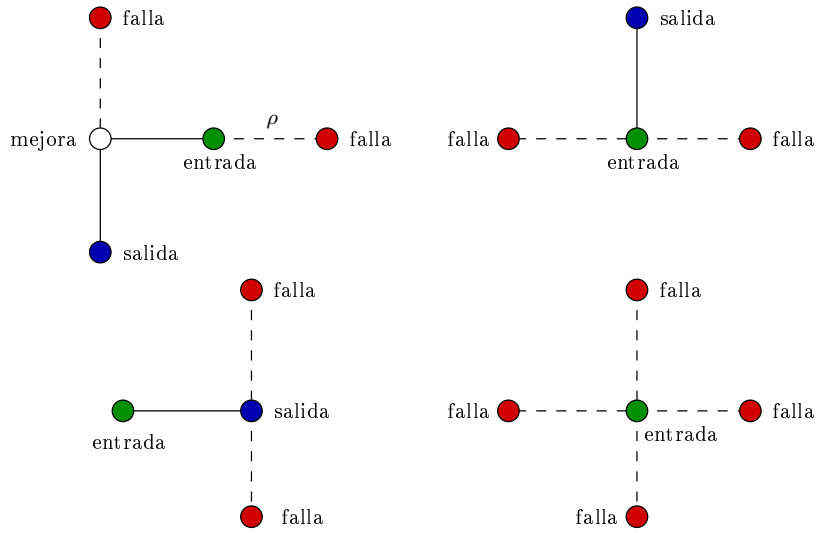


Figura 11: Movimiento exploratorio en $\mathbb{R}^2$.

Movimiento patrón

El movimiento patrón consiste en intentar un movimiento en la misma dirección que el último movimiento que resultó en una mejoría. Hacemos $b_3 \leftarrow 2b_2 - b_1$, si $f(b_3) < f(b_2)$, entonces b_3 será el nuevo punto inicial, de lo contrario b_2 será el punto inicial para la siguiente iteración. En la Figura 12 se muestran dos posibles movimientos patrón en $\mathbb{R}^2$.

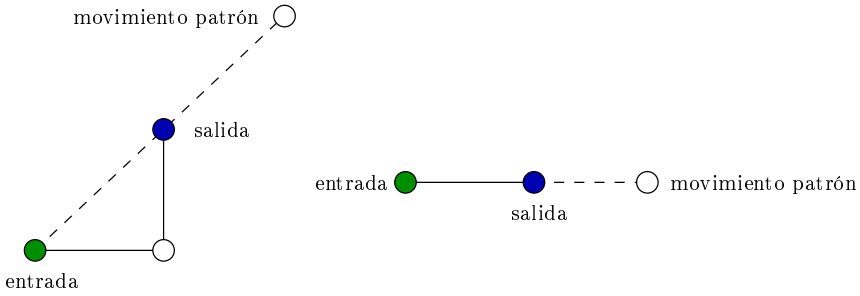


Figura 12: Movimiento exploratorio en $\mathbb{R}^2$.

En la Figura 13 se muestra un ejemplo de como sería la evolución de este método a base de los dos movimientos definidos. El punto inicial es b_1, a partir del cual se hace un movimiento exploratorio y se llega a b_2, se procede con un movimiento patrón y se llega a b_3. Con un movimiento exploratorio se llega a b_4, y el movimiento patrón ya no resulta en mejoría. Se intenta un movimiento exploratorio y no hay mejoría por lo que ρ se reduce a la mitad. Con dos movimientos exploratorios más se llega al mínimo en b_6.

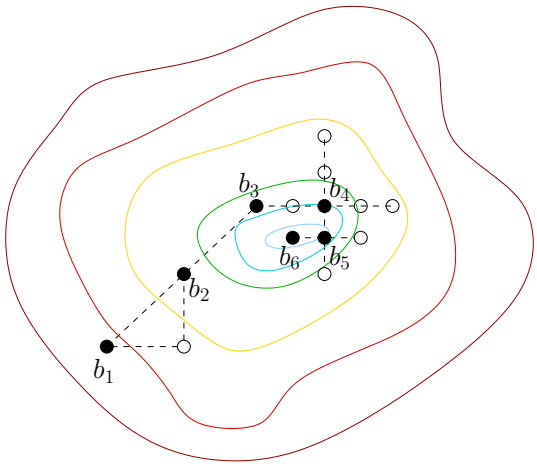


Figura 13: Optimización con método Hooke-Jeeves, curvas de nivel para $\mathbb{R}^2$.

IV.2.2 Nelder y Mead (simplex)

Este método como fue presentado por Nelder y Mead (1965), es utilizado para minimizar una función de n variables. El método depende de la comparación de los valores obtenidos al evaluar la función objetivo en los $(n+1)$ puntos que forman un simplex[1], seguido de reemplazar el (peor) vértice que genere el valor más grande por otro punto. El simplex se adapta a la forma del terreno en el espacio de búsqueda y se contrae cuando se encuentra en la cercanía de un mínimo; esto se logra con el uso de unas operaciones básicas que describiremos más adelante.

Algoritmo

Para utilizar este método debemos tener un simplex inicial. Este simplex inicial puede construirse a partir de una primera estimación o semilla inicial $x_1 \in \mathbb{R}^n$, y los otros n vértices pueden ser:

$$x_{i+1} = x_1 + \rho e_i,$$

para $i = 1, \ldots, n$, con ρ y e_i definidos como en el método anterior.

[1]Un simplex es formado por $n+1$ puntos en $\mathbb{R}^n$, que no están en un mismo hiperplano. Las coordenadas de cada punto corresponden a cada una de las n variables de la función objetivo.

Ahora definamos el peor (x_h), segundo peor (x_s) y mejor (x_l) vértice,

$$x_h := \{x_i | f(x_i) \geq f(x_j), \forall j = 1, \ldots, n+1\}$$
$$x_s := \{x_i | f(x_i) \geq f(x_j), \forall j = 1, \ldots, n+1, i \neq h\}$$
$$x_l := \{x_i | f(x_i) \leq f(x_j), \forall j = 1, \ldots, n+1\}$$

donde $f(\cdot)$ es la función objetivo como en el método anterior. Por simplicidad en la notación definimos:

$$f_i := f(x_i).$$

Siendo $\bar{x}$ el centroide de todos los vértices excepto x_h, las operaciones válidas sobre el simplex son:

- Calcular centroide $\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n+1} x_i - x_h\right)$.
- Reflexión $x_r = \bar{x} + \alpha(\bar{x} - x_h)$.
- Expansión $x_e = \bar{x} + \gamma(\bar{x} - x_h)$.
- Contracción $x_c = x_h + \beta(\bar{x} - x_h)$.
- Reducción $x_i = (x_i + x_l)/2$.

Típicamente $\alpha = 1$, $\gamma = 2$ y $\beta = 0.5$ ó -0.5.

Habiendo definido los vértices y las operaciones básicas que se pueden realizar sobre el simplex, el algoritmo queda de la siguiente manera.

1. Ordenar los vértices, $x_l \leq \cdots \leq x_s \leq x_h$. Calcular $\bar{x}$.
2. Hacer una reflexión; si $f_l \leq f_r \leq f_s$, entonces $x_h \leftarrow x_r$ y regresar a 1.
3. Si $f_r < f_l$, entonces hacer una expansión, de lo contrario ir a 5.
4. Si $f_e \leq f_r$, entonces $x_h \leftarrow x_e$ y regresar a 1, de lo contrario $x_h \leftarrow x_r$ y regresar a 1.
5. Si $f_s \leq f_r < f_h$, entonces hacer una contracción con $\beta = 0.5$.
6. Si $f_r \geq f_h$, entonces hacer una contracción con $\beta = -0.5$.

7. Si $f_c < f_h$, entonces $x_h \leftarrow x_c$, de lo contrario hacer una reducción.

8. Si se cumple el criterio de paro, entonces parar, de lo contrario regresar a 1.

IV.2.3 Recocido simulado

Este método está inspirado en el recocido utilizado en la metalurgia, donde a base de una combinación de calentamiento y enfriamiento controlado, los metales incrementan el tamaño de sus cristales y reducen sus defectos.

El método de recocido simulado es una técnica de optimización combinatoria de uso general propuesta por Kirkpatrick *et al.* (1983). Este método es una extensión del método desarrollado por Metropolis *et al.* (1953), para determinar el estado en equilibrio de átomos en una sustancia a una temperatura T.

Algoritmo

Haciendo analogía con el proceso de recocido en metalurgia, el recocido simulado se explica de la siguiente manera.

Se tiene una función objetivo análoga a la energía $E(\cdot)$ relacionada a una conformación S de n partículas. Siendo S_0 el estado inicial, hacemos $S \leftarrow S_0$ con energía correspondiente $e \leftarrow E(S)$. El proceso inicia con una temperatura T alta, de manera que las partículas puedan moverse libremente. Para una temperatura constante se realizan varias iteraciones donde se perturba el estado S, de tal manera que alguna(s) partícula(s) cambie(n) de posición y así obtengamos un nuevo estado S_{new}, con energía e_{new}. Para este nuevo estado se evalúa si hubo mejoría en la función objetivo o de energía, si la energía disminuyó ($e_{new} < e_{best}$) el estado S_{new} se toma como la nueva conformación ($S \leftarrow S_{new}$, $e \leftarrow e_{new}$). Si no hubo mejoría, el nuevo estado será aceptado dependiendo de una probabilidad $P(e, e_{new}, T)$; esta probabilidad debe de ser alta para temperaturas grandes y debe de ir disminuyendo conforme la temperatura disminuya, de tal manera que en un principio la aceptación será alta, aún cuando el nuevo estado no represente mejoría en la energía. Conforme T disminuya, el proceso se hará más selectivo hasta que sólo se acepten estados que mejoren la energía. Para decidir si un caso que no mejora la energía es aceptado, o no, se genera un número aleatorio con distribución uniforme (rand(0,1)) entre cero y uno, y se compara con el valor de la probabilidad. Si $\text{rand}(0,1) < P(e, e_{new}, T)$ el estado es aceptado, de lo contrario no.

Este proceso de generar nuevos estados perturbando el actual, debe hacerse varias veces, hasta que se considere que se ha alcanzado un equilibrio térmico para una temperatura T constante. Una vez alcanzado el equilibrio térmico, se disminuye la temperatura T y se repite el proceso anterior. La temperatura debe seguir disminuyendo hasta llegar a una temperatura en la que se considere que el sistema se ha enfriado. Además se puede ir guardando cual fue el estado S_{best} que minimizó la energía e_{best}, ya que este podría llegar a ser distinto que el último estado generado.

Para implementar este método debemos definir: la probabilidad de aceptación, la función vecindario, el tiempo para alcanzar equilibrio térmico, el perfil de temperatura y el criterio de paro.

Probabilidad de aceptación

La probabilidad de aceptación será una función del estado actual S, el nuevo estado S_{new} y la temperatura T. Una implementación típica es:

$$P(e, e_{new}, T) = \exp\left(\frac{e - e_{new}}{T}\right).$$

Definida la probabilidad de esta manera, la probabilidad de aceptar un estado con menos energía resulta mayor o igual a 1; y la de aceptar un estado con más energía será menor que 1, tendiendo a cero conforme $T \to 0$.

Función de vecindario

La función de vecindario $V : \mathbb{R}^n \to \mathbb{R}^n$, es la que generará el nuevo estado a partir del actual. El método de recocido simulado originalmente fue utilizado para problemas combinatorios, donde típicamente la función vecindario se definía como una permutación de los elementos de S. En el caso que presentamos en este trabajo, no se trata de un problema de combinatoria, sino de un problema de optimización en un espacio continuo.

La función de vecindario se ha definido de la siguiente manera:

$$V(S) = S + M\delta,$$

donde $\delta = (\epsilon)\text{rand}(0, 1)$ y M es un vector de dimensión n, cuyos elementos M_i son 1 ó -1 elegidos de manera aleatoria siguiendo una distribución uniforme discreta. Además

$\epsilon = d_{max} \frac{T}{T_{max}}$, de forma que su valor va disminuyendo conforme la temperatura T disminuye; d_{max} es el parámetro que limitará la distancia máxima que se puede mover el estado S sobre cada eje en $\mathbb{R}^n$ y T_{max} es la temperatura máxima o temperatura inicial.

Tiempo para alcanzar equilibrio térmico

Se debe definir cuantas iteraciones se realizarán para cada nivel constante de temperatura, de manera que se alcance el equivalente a un equilibrio térmico, es decir, que haya posibilidad de explorar las distintas conformaciones. Este tiempo puede definirse como un número finito de iteraciones, o un número de aceptaciones. En nuestra implementación utilizaremos un número de iteraciones máximas k_{max}.

Perfil de temperatura

La temperatura iniciará en un valor relativamente grande T_{max} e irá modificándose de la siguiente manera

$$T \leftarrow \alpha T$$

donde α es un número entre 0 y 1; típicamente $\alpha = 0.95$.

Criterio de paro

Típicamente el criterio de paro para este método es cuando se haya terminado el tiempo presupuestado (iteraciones) para optimización. Suele especificarse una temperatura mínima T_{min}, de manera que cuando $T < T_{min}$ se para el método. Y la solución será el estado S_{best} correspondiente a e_{best}.

Algoritmo

Habiendo definido lo anterior, el algoritmo puede quedar como se muestra en la Figura 14.

IV.3 Conclusiones del capítulo

Se presentaron las diversas funciones que se utilizarán como función objetivo al momento de sintonizar los parámetros de los controladores que se compararán. La idea es poder

```
T ← T_max                                   // T_max ∈ ℝ
k ← 0                                       // Contador para alcanzar equilibrio térmico.
S ← S_0                                     // S ∈ ℝ^n
e ← E(S)                                    // Función de energía E : ℝ^n → ℝ.
S_best ← S
e_best ← e                                  // T_min es la temperatura de paro.
while T > T_min do
  S_new ← V(S)                              // Función de vecindario
  e_new ← E(S_new)
  if e_new < e_best then
    S_best ← S_new                          // Disminuyó la energía.
    e_best ← e_new
  end if
  if random(0,1)< P(e, e_new, T) then
    S ← S_new                               // Aceptación del nuevo estado.
    e ← e_new
  end if
  k ← k + 1                                 // K_max es una constante predefinida.
  if k ≥ k_max then
    T ← αT      // Decremento en temperatura por un factor α predeterminado
    k ← 0       // Reiniciar contador de equilibrio térmico
  end if
end while
```

Figura 14: Pseudocódigo para el algoritmo de recocido simulado

encontrar la mejor combinación de parámetros que estabilice al péndulo invertido. La forma en la que podremos decidir si un controlador tiene un mejor desempeño que otro será midiendo el mismo con los índices presentados.

Está metodología fue presentada en (Abraján Guerrero y Márquez Martínez, 2010), específicamente para el control PID (ver Apéndice A), y se hizo una comparación con los resultados presentados por de Almeida *et al.* (2005), donde se utilizan técnicas de programación genética para sintonizar los parámetros del controlador. En estos dos trabajos se utilizan las reglas de Ziegler-Nichols para obtener una semilla para iniciar la sintonización, de igual manera se hará con el péndulo invertido.

Es importante destacar que las tres técnicas de optimización presentadas están basadas en heurísticas. Una diferencia importante entre ellas es que para Hooke-Jeeves y

Nelder-Mead, si se ejecuta el método de optimización repetidas veces partiendo siempre de la misma semilla, la solución que se obtenga siempre será la misma; en este sentido podemos decir que son técnicas determinísticas, a diferencia del recocido simulado. Para este último el ejecutar el algoritmo dos o más veces partiendo de la misma semilla, no implica que la aproximación del óptimo que se obtenga será la misma. Esto se debe a que en el recocido simulado, la forma en que el método avanza es estocástica, lo cual representa una ventaja, en el sentido que se permite hacer movimientos sin mejoría, con lo cual se exploran zonas que los otros métodos no explorarían. Sin embargo, con la reducción en la temperatura, el método empezará a buscar movimientos que sólo mejoren la función objetivo. Con esta variante podemos evitar quedar atascados en mínimos locales, una propiedad que los otros dos métodos no presentan.

Capítulo V

Plataforma de simulación y experimental

En los dos capítulos anteriores se ha presentado los controladores que se quiere utilizar y la metodología con la que se han de sintonizar sus parámetros. La obtención de los parámetros de los controladores se ha realizado tanto en simulación con el modelo del péndulo presentado en el Capítulo II como de manera experimental con el péndulo físico en el Laboratorio de Control.

V.1 Simulación

Para la sintonización a través de simulación, se han implementado los tres algoritmos de optimización en Matlab, de manera que a partir de un conjunto de parámetros iniciales se obtengan los parámetros que minimicen la función objetivo. Un esquema básico de como se implementó la sintonización se muestra en la Figura 15.

Dependiendo del algoritmo de optimización, la función objetivo y el controlador que se esté sintonizando es que se definen los bloques de ‘‘Evaluar función” y ‘‘Generar nuevos parámetros”. A continuación se detalla un poco sobre cada bloque.

V.1.1 Bloque: Parámetros iniciales

Dependiendo de que controlador se esté sintonizando es que se definen los parámetros iniciales. Estos parámetros iniciales serán los mismos para los distintos métodos de optimización y distintas funciones objetivo (excepto para el tiempo de establecimiento, donde se usará la combinación de parámetros sintonizados con las otras funciones objetivo que resulte en el menor tiempo de establecimiento).

PID

Para el control PID, se tienen tres parámetros: K_p, T_i y T_d. Los parámetros iniciales son elegidos utilizando el segundo método de Ziegler-Nichols. Los parámetros iniciales para la sintonización son: $K_p = -3.3$, $T_i = 2.607$ y $T_d = 0.6258$.

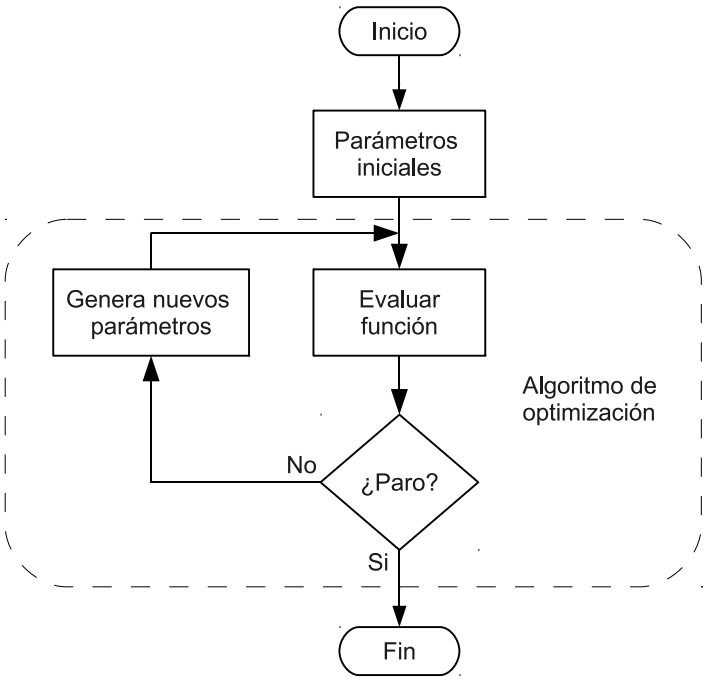


Figura 15: Esquema de sintonización.

Ubicación de polos

Dado que el modelo del péndulo es de cuarto orden, se cuenta con 4 polos (μ_1, μ_2, μ_3 y μ_4) a ubicar. Se ha decidido ubicarlos como dos pares de complejos conjugados, por lo que utilizaremos 4 parámetros (p_1, p_2, p_3 y p_4) para ubicar los 4 polos:

$$\begin{aligned} \mu_1 &= -p_1 + p_2\, i, & \mu_3 &= -p_3 + p_4\, i, \\ \mu_2 &= -p_1 - p_2\, i, & \mu_4 &= -p_3 - p_4\, i. \end{aligned}$$

A partir de estos polos se obtendrá la matriz K del control, como se vio en la Sección III.2.2. Los parámetros iniciales a utilizar en la sintonización son $\mathbf{p} = [4, 9, 6, 5]^T$.

Backstepping

Para el control por *backstepping* se utilizarán tres ganancias: k_1, k_2 y k_3. Como parámetros iniciales se tomará $\mathbf{k} = [1, 0, 1]^T$, con los cuales se obtiene un control *backstepping* con un solo grado de libertad como el mostrado en la Sección III.3.1.

V.1.2 Bloque: Genera nuevos parámetros

Este bloque depende de qué algoritmo de optimización se esté utilizando. Los algoritmos fueron programados de manera que se pudieran utilizar con funciones objetivo de distinto número de parámetros sin tener que modificar el programa. Dependiendo del algoritmo a utilizar los nuevos parámetros son generados, según se presentó en la Sección IV.2.

V.1.3 Bloque: Evaluar función

Este bloque es utilizado para elegir la función objetivo que se utilizará. En este trabajo se usaron las seis distintas funciones objetivo, presentadas en la Sección IV.1. Dependiendo cuál se desee minimizar, es que se define este bloque. Además de las funciones como se presentaron anteriormente, se agrega una penalización sobre la función objetivo bajo algunas condiciones, como se presenta a continuación.

Condiciones de Penalización

- Esfuerzo de control muy grande:

$$\left|u\right| > 3\, Newtons.$$

- Exceder restricciones físicas, x_1:

$$\left|x_1\right| > 0.6981\, rad.$$

- Exceder restricciones físicas, x_3:

$$\left|x_3\right| > 0.10\, m.$$

En caso que alguna de las tres condiciones anteriores se satisfaga, la funcion objetivo es penalizada con un factor de 10,

$$f \leftarrow 10\, f,$$

donde f es la función objetivo.

Para poder evaluar la función objetivo, se debe obtener la solución del péndulo con la retroalimentación del control a utilizar. Para esto se tiene programada una función de la

representación en espacio de estados del péndulo incluyendo el control, la cual se resuelve numéricamente en **Matlab** utilizando la función **ode45**, la cual es una implementación del método Runge-Kutta (orden 4 y 5) de paso variable. De esta manera se tiene acceso a la solución de todos los estados, con los cuales se puede evaluar la función objetivo.

V.1.4 Bloque: Paro

La sintonización termina, cuando se cumple la siguiente condición:

$$f_{i-k} - f_{i-k+1} \leq \epsilon \ \forall \ k = 1, 2, \cdots, n_{max};$$

donde n_{max} es un entero positivo y $\epsilon \approx 0^{+}$. De manera que el método se detiene cuando deja de haber mejoría después de n_{max} iteraciones. También puede pararse el método cuando llega a su número máximo de iteraciones permitido.

En el Apéndice B se muestran los programas utilizados para la sintonización.

V.2 Experimentación

Para la sintonización de manera experimental, se seguirá utlizando el esquema que se presentó para las simulaciones, mostrado en la Figura 15. Un cambio importante es que ahora en el bloque de ‘‘Evaluar función” en vez de resolver el sistema en simulación se llevará a cabo el experimento físicamente en el péndulo. Otra diferencia importante es que para el caso de simulaciones, se puede evaluar un conjunto de parámetros e inmediatamente evaluar el siguiente conjunto, simplemente se reasignan las condiciones iniciales del sistema al origen; mientras que para la experimentación, no se puede simplemente reiniciar condiciones iniciales, aquí se requiere llevar el sistema físicamente al origen para poder evaluar el siguiente conjunto de parámetros.

A continuación se describirán los componentes de hardware y software que fueron utilizados para poder sintonizar de manera experimental.

V.2.1 Hardware

El péndulo invertido ECP 550 cuenta con dos *encoders* que nos permiten medir la posición angular del primer eslabón y la posición lineal del segundo. También cuenta

con un actuador de corriente directa en la unión entre el primer y segundo eslabón.

Para poder leer los *encoders* y poder enviar una señal de control al actuador se utiliza la tarjeta de adquisición de datos Sensoray 626, el péndulo cuenta con su propia etapa de potencia.

V.2.2 Software

Se consideraron dos formas de realizar la sintonización, la primera fue utilizando Simulink y la segunda utilizando archivos MEX (Matlab executable).

Inicialmente se intentó utilizar Simulink, este ofrecía algunas ventajas como:

- Simplicidad para leer los *encoders* y enviar la señal de control.
- Facilidad para implementar los controladores y observador a través de bloques.

Sin embargo también presentaba algunas desventajas como:

- No se puede utilizar un modelo creado en Simulink (que opere en "*external mode*") como una función que se mande llamar desde la línea de comando o un m-file, de manera que pudiéramos utilizar los algoritmos de optimización como se utilizarían para la sintonización por simulación.
- Complejidad para utilizarse de manera iterativa.
- Difícil implementación de los algoritmos de optimización a través de bloques.

Por estas razones no se trabajó con Simulink.

La segunda opción, archivos MEX, consiste en un tipo de archivos que se utilizan en Matlab, los cuales permiten utilizar rutinas y librerías de C, C++ y Fortran. En este caso trabajamos con C/C++.

MEX

Para trabajar con estos archivos requerimos tener un archivo fuente MEX, con el cual al compilarlo se creará un archivo binario MEX, el cual podrá utilizarse como una función nativa de Matlab; permitiendo que la función sea llamada desde la línea de comandos o desde un "m-file".

Archivo fuente MEX

Un archivo fuente MEX debe contar con una rutina de enlace (*gateway routine*), `mexFunction()`, la cual se encargará de hacer la conexión entre Matlab y el código en C/C++, esta rutina la podemos ver como si fuera el "*main*" o rutina principal en un archivo en C/C++. Dentro de esta rutina se debe hacer la validación de los parámetros que se utilizarán en las rutinas en C/C++ y en caso de que las rutinas retornen algún parámetro se debe de asignar el espacio en memoria para hacerlo. Una vez validados los parámetros se pueden utilizar en las funciones escritas en C/C++, las cuales pueden aparecer en el mismo archivo antes de la función de enlace o en algún otro archivo como se hace normalmente en programas escritos en C/C++.

La Figura 16 muestra la estructura típica de un archivo fuente MEX. La librería `mex.h` debe ser incluida, posteriormente se pueden incluir cualesquier otras librerías que sean necesarias, según la aplicación.

Una vez que se cuenta con el archivo fuente MEX, éste se compila con el comando `mex`, para generar el archivo binario ejecutable; al momento de compilar se deben incluir los demás archivos que sirvan como recursos, separándolos por un espacio. Por ejemplo la instrucción:

`mex PID_Obs.cpp WIN626.obj signo.cpp`,

generaría el archivo binario ejecutable

`PID_Obs.mexw32`,

donde `WIN626.obj` y `signo.cpp` son recursos necesarios por el archivo fuente MEX `PID_Obs.cpp`.

El utilizar este tipo de archivos nos permite realizar un experimento físicamente como si simplemente se invocará un función de Matlab, siendo similar al caso de las simulaciones.

Implementación

Para la implementación, se hizo un programa en C++ que utiliza la tarjeta Sensoray 626 para controlar el péndulo, de manera que sólo se tuvieran que dar los parámetros del controlador al programa, y éste lo aplica al mecanismo, después de un tiempo definido por el usuario se conmuta a otro controlador (se eligió *backstepping*) para que el mecanismo sea llevado al origen nuevamente para dejarlo listo para otro experimento.

```
#include <mex.h>        // necesario para archivos MEX

/*****************************************************************
 * Se puede incluir cualquier otra cabecera (header) según se necesite
 *****************************************************************/

void funcion_1(int p1, int p2)
{
    /* Código en C de funcion_1 */
    /* La fución puede ser tipo void o cualquier otro */
}

/*******************************************
 * Aquí pueden aparecer más funciones en C
 *******************************************/

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    /*************************************************************
     * nlhs es el número de parámetros de entrada en Matlab
     * plhs[] es un vector con los parámetros de entrada
     * nrhs es el número de parámetros de salida en Matlab
     * prhs[] es el vector de los parámetros de salida
     *************************************************************/

    /* Código en C */
    /* Validar parámetros de entrada */
    /* Asignar espacio en memoria para parámetros de salida */
    funcion_1(p1, p2)    /* Llamar rutinas en C... */
    /* Asignar las variables de salida, si la función lo requiere */
}
```

Figura 16: Archivo fuente MEX, ejemplo.cpp

Ya teniendo un programa en C++ que permitiera usar la tarjeta Sensoray para hacer los experimentos con el péndulo, este programa fue adaptado para utilizarse como una función MEX. Se tomó el archivo que en C++ contenía el "main" y se le agregó la rutina de enlace. Se creó una función MEX para cada controlador a sintonizar, estás funciones reciben: el tiempo del experimento, los parámetros del controlador y la referencia (magnitud del escalón). Las funciones no regresan ningún parámetro, mas guardan en archivo la evolución de los estados medidos y estimados, así como la señal de control implementada. Con la información en estos archivos se determina el valor de la función objetivo que se esté minimizando.

Las funciones MEX desarrolladas realizan las siguientes tareas:

- Establecer conexión con la tarjeta Sensoray 626.
- Inicializar *encoders*, determinar offset.
- Controlar frecuencia de muestreo (1 kHz).
- Implementar el observador.
- Implementar el controlador que se esté sintonizando con los parámetros recibidos.
- Regresar el sistema al origen, conmutando a un control por *backstepping*.
- Guardar en archivo la evolución de los estados del sistema.
- Cerrar conexión con la tarjeta Sensoray 626.

En el Apéndice B.2.2, se muestra el código fuente de estas funciones.

Capítulo VI

Resultados en simulación

En este capítulo se muestran los resultados obtenidos en simulación al implementar la optimización de los parámetros para los distintos controladores aplicados al péndulo. En todos los casos, la sintonización se ha realizado evaluando la respuesta ante un escalón de magnitud 0.1745 *rad* (10 *grados*) en la referencia de la posición angular.

Para la simulación se utiliza el modelo del péndulo presentado en el Capítulo II, con los siguientes parámetros: $g = 9.81\,m/s^2$, $J = 0.022296\,kg \cdot m^2$, $m = 0.213\,kg$ y $r = 0.33\,m$; se agregó fricción viscosa a ambos eslabones (ver Apéndice B.2.1), mas ésta no es considerada en el diseño del controlador. El tiempo de simulación para cada combinación de parámetros es de 20 segundos. En el Apéndice B.1 se indican los parámetros que se utilizaron para realizar la sintonización con cada algoritmo.

Primero se presentan los resultados con control PID, luego con ubicación de polos y por último con *backstepping*.

VI.1 PID

Para la sintonización del PID, utilizando los cinco índices de desempeño definidos en la Sección IV.1.1, se toma como semilla los parámetros obtenidos por el segundo método de Ziegler-Nichols, es decir, $K_p = -3.3$, $T_i = 2.607$ y $T_d = 0.6258$. Para minimizar el tiempo de establecimiento, se elige como semilla alguna de las combinaciones de parámetros generadas con los índices de desempeño, la que haya obtenido menor tiempo de establecimiento y cumpla con las restricciones.

En la Figura 17 se muestra la respuesta del péndulo ante un cambio en la referencia, tipo escalón de magnitud 0.1745 *rad* en lazo cerrado, con los parámetros obtenidos por el segundo método de Ziegler-Nichols. El tiempo de establecimiento con estos parámetros es de 11.019 segundos. En la Figura 18 se muestra la señal de control con los mismos parámetros.

