TECNOLÓGICO
DE MONTERREY.

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Ciudad de México

División de Ingeniería y Arquitectura

Ingeniería en Electrónica y Comunicaciones

Departamento de Ingeniería Eléctrica y Electrónica

Control inteligente con aplicación al péndulo invertido

Autor: Eugenio Álvarez Ibargüengoitia [redacted]

Asesor: Dr. Pedro Ponce Cruz

México D. F. a 30 de abril de 2004.

INDICE

1. Introducción

El péndulo invertido es un ejemplo clásico de un sistema no lineal de naturaleza inestable, que ya ha sido solucionado de muchas formas pero que continúa siendo un ejemplo de optimización y prueba de nuevas técnicas de control.
El sistema de péndulo invertido está compuesto por una barra rígida y un carro al que la barra está unida por una chumacera que le da libertad de rotar; la barra asume una unión libre de fricción de un grado de libertad. El carro se mueve en una vía hacia su derecha o izquierda, dependiendo de la fuerza ejercida en él. El objetivo de control consiste en balancear la barra empezando desde condiciones no cero por medio de ejercer la fuerza apropiada al carro, y mantenerlo balanceado a pesar de posibles perturbaciones y de la inestabilidad del sistema [1].
Éste sistema dinámico se puede caracterizar por cuatro variables de estado, a saber:

$$x_s = (\theta \quad \dot{\theta} \quad x \quad \dot{x})^T \tag{1}$$

donde:

- θ es el ángulo de la barra con respecto a el eje vertical (o horizontal si así se desea),
- $\dot{\theta}$ es la velocidad angular de la barra,
- x es la posición del carro en la vía, y
- $\dot{x}$ es la velocidad del carro.

Como se mencionó, el objetivo de control es mantener el carro en su posición central x=0 de tal forma que el péndulo continúe en su posición vertical. Ello en nuestro caso significa,

$$\theta^d = \dot{\theta}^d = x^d = \dot{x}^d = 0 \tag{2}$$

Para el proyecto se asume el siguiente sistema, y cuyos parámetros son:

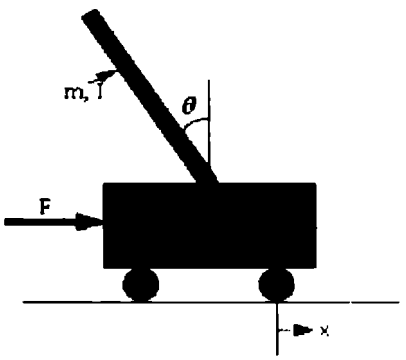


Figura 1. Diagrama esquemático del sistema de péndulo invertido

Tabla 1. Parámetros del péndulo invertido simulado.

M	Masa del carro	0.455 kg
m	Masa del péndulo	0.21 kg
l	Longitud del centro de masa del péndulo	0.305 m
I	Inercia del péndulo	0.006 $kg*m^2$

El péndulo invertido es de gran relevancia por su aplicación en otros sistemas reales. En el área militar es ampliamente desarrollado al compararlo con el lanzamiento de un cohete teledirigido, ya que al momento de la expulsión hay perturbaciones ocasionadas por la explosión y se debe mantener la posición de salida deseada. De la misma forma se han realizado trabajos para los lanzamientos espaciales. También, el caminar de un ser bípedo puede ser comparado a movimientos del péndulo invertido, ya que debe mantenerse en equilibrio [2].
Existen ya muchas soluciones al problema del péndulo invertido, lo que ha llevado a aumentar su complejidad; actualmente se analizan triples o cuádruples péndulos, con características flexibles (no barras rígidas) y en más de una dimensión. Tales sistemas tienen más entradas y salidas que controlar, haciéndolos mucho más inestables y no lineales.

1.1. Estado del arte

La ley de control mediante un PID convencional es complejo para sistemas con una entrada y dos salidas (SIMO), como en el caso del péndulo invertido. Por ello, las teorías de control moderno se utilizan generalmente para el diseño de control de éstos sistemas; dichas técnicas incluyen retroalimentación de estado, estrategias de control adaptivo, aplicación de redes neuronales para similar las combinaciones de control entrada/salida, controladores adaptivos o inteligentes mediante redes neuronales, y más recientemente integración de controladores con redes neuronales y lógica difusa, principalmente porque el control difuso requiere una ley de control experta del péndulo invertido escrita en reglas si-entonces. Uno de los últimos controladores diseñados (publicados en IEEE) utiliza algoritmos genéticos, redes neuronales y lógica difusa para sintonizar un controlador PID.

Muchas arquitecturas de redes neuronales se han propuesto para controlar el péndulo invertido [3]. Por ejemplo el control con modelado hacia delante se estudió por Jordan y Jacobs [4], donde el sistema de control aprende un modelo que relaciona el estado actual de la planta y la señal de control actual mediante una predicción de la falla futura. También, el aprendizaje de control del péndulo invertido por medio de un neuro-controlador fue propuesto por Kitamura y Sayito [5], ellos compusieron el sistema con un controlador neuronal, un generador de la salida deseada y un evaluador. En el generador de la salida deseada al siguiente tiempo, dada la posición y velocidad del carro, para moverlo a una posición especificada, el ángulo y la velocidad angular del péndulo son generados por un par de ecuaciones que ellos toman en consideración como conocimiento previo del comportamiento del péndulo. El evaluador se usa para decidir si la salida del controlador es correcta o falsa y dependiendo del caso, generar una señal maestra para el entrenamiento del neuro-controlador, basado en la diferencia entre el valor deseado y la salida de control.

Como se ha comentado anteriormente, existen cada vez más métodos de control y la gran mayoría son puestos a prueba ante el problema del péndulo invertido, que a su vez se ha hecho más complejo aumentando a usar barras flexibles libres en múltiples ejes.
El control inteligente ha cobrado un nuevo giro en el mundo del control, aplicando técnicas de lógica difusa, redes neuronales y algoritmos de optimización. Cada vez nuevos métodos son inventados, tanto al mejorar cada una de las técnicas, como al integrarlas aprovechando sus ventajas y capacidades.
En los últimos años, los trabajos de investigación han trabajado los siguientes temas, que marcan el estado del arte. Por ejemplo, se realiza el levantamiento desde la posición de equilibrio natural con el péndulo detenido abajo, hasta su equilibrio en la parte superior [6, 10]. También controladores neuronales con diferentes configuraciones [7, 8, 11], aplicaciones difusas al control del péndulo invertido [9, 13, 14, 15], aplicaciones con algoritmos genéticos [12].

1.2. Organización

El presente escrito comienza con el modelado del sistema, hasta obtener un diagrama de bloques en SIMULINK, para posteriormente linealizarlo para poder sintonizar un controlador LQR (Linear Quadratic Regulator). Para obtener mayor experiencia en el control del sistema, se obtuvieron controladores con PID y retroalimentación de estado con LQR, y se presentan sus resultados. Se continúa con un controlador difuso tipo Mamdani, detallando la configuración del sistema difuso. Posteriormente, se emplean redes neuronales como identificador del sistema, para utilizar dicha red como controlador inverso del sistema; se presentan algunos resultados y pruebas. En la sección posterior se emplea un sistema ANFIS (Adaptive-Network-based Fuzzy Inference System) para controlar el sistema. Se termina la simulación con una optimización del ANFIS empleando algoritmos genéticos. Por último, se describe y presentan los resultados de la implementación del péndulo invertido utilizando partes de una impresora.

2. Modelo del péndulo invertido

Empezamos considerando el sistema mediante los siguientes dos diagramas de cuerpo libre (figura 2):

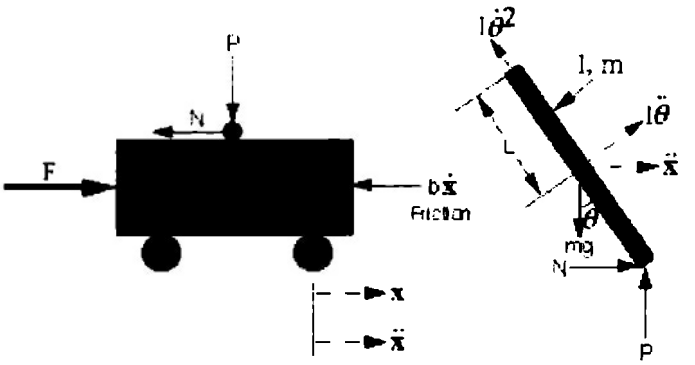


Figura 2. Diagramas de cuerpo libre del sistema de péndulo invertido

Se define el vector $q = (\theta, x)^T$

Realizamos el método Euler-Lagrange

$$\frac{d}{dt}\left(\frac{\partial \lambda}{\partial q}\right) - \frac{\partial \lambda}{\partial q} = \tau \tag{3}$$