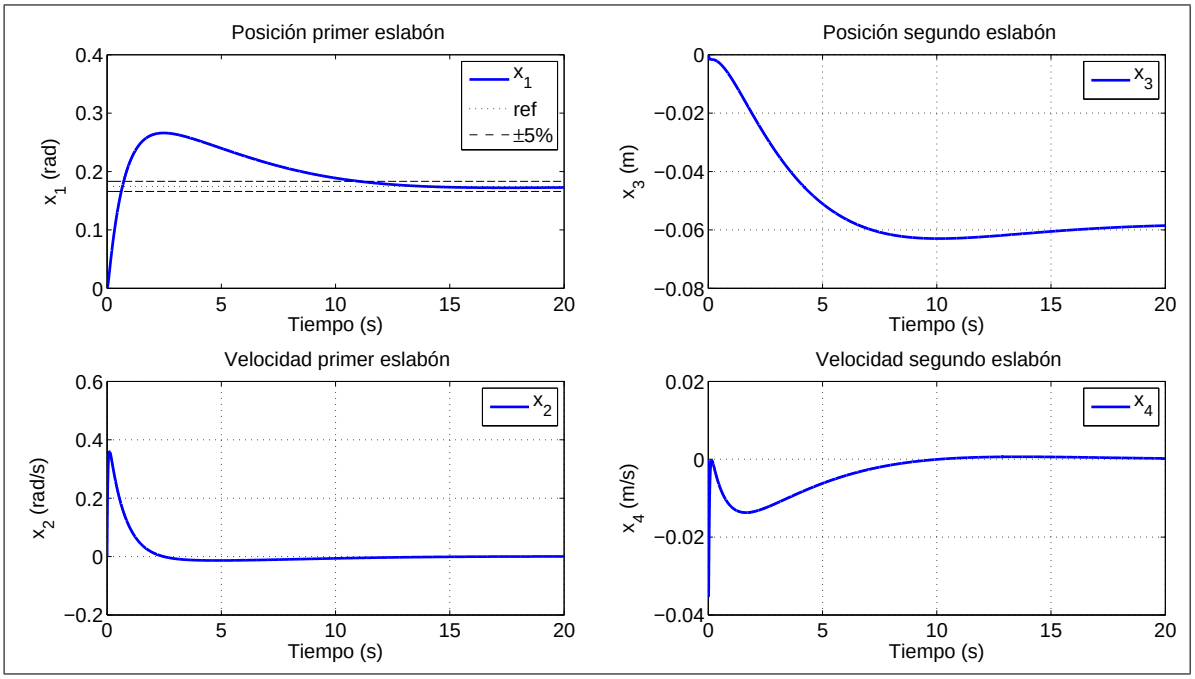


Figura 17: Respuesta ante un escalón de magnitud 0.1745 rad en la referencia, en lazo cerrado con un control PID con parámetros Ziegler-Nichols.

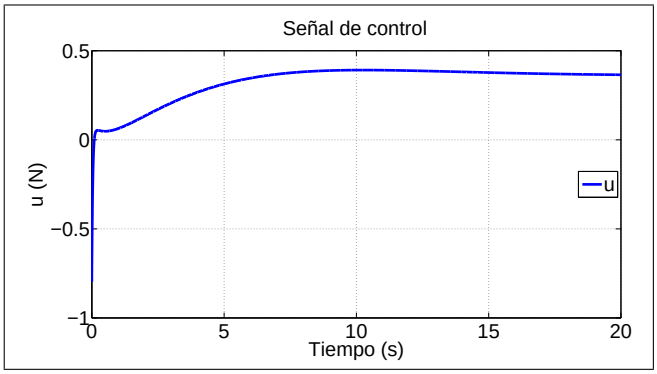


Figura 18: Control PID con parámetros Ziegler-Nichols.

La sintonización del controlador se hace a partir de esta respuesta.

VI.1.1 Hooke y Jeeves

Utilizando como semilla los parámetros de Ziegler-Nichols, se hizo la sintonización usando distintas funciones objetivo. Los parámetros obtenidos para cada función objetivo se muestran en la Tabla III. La columna T_s nos indica cuál es el tiempo de establecimiento que se obtiene con los parámetros sintonizados.

Tabla III: Control PID sintonizado con Hooke y Jeeves.

Función objetivo	K_p	T_i	T_d	T_s (s)
ITAE	-6.4926	0.6070	0.0754	2.556
MITAE	-6.4926	0.6070	0.0751	2.556
ISE	-14.9094	6.5445	0.0750	14.484
MISE	-14.9094	6.5425	0.0887	14.523
IAE	-6.4926	0.6070	0.0755	2.556
T_s	-6.4770	0.6048	0.2785	1.927

Tabla IV: Costo computacional al sintonizar el control PID con Hooke y Jeeves.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	31	200	297.55
MITAE	34	217	323.08
ISE	33	210	328.16
MISE	36	228	345.52
IAE	28	182	264.87
T_s	36	224	335.85

En dicha tabla se puede observar que el menor tiempo de establecimiento T_s obtenido al utilizar los índices de desempeño como función objetivo es de 2.556 segundos. Este tiempo es alcanzado usando los criterios ITAE, MITAE e IAE. Los parámetros que se obtienen son prácticamente los mismos (diferencia por diezmilésimas). Los índices ISE y MISE generan una respuesta con un T_s muy alto. Para el caso en que se toma el tiempo de establecimiento como función objetivo, se tomaron como semilla los parámetros obtenidos con el ITAE y se puede observar que el T_s se reduce un poco más, llegando a 1.927 segundos.

La Tabla IV muestra cuantas iteraciones fueron necesarias para minimizar cada función objetivo, así como cuantas veces fue necesario evaluar la función y el tiempo requerido para llevar acabo el proceso de optimización.

En la Figura 19 se muestra la respuesta del sistema ante una referencia tipo escalón de magnitud 0.1745 *rad*, para cada uno de los parámetros sintonizados. Las señales

generadas por los parámetros obtenidos con los criterios ITAE, MITAE e IAE casi no se distinguen entre si, puesto que los parámetros utilizados son prácticamente los mismos. Similar es el caso para ISE y MISE. La señal obtenida con T_s logra mejorar el tiempo de establecimiento.

A pesar de que las respuestas obtenidas con ITAE, MITAE e IAE al sintonizar los parámetros se aumentó el sobretiro y las oscilaciones, aun así las respuestas cumplen con la restricción que se tenía de no exceder 0.6981 rad.

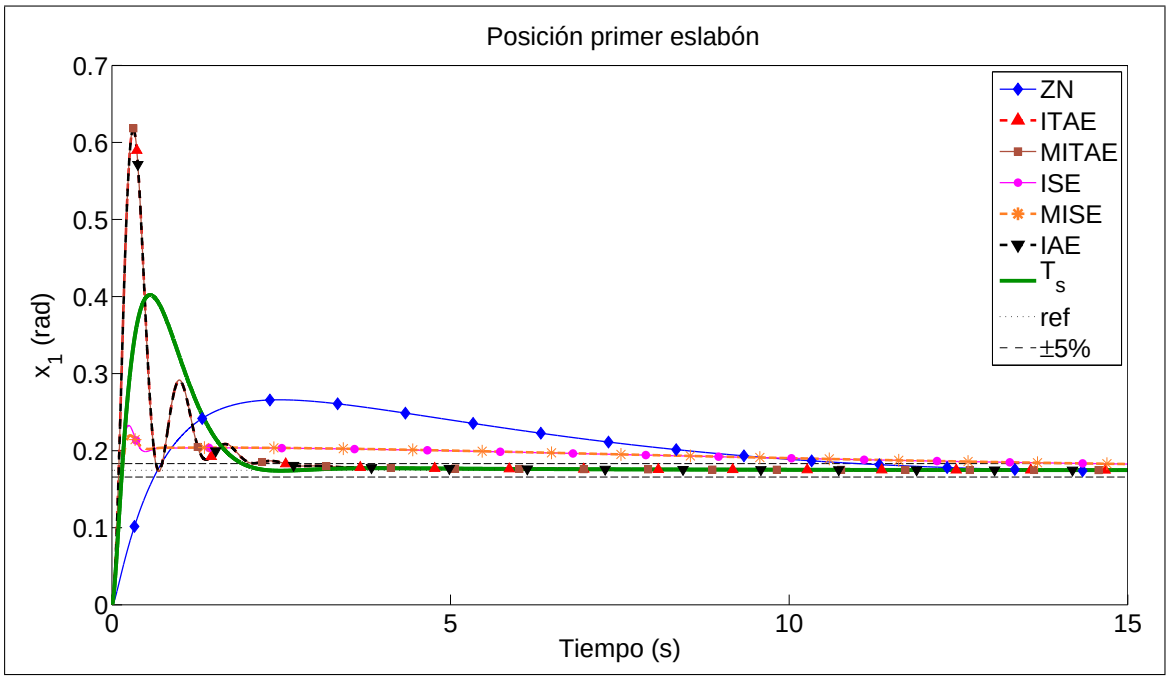


Figura 19: Posición angular x_1 usando PID sintonizado con Hooke y Jeeves.

La Figura 20 muestra la posición del segundo eslabón, la cual es importante conocer, aunque no la estamos intentando controlar, para asegurarnos que no exceda las restricciones físicas del sistema. Ninguno de los casos excedió los 0.10 m. También podemos ver que la señal de control cumplió con la restricción de no exceder los 3 N.

VI.1.2 Nelder y Mead

Ahora se presentan los resultados obtenidos para las sintonización del PID utilizando el algoritmo de Nelder y Mead para minimizar las funciones objetivo. En la Tabla V se muestran los parámetros obtenidos para cada una de las funciones objetivos, así como

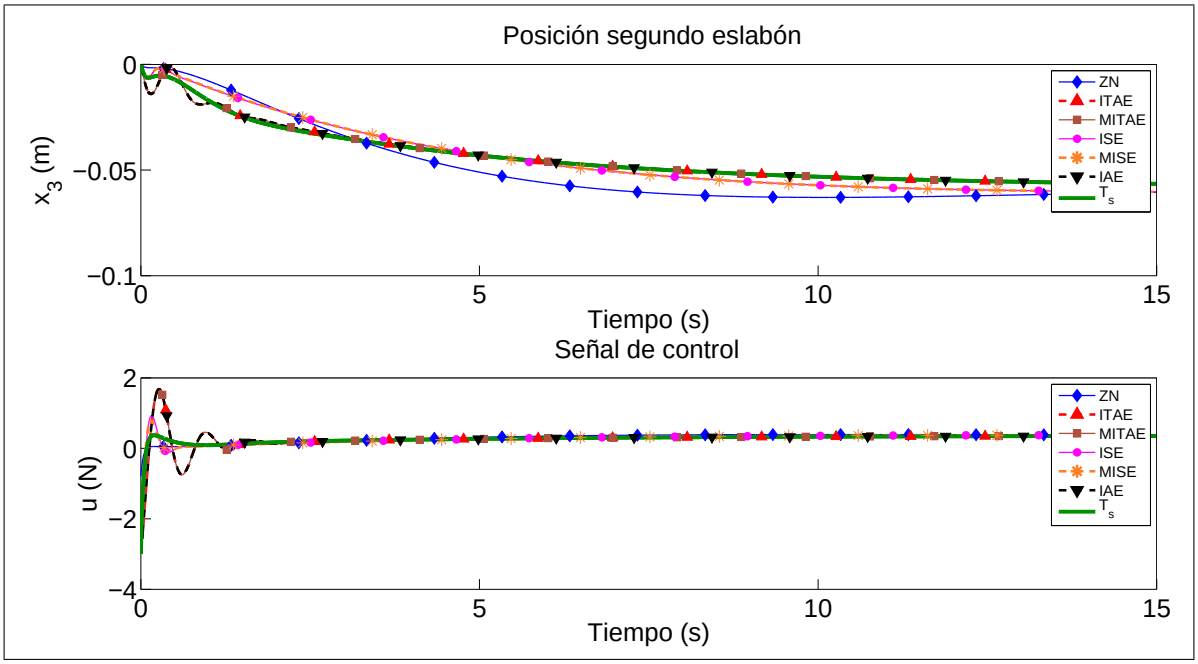


Figura 20: Posición de la barra x_3 y señal de control usando PID sintonizado con Hooke y Jeeves.

el tiempo de establecimiento del sistema. Para los 5 índices de desempeño se utilizaron los parámetros de Ziegler-Nichols como semilla y para el caso de T_s se utilizaron los parámetros obtenidos al minimizar el IAE, puesto que con este índice se obtuvo el menor tiempo de establecimiento. Podemos observar en la tabla, que al hacer la sintonización con T_s se obtiene un tiempo de establecimiento todavía menor.

Tabla V: Control PID sintonizado con Nelder y Mead.

Función objetivo	K_p	T_i	T_d	T_s (s)
ITAE	−4.9021	0.3990	0.1592	1.689
MITAE	−4.4041	0.3445	0.1968	1.941
ISE	−15.6761	10.4190	0.0635	19.450
MISE	−15.3450	8.3247	0.0855	16.938
IAE	−5.0683	0.4182	0.1426	1.654
T_s	−4.8619	0.3945	0.1726	0.954

La Tabla VI nos muestra el costo computacional requerido para la sintonización del

PID utilizando Nelder y Mead.

Tabla VI: Costo computacional al sintonizar el control PID con Nelder y Mead.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	98	180	265.12
MITAE	137	259	379.59
ISE	44	85	127.24
MISE	61	113	166.63
IAE	84	151	220.29
T_s	58	117	177.80

En la Figura 21 se muestra la respuesta del sistema con los parámetros obtenidos al hacer la sintonización. Nuevamente las señales generadas por los parámetros obtenidos con los criterios ITAE, MITAE e IAE son muy similares, así como las correspondientes a ISE y MISE. En la Figura 22 podemos observar que ninguno de los casos excede los límites establecidos para x_3 y la señal de control u.

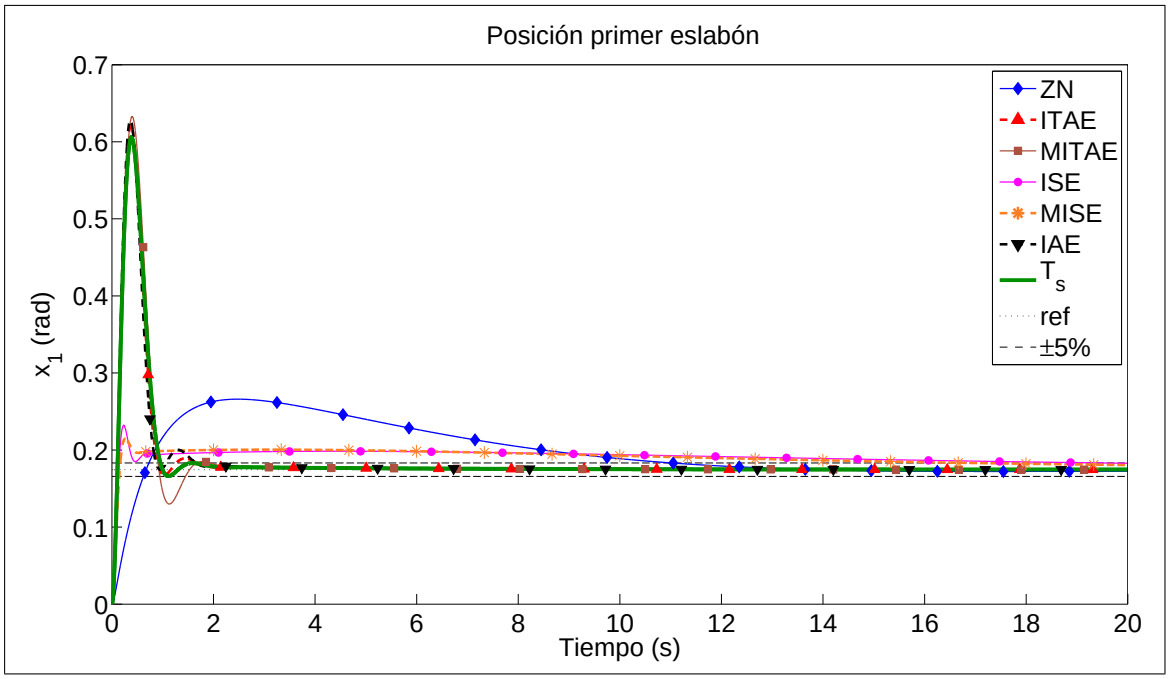


Figura 21: Posición angular x_1 usando PID sintonizado con Nelder y Mead.

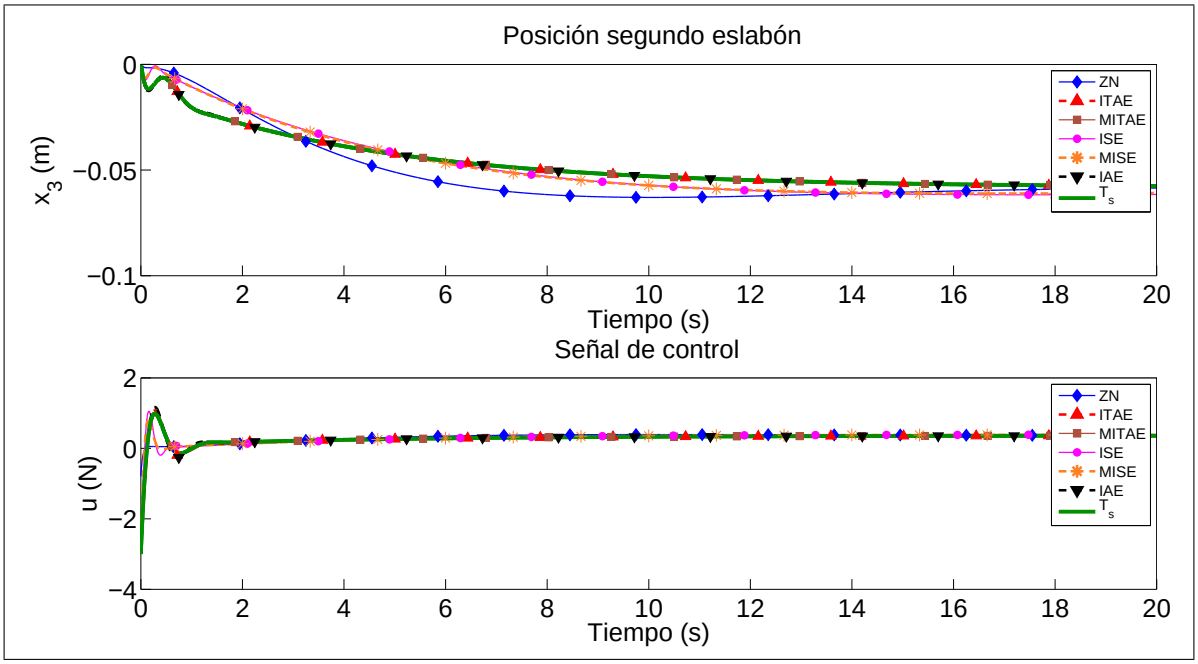


Figura 22: Posición de la barra x_3 y señal de control usando PID sintonizado con Nelder y Mead.

VI.1.3 Recocido simulado

A continuación se presentan los resultados obtenidos al sintonizar el PID utilizando el método de recocido simulado. Dado que el método utilizado es no determinístico, para poder compararlo con los otros dos métodos que sí lo son, se corre el método 30 veces, utilizando los parámetros de Ziegler-Nichols como semilla inicial. Con estas 30 corridas obtenemos el promedio y la desviación estándar para que sean estos datos los que comparemos con los otros dos métodos.

En la Tabla VII se muestra el promedio y desviación estándar del tiempo de establecimiento de las 30 corridas realizadas.

La Tabla VIII nos muestra el costo computacional, promedio y desviación estándar, requerido por las 30 corridas.

Ya que en la Tabla VII se muestran sólo los promedios obtenidos, se presenta en la Tabla IX, un caso particular para una optimización con cada una de las funciones objetivo. Se eligió el caso en el cual el tiempo de establecimiento es el más cercano al T_s promedio de las 30 corridas del algoritmo. En las figuras 23 y 24 se muestran las respuestas en el tiempo para este caso particular.

Tabla VII: Control PID sintonizado con recocido simulado; promedio y desviación estándar.

Función objetivo	T_s	
	Media	Desv. std.
ITAE	6.136	2.590
MITAE	3.982	2.546
ISE	12.316	4.747
MISE	12.150	4.409
IAE	8.820	2.685
T_s	2.085	0.265

Tabla VIII: Costo computacional al sintonizar el control PID con recocido simulado; promedio y desviación estándar.

Función objetivo	# iteraciones		Evaluaciones		Tiempo (s)	
	Media	Desv. std.	Media	Desv. std.	Media	Desv. std.
ITAE	32.90	45.29	187.93	15.51	416.33	171.55
MITAE	103.67	108.39	225.57	128.91	338.91	193.15
ISE	23.87	27.58	169.27	9.66	367.28	115.94
MISE	35.20	39.82	166.80	7.64	395.59	148.42
IAE	7.73	8.60	165.37	5.92	386.95	181.90
T_s	20.47	55.01	173.33	75.60	510.56	750.50

Tabla IX: PID sintonizado con recocido simulado; T_s cercano al promedio.

Función objetivo	K_p	T_i	T_d	T_s (s)
ITAE	-6.2165	1.5696	1.0059	6.154
MITAE	-2.6111	0.3135	0.4751	3.723
ISE	-7.8365	3.5799	0.6332	11.824
MISE	-6.6982	3.7149	0.2407	12.337
IAE	-3.8453	1.8872	0.2153	8.599
T_s	-5.0581	0.4215	0.1105	2.180

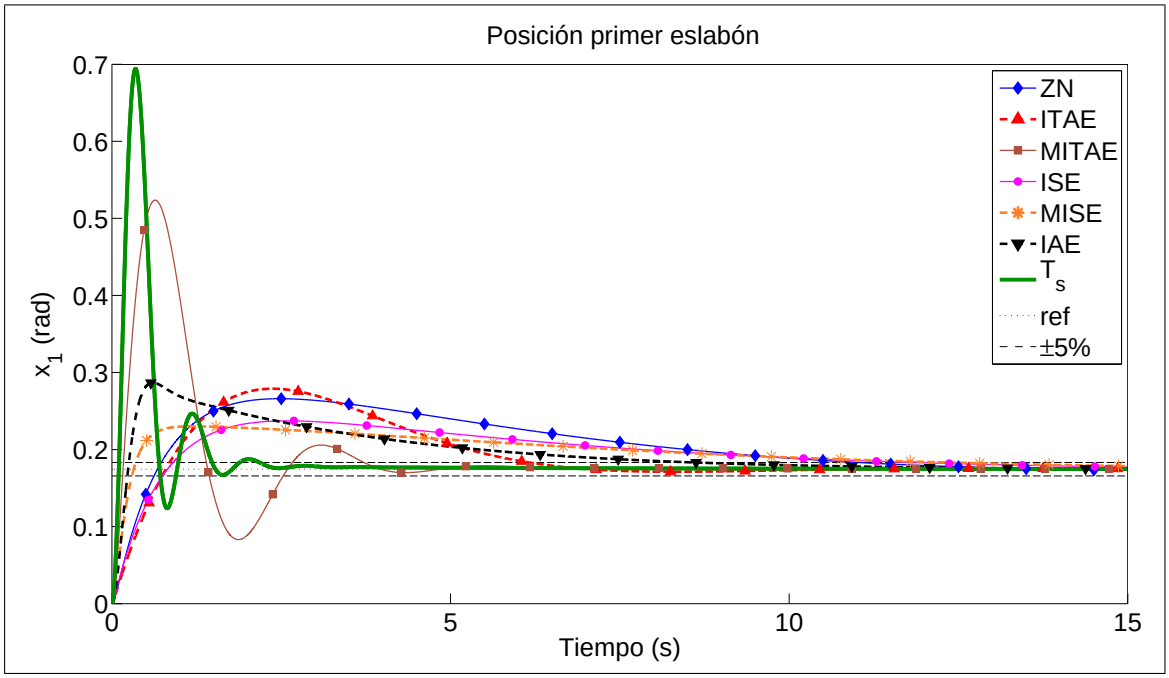


Figura 23: Posición angular x_1 usando PID sintonizado con recocido simulado.

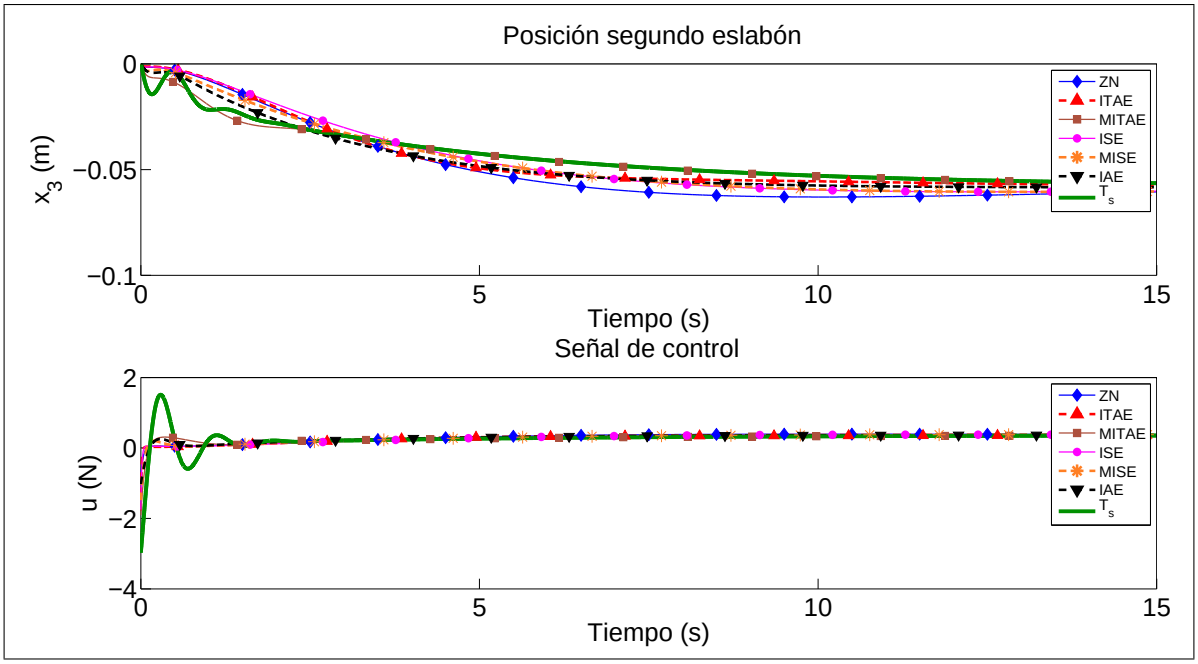


Figura 24: Posición de la barra x_3 y señal de control usando PID sintonizado con recocido simulado.

VI.2 Ubicación de polos

Se siguió la misma metodología para el control por ubicación de polos, en este caso se toma el vector $\mathbf{p} = [4, 9, 6, 5]^T$ como parámetros iniciales en la optimización. Los polos serán ubicados como dos pares de complejos conjugados: $-p_1 \pm p_2\, i$ y $-p_3 \pm p_4\, i$.

En la Figura 25 se muestra la evolución de los estados utilizando los parámetros iniciales. La Figura 26 muestra la señal de control.

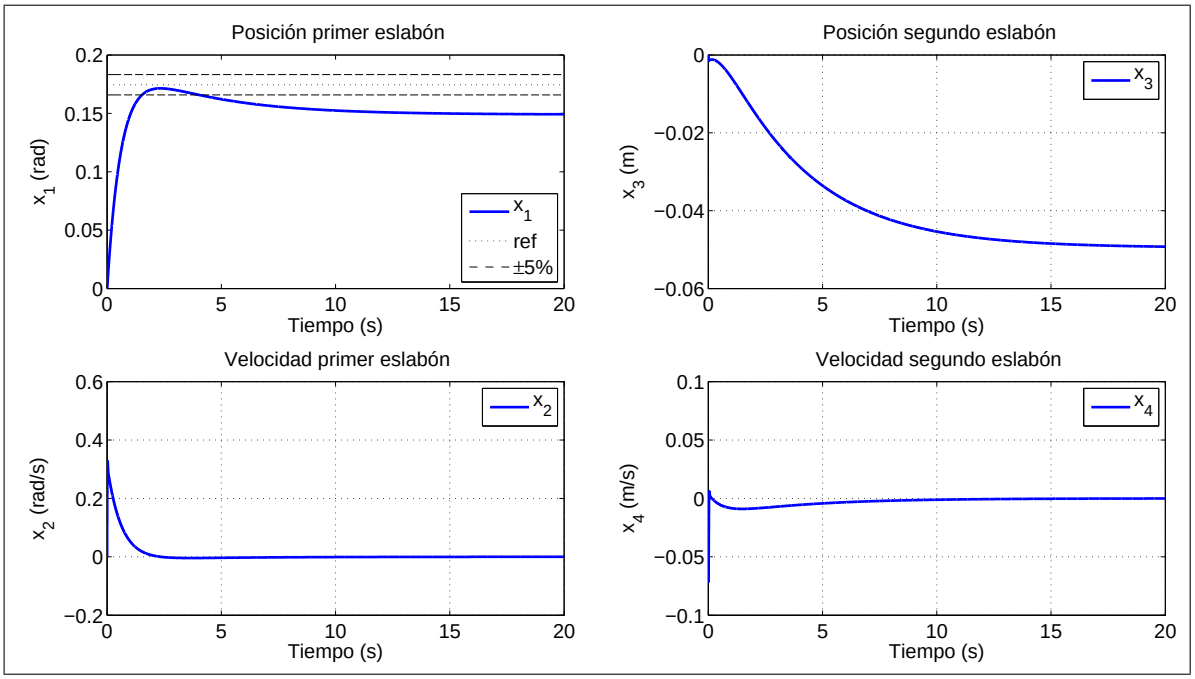


Figura 25: Respuesta ante un escalón de magnitud 0.1745 rad en la referencia, en lazo cerrado con un control por ubicación de polos, con parámetros iniciales.

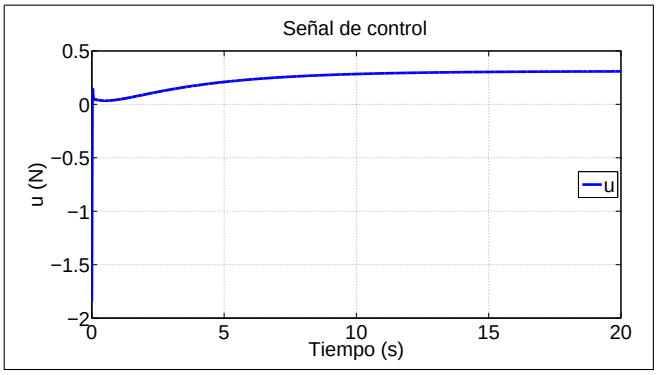


Figura 26: Control por ubicación de polos con parámetros iniciales.

Nótese que la respuesta no alcanza la referencia y no llega a la zona del ±5%.

A continuación se muestran los resultados obtenidos al sintonizar el control utilizando los tres métodos de optimización y las distintas funciones objetivo.

VI.2.1 Hooke y Jeeves

Partiendo del conjunto de polos definido por $\mathbf{p} = [4, 9, 6, 5]^T$ se realizó la sintonización de los parámetros para las distintas funciones objetivo utilizando el método de Hooke y Jeeves. En la Tabla X se muestran los resultados obtenidos. Para minimizar T_s, se utilizan los parámetros obtenidos usando el criterio ISE. La Tabla XI nos muestra el costo computacional requerido en cada caso.

Tabla X: Ubicación de polos sintonizado con Hooke y Jeeves.

Función objetivo	p_1	p_2	p_3	p_4	T_s (s)
ITAE	3.5503	9.4968	6.5005	5.0027	1.174
MITAE	3.1387	8.8281	5.7812	5.7129	1.095
ISE	3.5625	8.6875	6.6875	5.9375	1.083
MISE	3.7500	8.9688	6.5938	5.5312	1.164
IAE	3.5430	9.4766	6.4766	5.0234	1.174
T_s	3.5469	8.6875	7.3125	5.9375	1.037

Tabla XI: Costo computacional al sintonizar el control por ubicación de polos con Hooke y Jeeves.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	27	216	346.58
MITAE	200	1551	2486.09
ISE	16	123	197.46
MISE	18	140	226.81
IAE	21	169	273.88
T_s	23	186	345.22

En la Figura 27 se muestra la gráfica de las respuestas temporales con los parámetros obtenidos para las distintas funciones objetivo. La señal S_0 es la obtenida con los parámetros iniciales. Se puede observar que aunque con los parámetros iniciales no se llega a la referencia, una vez sintonizados sí se alcanza a entrar en la zona de $\pm 5\%$. La Figura 28 nos muestra la posición del segundo eslabón y la señal de control. Podemos observar que se cumple con las restricciones que se establecieron en ambos casos.

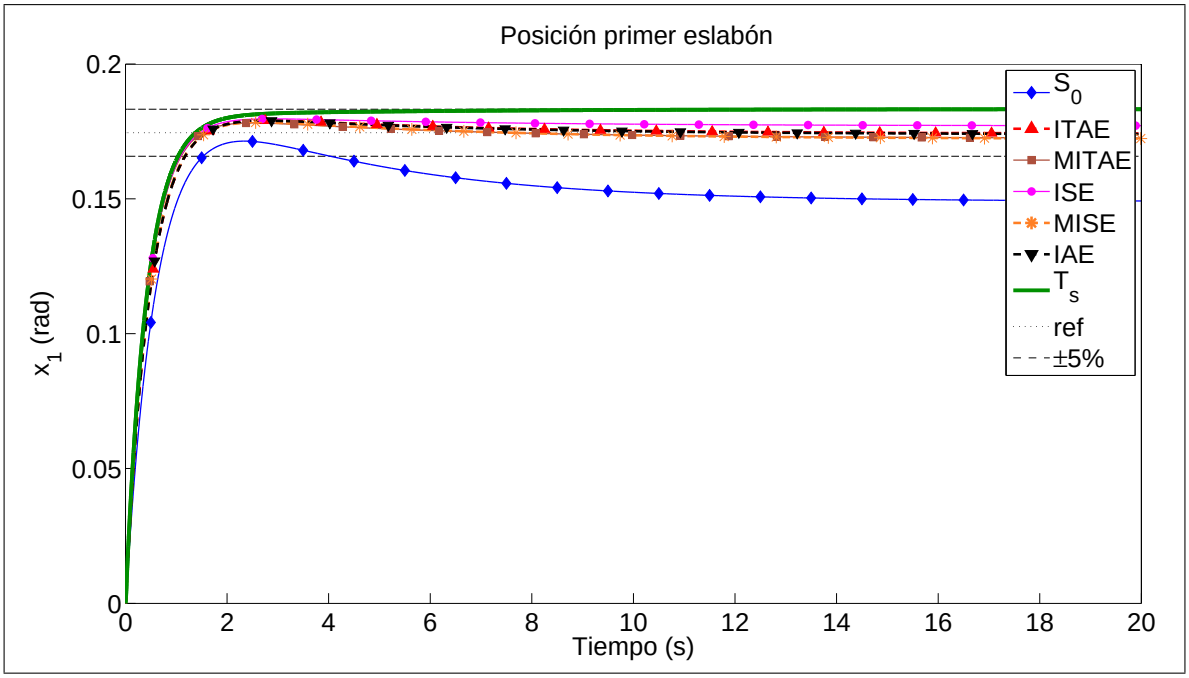


Figura 27: Posición angular x_1 usando ubicación de polos sintonizado con Hooke y Jeeves.

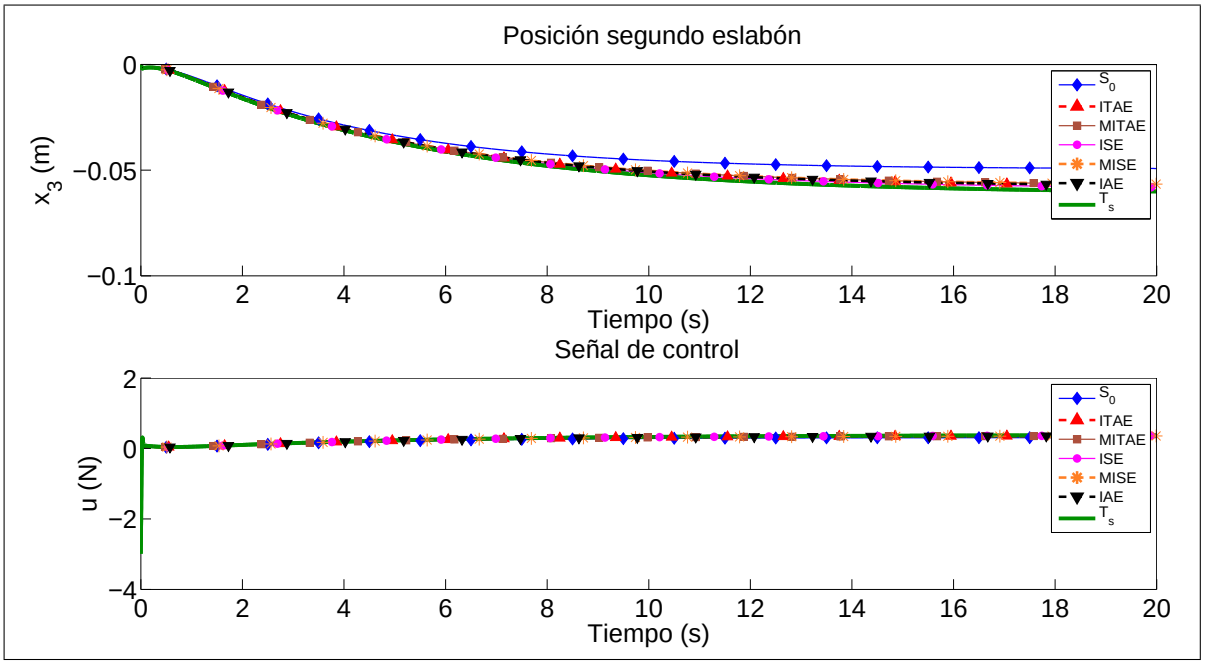


Figura 28: Posición de la barra x_3 y señal de control usando ubicación de polos sintonizado con Hooke y Jeeves.

VI.2.2 Nelder y Mead

En la Tabla XII se muestran los resultados al hacer la sintonización utilizando el método de Nelder y Mead. Para minimizar T_s, se utilizaron los parámetros obtenidos con el criterio MISE. En la Tabla XIII se muestra el costo computacional requerido por la sintonización.

Tabla XII: Ubicación de polos sintonizado con Nelder y Mead.

Función objetivo	p_1	p_2	p_3	p_4	T_s (s)
ITAE	1.45×10^{-4}	5.4767	5.1000	7.3622	0.613
MITAE	7.73×10^{-6}	5.4705	7.0261	6.8608	0.524
ISE	1.5409	8.4841	1.3208	5.8061	0.345
MISE	1.6281	9.0348	1.1764	5.7132	0.326
IAE	3.9615	11.8495	0.3431	5.5660	0.428
T_s	1.6998	9.5324	1.0288	5.8684	0.272

Tabla XIII: Costo computacional al sintonizar el control por ubicación de polos con Nelder y Mead.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	129	199	311.03
MITAE	108	158	241.38
ISE	52	94	139.36
MISE	53	93	140.90
IAE	157	263	391.70
T_s	39	90	149.34

En la Figura 29 se muestra la gráfica de la respuesta ante el escalón en la referencia utilizando los distintos parámetros obtenidos en la sintonización. La Figura 30 muestra la gráfica de la posición del segundo eslabón y la señal de control.

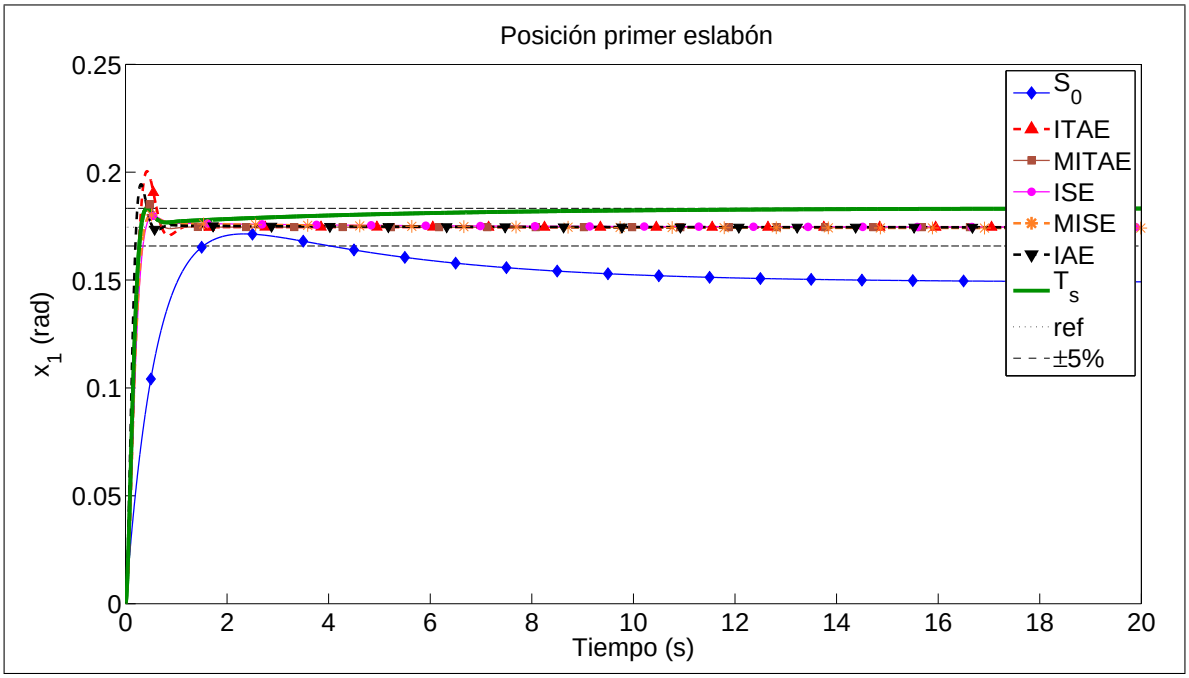


Figura 29: Posición angular x_1 usando ubicación de polos sintonizado con Nelder y Mead.

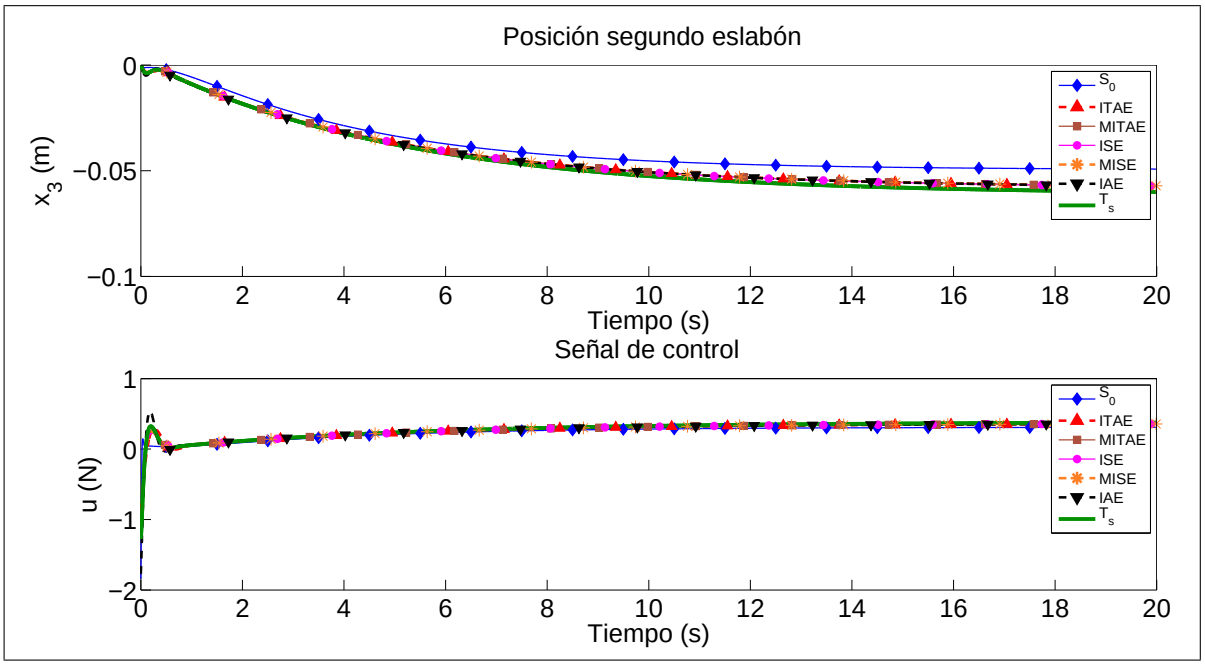


Figura 30: Posición de la barra x_3 y señal de control usando ubicación de polos sintonizado con Nelder y Mead.

VI.2.3 Recocido simulado

En las tablas XIV y XV, se muestran los promedios y desviación estándar de los tiempos de establecimiento y costo computacional, respectivamente.

Tabla XIV: Ubicación de polos sintonizado con recocido simulado; promedio y desviación estándar.

Función objetivo	T_s	
	Media	Desv. std.
ITAE	2.280	4.841
MITAE	5.357	8.221
ISE	5.291	8.258
MISE	6.515	8.983
IAE	5.317	8.243
T_s	0.278	0.005

Tabla XV: Costo computacional al sintonizar el control por ubicación de polos con recocido simulado; promedio y desviación estándar.

Función objetivo	# iteraciones		Evaluaciones		Tiempo (s)	
	Media	Desv. std.	Media	Desv. std.	Media	Desv. std.
ITAE	124.13	93.21	211.03	90.65	412.38	491.67
MITAE	77.43	85.87	200.90	98.90	315.43	153.45
ISE	32.23	36.97	148.73	31.08	249.78	59.04
MISE	33.47	32.76	157.73	26.42	265.05	59.40
IAE	27.37	31.89	160.03	57.65	268.63	101.35
T_s	0.13	0.73	28.23	53.39	181.06	714.34

En la Tabla XVI se muestran los parámetros con los que se obtuvieron los tiempos de establecimiento más cercanos a los tiempos promedio mostrados en la Tabla XIV. En las figuras 31 y 32 se muestran las respuestas temporales y señales de control.

Tabla XVI: Ubicación de polos sintonizado con recocido simulado; T_s cercano al promedio.

Función objetivo	p_1	p_2	p_3	p_4	T_s (s)
ITAE	2.8486	16.8360	0.3267	5.4846	3.111
MITAE	0.4726	9.2136	5.5594	2.1153	1.376
ISE	0.6687	5.5249	0.3046	7.6176	1.437
MISE	1.8816	9.4842	6.3702	3.5582	1.196
IAE	3.3513	10.2406	7.3544	3.2123	1.275
T_s	2.4924	14.0900	0.8082	5.5250	0.279

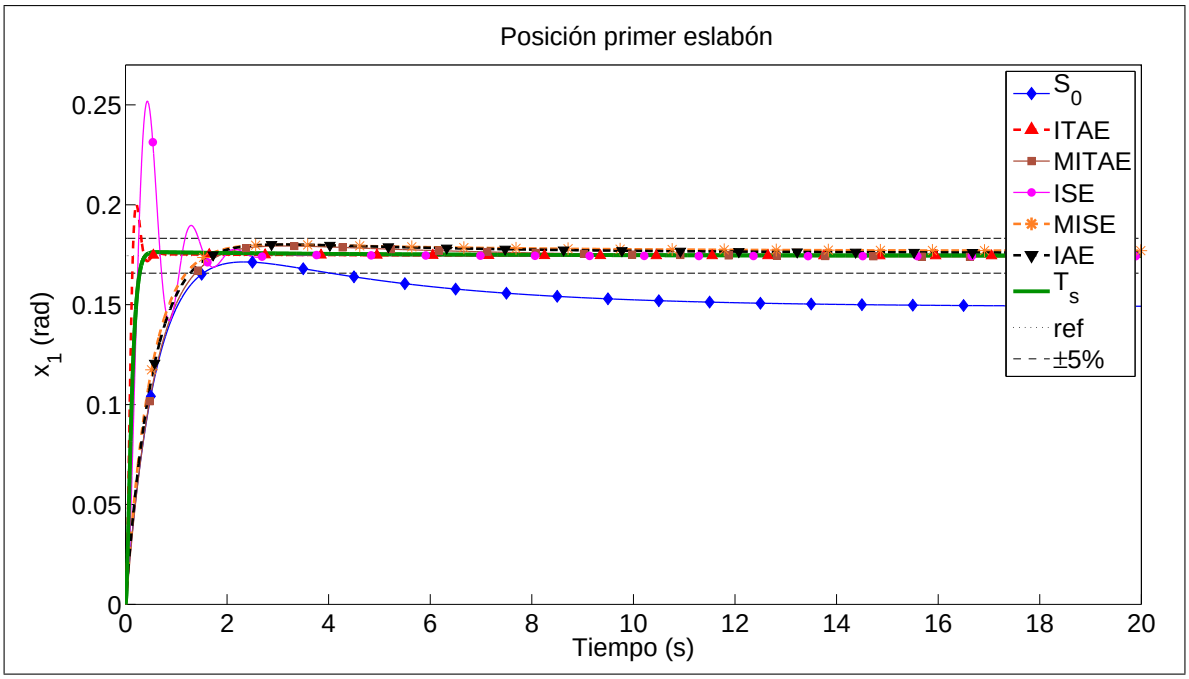


Figura 31: Posición angular x_1 usando ubicación de polos sintonizado con recocido simulado.

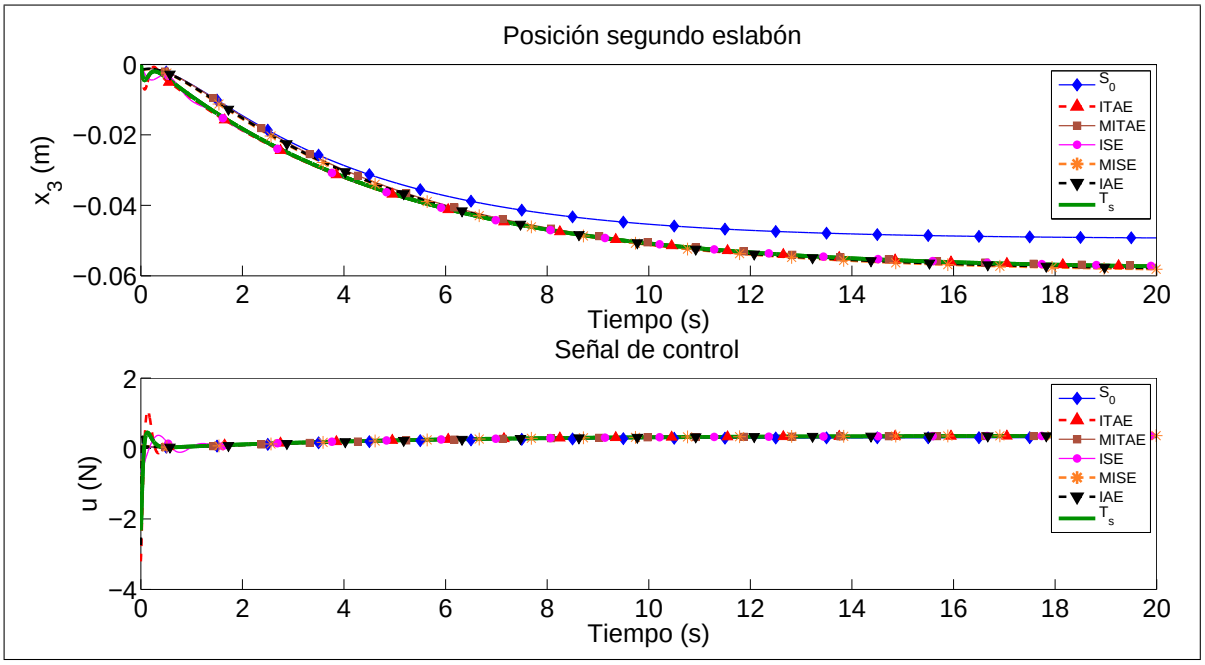


Figura 32: Posición de la barra x_3 y señal de control usando ubicación de polos sintonizado con recocido simulado.

VI.3 *Backstepping* caso 1

Para *backstepping* se presentan dos casos. En este primer caso se sintonizan las tres ganancias del controlador que se presentaron en la Sección III.3.2. Como parámetros iniciales se tiene el vector $\mathbf{k} = [1, 0, 1]^T$. La respuesta del sistema en lazo cerrado, cuando la referencia es un escalón de magnitud 0.1745 rad se presenta en la Figura 33. El tiempo de establecimiento obtenido es de 3.744 segundos. El esfuerzo control se muestra en la Figura 34.

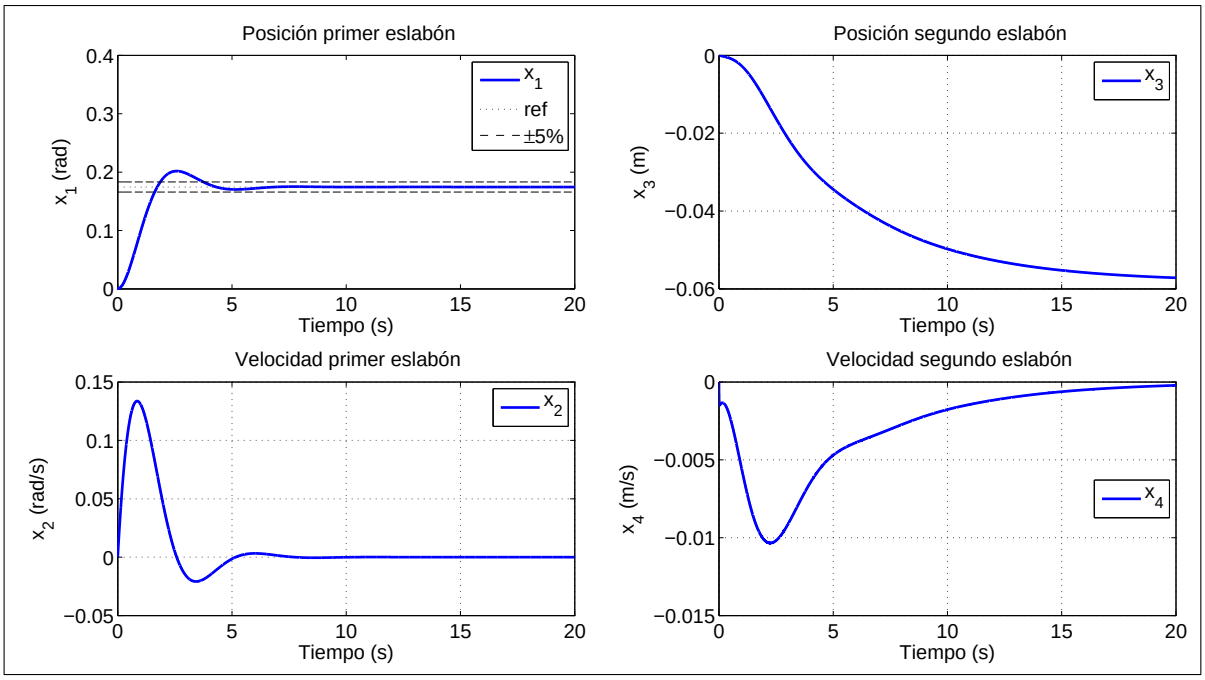


Figura 33: Respuesta ante un escalón de magnitud 0.1745 rad en la referencia, en lazo cerrado con un control *backstepping*, con parámetros iniciales.

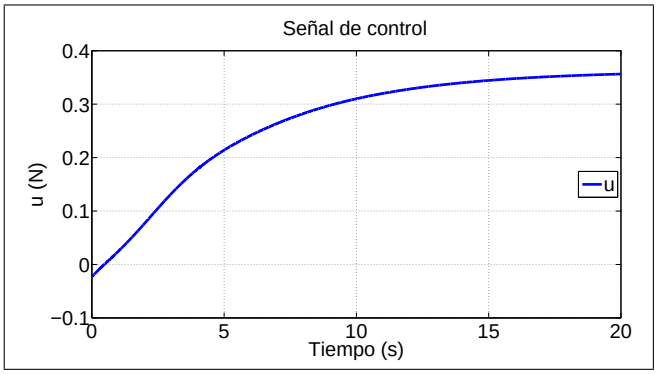


Figura 34: Control *backstepping* con parámetros iniciales.

VI.3.1 Hooke y Jeeves

En la Tabla XVII, se muestran los parámetros y tiempos de establecimiento obtenidos con cada función objetivo al sintonizar con Hooke y Jeeves. Para la sintonización utilizando T_s como función objetivo, se toman los parámetros obtenidos con el IAE. La Tabla XVIII muestra el costo computacional requerido para la sintonización.

Tabla XVII: *Backstepping* caso 1 sintonizado con Hooke y Jeeves.

Función objetivo	k_1	k_2	k_3	T_s (s)
ITAE	1.0098	0.5000	1.0000	0.776
MITAE	1.0098	0.5000	1.0000	0.776
ISE	19.1699	0.5000	11.5000	0.217
MISE	19.1699	0.5000	11.5000	0.217
IAE	18.7020	0.5000	11.7500	0.215
T_s	18.7020	0.5000	11.7500	0.215

Tabla XVIII: Costo computacional al sintonizar *backstepping* caso 1 con Hooke y Jeeves.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	19	115	293.32
MITAE	19	115	291.63
ISE	35	191	503.94
MISE	35	191	503.91
IAE	97	526	1359.10
T_s	16	91	245.39

En la Figura 35 se muestra la respuesta en el tiempo con los distintos parámetros. La Figura 36 muestra la posición del segundo eslabón y la señal de control.

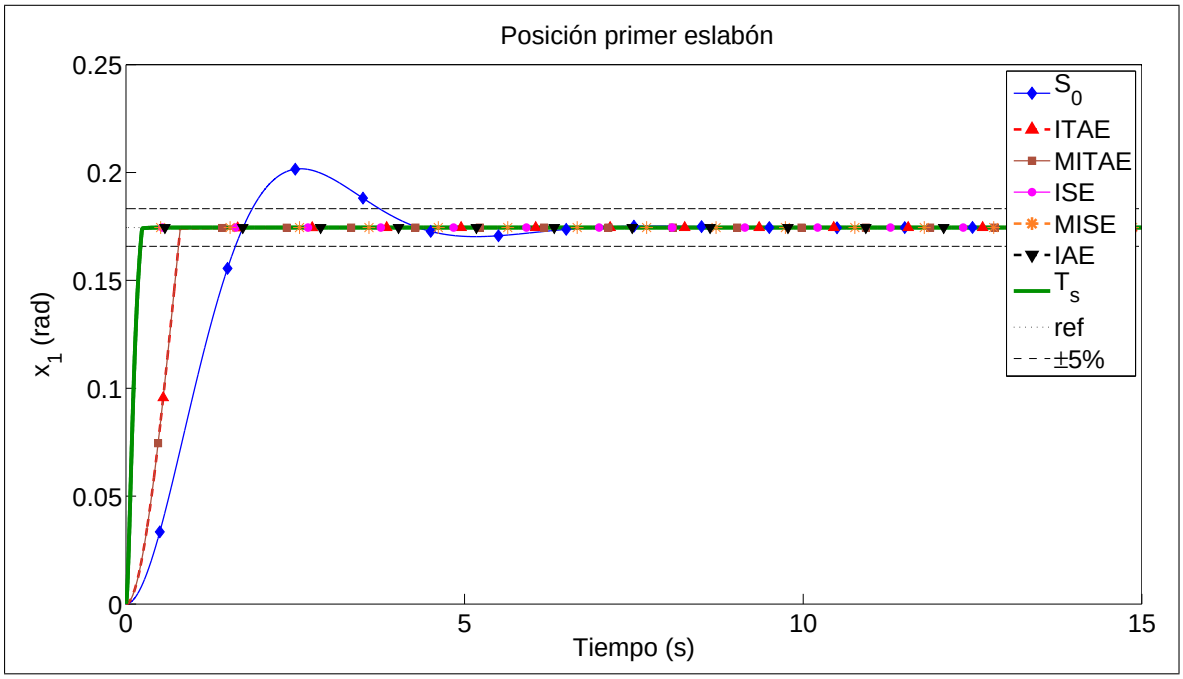


Figura 35: Posición angular x_1 usando *backstepping* caso 1 sintonizado con Hooke y Jeeves.

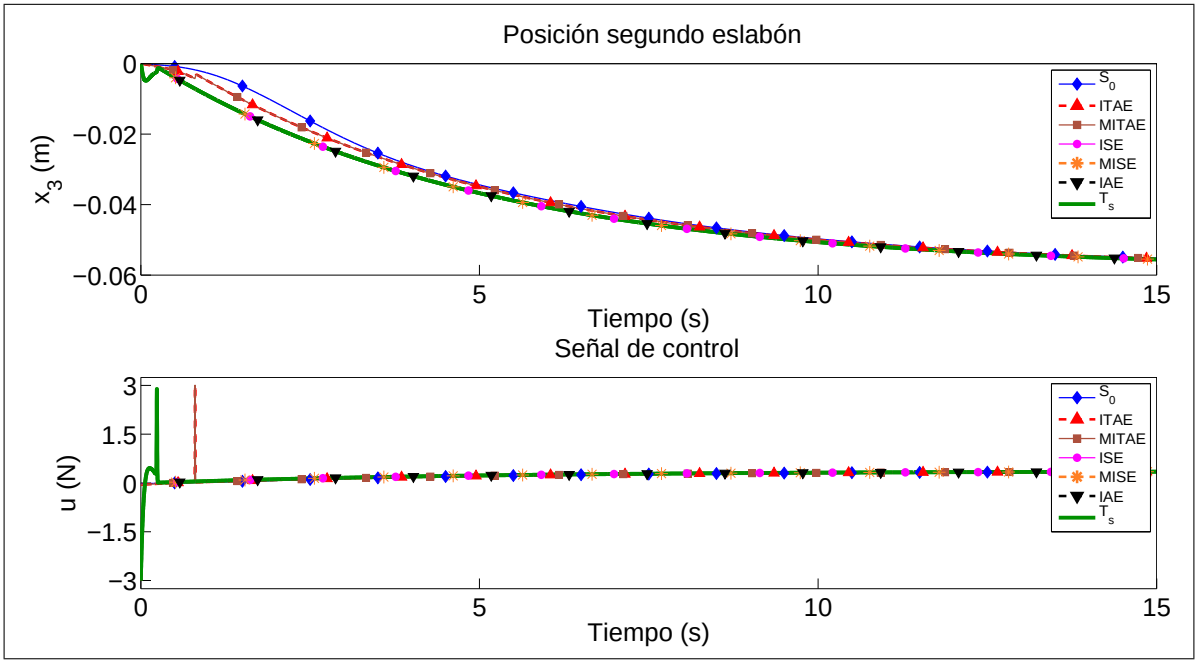


Figura 36: Posición de la barra x_3 y señal de control usando *backstepping* caso 1 sintonizado con Hooke y Jeeves.

VI.3.2 Nelder y Mead

La Tabla XIX muestra los parámetros y tiempos de establecimiento obtenidos al sintonizar utilizando Nelder y Mead con las distintas funciones objetivo. En la Tabla XX se muestra el costo computacional requerido.

Tabla XIX: *Backstepping* caso 1 sintonizado con Nelder y Mead.

Función objetivo	k_1	k_2	k_3	T_s (s)
ITAE	0.4601	0.3246	2.7408	0.685
MITAE	0.2644	0.3190	8.9541	0.556
ISE	17.8433	0.2391	13.0854	0.239
MISE	17.8433	0.2391	13.0854	0.239
IAE	0.0635	0.3091	4.4846	0.659
T_s	17.8194	0.2776	13.0553	0.232

Tabla XX: Costo computacional al sintonizar *backstepping* caso 1 con Nelder y Mead.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	60	120	257.58
MITAE	85	157	345.64
ISE	40	81	159.13
MISE	40	81	159.73
IAE	91	172	363.99
T_s	30	77	162.06

En las figuras 37 y 38 se muestran las posiciones de los eslabones y las señales de control, ante un escalón en la referencia.

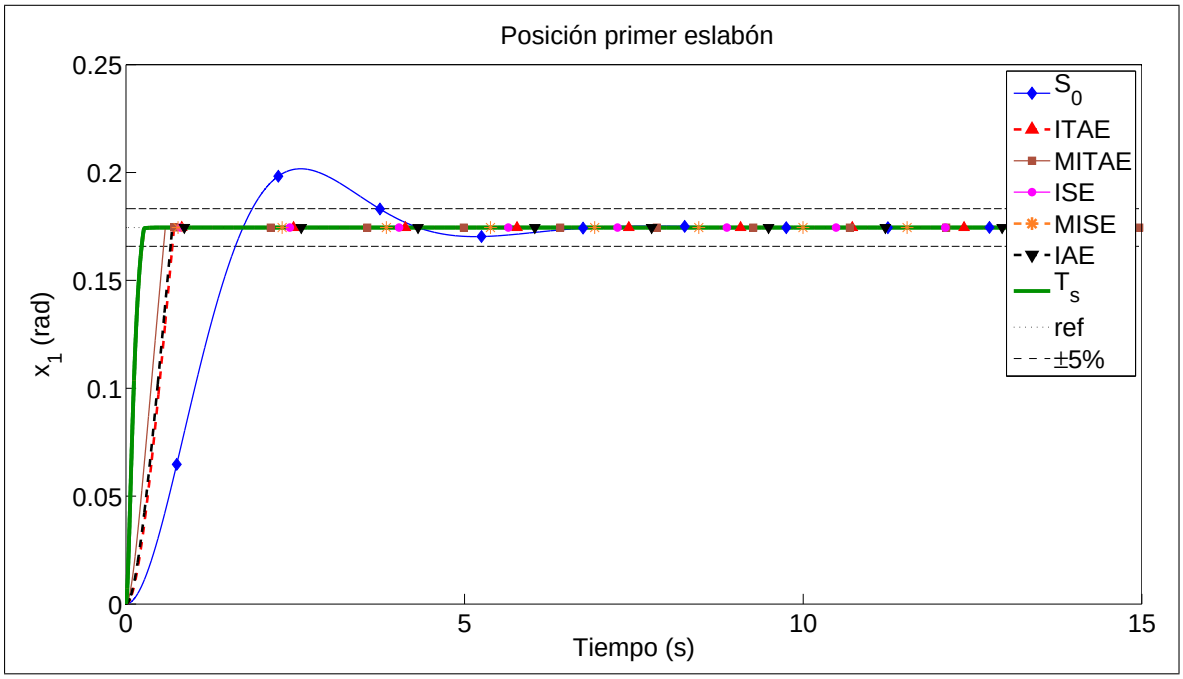


Figura 37: Posición angular x_1 usando *backstepping* caso 1 sintonizado con Nelder y Mead.

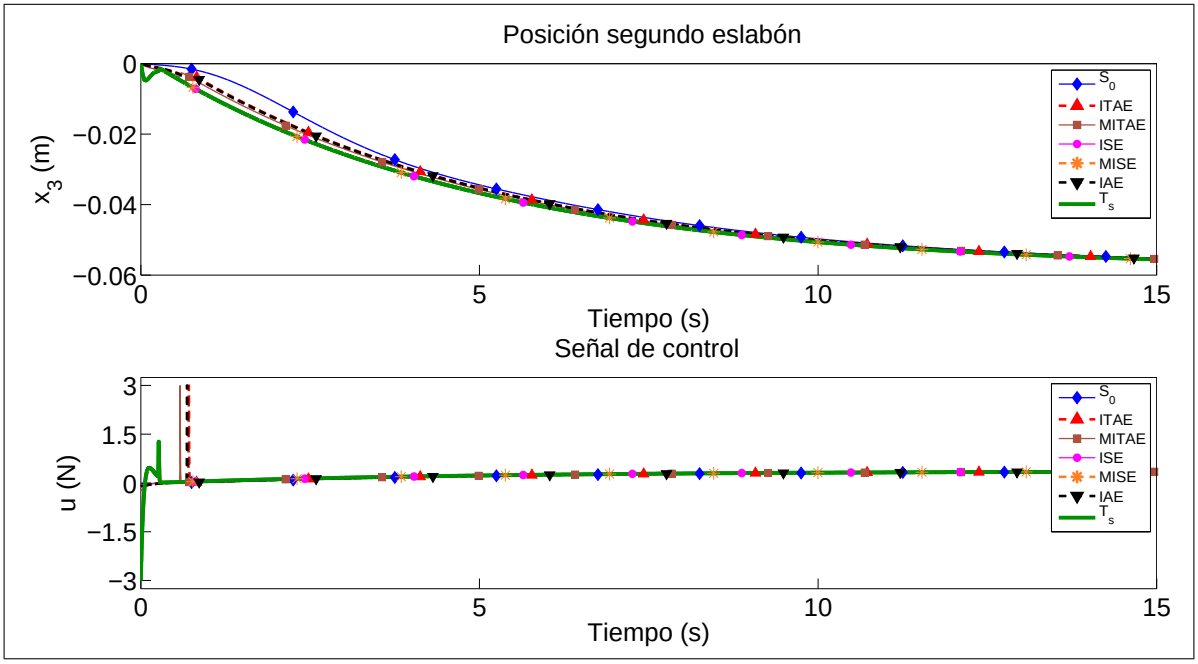


Figura 38: Posición de la barra x_3 y señal de control usando *backstepping* caso 1 sintonizado con Nelder y Mead.

VI.3.3 Recocido simulado

En las tablas XXI y XXII se muestran los promedios y desviación estándar de los tiempos de establecimiento y costo computacional al sintonizar con recocido simulado.

Tabla XXI: *Backstepping* caso 1 sintonizado con recocido simulado; promedio y desviación estándar.

Función objetivo	T_s	
	Media	Desv. std.
ITAE	0.117	0.079
MITAE	0.189	0.136
ISE	1.104	1.250
MISE	0.562	0.652
IAE	0.760	1.051
T_s	0.280	0.021

Tabla XXII: Costo computacional al sintonizar *backstepping* caso 1 con recocido simulado; promedio y desviación estándar.

Función objetivo	# iteraciones		Evaluaciones		Tiempo (s)	
	Media	Desv. std.	Media	Desv. std.	Media	Desv. std.
ITAE	46.70	35.40	165.77	11.98	1578.89	581.75
MITAE	46.63	30.89	164.83	9.11	1552.60	785.63
ISE	24.20	25.19	162.43	5.63	1314.70	510.56
MISE	37.50	36.44	163.27	8.30	1546.70	877.03
IAE	29.37	28.23	167.30	8.19	1767.30	848.78
T_s	6.67	15.19	162.47	9.15	2822.80	2235.30

En la Tabla XXIII se muestran los parámetros y tiempos de establecimiento de un caso particular, este caso fue elegido de entre las 30 corridas, siendo estos parámetros los que generaban el tiempo de establecimiento más cercano al promedio de las 30 corridas.

En las figuras 39 y 40 se muestra la respuesta temporal y las señales de control para los parámetros de la Tabla XXIII. Se puede apreciar que las señales de control

correspondientes a los criterios ITAE y MITAE exceden la restricción establecida de $3\,N$, a pesar de que la solución es penalizada en estos casos.

Tabla XXIII: *Backstepping* caso 1 sintonizado con recocido simulado; T_s cercano al promedio.

Función objetivo	k_1	k_2	k_3	T_s (s)
ITAE	0.0438	4.9517	2.9429	0.160
MITAE	1.3514	2.0924	3.1863	0.246
ISE	2.7844	0.0825	1.6041	1.139
MISE	7.4049	1.6897	1.1488	0.561
IAE	1.4855	0.5087	1.1078	0.754
T_s	8.5816	0.2285	14.0590	0.288

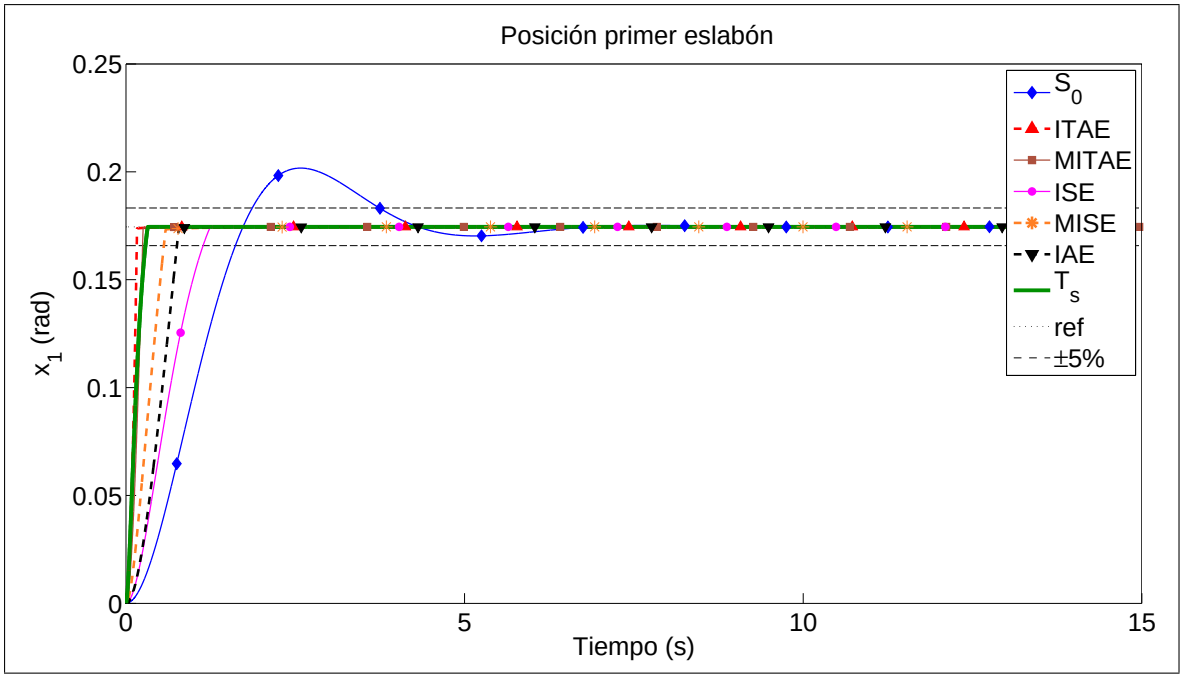


Figura 39: Posición angular x_1 usando *backstepping* caso 1 sintonizado con recocido simulado.

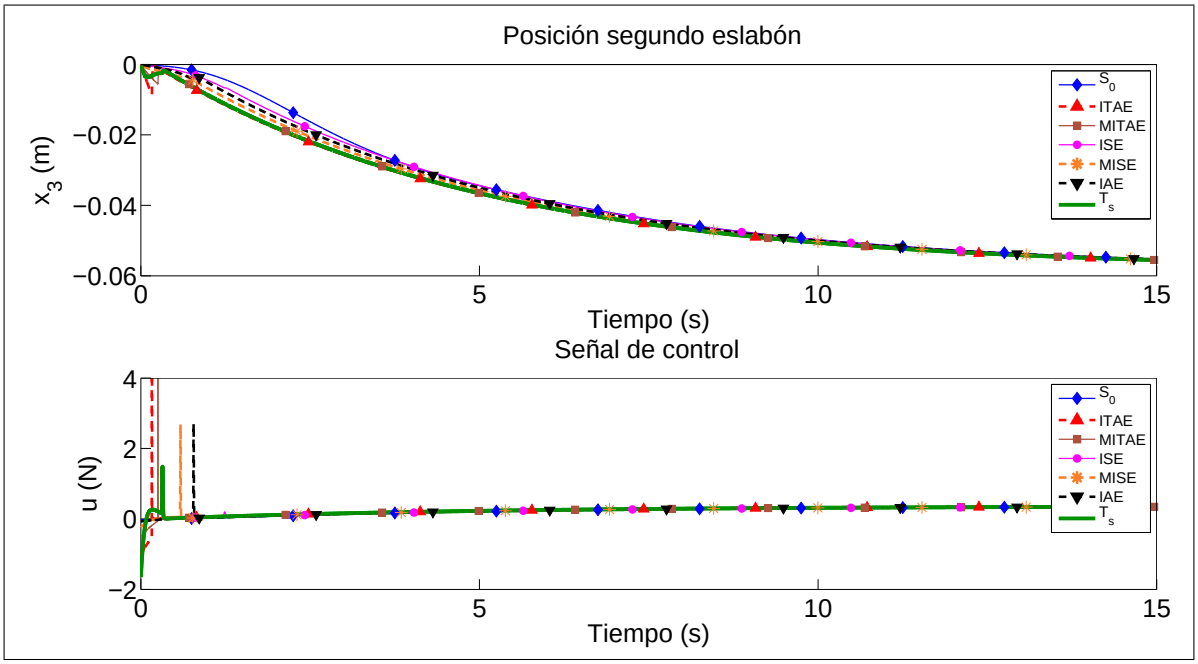


Figura 40: Posición de la barra x_3 y señal de control usando *backstepping* caso 1 sintonizado con recocido simulado.

VI.4 *Backstepping* caso 2

Para este segundo caso se vuelve a utilizar el mismo vector de parámetros iniciales $\mathbf{k} = [1, 0, 1]^T$, cuya respuesta se muestra en las figuras 33 y 34. La diferencia con el caso anterior es que sólo se sintonizan dos ganancias: k_1 y k_3. La ganancia correspondiente a la función signo (en realidad $\text{sech}^2(\lambda\, x_1)$), k_2, permanece en cero.

VI.4.1 Hooke y Jeeves

En la Tabla XXIV se muestran los parámetros obtenidos al sintonizar con Hooke y Jeeves. La Tabla XXV muestra el costo computacional requerido.

Tabla XXIV: *Backstepping* caso 2 sintonizado con Hooke y Jeeves.

Función objetivo	k_1	k_2	k_3	T_s (s)
ITAE	16.3486	0	15.5000	0.289
MITAE	16.3486	0	15.5000	0.289
ISE	16.3477	0	15.5000	0.289
MISE	16.3477	0	15.5000	0.289
IAE	16.3491	0	15.5000	0.289
T_s	16.3486	0	15.5000	0.289

Tabla XXV: Costo computacional al sintonizar *backstepping* caso 2 con Hooke y Jeeves.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	31	106	163.01
MITAE	31	106	163.12
ISE	30	102	156.95
MISE	30	102	156.92
IAE	34	118	181.63
T_s	16	61	95.70

En las figuras 41 y 42 se muestra la respuesta temporal y la respectiva señal de control.

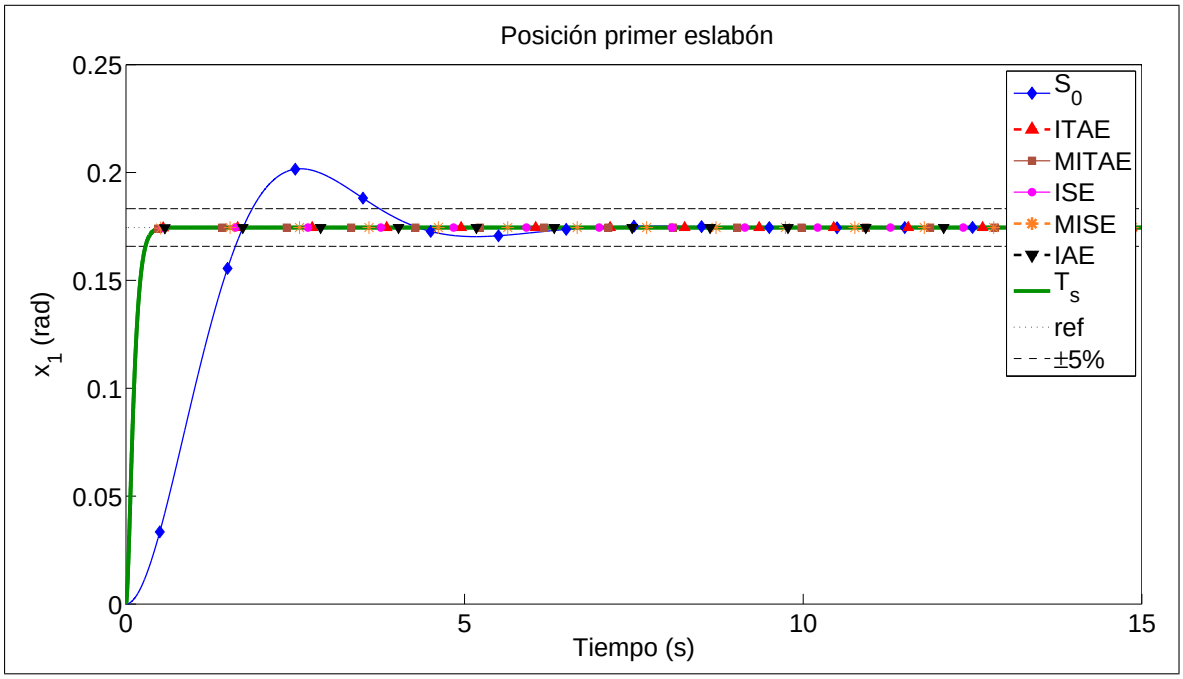


Figura 41: Posición angular x_1 usando *backstepping* caso 2 sintonizado con Hooke y Jeeves.

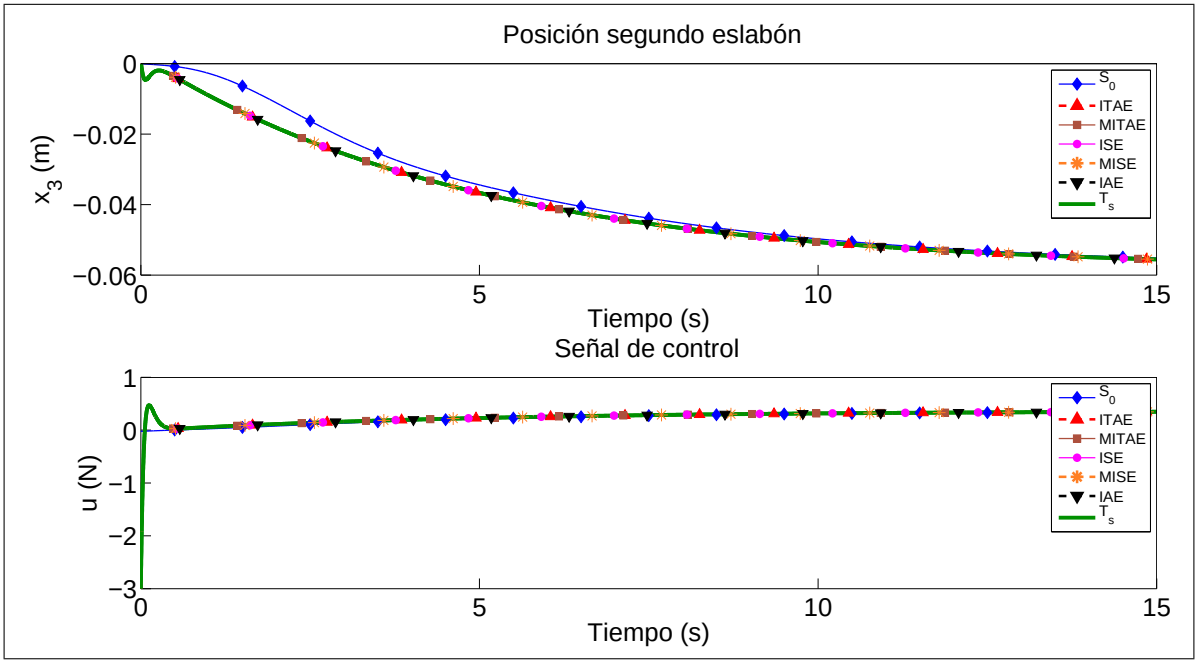


Figura 42: Posición de la barra x_3 y señal de control usando *backstepping* caso 2 sintonizado con Hooke y Jeeves.

VI.4.2 Nelder y Mead

Las tablas XXVI y XXVII muestran los resultados de la sintonización y el costo computacional requerido utilizando Nelder y Mead.

Tabla XXVI: *Backstepping* caso 2 sintonizado con Nelder y Mead.

Función objetivo	k_1	k_2	k_3	T_s (s)
ITAE	15.8877	0	15.9502	0.289
MITAE	15.3219	0	16.4862	0.290
ISE	15.7891	0	16.0391	0.289
MISE	15.7891	0	16.0391	0.289
IAE	15.7627	0	16.0752	0.289
T_s	15.8877	0	15.9502	0.289

Tabla XXVII: Costo computacional al sintonizar *backstepping* caso 2 con Nelder y Mead.

Función objetivo	# iteraciones	Evaluaciones	Tiempo (s)
ITAE	31	56	86.196
MITAE	26	52	79.869
ISE	30	56	86.053
MISE	30	56	86.172
IAE	35	64	98.622
T_s	15	52	81.489

En las figuras 43 y 44 se muestra la respuesta temporal y la señal de control con los parámetros sintonizados.

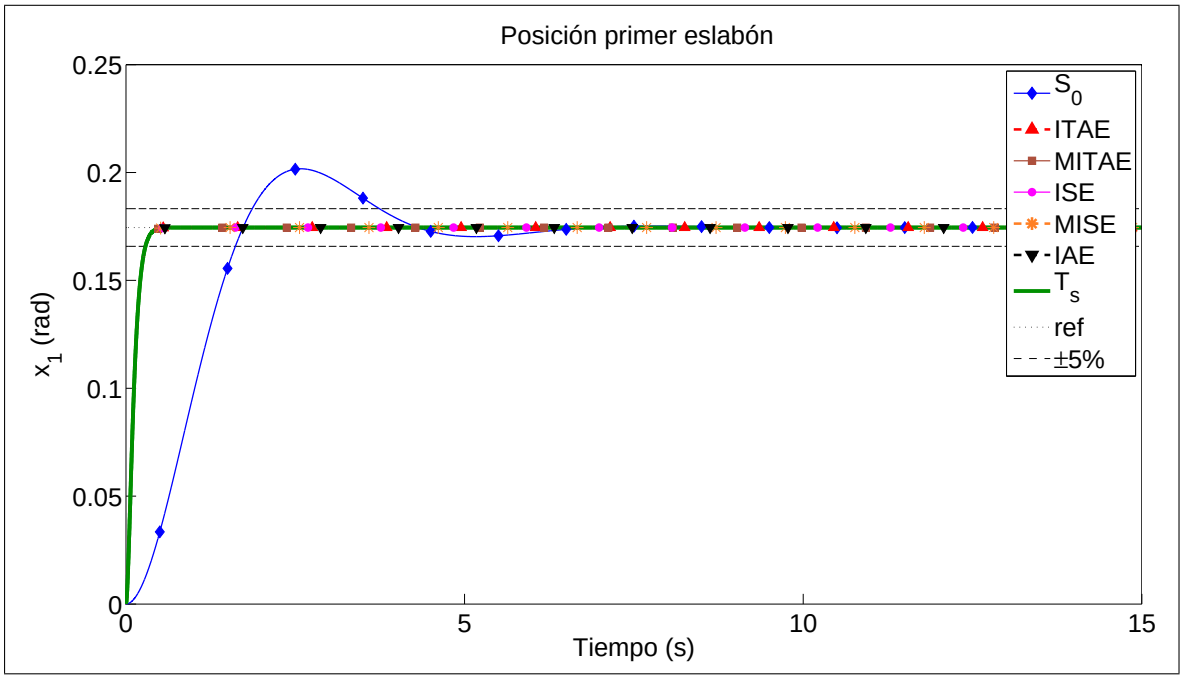


Figura 43: Posición angular x_1 usando *backstepping* caso 2 sintonizado con Nelder y Mead.

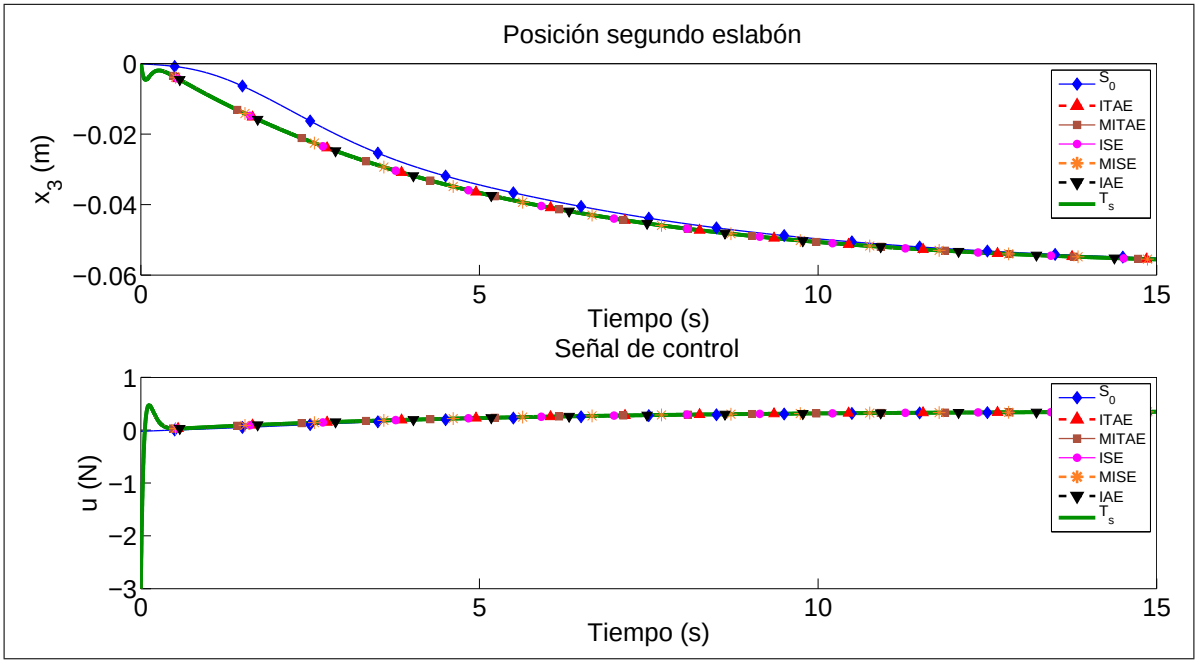


Figura 44: Posición de la barra x_3 y señal de control usando *backstepping* caso 2 sintonizado con Nelder y Mead.

VI.4.3 Recocido simulado

En las tablas XXVIII y XXIX se muestran los promedios y desviación estándar de los tiempos de establecimiento y costo computacional de las 30 corridas con recocido simulado.

Tabla XXVIII: *Backstepping* caso 2 sintonizado con recocido simulado; promedios y desviación estándar.

Función objetivo	T_s	
	Media	Desv. std.
ITAE	0.724	0.216
MITAE	0.679	0.141
ISE	0.776	0.274
MISE	0.729	0.227
IAE	0.673	0.213
T_s	0.318	0.024

Tabla XXIX: Costo computacional al sintonizar *backstepping* caso 2 con recocido simulado; promedios y desviación estándar.

Función objetivo	# iteraciones		Evaluaciones		Tiempo (s)	
	Media	Desv. std.	Media	Desv. std.	Media	Desv. std.
ITAE	55.03	39.94	163.60	6.92	254.05	9.85
MITAE	61.80	38.71	159.40	4.04	249.94	5.09
ISE	44.70	37.63	161.13	6.02	250.55	8.22
MISE	47.33	30.31	161.77	7.12	251.06	9.66
IAE	54.30	25.39	159.90	5.44	282.20	187.17
T_s	15.40	17.16	156.10	4.49	247.27	6.83

En la Tabla XXX se muestra los parámetros y tiempos de establecimiento que corresponden a una corrida que se acerca más al promedio de los tiempos de establecimiento de las 30. En la Figura 45 se muestra la respuesta con dichos parámetros. En la Figura 46 se muestra la posición del segundo eslabón y la señal de control.

Tabla XXX: *Backstepping* caso 2 sintonizado con recocido simulado; T_s cercano al promedio.

Función objetivo	k_1	k_2	k_3	T_s (s)
ITAE	5.9891	0	5.7674	0.720
MITAE	5.8952	0	6.7395	0.681
ISE	4.2114	0	8.4052	0.786
MISE	4.6818	0	7.9251	0.740
IAE	4.9569	0	9.9909	0.681
T_s	13.1512	0	16.1474	0.317

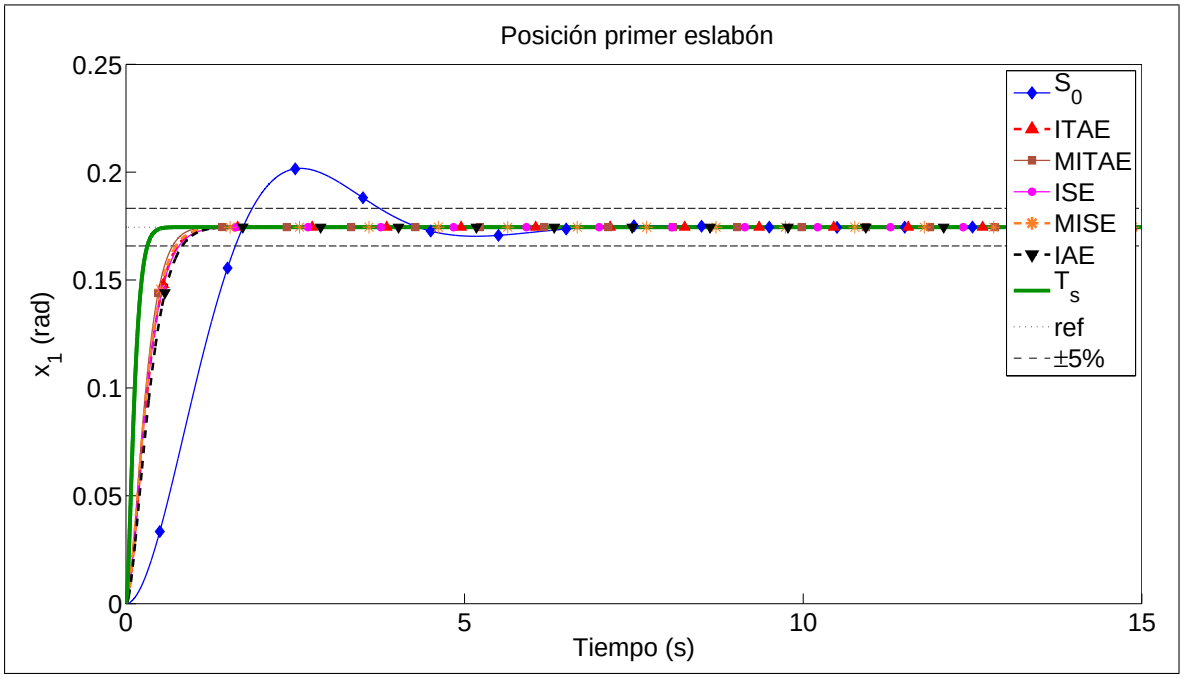


Figura 45: Posición angular x_1 usando *backstepping* caso 2 sintonizado con recocido simulado.

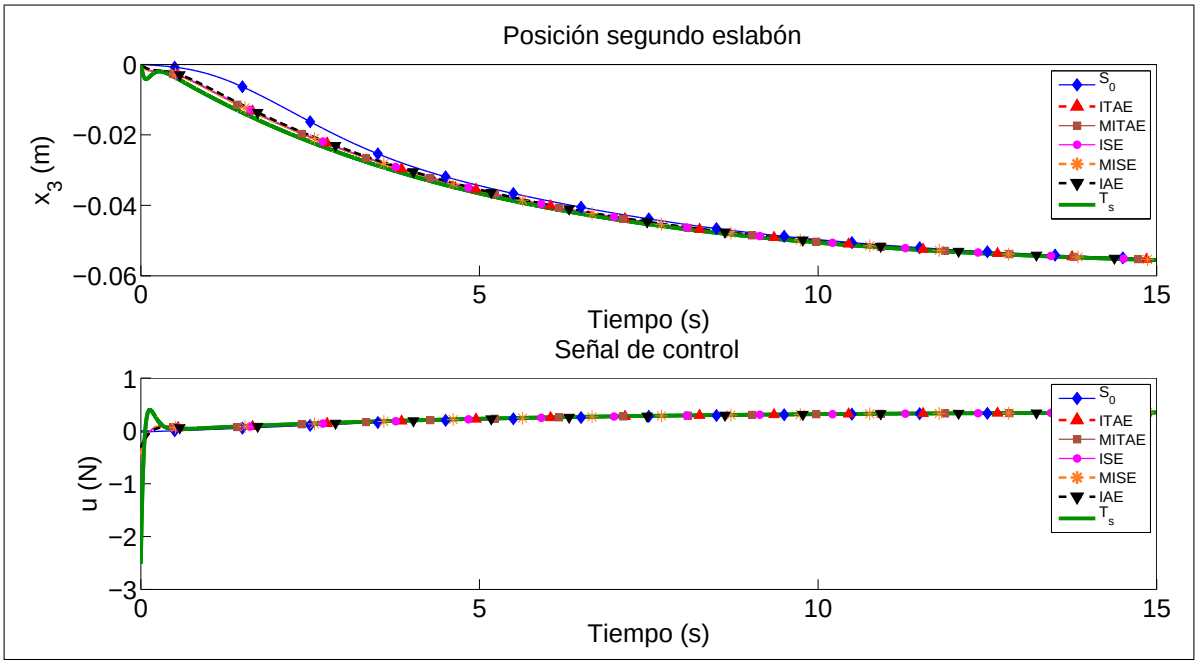


Figura 46: Posición de la barra x_3 y señal de control usando *backstepping* caso 2 sintonizado con recocido simulado.

Capítulo VII

Resultados experimentales

En este capítulo se muestran los resultados obtenidos experimentalmente en el laboratorio con el péndulo ECP 505. Sólo se hizo la sintonización con el método de Nelder y Mead utilizando el ITAE y T_s como funciones objetivo. Se sintonizaron los controles PID, ubicación de polos y dos casos de *backstepping*. Los experimentos siguieron siendo con un escalón de magnitud 0.1745 *rad* en la referencia de la posición angular, como en el capítulo anterior.

Como se mencionó con anterioridad, para poder realizar la evaluación iterativa de distintos conjuntos de parámetros, el sistema mecánico debe ser regresado al origen después de evaluar cada conjunto. Esta operación es realizada de manera automática. Cada evaluación de un conjunto de parámetros tiene una duración de 40 segundos, 20 para evaluar los parámetros y 20 para regresar al origen (siempre con un control por *backstepping*).

VII.1 PID

Para obtener una primer aproximación de los parámetros en el control PID se utilizó el segundo método de Ziegler-Nichols de manera experimental, los parámetros son: $K_p = -9.6$, $T_i = 0.7731$ y $T_d = 0.1855$. A partir de estos parámetros se minimiza el ITAE con el método Nelder y Mead obteniendo: $K_p = -9.3698$, $T_i = 0.8490$ y $T_d = 0.1492$. En la Figura 47, se muestra la respuesta con los parámetros de Ziegler-Nichols y con los obtenidos al minimizar el ITAE; en la Figura 48 se muestra la posición del segundo eslabón y la señal de control.

Ahora se utilizan lo parámetros obtenidos con el ITAE para minimizar el T_s. Los nuevos parámetros que se obtienen son: $K_p = -9.0990$, $T_i = 0.6198$ y $T_d = 0.2117$. En la Figura 49, se muestran las respuestas del sistema con los parámetros correspondientes al ITAE y T_s. En la Figura 50 se muestran las posiciones del segundo eslabón y señales de control correspondientes.

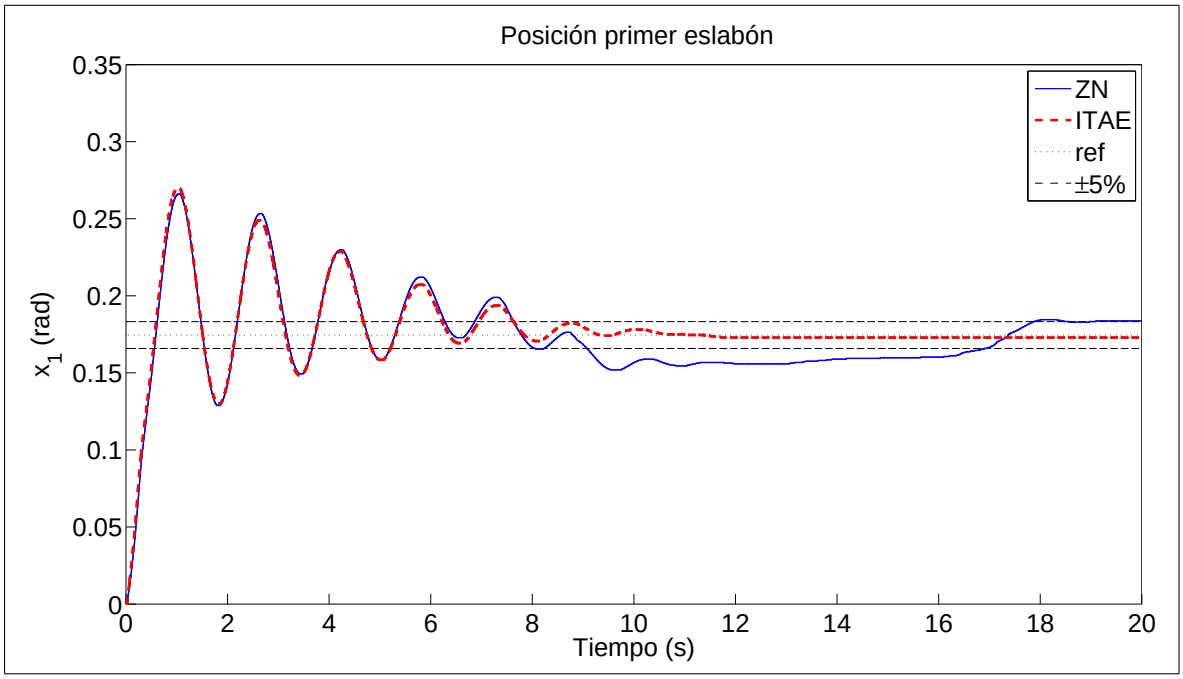


Figura 47: Posición angular x_1 usando PID - optimización experimental del ITAE.

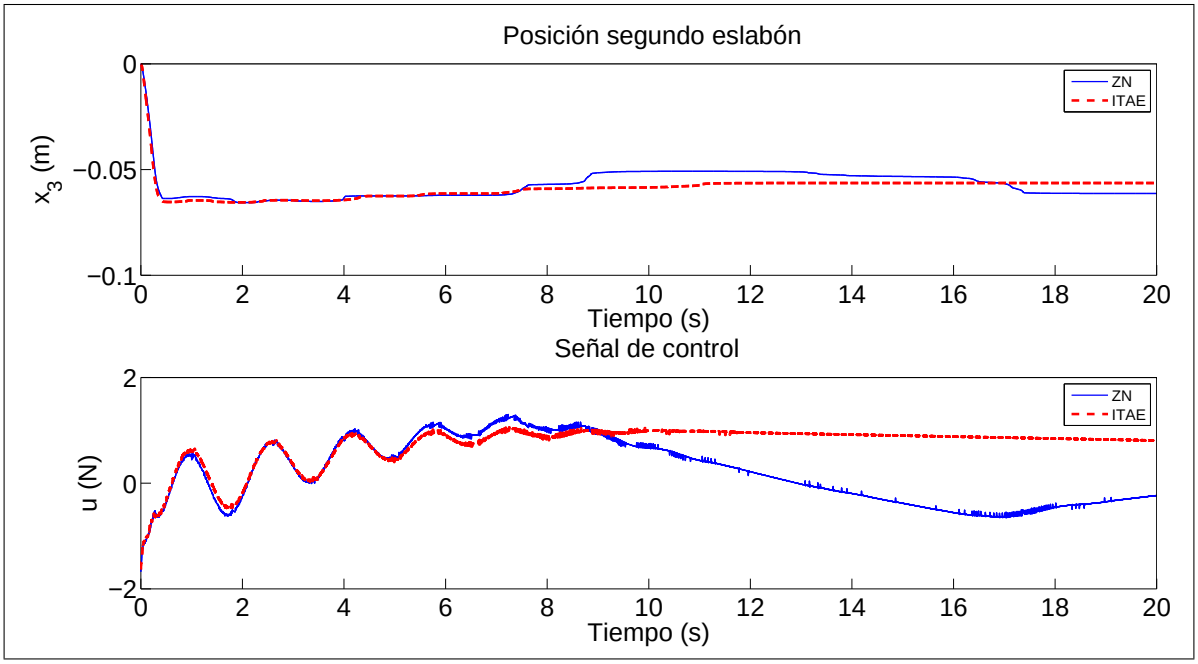


Figura 48: Posición de la barra x_3 y señal de control usando PID - optimización experimental del ITAE.

Se puede observar que la señal correspondiente al ITAE en las figuras 47 y 49 no es exactamente la misma. Esto es debido a que al momento de realizar la sintonización de manera experimental cuando se realiza un experimento con los mismos parámetros no siempre se obtiene el mismo resultado, a diferencia de las simulaciones.

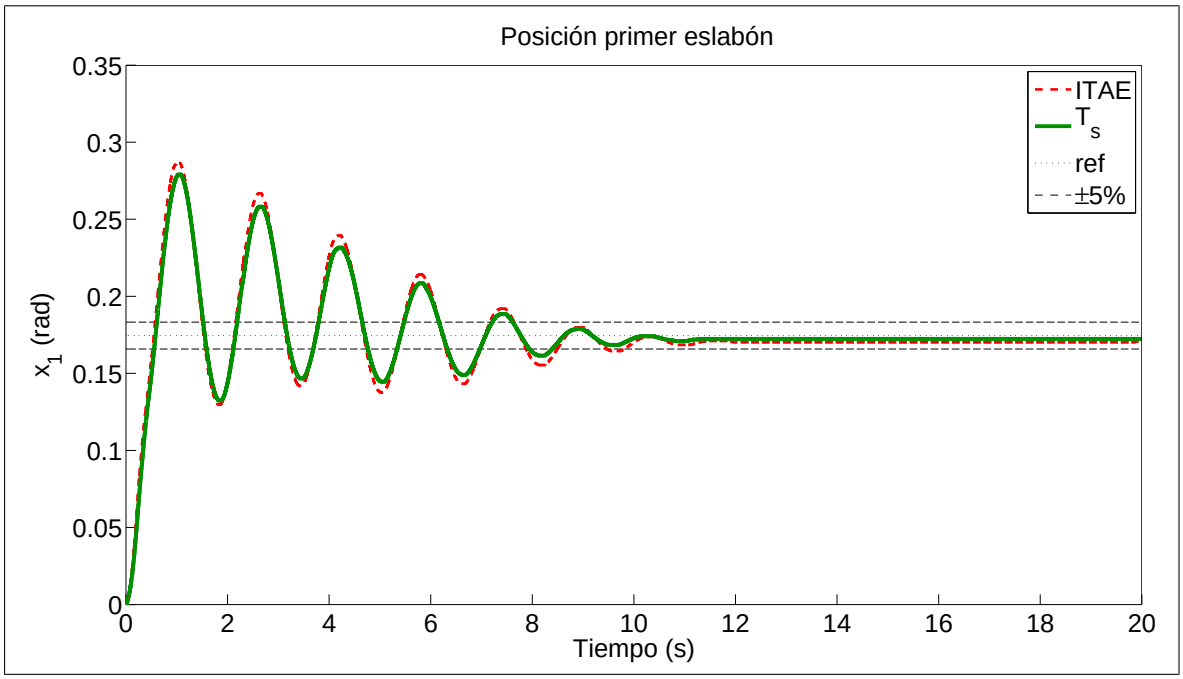


Figura 49: Posición angular x_1 usando PID - optimización experimental del T_s.

En la Tabla XXXI se muestran los tiempos de establecimiento de las señales mostradas en las figuras 47 y 49. Para el ITAE se muestran dos tiempos distintos, el primero es el que se obtuvo al utilizar ITAE como función objetivo y el segundo es cuando los parámetros de ITAE fueron utilizados como semilla para minimizar el T_s. El tiempo obtenido para T_s como función objetivo es mayor que el de ITAE[2], sin embargo es menor que el de ITAE[3]. Esto indica que al momento de hacer la sintonización con T_s como función objetivo, sí se logró minimizar la función; pero también podemos ver que hay poca repetibilidad de resultados experimentalmente.

[2]Obtenido al minimizar el ITAE.
[3]Semilla para minimizar el T_s.

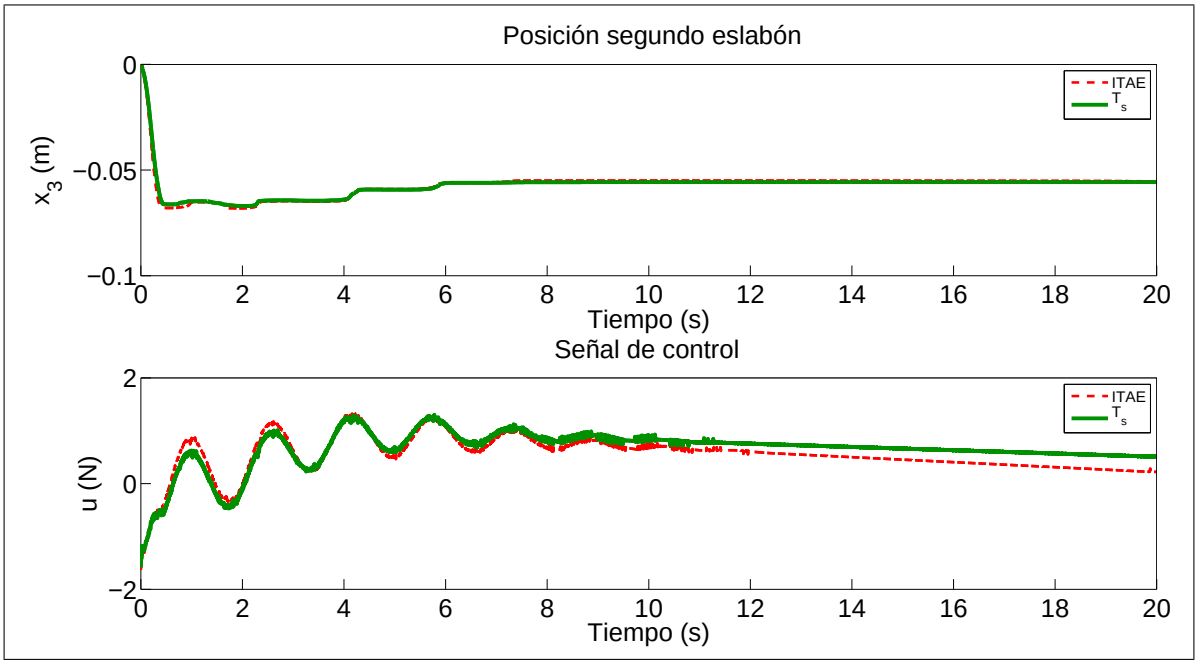


Figura 50: Posición de la barra x_3 y señal de control usando PID - optimización experimental del T_s.

Tabla XXXI: Tiempos de establecimiento, PID experimental.

Señal	T_s (s)
ITAE[2]	7.638
ITAE[3]	9.802
T_s	8.387

VII.2 Ubicación de polos

Para la sintonización experimental se parte de los mismos parámetros que en el caso de simulación, el vector $\mathbf{p} = [4, 9, 6, 5]^T$. Al sintonizar con el ITAE como función objetivo, los parámetros que se obtienen son $\mathbf{p} = [3.5696, 9.2706, 6.1671, 5.3019]^T$. En las figuras 51 y 52 se muestra la respuesta en el tiempo con los parámetros iniciales y finales. En este caso no es posible comparar el tiempo de establecimiento, ya que ninguna de las dos señales permanece en la región del $\pm 5\%$. Sin embargo, el ITAE sí presenta una mejoría. El ITAE se reduce de 10.2688 a 1.4923.

[2]Obtenido al minimizar el ITAE.
[3]Semilla para minimizar el T_s.

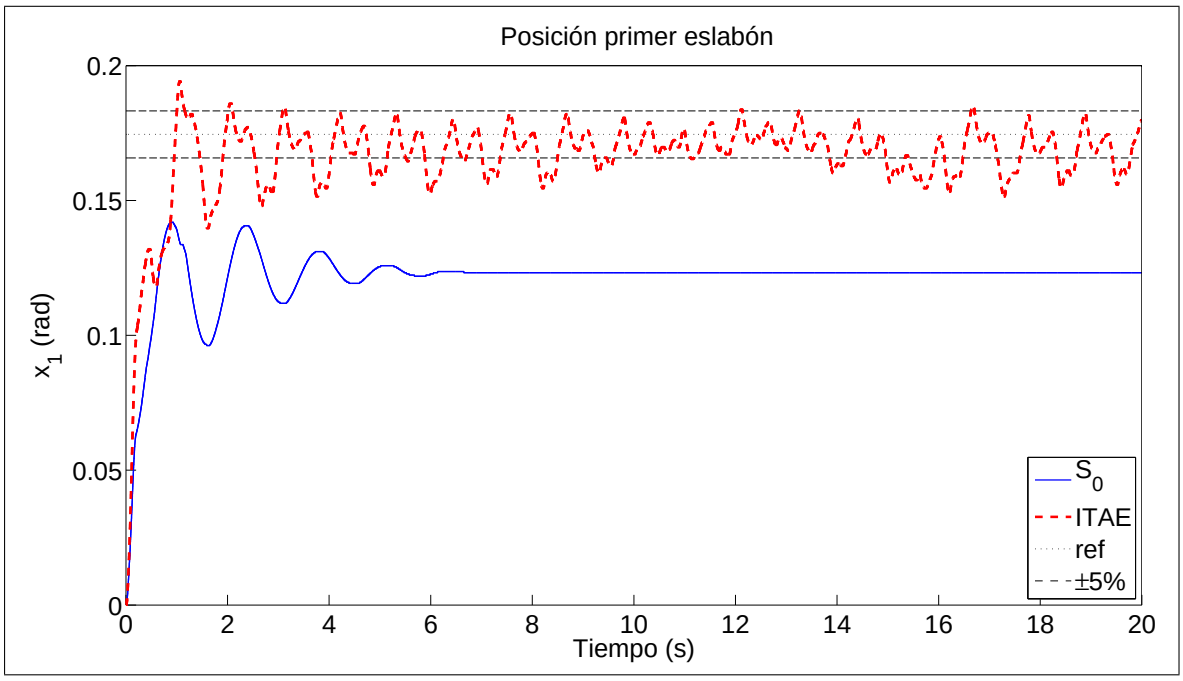


Figura 51: Posición angular x_1 usando ubicación de polos - optimización experimental del ITAE.

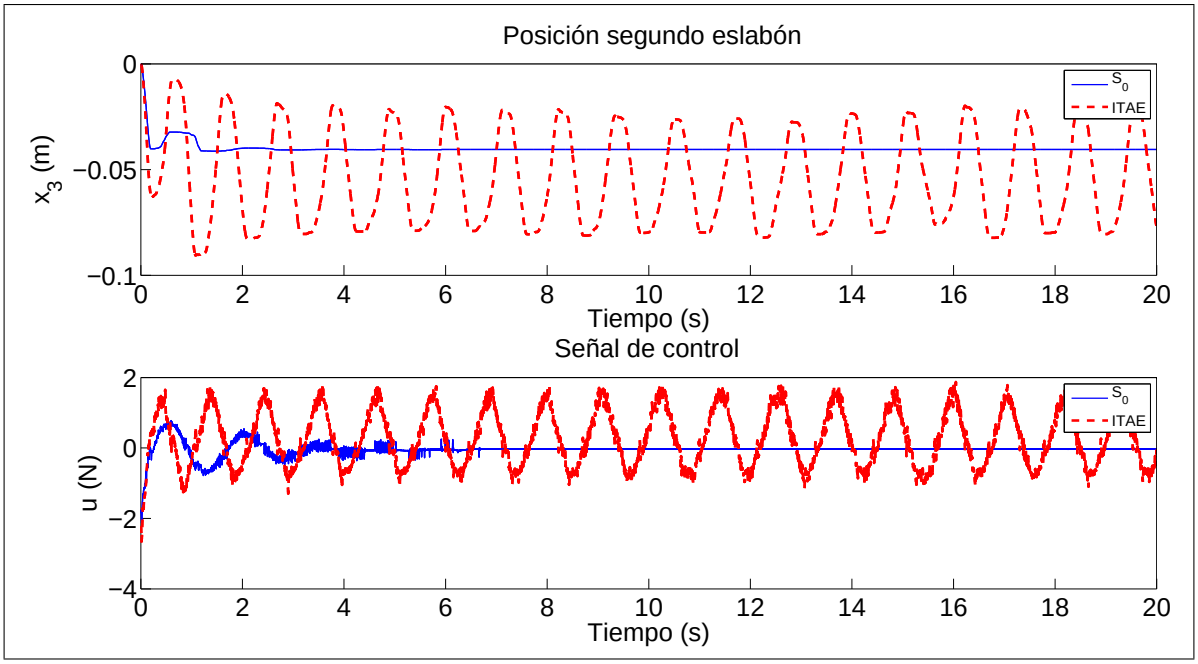


Figura 52: Posición de la barra x_3 y señal de control usando ubicación de polos - optimización experimental del ITAE.

VII.3 *Backstepping* caso 1

En los experimentos con *backstepping* se retroalimenta la perturbación estimada con el observador, esto nos permite compensar variaciones paramétricas o dinámicas no modeladas, como la fricción.

Como parámetros iniciales se tiene el vector $\mathbf{k} = [1, 0, 1]^T$. Para este primer caso, el experimento solamente considera el tercer elemento del vector, es decir k_3. Esta configuración equivale al control por *backstepping* como se presentó en la Sección III.3.1, donde sólo se tiene que elegir una ganancia.

Al realizar la sintonización utilizando el ITAE como función objetivo, se obtienen los siguientes parámetros: $\mathbf{k} = [1, 0, 21.6049]^T$. En la Figura 53 se puede ver la respuesta con los parámetros iniciales y con los obtenidos con el ITAE.

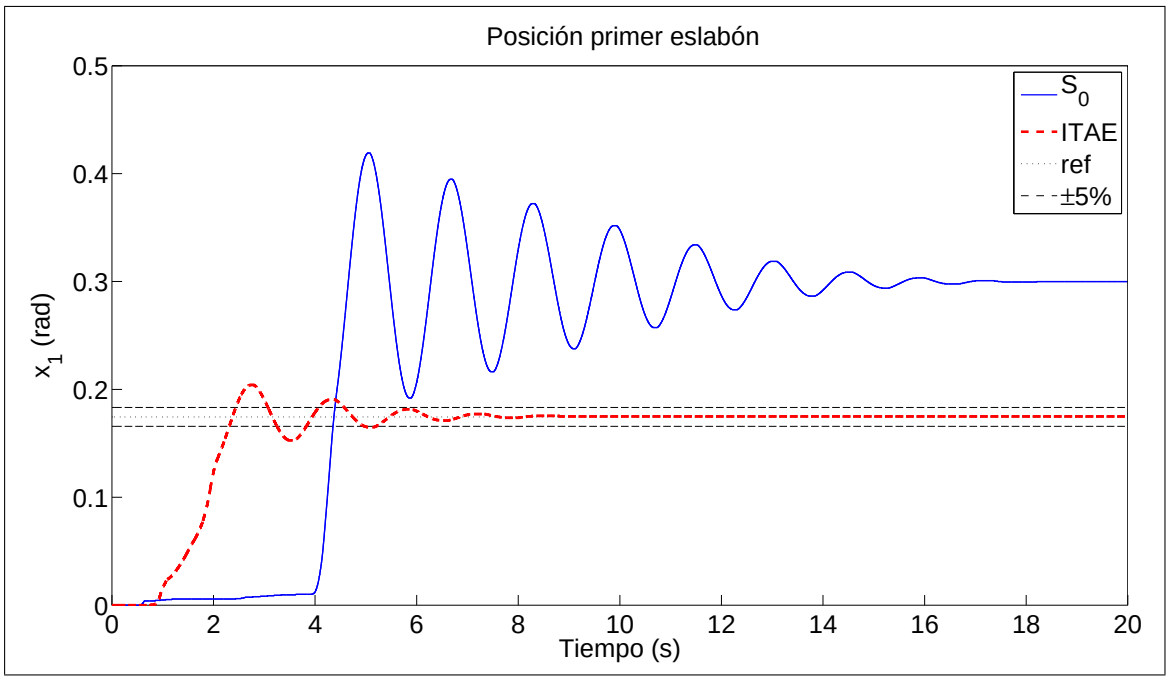


Figura 53: Posición angular x_1 usando *backstepping* caso 1 - optimización experimental del ITAE.

El tiempo de establecimiento para la señal del ITAE es 5.195 segundos, mientras que la otra respuesta no llega a la referencia. En la Figura 54 podemos ver que para el conjunto inicial de parámetros, la posición del segundo eslabón alcanzó los $0.10\,m$, por lo que se activó un bloque de seguridad mandando la señal de control a $0\,N$, para proteger el mecanismo. La Figura 55 nos muestra la perturbación estimada para el

primer eslabón, ω_2 como se vio en la Sección III.4.2.

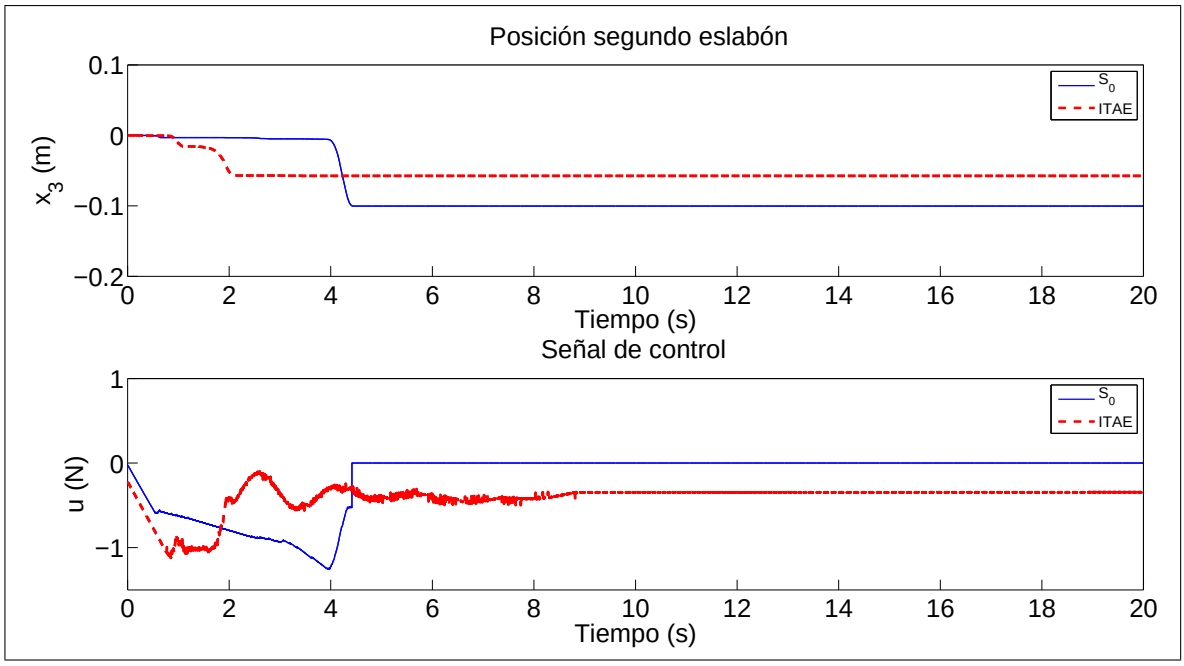


Figura 54: Posición de la barra x_3 y señal de control usando *backstepping* caso 1 - optimización experimental del ITAE.

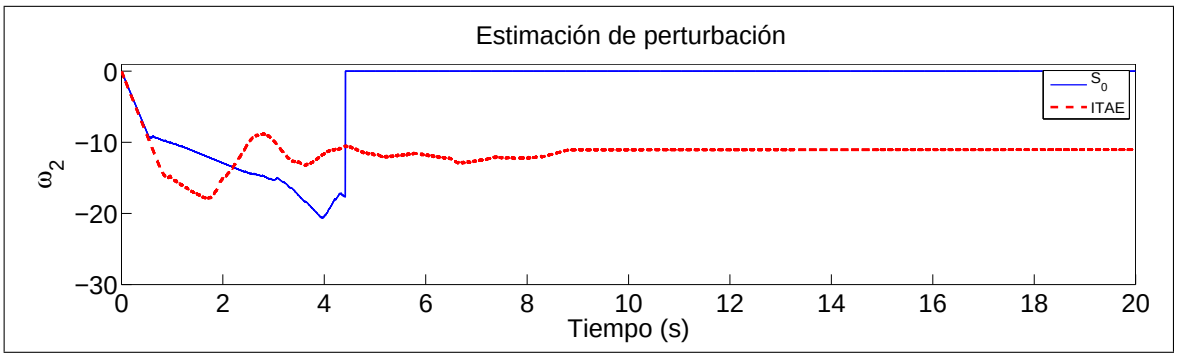


Figura 55: Estimación de perturbación del primer eslabón usando *backstepping* caso 1.

VII.4 *Backstepping* caso 2

En este segundo caso experimental de *backstepping*, se parte del vector $\mathbf{k} = [1, 0, 200]^T$ como parámetros iniciales, con el cual se obtiene un tiempo de establecimiento de 5.611 segundos, estos parámetros son los que se han utilizado para el control encargado de regresar el mecanismo al origen después de cada experimento en las iteraciones del proceso de optimización.

En este caso se sintonizan dos ganancias, k_1 y k_3. Después de sintonizar utilizando el ITAE como función objetivo, los parámetros obtenidos son: $\mathbf{k} = [1.2812, 0, 206.25]^T$, con un tiempo de establecimiento de 4.041 segundos. Estos parámetros se utilizan como semilla para sintonizar utilizando T_s como función objetivo, los parámetros que se obtienen son: $\mathbf{k} = [1.2813, 0, 203.75]^T$, con un tiempo de establecimiento de $3.3560\,s$.

En la Figura 56, se muestra la respuesta al escalón con los tres conjuntos de parámetros. En la Figura 57 se muestra la posición del segundo eslabón y la señal de control. En la Figura 58 se muestra la perturbación estimada, ω_2, para cada conjunto de parámetros.

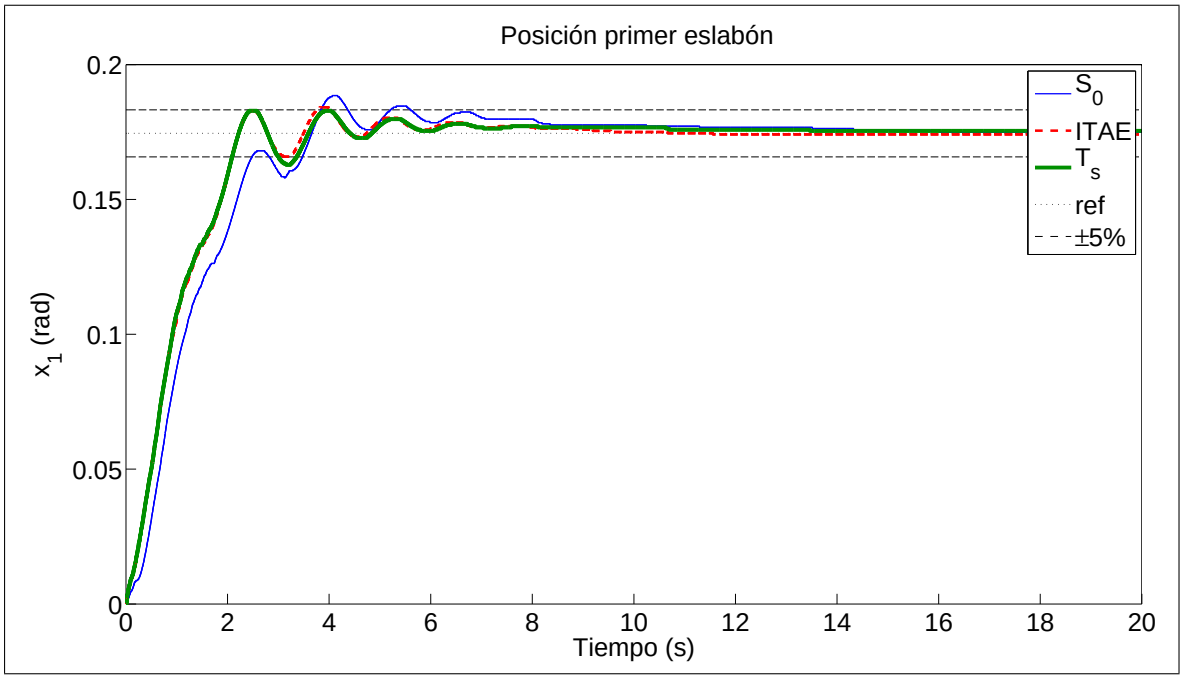


Figura 56: Posición angular x_1 usando *backstepping* caso 2 - sintonización experimental.

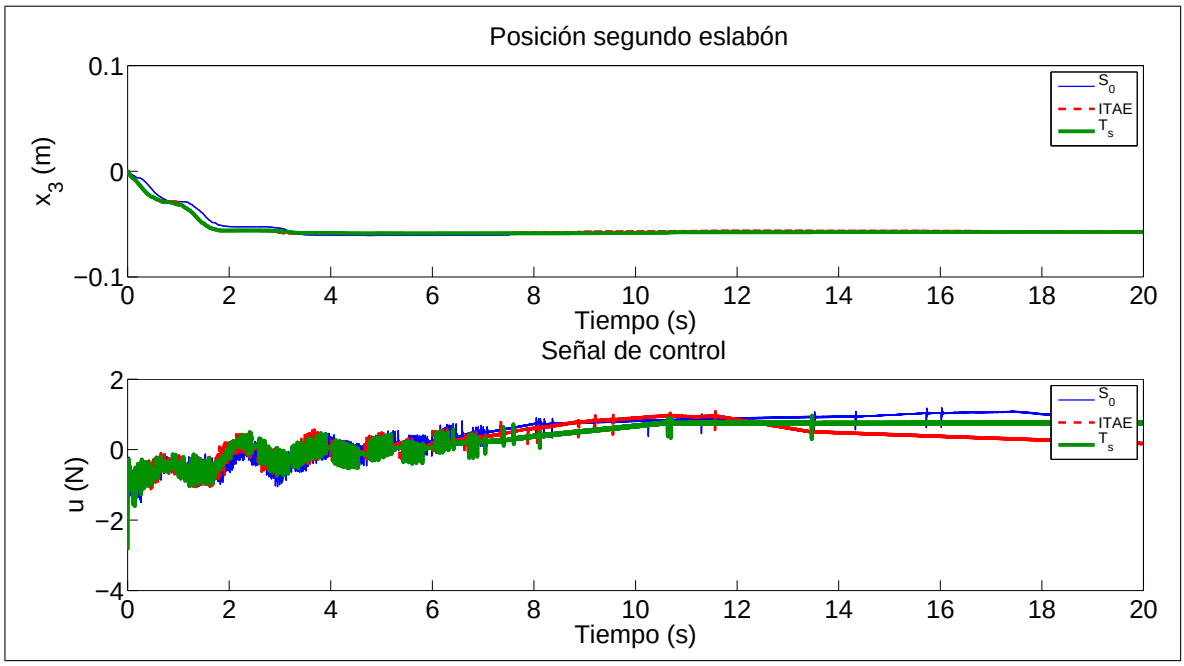


Figura 57: Posición de la barra x_3 y señal de control usando *backstepping* caso 2 - sintonización experimental.

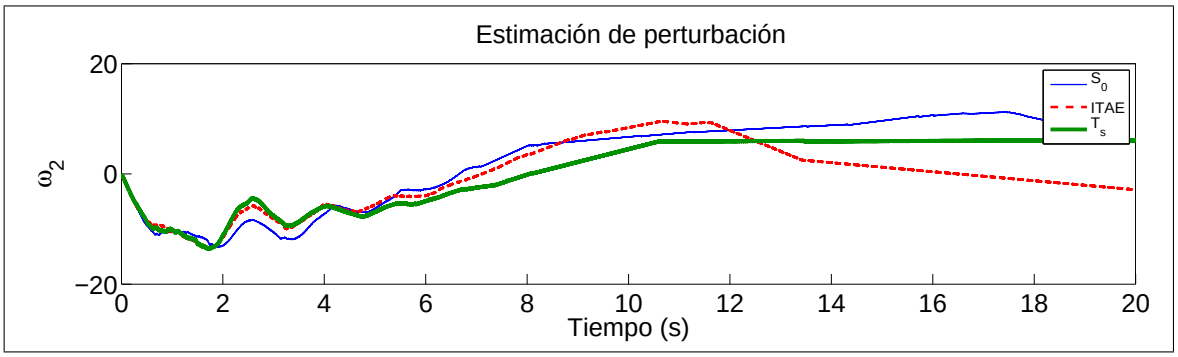


Figura 58: Estimación de perturbación del primer eslabón usando *backstepping* caso 2.

Capítulo VIII

Discusión y conclusiones

En el capítulo anterior se presentaron resultados al implementar la metodología propuesta, tanto en simulación como experimentalmente. A continuación se analizan dichos resultados, se presentan las conclusiones y trabajo a futuro.

VIII.1 Análisis de resultados

VIII.1.1 Simulación

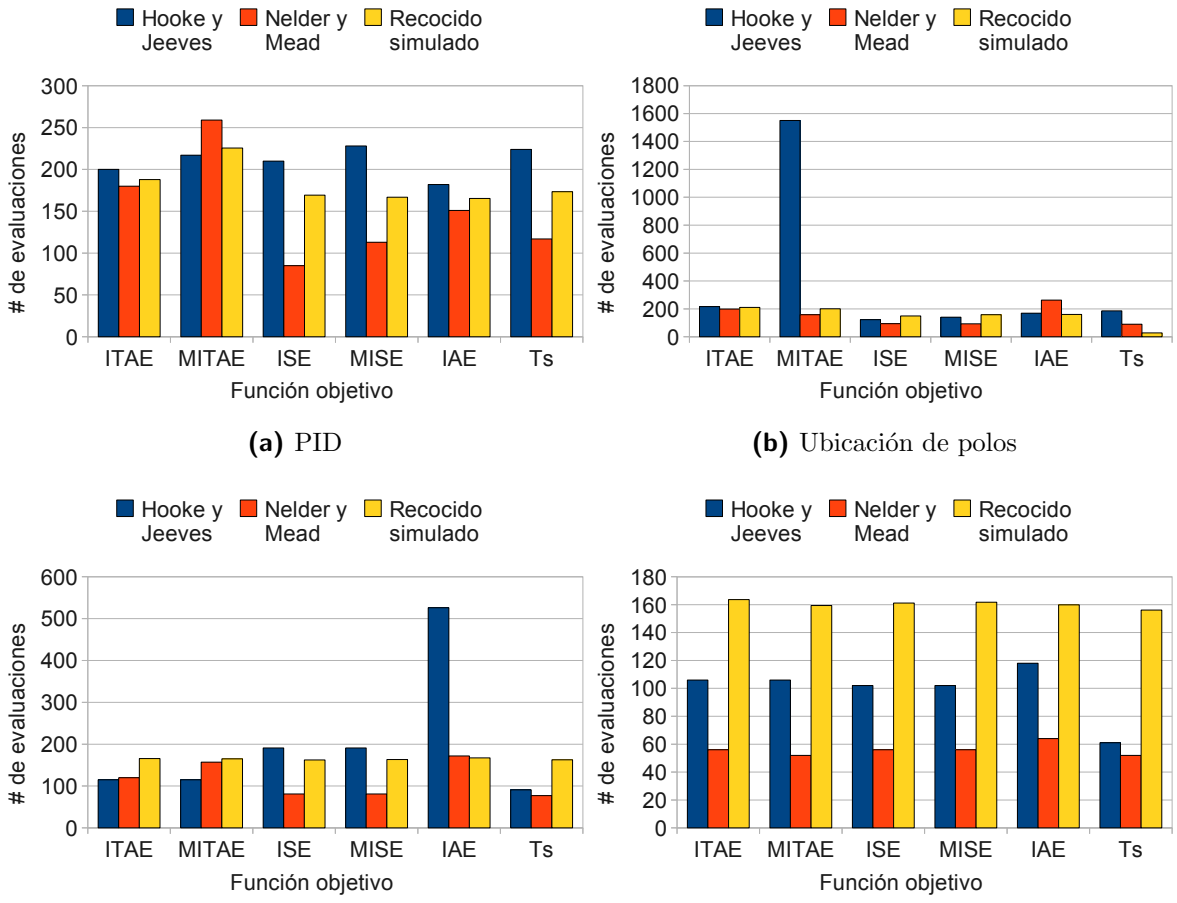
Al inicio de este trabajo de tesis, se decidió tomar el tiempo de establecimiento como criterio a minimizar. Sin embargo, desde las primeras pruebas se notó que es una función con muchos mínimos locales, lo cual conducía a que no se obtuviera mucha mejoría con respecto a la sintonización tomada al inicio del proceso de optimización. Se obtuvieron mejores resultados optimizando primero una función objetivo basada en la señal de error, y usar después el tiempo de establecimiento solo para afinar el resultado.

- Para el contol PID, los mejores tiempos de establecimiento T_s fueron obtenidos al utilizar los criterios ITAE ($1.689\,s$) e IAE ($1.654\,s$), con el método Nelder y Mead.
- En el control por ubicación de polos los mejores tiempos fueron obtenidos con los criterios ISE ($0.345\,s$) y MISE ($0.326\,s$), una vez más con el método Nelder y Mead.
- Para el caso 1 de *backstepping*, los mejores tiempos se obtuvieron con los criterios IAE ($0.215\,s$), ISE ($0.217\,s$) y MISE ($0.217\,s$), esta vez fue utilizando el método Hooke y Jeeves. Con recocido simulado se obtienen buenos resultados considerando el criterio de tiempo de establecimiento, pero las señales de control sobrepasan el máximo establecido de $3\,N$.
- En el caso 2 de *backstepping*, el tiempo de establecimiento fue prácticamente el mismo al utilizar el método Hooke y Jeeves o Nelder y Mead ($0.289\,s$), con cualquiera de los índices de desempeño como criterio.

- Con el método recocido simulado, aunque sí se logra mejorar los tiempos de establecimiento que se obtienen con las semillas utilizadas, no mejora a los de los otros dos métodos. En este trabajo se presentaron los resultados promedio con su desviación estándar y estos valores quedan por encima de los obtenidos con Hooke y Jeeves o Nelder y Mead.

Costo computacional

Un aspecto que es importante considerar para poder decidir cual de los tres algoritmos utilizados conviene utilizar, es el costo computacional requerido para llegar a los resultados. La Figura 59 muestra un comparativo entre los tres algoritmos, donde la meta es minimizar la función objetivo.



(a) PID

(b) Ubicación de polos

(c) *Backstepping* - caso 1

(d) *Backstepping* - caso 2

Figura 59: Comparación de costo computacional con respecto al algoritmo de optimización.

En la Figura 59a se puede observar que el algoritmo Nelder y Mead es el que requirió menos evaluaciones para sintonizar el control PID, con la excepción del caso en que se utiliza el criterio MITAE como función objetivo. Para el control por ubicación de polos (Figura 59b), nuevamente es Nelder y Mead el algoritmo que requiere menos evaluaciones en la mayoría de los casos; es el recocido simulado quien requiere menos (en promedio) con los criterios de IAE y T_s. En el primer caso de *backstepping* (Figura 59c), Nelder y Mead requiere menos evaluaciones para tres criterios: ISE, MISE y T_s; para los criterios ITAE y MITAE, Hooke y Jeeves requiere menos evaluaciones, mientras que recocido simulado fue más rápido para el IAE. En el segundo caso de *backstepping* (Figura 59d) Nelder y Mead fue el algoritmo que requirió menos evaluaciones con todas las funciones objetivo.

VIII.1.2 Experimental

Después de haber obtenido los resultados en simulación, se intentó validar estos en el sistema físico, es decir en el laboratorio. Al hacer algunas pruebas con los parámetros que se habían obtenido, se notó que la respuesta en el laboratorio no coincidía con lo que se había hecho en simulación. Estas diferencias se han atribuido a variaciones paramétricas y dinámicas no modeladas, como lo es la fricción, ya que el sistema físico presenta una alta fricción seca, misma que no fue incluida en el modelo que se utilizó en simulaciones. Por estas razones es que se realizó la sintonización de manera experimental.

En el caso de la sintonización del PID, se pudo minimizar el ITAE y a partir de éste minimizar el tiempo de establecimiento. En los resultados que se presentaron, se puede observar que aunque al parecer la posición angular ha llegado a la referencia (en la banda del $\pm 5\%$ de error, ver figuras 47 y 49), el control no se ha estabilizado (ver figuras 48 y 50), esto debido a que la parte integral del control intenta hacer cero el error, pero el sistema se encuentra atascado por la fricción.

En la sintonización del control por ubicación de polos, no se logra obtener un muy buen resultado. Con los parámetros que se utilizan como semilla el sistema queda lejos de la referencia, similar a la simulación. Conforme se hizo la sintonización, el sistema se empezó a acercar a la referencia, mas no se logró estabilizar, numéricamente el criterio ITAE se redujo, pero el resultado no fue bueno.

Para el *backstepping* se presentaron dos casos, en el primero se pudo observar que

aunque la semilla generó una respuesta que estaba por exceder los límites permitidos, el método pudo llegar a una combinación de parámetros que si estabilizaban el sistema. La cancelación de perturbaciones, gracias al observador, ayudó a obtener buenos resultados.

VIII.2 Conclusiones

En el trabajo se propuso una metodología para sintonizar controladores, independientemente de la estructura del mismo. Se mostró la sintonización con tres algoritmos distintos: Hooke y Jeeves, Nelder y Mead y recocido simulado.

Entre estos tres algoritmos de acuerdo a los resultados que se presentaron en la sección anterior, con el algoritmo de Nelder y Mead se obtienen buenos resultados y es además, el que presenta un menor costo computacional, ya que en general requiere menos evaluaciones para llegar al óptimo.

De entre las distintas funciones objetivo que se utilizaron, no hubo una con la que se obtuvieran los mejores resultados en todos los casos. Por ejemplo para el control PID los criterios ITAE, MITAE e IAE arrojaron los mejores tiempos de establecimiento, sin embargo para el control por ubicación de polos fueron los criterios ISE y MISE. En el control *backstepping* no hubo una tendencia definida. Es importante hacer notar que los criterios ISE y MISE arrojaron tiempos de establecimiento muy altos en la sintonización del PID.

La sintonización que se llevó a cabo en simulaciones fue satisfactoria y nos permitió obtener las siguientes conclusiones:

- No es recomendable utilizar el tiempo de establecimiento T_s como función objetivo desde un inicio. Por ejemplo, en los resultados presentados para el control por ubicación de polos, la semilla que se utilizó como conjunto inicial de parámetros, resultaba en una respuesta en el tiempo que no llegaba a la referencia (Ver Figura 25). De haber utilizado este conjunto de parámetros para minimizar T_s, lo más probable es que no se hubiera obtenido ninguna mejoría, ya que los parámetros cercanos a la semilla tampoco alcanzarían la referencia. En cambio con los otros criterios, se busca minimizar una función basada en el error. El ITAE inclusive hace énfasis en el error en estado estacionario.

- En el caso del control por ubicación de polos, se puede observar, que los parámetros

que se obtienen al sintonizar utilizando como función objetivo los índices de desempeño, sí presentan mejoría (ver figuras 27 y 29). Sin embargo al sintonizar con T_s como función objetivo, la respuesta que se obtiene está en el límite, puesto que numéricamente el tiempo de establecimiento se reduce, pero se puede observar en las gráficas, que la señal correspondiente está muy cerca del límite superior de la banda del $\pm 5\%$ de error; podríamos considerar que este resultado es sensible al tiempo de simulación que se elija. Un problema muy relacionado a esta situación es que por las características del controlador, si se elige una referencia distinta a la que se utilizó para sintonizar, no podemos garantizar que la respuesta sea igual. Esto debido a que la acción de control es puramente proporcional, además de que el control supone un sistema lineal, y mientras más se aleje la referencia, el error en estado estacionario crecerá.

- Los controles PID y *backstepping* no presentan los inconvenientes mencionados con ubicación de polos. El PID cuenta con la acción integral que le permite reducir o eliminar el error en estado estacionario. El control por *backstepping*, por como se diseña el control permite cancelar la dinámica del sistema e impone la propia, dependiendo de los parámetros elegidos.

- En algunas ocasiones al utilizar los criterios ITAE, MITAE e IAE, se obtuvieron respuestas con un sobretiro considerable (ver figuras 19 y 21), sin embargo en ninguno de los casos se excede el límite que se estableció. No se excede debido a las penalizaciones que se utilizaron al evaluar la función objetivo en la sintonización. Si se desea obtener una respuesta con sobretiros menores se puede reducir más este límite máximo. De igual manera se podrían incluir algunas otras penalizaciones como tiempo de subida.

Sobre la sintonización experimental cabe destacar los siguientes puntos:

- Se logró implementar una plataforma que permite hacer la sintonización haciendo los experimentos con el sistema físico. Se implementaron funciones MEX con lenguaje C/C++ lo cual da más flexibilidad de programación que Simulink.

- El utilizar los archivos MEX permite realizar la sintonización con el mismo esquema propuesto en simulación.

- De manera experimental la sintonización es más sensible a variaciones entre experimentos, es decir no se puede garantizar que dos experimentos sean iguales, siempre habrá pequeñas variaciones físicas, como condiciones iniciales, que nos lleven a resultados distintos.

Como propuesta final se sugiere utilizar el algoritmo Nelder y Mead con ITAE o IAE como función objetivo. Si se desea dar énfasis a la reducción del error en estado estacionario, utilizar ITAE. Posteriormente se puede hacer una sintonización con T_s como función objetivo.

VIII.3 Trabajo a futuro

Como trabajo futuro se proponen los siguientes temas:

- Considerar optimización multiobjetivo, en donde se pueda incluir el control en la función a minimizar.
- Utilizar otros métodos de optimización, heurísticos o no.
- Comparar distintos controladores, aplicados a un mismo sistema, sintonizando sus parámetros para ver con cual se obtiene un mejor desempeño.

Referencias

Abraján Guerrero, R. y Márquez Martínez, L. A. (2010). Automatic pid tuning, a heuristic optimization approach. En *Congreso Nacional de Control Automático 2010*, Puerto Valarta, Jalisco, México. AMCA. 6-8 de octubre.

Alvarez, J., Orlov, I., y Acho, L. (2000). An invariance principle for discontinuous dynamic systems with application to a Coulomb friction oscillator. *Journal of Dynamic Systems, Measurement, and Control*, **122**: 687--690.

Aström, K. y Hägglund, T. (1988). *Automatic tuning of PID controllers*. Instrument Society of America Research Triangle Park, NC. ISBN 1556170815. 141 p.

Aström, K., Hägglund, T., Hang, C., y Ho, W. (1993). Automatic tuning and adaptation for pid controllers-a survey. *Control Engineering Practice*, **1**(4): 699--714.

Boyd, S. y Barratt, C. (1991). *Linear controller design: Limits of performance*. Prentice Hall (Englewood Cliffs, NJ).

Brizuela Rodriguez, C. A. (1994). *Diseño de controladores para una clase de sistemas no lineales empleando técnicas de optimización convexa*. Tesis de maestría, Centro de Investigación Científica y Educación Superior de Ensenada. Ensenada, BC, México. 102 p.

Chien, K., Hrones, J., y Reswick, J. (1952). On the automatic control of generalized passive systems. *Transactions ASME*, **74**: 175--181.

de Almeida, G. M., e Silva, V. V. R., Nepomuceno, E. G., y Yokoyama, R. (2005). Application of genetic programming for fine tuning PID controller parameters designed through Ziegler-Nichols technique. En L. Wang, K. Chen, y Y. S. Ong, editores, *Advances in Natural Computation*. Springer. 313-322.

Dorf, R. C. (1967). *Modern Control Systems*. Addison-Wesley, Reading, MA. 387 p.

Dorf, R. C. y Bishop, R. H. (1996). Design using performance indices. En W. S. Levin, editor, *The control handbook*. CRC Press. 169-172.

Estrada García, H. J. (2008). *Sincronización de sistemas mecánicos con retardos en la transmisión de datos*. Tesis de doctorado, Centro de Investigación Científica y Educación Superior de Ensenada. Ensenada, BC, México. 139 p.

Graham, F. D. y Lathrop, R. C. (1953). Synthesis of ‘‘optimum’’ transient response - criteria and standard forms. 45 p.

Hooke, R. y Jeeves, T. A. (1961). ‘‘Direct Search’’Solution of Numerical and Statistical Problems. *Journal of the ACM (JACM)*, **8**(2): 212--229.

Kelly, R. y Santibáñez, V. (2003). *Control de movimiento de robots manipuladores*. Pearson Educación, SA. ISBN 8420538310. 323 p.

Khalil, H. K. (2002). *Nonlinear systems*. Prentice Hall, Upper Saddle River, NJ, 3ra. edición. ISBN 0130673897. 750 p.

Kirkpatrick, S., Gelatt, C. D., y Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**(4598): 671--680.

Kwakernaak, H. y Sivan, R. (1972). *Linear optimal control systems*, Vol. 188. Wiley-Interscience New York. 575 p.

Levant, A. (2003). Higher-order sliding modes, differentiation and output-feedback control. *International Journal of Control*, **76**(9): 924--941.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., *et al.* (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, **21**(6): 1087--1092.

Nelder, J. A. y Mead, R. (1965). A simplex method for function minimization. *The computer journal*, **7**(4): 308--312.

Ogata, K. (2009). *Modern control engineering*. Prentice Hall. ISBN 0136156738. 912 p.

Purandare, R. (2007). *A buoyancy-propelled airship*. Tesis de doctorado, New Mexico State University. Las Cruces, NM, EUA. 232 p.

Roberge, J. K. (1960). *The mechanical seal*. Tesis de maestría, Massachusetts Institute of Technology. Cambridge, MA, EUA. 45 p.

Rosas Almeida, D. I., Alvarez, J., y Fridman, L. (2007). Robust observation and identification of nDOF Lagrangian systems. *International Journal of Robust and Nonlinear Control*, **17**(9): 842--861.

Rosas Almeida, D. I., Álvarez Gallegos, J., Rosas Méndez, P. L. A., y Silva, A. (2009). Observación del estado y estimación de perturbaciones en sistemas no lineales en forma normal. En *Congreso Nacional de Control Automático 2009*, Zacatecas, México. AMCA.

Spong, M. (1998). Underactuated mechanical systems. En B. Siciliano y K. P. Valavanis, editores, *Control Problems in Robotics and Automation*. Springer. 135-150.

Vrancic, D., Peng, Y., y Strmcnik, S. (1998). A new pid controller tuning method based on multiple integrations. *Control Engineering Practice*, **7**(5): 623--633.

Wu, X., Moog, C., Márquez-Martínez, L. A., y Huu, Y. (2010). Modelling and control of a complex buoyancy-driven airship. En *Proceedings of the 8th IFAC Symposium on Nonlinear Control Systems*, Bologna, Italy. 1-3 de septiembre.

Yachen, Z. y Yueming, H. (2008). On pid controllers based on simulated annealing algorithm. En *27th Chinese Control Conference, 2008. CCC 2008.*, páginas 225--228. IEEE. 16-18 de julio.

Ziegler, J. G. y Nichols, N. B. (1942). Optimum settings for automatic controllers. *Transactions ASME*, **62**: 759--768.

Apéndice A

Congreso nacional AMCA 2010

En este apéndice se presenta una copia del artículo que fue presentado en el Congreso Nacional 2010 de la Asociación de México de Control Automático, llevado a cabo del 6 al 8 de octubre de 2010 en Puerto Vallarta, Jalisco.

Automatic PID tuning, a heuristic optimization approach

Rodrigo Abrajan-Guerrero, Luis Alejandro Marquez-Martinez
CICESE, Carr. Ensenada-Tijuana #3918, CP 22860, Ensenada B.C. MEXICO
rabrajan@cicese.mx, lmarquez@cicese.mx
Phone: (52)-646-175-0500

***Abstract*— This paper presents a method to tune a PID controller based on three heuristic optimization techniques: Hooke and Jeeves, Nelder and Mead (simplex), and simulated annealing. The proposed method uses Ziegler-Nichols tuning formula to obtain a first approximation of the controller, then it is optimized by minimizing a cost function defined by a performance index.**

Keywords: PID controllers, optimization, heuristic searches, settling times, performance indices.

I. Introduction

One of the most used controllers today in industry is the proportional-integral-derivative (PID). These controllers are often used because of their simple structure and robustness. A problem that can arise with these controllers is that a bad tuning may result in a poor performance or even lead to instability.

Several attempts have been made to find a tuning formula for PID controllers. The Ziegler-Nichols (ZN) method (Ziegler & Nichols, 1942) gives the parameter values as a function of the ultimate gain (K_u) and period (T_u) of the system. The performance obtained by using ZN is not always the best. The formula was designed to give an overshoot around 25%, which is why it is convenient to fine tune the parameters to achieve an acceptable performance.

This has motivated a lot of research and several methods for PID tuning have been proposed. A modification in the ZN coefficients was proposed in (Chien, Hrones & Reswick, 1952) to obtain a tuning method with an improved damping (CHR). A refinement to the ZN formula (RZN) (Astrom et al., 1993) was done by adding a fourth parameter β. A tuning method based on a magnitude optimum frequency criterion was proposed in (Vranci, Peng & Strmenik, 1998). Also in the last years some tuning has been made using Genetic Programming techniques (de Almeida et al., 2005).

In this paper we propose the use of three heuristic optimization techniques that have shown good results: Hooke and Jeeves (Hooke & Jeeves, 1961), Nelder and Mead (simplex) (Nelder & Mead, 1965) and simulated annealing (Aarts & Lenstra, 2003).

This paper is organized as follows. Section II recalls the ZN PID tuning method, the three optimization methods that will be used in later sections and also several performance indices are presented. Section III consists of a proposed implementation of the optimization methods to find the PID parameters. Section IV shows the results of some numerical simulations done using the proposed approach. Section V concludes the paper.

II. Mathematical Preliminaries

Here we recall the ZN tuning method, define the optimization methods we will use later in the paper, as well as some performance indices which will be used later with the optimization methods.

ZN-PID Controller Overview

The considered PID structure is:

$$PID = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{1}$$

There are two Ziegler-Nichols methods to determine the parameters of a PID controller: a step response method and the frequency response method. Here we will show the second one only, which is the most commonly used.

For the frequency response method, the formulas are given in function of the ultimate gain K_u and the ultimate period T_u. An easy way to find this parameters is to connect a controller to the plant with only proportional control action, that is, $T_i = \infty$ and $T_d = 0$. Once connected this way you have to start incrementing K_p, until you get a sustained oscillation as an output. The value of K_p required to sustain the oscillation will be K_u, and the period of the oscillations will be T_u. The formulas for the controller parameters are shown in Table I.

TABLE I
Ziegler-Nichols frequency response tuning formulas.

Controller	K	T_i	T_d
P	$0.5K_u$		
PI	$0.4K_u$	$0.8T_u$	
PID	$0.6K_u$	$0.5T_u$	$0.12T_u$

Optimization Methods

We have selected three heuristic optimization methods: Hooke-Jeeves (HJ), Nelder-Mead (NM), and simulated annealing (SA). They were selected because of their simplicity and ease of implementation. They only require to be able to compute a cost function at any given point. Each method is explained next.

Hooke-Jeeves: This method basically consists of the repetition of an exploratory movement followed by a pattern movement until a stopping criteria is met.

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a cost function and $b \in \mathbb{R}^n$ a starting point. Define e_i as a base of orthonormal vectors for $i = 1, 2, ..., n$, and ρ an initial exploratory increment.

The exploratory movement works as follows. Let $b_1 \leftarrow b$, compute $f(b_1 + \rho e_1)$. If the movement from b_1 to $b_1 + \rho e_1$ is an improvement, then $b_1 \leftarrow b_1 + \rho e_1$. If it does not improve, compute $f(b_1 - \rho e_1)$. If it improves, then $b_1 \leftarrow b_1 - \rho e_1$, on the contrary b_1 remains the same. The next step is to repeat the previous procedure, but changing e_1 for e_2, then e_3, until you finish with e_n. Now, if $b_1 = b$, make $\rho \leftarrow \frac{\rho}{2}$. If $b_1 \neq b$, then do a pattern movement.

The pattern movement basically consists in moving in the same direction that just yield an improvement in the cost function. This means you have to do $b_3 \leftarrow 2b_2 - b$. If $f(b_3) < f(b_2)$, then $b \leftarrow b_3$, otherwise $b \leftarrow b_2$. Now return to an exploratory movement unless the stopping criteria is met.

Nelder-Mead (simplex): This method is based on some basic operations over a simplex. A simplex is defined by $n+1$ vertices over $\mathbb{R}^n$, that do not belong to a hyperplane. To use this method you first need to create an initial simplex. The initial simplex may be created from an initial guess, x_1; the other n vertices, may be: $x_{i+1} = x_1 + \rho e_i$. ($f(\bullet)$, ρ and e_i are defined as in the previous method) We define the worst, second worst, and best vertices as follows: $x_h := \{x_i | f(x_i) \geq f(x_j), \forall j = 1, .., n+1\}$, $x_s := \{x_i | f(x_i) \geq f(x_j), \forall j = 1, .., n+1, i \neq h\}$, $x_l := \{x_i | f(x_i) \leq f(x_j), \forall j = 1, .., n+1\}$. For ease of notation we define $f_i := f(x_i)$. Being $\bar{x}$ the centroid of all vertices except x_h, the valid operations are:

- Reflection $x_r = \bar{x} + \alpha(\bar{x} - x_h)$
- Expansion $x_e = \bar{x} + \gamma(\bar{x} - x_h)$
- Contraction $x_c = x_h + \beta(\bar{x} - x_h)$
- Reduction $x_i = (x_i + x_l)/2$

Typically $\alpha = 1$, $\gamma = 2$, and $\beta = 0.5$ or -0.5.
The algorithm is as follows.

1) Order the vertices, $x_l \leq \cdots \leq x_s \leq x_h$. Calculate $\bar{x}$.
2) Reflect; if $f_l \leq f_r \leq f_s$, then $x_h \leftarrow x_r$ and go to 1.
3) If $f_r < f_l$, then expand, else go to 5.
4) If $f_e \leq f_r$, then $x_h \leftarrow x_e$ and go to 1, else $x_h \leftarrow x_r$ and go to 1.
5) If $f_s \leq f_r < f_h$, then contract with $\beta = 0.5$.
6) If $f_r \geq f_h$, then contract with $\beta = -0.5$.
7) If $f_c < f_h$, then $x_h \leftarrow x_c$, else make reduction.
8) If stopping criteria is met, then stop, else go to 1.

Simulated Annealing: This method was inspired by annealing in metallurgy, where by a combination of heating and controlled cooling the material increases the size of its crystals and reduces defects.

Any point or combination of variables of the search space will be called a state, s. The cost function to minimize will be the energy of the states, $e = E(s)$.

For this method the way of changing from one state to another will be decided probabilistically. It is necessary that the user defines the neighbors of a state; if it is a continuous problem it has to be discretized. It also needs to be defined how the new neighbors will be selected. All this is specified by the user.

The user also has to define an acceptance probability function $P(e, e_{new}, T)$, which depends on the energies of the current and new state and a "temperature" parameter. This probability should be non zero when $e_{new} > e$ if $T \neq 0$, enabling the state to change even when it is to a worse state. As T approaches zero, this probability decreases. When $T = 0$, the probability should be zero if $e_{new} > e$.

When $e_{new} \leq e$ the acceptance probability can be 1 or it may also depend on the parameter T. If the user decides to make it a function of T, it should be always non zero and as T approaches zero the acceptance probablity should approach 1.

The annealing schedule also has to be defined by the user. That is, how the temperature will vary through iterations. Having defined all that, the algorithm works as follows.

1) Initialize $s \leftarrow s_0$, $e \leftarrow E(s)$.
2) $s_{best} \leftarrow s$, $e_{best} \leftarrow e$
3) $s_{new} \leftarrow neighbor(s)$
4) $e_{new} \leftarrow E(s_{new})$
5) If $e_{new} < e_{best}$, then $s_{best} \leftarrow s_{new}$, $e_{best} \leftarrow e_{new}$.
6) If $P(e, e_{new}, T) > random()$, then $s \leftarrow s_{new}$, $e \leftarrow e_{new}$.
7) $k \leftarrow k + 1$
8) If stopping criteria is met or scheduled time is over, then stop, else go to 3.

Performance Indices

To compare two systems, performance indices will be considered as in (Dorf & Bishop, 1996). A performance index must be always positive or zero. The best system is defined as the system that minimizes the index. The indices are calculated for a finite period of time T. For the use of the indices we first define the error, $e(t) = y_{ref}(t) - y(t)$; we also define a modified version for the error:

$$\hat{e}(t) = \begin{cases} 10e(t) & \text{for} \quad e(t) < 0 \\ e(t) & \text{for} \quad e(t) \geq 0 \end{cases}.$$

Integral of time multiplied by the absolute magnitude of the error (ITAE):

$$ITAE = \frac{1}{T} \int_0^T t|e(t)|\, dt \tag{2}$$

Modified integral of time multiplied by the absolute magnitude of the error (MITAE):

$$MITAE = \frac{1}{T} \int_0^T t|\hat{e}(t)|\, dt \tag{3}$$

Integral of the square of the error (ISE):

$$ISE = \frac{1}{T} \int_0^T e^2(t)\, dt \tag{4}$$

Modified integral of the square of the error (MISE):

$$MISE = \frac{1}{T}\int_0^T \dot{e}^2(t)\,dt \tag{5}$$

Integral of the absolute magnitude of the error (IAE):

$$IAE = \frac{1}{T}\int_0^T |e(t)|\,dt \tag{6}$$

III. Proposed Approach

We consider the tuning of the PID as an optimization problem. To do this we need the following:

- Select a performance index from section II to be used as cost function J.
- Select an optimization algorithm from section II.
- Define the stopping criteria.
- Use the ZN formulas from section II to give an initial guess of the controller parameters in the search of an optimum.

The algorithms HJ and NM will stop and return an optimum if any of the following occur:

- $J_i < J_{max}$; where J_i is the value of the cost function at the $i-th$ iteration, and J_{max} is a good enough cost function value, defined by the user.
- $J_{i-k} - J_{i-k+1} \leq \epsilon \ \forall \ k = 1, 2, \cdots, n_\epsilon$. This means that J has not improved (in at least ϵ) for the last n_ϵ iterations (n_ϵ is user defined).

The SA algorithm will stop if $J_i < J_{max}$ or if it reaches a maximum number of iterations defined by the user.

IV. Numerical Simulations

The three optimization algorithms from section II were implemented in Matlab®software. The algorithms were used to tune a PID controller for two nominal plants taken from (de Almeida et al., 2005) where a Genetic Programming (GP) algorithm is used to tune the PID controller. The simulations were done using a large time-delay plant and a high-order process. In each case optimization was made with the three algorithms and all the performance indices.

TABLE II

Hooke-Jeeves optimization results for $G_1(s)$.

Cost Function	K_p	T_i	T_d	Settling Time
$ITAE$	0.5552	3.6482	1.4411	9.461
$MITAE$	0.6450	5.0594	1.2428	20.167
ISE	0.6255	3.2547	2.3014	22.129
$MISE$	0.6216	4.0633	1.7623	17.029
IAE	0.5864	3.5496	1.7409	8.617
$S.TIME$	0.5864	3.5105	1.7721	8.562

The objective is to minimize the settling time, however using this parameter as cost function, leads to a local minimum which is improved if other cost functions are used. As shown in Table II the best results were obtained using $ITAE$ and IAE, this is why we show results for the five performance indices only in the first simulation, and in next simulations we only show $ITAE$ and IAE. Once the best settling time is obtained from either $ITAE$ or IAE, this parameters are used as an initial guess for optimization using settling time as cost function.

Plant 1, large time-delay system

The first considered system is the following:

$$G_1(s) = \frac{e^{-5s}}{(s+1)^2} \tag{7}$$

A third order Pade approximation was used for the time-delay in simulations, for comparison with (de Almeida et al., 2005). The PID parameters obtained with ZN are: $K_p = 0.77$, $T_i = 13.20$, and $T_d = 2.11$. This parameters were used as initial guess for the optimization methods.

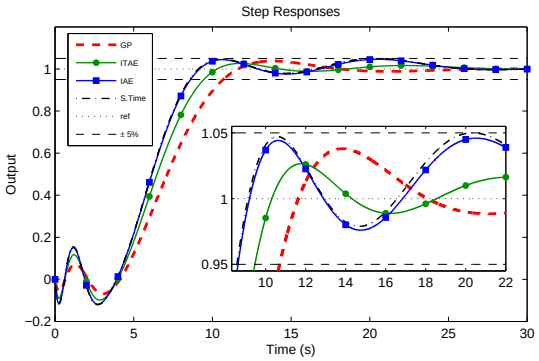


Fig. 1. Hooke-Jeeves tuned PID step responses for $G_1(s)$

Table II shows the optimum parameters obtained by the HJ algorithm for each cost function, as well as the settling time corresponding to each parameter configuration. Figure 1 shows the step response of the system with the controller tuned with each cost function, it also includes the one generated by GP from (de Almeida et al., 2005) for comparison. The settling time for GP is 10.697 seconds; its PID parameters are: $K_p = 0.49$, $T_i = 3.56$, and $T_d = 0.99$.

TABLE III

Nelder-Mead optimization results for $G_1(s)$.

Cost Function	K_p	T_i	T_d	Settling Time
$ITAE$	0.4805	3.6542	0.8447	11.023
IAE	0.6259	4.3192	1.0701	20.4980
$S.TIME$	0.4961	3.6542	0.8447	10.604

The results of optimization with NM are shown in Table III and Figure 2.

TABLE IV

Simulated Annealing optimization results for $G_1(s)$.

Cost Function	K_p	T_i	T_d	Settling Time
$ITAE$	0.56	3.63	1.49	9.325
IAE	0.59	3.51	1.78	20.901
$S.TIME$	0.58	3.45	1.76	8.625

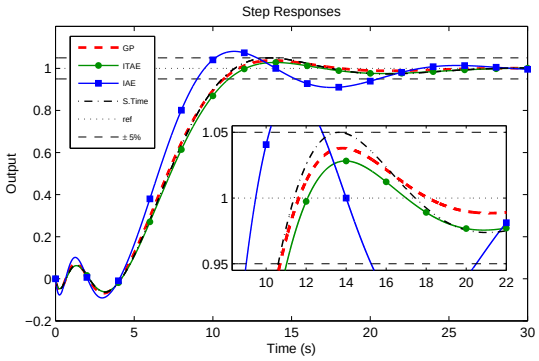


Fig. 2. Nelder-Mead tuned PID step responses for $G_1(s)$

Table IV and Figure 3 show the parameters and step response when SA is used to tune the PID controller for plant $G_1(s)$.

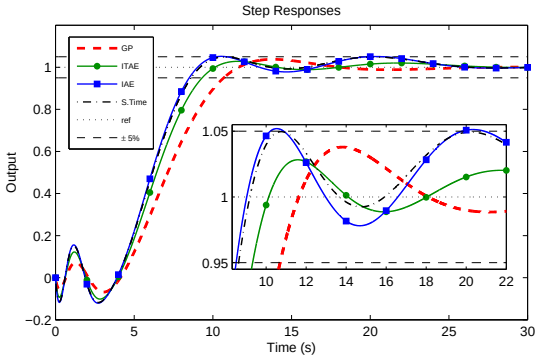


Fig. 3. Simulated Annealing tuned PID step responses for $G_1(s)$

Plant 2, High-Order Process

The second considered system is:

$$G_2(s) = \frac{1}{(1+s)^8} \tag{8}$$

TABLE V

HOOKE-JEEVES OPTIMIZATION RESULTS FOR $G_2(s)$.

Cost Function	K_p	T_i	T_d	Settling Time
$ITAE$	0.8205	4.9223	1.9661	9.423
IAE	0.9025	4.6762	2.4700	23.196
$S.TIME$	0.8205	4.6118	2.4661	9.053

The ZN formula yields the following PID parameters: $K_p = 2.34$, $T_i = 10.77$, and $T_d = 1.72$.

Table V shows the parameters obtained with HJ. The step responses for HJ are shown in Figure 4. The PID parameters for the GP tuning are: $K_p = 0.68$, $T_i = 4.36$, and $T_d = 1.47$. Its settling time is 11.119 seconds.

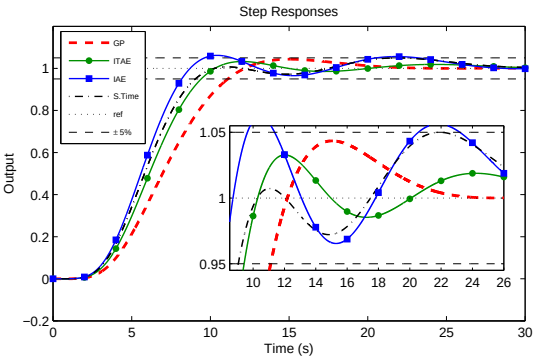


Fig. 4. Hook-Jeeves tuned PID step responses for $G_2(s)$

The optimum parameters obtained with the NM method are shown in Table VI, the step response for each tuned controller using NM can be seen in Figure 5.

TABLE VI

NELDER-MEAD OPTIMIZATION RESULTS FOR $G_2(s)$.

Cost Function	K_p	T_i	T_d	Settling Time
$ITAE$	0.8141	5.1203	1.6723	9.679
IAE	0.9376	5.3935	1.9002	18.613
$S.TIME$	0.8702	5.3026	1.9688	9.088

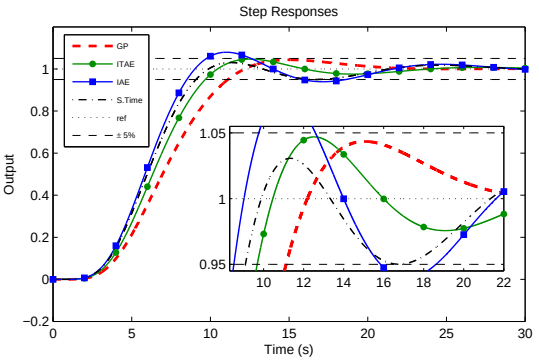


Fig. 5. Nelder-Mead tuned PID step responses for $G_2(s)$

Table VII and Figure 6 show the parameters and step responses resulting from SA optimization.

V. CONCLUSIONS

In this paper we have presented a simple method to tune a PID controller. We give three options of optimization algorithms, which are easy to implement. In the simulations we compared to a previously reported controller tuned with Genetic Programming and the results show an improvement in the settling time.

The best results were obtained using $ITAE$ and IAE as cost functions. These can be combined with a settling time

TABLE VII

SIMULATED ANNEALING OPTIMIZATION RESULTS FOR $G_2(s)$.

Cost Function	K_p	T_i	T_d	Settling Time
$ITAE$	0.83	4.92	1.98	9.298
IAE	0.89	4.73	2.43	22.699
$S.TIME$	0.90	4.89	2.40	8.343

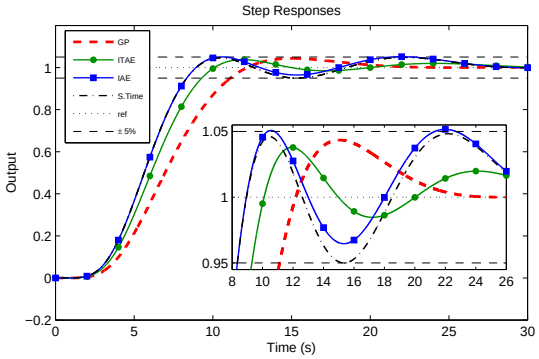


Fig. 6. Simulated Annealing tuned PID step responses for $G_2(s)$

cost function, if what we are interested in is a small settling time.

To minimize the settling time, we found that the best results are obtained by, starting from a ZN-tuned PID, perform an optimization using the ITAE or IAE as cost function, and then use the result to optimize the settling time-based cost function.

Even though all of the simulations were carried on linear plants, the same method can be used for nonlinear systems with other type of controllers.

REFERENCES

Aarts, E. & Lenstra, J. K. (2003). Simulated annealing. *Local search in combinatorial optimization*, Princeton University Press. 91–120

Astrom, K. J., Hagglund, T., Hang, C. C., & Ho, W. K. (1993). Automatic tuning and adaptation for PID controller - a survey. *Control Engineering Practice* **4**, 699–714

Chien, Hrones & Reswick (1952). On the automatic tuning of generalized passive systems. *Transactions ASME* **74**, 175–185

de Almeida, G.M. et al. (2005). Application of Genetic Programming for Fine Tuning PID Controller Parameters Designed Through Ziegler-Nichols Technique. *In Advances in Natural Computation*, 313–322

Dorf, R. C. & Bishop, R. H. (1996). Design Using Performance Indices, in Levine, W.S. (ed.). *The control handbook*, CRC Press, 169–172

Hooke, R. & Jeeves, T.A. (1961). Direct Search Solution of Numerical and Statistical Problems. *J. ACM* **8(2)**, 212–229

Nelder, J. A. & Mead, R. (1965). A simplex method for function minimization. *Computer Journal* **7**, 308–313

Vrancic, D., Peng, Y. & Strmenik, S. (1998). A new PID controller tuning method based on multiple integrations. *Control Engineering Practice* **7**, 623–633

Ziegler, J. G. & Nichols, N. B. (1942). Optimum settings for automatic controllers. *Transactions ASME* **62**, 759–768

Apéndice B

Programas en Matlab y C/C++

En este apéndice se muestra el código de lo programas que fueron utilizados para realizar la sintonización tanto en simulación como de manera experimental.

B.1 Métodos de optimización multivariable

A continuación se muestran los programas utilizados para implementar los algoritmos de optimización. Un parámetro en común que reciben los métodos es `funObj`, el cual es el nombre de la función que intentarán minimizar.

B.1.1 Hooke y Jeeves

Para la sintonización por simulación, se utilizaron los siguientes parámetros:
`x0` varía la semilla según el control a sintonizar,
`n=1,2,3` o `4` dependiendo del espacio de búsqueda,
`h=0.5, N=200, eps=0.0001, nEps=15`,
`funObj` cambia dependiendo del control a utilizar y
`op` también cambia dependiendo que función objetivo se deseé minimizar.

```
1  function [optimo,opt,hhist,mov,patron,Error]=HJ_nVar(x0,h,n,N,eps,...
2      nEps,funObj,op)
3
4  % [optimo,opt,hhist,mov,patron,Error]=HJ_nVar(x0,h,n,N,eps,nEps,
5  %   @funObj,op)
6  %
7  % Busca un mínimo para funObj de 4 variables, con el método de Hooke-
8  % Jeeves, haciendo búsqueda local en todos los ejes en R^n, a una
9  % distancia incicial h. De tener mejoría en algún movimiento, se
10 % intenta un movimiento patrón. Si en un punto ya no se encuentra
11 % mejoría en ningún sentido a una distancia de h, se cambia la
12 % distancia de búsqueda h <-- h/2.
13 %
14 % En optimo se regresa el punto que minimiza la función objetivo.
15 % En opt se regresa la trayectoria que se ha seguido hasta el óptimo.
```

```
16 % En hhist, el historial de la distancia h, conforme va avanzando.
17 % En mov, el historial de movimientos que genera el nuevo punto.
18 % En patron se indica si se realizó un movimiento patrón o no.
19 % En Error se regresa el error generado por "optimo".
20 %
21 % Parámetros de entrada:
22 % x0:     punto inicial en R^n
23 % h:      incremento ortogonal para simplex inicial
24 % n:      dimensión del problema (número de variables)
25 % N:      número máximo de iteraciones
26 % eps:    valor cercano a cero para evaluar cuando ya no hay mejoría
27 % nEps:   número máximo de iteraciones permitidas sin tener mejoría
28 % funObj: nombre de la función que se desea minimizar
29
30 movv=zeros(n,1);
31 vecH=zeros(n,1);
32 xxe=x0';
33 ee=funObj(xxe,op);
34 error_anterior=ee;
35 noMejoro=0;
36 iteracion=0;    %en caso de que no haya mejoría
37
38 opt(1,:)=xxe;
39 for i=1:N
40     hhist(i)=h;
41         %Búsqueda Local
42     for j=1:n,
43         vecH(j)=h;
44         x=xxe+vecH;
45         e=funObj(x,op);
46         if(e>=ee)
47             x=xxe-vecH;
48             e=funObj(x,op);
49             movv(j)=-1;
50         else
51             movv(j)=1;
52         end
53         if(e<ee)
54             xxe=x;
55             ee=e;
56         else
57             movv(j)=0;
58         end
59         vecH(j)=0;
60     end
61         %Hacer movimiento patrón
62     if(sum(movv.^2)~=0)
63         x=xxe+h*movv;
64         e=funObj(x,op);
65         iteracion=i;
66         if(e<ee)
67             xxe=x;
```

```
68              ee=e;
69              patron(i)=1;
70          else
71              patron(i)=0;
72          end
73      else
74          patron(i)=0;
75      end
76      mov(i,:)=movv';
77          %Reducción de h
78      if(error_anterior==ee)
79          h=h/2;
80      end
81      opt(i+1,:)=xxe';
82          %Criterios de paro
83      if(error_anterior-ee<eps)
84          noMejoro=noMejoro+1;        %No mejoró
85      else
86          noMejoro=0;                 %Sí mejoró
87      end
88      if(noMejoro==nEps)          %Paro, por no mejorar...
89          break;
90      end
91      error_anterior=ee;
92      es(i)=ee;
93  end
94  Error=ee;
95  optimo=xxe';
```

B.1.2 Nelder y Mead

Para la sintonización por simulación, se utilizaron los mismos parámetros que se presentaron para Hooke y Jeeves, ya que los dos métodos reciben los mismos argumentos. En el caso en que se utilizó este método para sintonizar experimentalmente se utilizaron los siguientes parámetros:

`x0` varía la semilla según el control a sintonizar,

`n=1,2,3` o `4` dependiendo del espacio de búsqueda,

`h=0.5, N=40, eps=0.0001, nEps=10`,

`funObj` cambia dependiendo del control a utilizar y

`op=1` o `6` representando ITAE o T_s.

```
1  function [optimo,xn,paso,error]=Simplex_nVar(x0,h,n,N,eps,nEps,...
2      funObj,op)
3
```

```
4  % [optimo,xn,paso,error]=Simplex_nVar(x0,h,n,N,eps,nEps,@funObj,op)
5  %
6  % Busca un mínimo para funObj de n variables, con el método de Nelder–
7  % Mead, construyendo un simplex (n+1 puntos en R^n) a partir de x0
8  % considerando un incremento h ortogonal.
9  %
10 % En xn se regresa las coordenadas de los los n+1 puntos del simplex,
11 % para cada iteración; xn es una matriz en R^3.
12 %
13 % xn(:,:,1)      % representaría las coordenadas de los n+1 puntos
14 %                % en la i–ésima iteración
15 %                % en este ejemplo es para n=4, con el simplex inicial
16 %                % cada renglón representa un punto del simplex
17 %                % los renglones están de mejor a peor, de arriba a
18 %                % abajo
19 % ans =
20 %          0         0         0         0
21 %     0.5000         0         0         0
22 %          0         0         0    0.5000
23 %          0         0    0.5000         0
24 %          0    0.5000         0         0
25 % En paso se regresa la operación que se realizó en cada iteración,
26 % éstas pueden ser:
27 %
28 %           're'    Reflexión
29 %           'ex'    Expansión
30 %           'co'    Contracción o Contracción Inversa
31 %           'rd'    Reducción
32 %
33 % En error se regresa el error mínimo de cada iteración
34 % En optimo se regresa las coordenadas del punto que generó el mínimo
35 % error.
36 %
37 % Parámetros de entrada:
38 % x0:     punto inicial en R^n
39 % h:      incremento ortogonal para simplex inicial
40 % n:      dimensión del problema (número de variables)
41 % N:      número máximo de iteraciones
42 % eps:    valor cercano a cero para evaluar cuando ya no hay mejoría
43 % nEps:   número máximo de iteraciones permitidas sin tener mejoría
44 % funObj: nombre de la función que se desea minimizar
45
46 %***** Simplex Inicial *********
47 x(1,:)=x0;
48 for i=2:n+1,
49     x(i,:)=x(1,:);
50     x(i,i-1)=x(1,i-1)+h;
51 end
52 noMejoro=0;
53 %***** Ordenar Simplex y Obtener Centroide ******
54 %Primero calcular f(xi)
55 tic
```

```
for i=1:n+1,
    f(i)=funObj(x(i,:),op);
end
%Ordenar
for i=1:n,
    for j=i+1:n+1,
        if(f(j)<f(i))
            buff=f(i);
            f(i)=f(j);
            f(j)=buff;
            xbuff=x(i,:);
            x(i,:)=x(j,:);
            x(j,:)=xbuff;
        end
    end
end
%Centroide
for i=1:n,
    xc(i)=(sum(x(1:n,i)))/n;
end
xn(:,:,1)=x;
error(1)=f(1);
reduccion=0;
for ind=1:N,
    ind;
    salir=0;
    %***** Reflejar ******
    for i=1:n,
        xr(i)=xc(i)+(xc(i)-x(n+1,i));
    end
    fr=funObj(xr,op);
    if(fr<=f(n)&&fr>=f(1))                          %fl<=fr<=fs
        x(n+1,:)=xr;                               %xh<-xr
        f(n+1)=fr;
        paso(ind,:)='re';
        salir=1;
    end
    if(salir==0)
        if(fr<f(1))                                %Expansión
            xe=xc+2*(xc-x(n+1,:));
            fe=funObj(xe,op);
            if(fr<fe)
                x(n+1,:)=xr;
                f(n+1)=fr;
                paso(ind,:)='re';
            else
                x(n+1,:)=xe;
                f(n+1)=fe;
                paso(ind,:)='ex';
            end
            salir=1;
        end
```

```
108     end
109     if(salir==0)
110        if(fr>f(n))
111            if(fr<f(n+1))
112                xo=xc+0.5*(xc-x(n+1,:));          %Contracción
113                paso='co';
114            else
115                xo=xc-0.5*(xc-x(n+1,:));          %Contracción Inversa
116            end
117                fo=funObj(xo,op);
118            if(fo<f(n+1))
119                x(n+1,:)=xo;
120                f(n+1)=fo;
121                paso(ind,:)='co';
122            else
123                for i=2:n+1,                      %Reducción
124                    x(i,:)=(x(i,:)+x(1,:))/2;
125                end
126                reduccion=1;
127                paso(ind,:)='rd';
128            end
129            salir=1;
130        end
131     end
132     %****** Ordenar Simplex y Obtener Centroide ******
133     %Primero calcular f(xi)
134     if(reduccion==1)
135         reduccion=0;
136         for i=2:n+1,
137             f(i)=funObj(x(i,:),op);
138         end
139     end
140     %Ordenar
141     for i=1:n,
142         for j=i+1:n+1,
143             if(f(j)<f(i))
144                 buff=f(i);
145                 f(i)=f(j);
146                 f(j)=buff;
147                 xbuff=x(i,:);
148                 x(i,:)=x(j,:);
149                 x(j,:)=xbuff;
150             end
151         end
152     end
153     %Centroide
154     for i=1:n,
155         xc(i)=(sum(x(1:n,i)))/n;
156     end
157     xn(:,:,ind+1)=x;
158     error(ind+1)=f(1);                    %Historial de errores
159     if(ind~=1)
```

```
160         if(error(ind-1)-f(1)<eps)
161             noMejoro=noMejoro+1;          %No mejora
162         else
163             noMejoro=0;                   %Sí mejora
164         end
165     end
166     if(noMejoro==nEps)          %Paro, por no mejorar...
167         break;
168     end
169     if(f(1)<0)                  %Condición de
170         break;                  %paro
171     end
172 end     %for i=1:N;
173 toc;
174 optimo=x(1,:)
```

B.1.3 Recocido simulado

Para la sintonización por simulación, se utilizaron los siguientes parámetros:

`x0` varía la semilla según el control a sintonizar,

`dmax=2, Tmax=10, Tmin=0.1, Kmax=15, alpha=0.7, nEps=15`

`funObj` cambia dependiendo del control a utilizar y

`op` también cambia dependiendo que función objetivo se deseé minimizar.

```
1  function [sbest,ebest,itbest,iteraciones]=SA_nVar(s0,dmax,Tmax,...
2      Tmin,Kmax,alpha,nEps,funObj,op)
3
4  % [sbest,ebest,itbest,iteraciones]=SA_nVar(s0,dmax,Tmax,Tmin,
5  %   Kmax,alpha,Neps,@funObj,op)
6  %
7  % Método de recocido simulado (simulated annealing) para minimizar una
8  % función objetivo de n variables.
9  %
10 % Variables de entrada
11 % s0:            estado inicial (combinación inicial)
12 % dmax:          distancia máxima para la discretización del espacio
13 %                de búsqueda
14 % Tmax:          temperatura máxima o inicial
15 % Tmin:          temperatura mínima o donde se detiene el método
16 % Kmax:          cantidad max/min de iteraciones aceptadas para una
17 %                misma temperatura (equilibrio térmico)
18 % alpha:         factor de decaimiento de la temperatura
19 %                (perfil de temp.)
20 % Neps:          Intentos sin mejorar (aceptación) en una temperatura
21 %                fija.
22 % funObj:        función objetivo a minimizar
23 %
```

```
% Variables de salida
% sbest:         la combinación que minimiza la función objetivo
% ebest:         el valor mínimo de la función evaluada en sbest
% itbest:        la iteración en donde se encontró el mínimo
% iteraciones:   total de iteraciones realizadas
%

T=Tmax;
k=0;
s=s0;
e=funObj(s,op);
sbest=s;
ebest=e;
noMejoro=0;
itbest=0;
iteraciones=0;
while(T>Tmin)
    iteraciones=iteraciones+1;
    snew=neighbour(s,T,Tmax,dmax);
    enew=funObj(snew,op);
    if(enew<ebest)
        sbest=snew;
        ebest=enew;
        itbest=iteraciones;
    end
    if (rand<acceptProb(e,enew,T))
        s=snew;
        e=enew;
        k=k+1;
        noMejoro=0;
    else
        noMejoro=noMejoro+1;
    end

    if (noMejoro>nEps)
        break;
    end
    if(k>=Kmax)
        T=alpha*T;
        k=0;
        Kmax=.95*Kmax;
    end
end
```

```
function snew=neighbour(s,T,Tmax,dmax)

% snew=neighbour(s,T,Tmax,dmax)
%
% Genera un vecino, tomando dmax como el incremento, para discretizar.
% s:    estado actual
```

```
7  % T:    Temperatura actual
8  % Tmax: Temperatura máxima
9  % dmax: distancia máxima para generar vecino
10
11 epsilon=dmax*T/Tmax;              %disminuye con la evolución de T
12 for i=1:length(s),
13     aleatorio=rand(1,1);
14     delta(i)=epsilon*aleatorio;     % un delta para cada eje
15     M(i,i)= MasMenosUno;
16 end
17 snew=s+(M*delta')';
18
19 if(s==snew)
20     snew=neighbour(s,T,Tmax,dmax);
21 end
22
23 function y=MasMenosUno()          %Números aleatorios +1 y -1
24 if(rand<.5)
25     y=1;
26 else
27     y=-1;
28 end
```

```
1  function p=acceptProb(e,enew,T)
2
3  % p=acceptProb(e,enew,T)
4  %
5  % Probabilida de aceptación para cambiar de un estado s a snew.
6  % e = E(s), energía del estado actual
7  % enew = E(snew), energía del nuevo estado
8  % T es la temperatura actual, debe de ir descendiendo tendiendo a 0.
9
10 p=exp((e-enew)/T);
```

B.2 funObj

Esta función, `funObj`, es en realidad la que define que función objetivo se minimizará, de las definidas en la Sección IV.1, además aquí podemos indicar si la sintonización se hará con el sistema simulado o de manera experimental con el péndulo en el laboratorio. A continuación mostramos el código para simular el sistema con un control PID. Posteriormente se mostrará el código necesario para hacer el experimento físicamente.

B.2.1 Simulación

A continuación mostramos la función para simular el sistema con un control PID. Esta función evalua el desempeño del controlador con los parámetros `x` bajo la función objetivo elegida por `op`.

```
function z = fevaluarPID(x,op)
% z = fevaluarPID(x,op)
%
% Salida:
% z     es el valor de la función objetivo evaluada con parámetros x
% Entrada:
% x     los parámetros a evaluar
% op    elige la función objetivo a evaluar
%       1.ITAE  2.MITAE  3.ISE   4.MISE   5.IAE   6.Settling Time

if(nargin==2)           % si no se especifica op, se considera 1
    error=op;
else
    error=1;
end

load num                % para saber cuantas veces se evaluó la
num=num+1;              % función
disp(num)
save num num

t=0:0.001:20;           % tiempo de simulación
xd=10*pi/180;           % referencia en radianes
x0=[0,0,0,0,xd];        % condiciones iniciales, son cinco sólo para
                        % el PID, ya que se usa un quinto estado para
                        % obtener la integral del error.
Kp=x(1);
Ti=x(2);
Td=x(3);

[t,x]=ode45(@pendulo_PID,t,x0,'',xd,Kp,Ti,Td);  % Simular sistema

yRef=ones(length(t),1)*xd;                      % Referencia
z=itae(t,yRef,x(:,1),error);                    % Calcular error

% ******* Penalizar por control de magnitud grande *******
for i=1:length(t)
    u(i)=miPID(x(i,:),xd,Kp,Ti,Td);
end
if(max(abs(u))>3)           %3N
    z=z*10;
end
% ******* Penalizar por control de magnitud grande *******

```

```
% ******* Penalizar por exceder límite de x3 *******
if(max(abs(x(:,3)))>.1)      %10 cm
    z=z*10;
end
% ******* Penalizar por exceder límite de x3 *******

% ******* Penalizar por exceder límite de x1 *******
if(max(abs(x(:,1)))>.6981)  %40 grados
    z=z*10;
end
% ******* Penalizar por exceder límite de x1 *******
```

En la siguiente función se tiene el modelo del péndulo en espacio de estados con un control PID. Se agregó fricción viscosa (`fv1` y `fv2`)[4] al modelo en los dos eslabones.

```
function y=pendulo_PID(t,x,xd,Kp,Ti,Td)

r=0.33;
m=0.213;
J=0.02229555312;
g=9.81;

fv1=0.01250;    %fricción viscosa primer eslabón
fv2=3.18278;    %fricción viscosa segundo eslabón

u=miPID(x,xd,Kp,Ti,Td);

x1dot=x(2);
x2dot=-1/(J+m*x(3)^2)*(m*r*x(3)*x(2)^2 + 2*m*x(3)*x(4)*x(2) + ...
       m*g*x(3)*cos(x(1)) -fv1*x(2) +r*u);
x3dot=x(4);
x4dot=1/(J+m*x(3)^2)*((J*x(3)+m*x(3)^3 + m*r^2*x(3))*x(2)^2 + ...
      2*m*r*x(3)*x(4)*x(2) - (J + m*x(3)^2)*g*sin(x(1)) + ...
      m*g*r*x(3)*cos(x(1)) -fv2*x(4) + (J/m + x(3)^2 + r^2)*u);
x5dot=xd-x(1);  %para obtener la integral del error

y=[x1dot;x2dot;x3dot;x4dot;x5dot];
```

Para generar el control se utiliza la siguiente función.

```
function u=miPID(x,xd,Kp,Ti,Td)

e=xd-x(1);                              %error
u=Kp*e+(Kp/Ti)*x(5)-Kp*Td*x(2);         %control
```

Para los otros dos tipos de controles (Ubicación de polos y Backstepping) se utilizan programas similares, donde lo que cambia es la función que genera el control, y los

[4]Estos coeficientes fueron identificados experimentalmente, sin considerar fricción seca. La fricción no se contempla en el diseño del controlador.

parámetros que se envían entre métodos.

Por último se muestra el código de la función que evalúa la función objetivo elegida por op, esta función es utilizada de la misma manera en el caso del experimento físico.

```
function e=itae(t,yRef,y,op)
% e=itae(t,yRef,y)
% Salida:
% e     el error elegido por op, o tiempo de establecimiento en su
%       caso
%
% Entrada:
% t     tiempo
% yRef  referencia
% y     salida del sistema
% op    selector del criterio de error
% 1.ITAE    2.MITAE  3.ISE   4.MISE  5.IAE   6.Settling Time

if(nargin==4)
    error=op;
else
    error=1;
end

switch error
    case 1
        %**** ITAE ****
        T=t(2)-t(1);
        e=t.*abs(y-yRef);
        e=sum(e)*T;
        %e=e/t(end);
    case 2
        %**** MITAE ****
        T=t(2)-t(1);
        tmax=length(t);
        e=0;
        for i=1:tmax,
            dif=y(i)-yRef(i);
            if(dif>0)                   % ***** Sobretiro *****
                e=e+10*t(i)*abs(dif);
            else                        % ***** No Sobretiro *****
                e=e+t(i)*abs(dif);
            end
        end
        e=e*T;
        %e=e/t(end);
    case 3
        %**** ISE ****
        T=t(2)-t(1);
        e=(yRef-y).^2;
        e=sum(e)*T;
```

```
47          %e=e/t(end);
48      case 4
49          %**** MISE ****
50          T=t(2)-t(1);
51          tmax=length(t);
52          e=0;
53          for i=1:tmax,
54              dif=y(i)-yRef(i);
55              if(dif>0)                   % ***** Sobretiro *****
56                  e=e+10*(dif)^2;
57              else                        % ***** No Sobretiro *****
58                  e=e+(dif)^2;
59              end
60          end
61          e=e*T;
62          %e=e/t(end);
63      case 5
64          %**** IAE ****
65          T=t(2)-t(1);
66          e=abs(yRef-y);
67          e=sum(e)*T;
68          %e=e/t(end);
69      case 6
70          %**** Settling Time ****
71          e=abs((yRef-y)./yRef);
72          a=find(e>0.05);
73          if(isempty(a))
74              e=0;
75          else
76              e=t(a(length(a)));
77          end
78  end
```

B.2.2 Experimento físico

Para hacer la sintonización de manera experimental, se implementaron funciones MEX, las cuales permiten utilizar rutinas escritas en C/C++ y llamarlas desde Matlab.

A continuación se muestra la función en Matlab que se utilizaría para realizar un experimento con un control PID. El parámetro `x` representa los parámetros del controlador y `perf_Index` selecciona la función objetivo que se evaluará.

```
1  function error=Evalua_PID(x,perf_Index)
2
3  xd=0.1745;          % Entrada tipo escalón de 10 grados
4
5  Kp=x(1);
6  Ti=x(2);
```

```
7  Td=x(3);
8  PID_Obs(40,Kp,Ti,Td,xd);     % Función C-MEX
9                               % genera: file,file2,file3
10 load('file.txt');
11 load('file2.txt');
12 load('file3.txt');
13 load('num');          % Contador de evaluaciones de la función
14 load('folder');       % Nombre de la carpeta donde se guardarán los
15                       % resultados de los experimentos
16 t=0:0.001:20;
17 error=itae(t',file(1:20001,8),file(1:20001,2),perf_Index);
18                       % la función itae es la misma que se usa para el
19                       % caso de simulaciones
20 disp(['Error: ',num2str(error)]);
21 disp(' ');
22 save Datos file file2 file3 Kp Ti Td error
23                       % se guardan los datos de cada experimento
24 if(num>9)
25     filename=['Resultados_PID_Simplex\',folder,'\Corrida_',...
26         num2str(num),'.mat'];
27 else
28     filename=['Resultados_PID_Simplex\',folder,'\Corrida_0',...
29         num2str(num),'.mat'];
30 end
31 num=num+1;
32 save num num
33 copyfile('Datos.mat',filename);
```

El siguiente código muestra la función MEX utilizada para un control PID. Entre las líneas 297 y 306 es donde se implementa el control deseado, para el caso en que se quiera usar otro control, se comentaría la línea con el PID y se habilitaría la línea con el control deseado, de igual manera se tendrían que ajustar los parámetros de entrada de la función.

```
1  #include <mex.h>
2
3  /*********************************************************
4   *  PID_Obs.cpp
5   *
6   *  Control PID, con observador discontinuo.
7   *  Backstepping para regresar el mecanismo al origen.
8   *
9   *  PID_Obs(tiempo,Kp,Ti,Td,xd)
10  *  tiempo: tiempo del experimento
11  *  Kp    : ganancia proporcional
12  *  Ti    : tiempo de integración
13  *  Td    : tiempo de derivación
14  *  xd    : referencia
```

```
15  *
16 ************************************************************/
17
18 #include <conio.h>
19 #include <iostream>
20 #include <string>
21 #include <stdio.h>
22 #include <wtypes.h>
23 #include <winbase.h>
24 #include <math.h>
25 #include <time.h>
26 #include <mmsystem.h>
27
28 #include "signo.h"
29 #include "d:\rodrigo\C_Files\sensoray\Drivers\626\Win626.h"
30 #include "d:\rodrigo\C_Files\sensoray\Drivers\626\App626.h"
31
32
33 #define SAMPLE_RATE 1000.0
34 #define EOPL                0x80
35 #define PERFORMANCE_COUNTS  3570000       //Empíricamente...
36 #define RANGE_5V            0x10
37
38 #define     r           0.33
39 #define     m           0.213
40 #define     J           0.020347774776059
41 #define     g           9.81
42
43 #define a               0
44 #define b               500
45 #define PI              3.1415926535897931
46 #define A_RAD           .0004353256224965591
47 #define A_MTS           .00001991758241758242
48 #define x3_MAX          0.10
49 #define SIZE_VEC        9
50 /************ Encoders ***************/
51 #define CRA1            0x00
52 #define CRB1            0x02
53 #define PRE1ALSW        0x0C    /* register offset values */
54 #define PRE1AMSW        0x0E
55 #define LATCH1ALSW      0x0C
56 #define LATCH1AMSW      0x0E
57
58 #define CRA2            0x04
59 #define CRB2            0x06
60 #define PRE2ALSW        0x14
61 #define PRE2AMSW        0x16
62 #define LATCH2ALSW      0x14
63 #define LATCH2AMSW      0x16
64 /************ Encoders ***************/
65
66 using namespace std;
```

```
67
68 typedef DWORD HBD;
69
70 /* sleepfor toma un LARGE_INTEGER (64 bits) y se detiene después de
71  * pasar el número de performance_clicks indicados en la entrada, para
72  * de esta manera controlar el periodo de muestreo, approx. 1ms. En la
73  * computadora donde se hicieron los experimentos se tienen 3570000
74  * performance_clicks por segundo, de esta manera se obtiene un
75  * desempenho en tiempo real aproximado */
76     void sleepfor(LARGE_INTEGER from, long performance_clicks)
77     {
78         LARGE_INTEGER pc0;
79         long pc_diff;
80
81         QueryPerformanceCounter(&pc0);
82         pc_diff = (long) pc0.QuadPart - (long) from.QuadPart;
83
84         if(pc_diff > performance_clicks)
85             printf(".");
86
87         while(pc_diff < performance_clicks)
88         {
89             QueryPerformanceCounter(&pc0);
90             pc_diff = (long) pc0.QuadPart - (long) from.QuadPart;
91         }
92
93         return;
94     }
95
96 void PID_Obs(double max_time[], double _Kp[], double _Ti[],
97             double _Td[], double Ref[])
98 {
99     HBD hbd=0;
100    BYTE poll_list[16] = {0x10 | EOPL, 0x11, 0x12, 0x13, 0x14, 0x15, \
101                          0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, \
102                          0x1d, 0x1e, 0x1f | EOPL};
103    char key = 0;
104    int i,ncount, dacdata = 0;
105    int max_count,mitad,ultimo;
106    int frenando;
107    int origen=0;
108    int salir=0;
109    float tiempo_salida=0;
110    int output1, output2, output0;
111    float control;
112    float xd=(float)0.1745;
113    float k=200;                //Ganancia Backstepping
114    float Kp,Ti,Td;     //Ganancias PID
115    float output1Float, output2Float, output0Float;
116    int exceed=0;
117    int seconds=0;
118    long sample_period;
```

```
119     LARGE_INTEGER hr0;
120     double elapsed_time,freq;
121     clock_t start,stop;
122 
123     float t=0;
124 
125     Kp=_Kp[0];
126     Ti=_Ti[0];
127     Td=_Td[0];
128     printf("Kp = %.4f \nTi = %.4f \nTd = %.4f\n",Kp,Ti,Td);
129 
130     /********** Encoders **********/
131     WORD ldata, hdata, ldata2, hdata2; /* unsigned short (16bits) */
132     int enc_count, enc_count2;     /* encoder 1 y 2 */
133 
134     int start_count, start_count2;   /* para el offset */
135     float x1, x3, x2, x4;
136     float x1_ant=0, x3_ant=0;
137     /********** Encoders **********/
138 
139     /************** Observador ***************/
140     float x_vec[SIZE_VEC]={0.0};  //Estados observados y perturbación
141     float xt1,xt2,xt3,xt4,pert;
142     float error, error_dot, int_error;
143     /************** Observador ***************/
144 
145 //---*****WARNING!*****---DO NOT CHANGE THE CODE BETWEEN WARNINGS------
146     // periodo de muestreo en performance counts
147     sample_period = PERFORMANCE_COUNTS/SAMPLE_RATE;
148     poll_list[0] = 0 | RANGE_5V | EOPL;
149     // Prioridad del programa
150     SetPriorityClass(GetCurrentProcess(),REALTIME_PRIORITY_CLASS);
151     // Inicializar la tarjeta Sensoray 626
152     S626_DLLOpen();
153     S626_OpenBoard( hbd, 0, 0, 3);
154     // Inicializar el ADC
155     S626_ResetADC(hbd, poll_list);
156 //--***** END WARNING! *****-----------------------------------------------
157 
158     /********* Preparar archivos para guardar los datos **********/
159     FILE *file, *file2, *file3;
160     file = fopen("file.txt","w");
161     file2 = fopen("file2.txt","w");
162     file3 = fopen("file3.txt","w");
163     /********* Preparar archivos para guardar los datos **********/
164 
165     /********* inicializar encoders **********/
166     S626_RegWrite(hbd,CRA1,0x0200);
167     S626_RegWrite(hbd,CRB1,0x0C50);
168     S626_RegWrite(hbd,CRA2,0x0200);
169     S626_RegWrite(hbd,CRB2,0x0C50);
170     /********* inicializar encoders **********/
```

```
171
172     /************ Primer Lectura Encoders (OFFSET) ************/
173     for(i=0;i<10;i++)
174     {
175     /* read in low 16 bits into variable ldata */
176     ldata = S626_RegRead(hbd, LATCH1ALSW);
177     ldata2 = S626_RegRead(hbd,LATCH2ALSW);
178     /* read in bits 23 to 16 into variable hdata */
179     hdata = S626_RegRead(hbd, LATCH1AMSW);
180     hdata2 = S626_RegRead(hbd, LATCH2AMSW);
181     /*left shift hdata by 16 bits */
182     enc_count = ((int) hdata) << 16;
183     start_count = enc_count | ((int) ldata);
184     enc_count2 = ((int) hdata2) << 16;
185     start_count2 = enc_count2 | ((int) ldata2);
186     }
187     printf("Start 1: %d\nStart 2: %d\n",start_count, start_count2);
188     /************ Primer Lectura Encoders (OFFSET) ************/
189
190     printf("Sart, Now!!!");
191     mexEvalString("drawnow;");
192
193     // Inicia un reloj de sistema y contador para determinar la
194     // frecuencia de adquisición de datos
195     start = clock();
196     ncount=0;
197     // obtener los performance counts del sistema AHORA, en hr0
198     QueryPerformanceCounter(&hr0);
199
200     t=0;
201     max_count=(int)max_time[0]*1000+1;
202     mitad=(max_count-1)/2+1;
203
204     output0=0;   // control
205     control=(float)0;
206     frenando=0;
207     //xd=(float)0.1745;
208     xd=Ref[0];
209     // correr hasta que se acabe el tiempo (elegido por el usuario)
210     while(ncount<max_count)
211     {
212         /*********** Leer Encoders ***********/
213             ldata = S626_RegRead(hbd, LATCH1ALSW);
214             hdata = S626_RegRead(hbd, LATCH1AMSW);
215             enc_count = ((int) hdata) << 16;
216             enc_count = enc_count | ((int) ldata);
217             enc_count -= start_count; // so we begin from 0
218             ldata2 = S626_RegRead(hbd, LATCH2ALSW);
219             hdata2 = S626_RegRead(hbd, LATCH2AMSW);
220             enc_count2 = ((int) hdata2) << 16;
221             enc_count2 = enc_count2 | ((int) ldata2);
222             enc_count2 -= start_count2; // so we begin from 0
```

```
223         /*********** Leer Encoders ***********/
224
225         x1=(float)enc_count*A_RAD;      //ángulo teta (radianes)
226         x3=(float)enc_count2*A_MTS;     //distancia x (mts)
227         x2=(x1-x1_ant)*1000;        //igual que entre 1ms
228         x4=(x3-x3_ant)*1000;        //igual que entre 1ms
229         x1_ant=x1;
230         x3_ant=x3;
231
232         if(t>0)
233         {
234             /********* Observador *******/
235             Observador(t,x_vec,x1,x3,control,xd);
236             xt1=x_vec[0];
237             xt2=x_vec[2];
238             xt3=x_vec[4];
239             xt4=x_vec[6];
240             pert=x_vec[3];
241             int_error=x_vec[8];
242             /********* Observador *******/
243         }
244         else
245         {
246             xt1=0;
247             xt2=0;
248             xt3=0;
249             xt4=0;
250             pert=0;
251             int_error=0;
252         }
253         error=xd-x1;
254         error_dot=-xt2;
255         if(ncount>mitad && exceed!=3)   /// Regresar al Origen ///
256         {
257             if(frenando<1000)
258             {
259                 if(x2==0)
260                 {
261                     frenando++;
262                     ultimo=ncount;
263                 }
264                 else
265                 {
266                     frenando=0;
267                 }
268                 control=0;
269                 x_vec[2]=0;
270                 x_vec[3]=0;
271                 x_vec[5]=0;
272                 x_vec[7]=0;
273             }
274             else
```

```
275             {
276                 exceed=2;
277                 xd=0;
278                 control=Backstepping(x1,xt2,x3,xt4,1,0,k,xd,pert);
279                 if(x2==0 && x4==0 && x1>-.0001745 && x1<.0001745)
280                 {               // error menor a 0.01 grados
281                     origen++;
282                 }
283                 else
284                 {
285                     origen=0;
286                 }
287                 if(origen>199)
288                 {
289                     //salir=1;
290                     exceed=3;
291                     control=0;
292                     tiempo_salida=t;
293                 }
294             }
295         }
296         else
297         {               ///// Control PID /////
298             control=PID(error,error_dot,int_error,Kp,Ti,Td);
299                         ///// Control Polos /////
300             //control=POLOS(x1,xt2,x3,xt4,K,Kb,xd);
301                         ///// Control Backstepping /////
302             //control=Backstepping(x1,xt2,x3,xt4,k1,k2,k3,xd,pert);
303             /** Retroalimentar la perturbación para PID o Polos ***/
304             //control=control+pert*(J+m*x3*x3)/r;
305             /** Retroalimentar la perturbación para PID o Polos ***/
306         }
307         if((x3>-x3_MAX && x3<x3_MAX && exceed==0) || exceed==2)
308         {
309             output0Float=control*192.2901;
310                                 //output0=control/kf*(8191/10)
311             output0=(int)output0Float;
312         }
313         else
314         {
315             /**** Paro de seguridad ****/
316             output0=0;
317             if(exceed!=3)
318                 exceed=1;
319         }
320         output1Float=x1*81.91*180/3.1416;
321         output1=(int)output1Float;
322         output2Float=x3*16382;
323         output2=(int)output2Float;
324         S626_WriteDAC(hbd,0,output0);
325         S626_WriteDAC(hbd,0,output0);
326         S626_WriteDAC(hbd,1,output1);
```

```
327         S626_WriteDAC(hbd,1,output1);
328         S626_WriteDAC(hbd,2,output2);
329         S626_WriteDAC(hbd,2,output2);
330         /********* Print to File *************/
331         fprintf(file,"%d %3.8f %3.8f %3.8f %3.8f %3.8f %d %3.8f\n",
332                 ncount,x1,x2,x3,x4,control,output0,xd); //ncount en ms
333         fprintf(file2,"%d %3.8f %3.8f %3.8f %3.8f %3.8f\n",
334                 ncount,xt1,xt2,xt3,xt4,pert);
335         fprintf(file3,"%d %3.8f %3.8f %3.8f\n",
336                 ncount,error,error_dot,int_error);
337         /********* Print to File *************/
338         ncount++;
339         if(salir==1)
340             break;
341         // this loop should execute once per msec
342         sleepfor(hr0,sample_period);
343         QueryPerformanceCounter(&hr0); // counts from right NOW
344         t+=0.001f;
345     }   // while
346     stop = clock();
347
348     /******** Calcula tiempo de Ejecución, y Muestreo *******/
349     elapsed_time = (double) (stop - start) / CLOCKS_PER_SEC;
350     freq = ((double)ncount)/elapsed_time;
351     printf("\n%d cycles in %f sec (%f Hz, %f sec))\n",
352         ncount,elapsed_time, freq, 1.0/freq);
353     /******** Calcula tiempo de Ejecucion, y Muestreo *******/
354
355     /******* Cierra Tarjeta y pone Ceros en las salidas ******/
356     // Se escribe dos veces la salida debido a un posible error
357     // reportado con la tarjeta
358     for(i=0; i<4; i++)
359     {
360         S626_WriteDAC(hbd,(WORD) i,0);
361         S626_WriteDAC(hbd,(WORD) i,0);
362     }
363     S626_CloseBoard( hbd );
364     S626_DLLClose();
365     /******* Cierra Tarjeta y pone Ceros en las salidas ******/
366
367     fclose(file);
368     fclose(file2);
369     fclose(file3);
370     // tiempo en que regresó al origen
371     printf("tiempo_salida = %.3f\n",tiempo_salida);
372 }
373
374 void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
375                 const mxArray *prhs[])
376 {
377     double *Kp, *Ti, *Td, *time, *Ref;
378     mwSize mrows, ncols;
```

```
379     // Verificar número de entradas y salidas.
380     if(nrhs!=5)
381         mexErrMsgTxt("Se requieren CINCO entradas,\n"
382                 "el tiempo de captura, Kp, Ti, Td y xd");
383     else if(nlhs!=0)
384         mexErrMsgTxt("Muchos argumentos de salida");
385         mrows=mxGetM(prhs[0]);
386     ncols=mxGetN(prhs[0]);
387     if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
388         !(mrows==1 && ncols==1) )
389         mexErrMsgTxt("La entrada debe ser un escalar no complejo");
390     mrows=mxGetM(prhs[1]);
391     ncols=mxGetN(prhs[1]);
392     if( !mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
393         !(mrows==1 && ncols==1) )
394         mexErrMsgTxt("La entrada debe ser un escalar no complejo");
395     mrows=mxGetM(prhs[2]);
396     ncols=mxGetN(prhs[2]);
397     if( !mxIsDouble(prhs[2]) || mxIsComplex(prhs[2]) ||
398         !(mrows==1 && ncols==1) )
399         mexErrMsgTxt("La entrada debe ser un escalar no complejo");
400     mrows=mxGetM(prhs[3]);
401     ncols=mxGetN(prhs[3]);
402     if( !mxIsDouble(prhs[3]) || mxIsComplex(prhs[3]) ||
403         !(mrows==1 && ncols==1) )
404         mexErrMsgTxt("La entrada debe ser un escalar no complejo");
405     mrows=mxGetM(prhs[4]);
406     ncols=mxGetN(prhs[4]);
407     if( !mxIsDouble(prhs[4]) || mxIsComplex(prhs[4]) ||
408         !(mrows==1 && ncols==1) )
409         mexErrMsgTxt("La entrada debe ser un escalar no complejo");
410     time=mxGetPr(prhs[0]);
411     Kp=mxGetPr(prhs[1]);
412     Ti=mxGetPr(prhs[2]);
413     Td=mxGetPr(prhs[3]);
414     Ref=mxGetPr(prhs[4]);
415     //Rutina en C/C++ adaptada
416     PID_Obs(time,Kp,Ti,Td,Ref);
417 }
```

El siguiente código muestra el archivo `signo.cpp`, el cual implementa algunas funciones que son requeridas por las funciones MEX implementadas en los controladores y el observador.

```
1 #include <math.h>
2
3 #define     A           200
4 #define     B           48
5 #define     C           17
```

```
6  #define     c11         A
7  #define     c12         A
8  #define     c13         A
9  #define     c14         A
10 #define     c21         B
11 #define     c22         B
12 #define     c23         B
13 #define     c24         B
14 #define     c31         C
15 #define     c32         C
16 #define     c33         C
17 #define     c34         C
18 #define     r           0.33
19 #define     m           0.213
20 #define     J           0.020347774776059
21 #define     g           9.81
22 #define     h           0.001
23 #define     SIZE_VEC    9
24
25 float sign(float x)
26 {
27     if(x>0)
28         return 1;
29     else if(x==0)
30         return 0;
31     return -1;
32 }
33
34 float sech(float x)
35 {
36     return 1/cosh(x);
37 }
38
39 void constVector(float vec_x[], float cte, float vec_y[])
40 {
41     int i;
42     for(i=0;i<SIZE_VEC;i++)
43     {
44         vec_y[i]=vec_x[i]*cte;
45     }
46 }
47
48 void sumaVectores(float vec_y[], float vec_a[], float vec_b[])
49 {
50     int i;
51     for(i=0;i<SIZE_VEC;i++)
52     {
53         vec_y[i]=vec_a[i]+vec_b[i];
54     }
55 }
56
57 void ObsPert(float t,float xtt[], float x1, float x3, float u,
```

```
58          float x_est[], float xd)
59 {
60     float x1t,x2t,x3t,x4t,w1,w2,w3,w4,e1,e2,e3,e4,error;
61 
62     error=xd-x1;
63 
64     x1t=xtt[0];
65     w1=xtt[1];
66     x2t=xtt[2];
67     w2=xtt[3];
68 
69     e1=x1-x1t;
70     e2=w1-x2t;
71 
72     x3t=xtt[4];
73     w3=xtt[5];
74     x4t=xtt[6];
75     w4=xtt[7];
76 
77     e3=x3-x3t;
78     e4=w3-x4t;
79 
80     x_est[0]=c11*e1+w1;
81     x_est[1]=c21*e1+c31*sign(e1);
82     x_est[2]=(float)(-1/(J+m*x3*x3)*(m*r*x3*x2t*x2t + 2*m*x3*x4t*x2t +
83             m*g*x3*cos(x1) +r*u) + c12*(e2) + w2);
84     x_est[3]=c22*(e2) + c32*sign(e2);
85 
86     x_est[4]=c13*e3 + w3;
87     x_est[5]=c23*e3 + c33*sign(e3);
88     x_est[6]=(float)(1/(J+m*x3*x3)*((J*x3+m*x3*x3*x3 + m*x3*x3*x3 +
89             m*r*r*x3)*x2t + 2*m*r*x4t*x2t - (J + m*x3*x3)*g*sin(x1) +
90             m*g*r*x3*cos(x1) + (J/m + x3*x3 + r*r)*u) +
91             c14*e4 + w4);
92     x_est[7]=c24*e4 + c34*sign(e4);
93     x_est[8]=error;  //Para obtener la integral del error, para el PID
94 }
95 
96 void Observador(float t,float xtt[],float x1,float x3,float u,float xd)
97 {
98     //t es el tiempo anterior, xtt el estado anterior,
99     //x1 y x3 las mediciones actuales,  u el control actual
100    //////// ode4 ////////
101    float k1[SIZE_VEC],k2[SIZE_VEC],k3[SIZE_VEC],k4[SIZE_VEC];
102    float k_buff[SIZE_VEC], xtt_buff[SIZE_VEC];
103    int   i;
104 
105    for(i=0;i<SIZE_VEC;i++)
106    {
107        xtt_buff[i]=xtt[i];     //Respaldar xtt
108    }
109 
```

```
110     ObsPert(t,xtt,x1,x3,u,k1,xd);                    //k1
111
112     constVector(k1,(float)h/2,k_buff);               //k_buff=k1*h/2
113     sumaVectores(xtt_buff,xtt,k_buff);           //xtt_buff=xtt+1/2*h*k1
114     ObsPert(t+(float)h/2,xtt_buff,x1,x3,u,k2,xd);    //k2
115
116     constVector(k2,(float)h/2,k_buff);               //k_buff=k2*h/2
117     sumaVectores(xtt_buff,xtt,k_buff);           //xtt_buff=xtt+1/2*h*k2
118     ObsPert(t+(float)h/2,xtt_buff,x1,x3,u,k3,xd);    //k3
119
120     constVector(k3,(float)h,k_buff);         //k_buff=k3*h
121     sumaVectores(xtt_buff,xtt,k_buff);       //xtt_buff=xtt+h*k3
122     ObsPert(t,xtt_buff,x1,x3,u,k4,xd);       //k4
123
124     sumaVectores(k_buff,k2,k3);                //k_buff=k2+k3
125     constVector(k_buff,2,k_buff);              //k_buff=2*(k2+k3)
126     sumaVectores(k_buff,k_buff,k4);            //k_buff=2*(k2+k3)+k4
127     sumaVectores(k_buff,k_buff,k1);            //k_buff=k1+2*(k2+k3)+k4
128     constVector(k_buff,(float)h/6,k_buff);
129                                  //k_buff=h/6*(k1+2*(k2+k3)+k4)
130     sumaVectores(xtt,xtt,k_buff);       //xtt=xtt+h/6*(k1+2*(k2+k3)+k4)
131
132     ///// en xtt se regresa la predicción /////
133 }
134
135 float Backstepping(float x1, float x2, float x3, float x4, float k1,
136         float k2, float k3, float xd, float pert)
137 {
138     float ua;
139     float L;
140     float xt1;
141     L=10000;
142
143     xt1=x1-xd;
144     ua=-xt1 -(k1+L*k2*((sech(L*xt1))*(sech(L*xt1))))*x2
145             -k3*(k1*xt1+x2+k2*tanh(L*xt1));
146     return (float)(-(1/r)*(m*r*x3*x2*x2 + 2*m*x3*x4*x2
147             + m*g*x3*cos(x1) + (J+m*x3*x3)*(ua-pert)));
148 }
149
150 float PID(float error, float error_dot, float int_error, float Kp,
151         float Ti, float Td)
152 {
153     return Kp*(error+int_error/Ti+error_dot*Td);
154 }
155
156 float POLOS(float x1, float x2, float x3, float x4, float K[],
157         float Kb, float xd)
158 {
159     return -K[0]*(x1-xd)-K[1]*x2-K[2]*x3-K[3]*x4;
160 }
```