Donde $\lambda = k_{TOTAL} - U_{TOTAL}$

Es la diferencia de energías cinética y potencial.

$$k_1 = \frac{1}{2} m_1 v_1^{\ 2} = \frac{1}{2} m_1 \dot{x}^2 \qquad U_1 = m_1 g l$$

$$k_2 = \frac{1}{2} m_2 v_2^{\ 2} + \frac{1}{2} I \omega^2 \qquad U_2 = m_2 g (l + l\cos\theta)$$

Y sabemos que $k_{Tot} = \frac{1}{2} q^T H(q) q$

Donde $H(q) = H(q)^T \rangle 0$

Obtenemos

$$H(q) = \begin{bmatrix} I + m_2 l^2 & -m_2 l\cos\theta \\ -m_2 l\cos\theta & m_1 + m_2 \end{bmatrix} \tag{4}$$

Entonces la ecuación de Euler-Lagrange resulta [16]

$$H(q)\ddot{q} + \left(\dot{H}(q)\dot{q} - \frac{\partial k}{\partial q}\right) + \frac{\partial U}{\partial q} = \tau = H(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{5}$$

Donde C define el término de Coriolis, y g el de la gravedad. El procedimiento completo se encuentra en el apéndice 2.

Luego,

$$\ddot{q} = \begin{pmatrix} \ddot{\theta} \\ \ddot{x} \end{pmatrix} = H(q)^{-1}\left(\tau - C(q,\dot{q})\dot{q} - g(q)\right) \tag{6}$$

Asumiendo que no hay fricción (coeficiente b=0), y considerando el modelo únicamente del péndulo sin las implicaciones de control de motor para ejercer la fuerza de balanceo (F= τ), podemos definir el sistema por las siguientes ecuaciones diferenciales de segundo orden.

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\frac{-\tau - m_2 l\dot{\theta}^2 \sin\theta}{m_1 + m_2}\right)}{l\left(\frac{4}{3} - \frac{m_2\cos^2\theta}{m_1 + m_2}\right)} \tag{7}$$

$$\ddot{x} = \frac{\tau + m_2 l\left(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right)}{m_1 + m_2} \tag{8}$$

El sistema, con las condiciones ya comentadas, fue validado por varios artículos [17, 18], y siguiendo las ecuaciones se construyó el modelo en Simulink/MATLAB como se muestra en la figura 5.

Encontramos que el sistema responde a una señal escalón como se muestra en la figura 3:

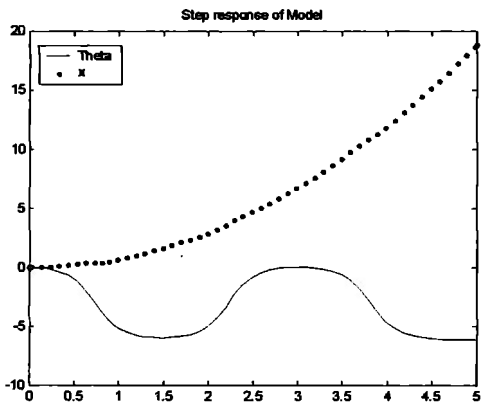


Figura 3. Respuesta a escalón del péndulo invertido.

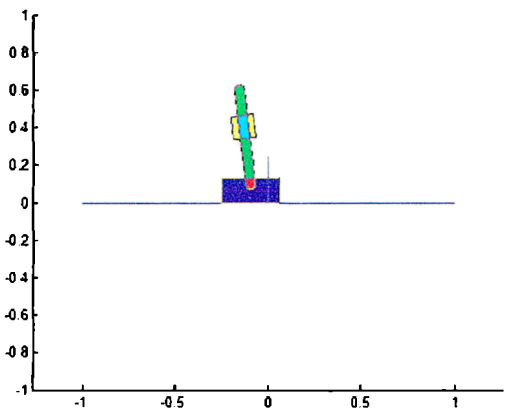


Figura 4. Simulador gráfico en MATLAB del péndulo invertido

Los detalles de modelación, con el procedimiento completo, se muestran en el apéndice 2. Ya que se tuvo el simulador del péndulo invertido, se implementó una etapa gráfica (figura 4). Después de comprender el simulador realizado por el Dr. Ernesto Olguín [37], se modificó la parte gráfica para implementarse específicamente en el modelo previamente expuesto. El programa del simulador se muestra en el apéndice 6.

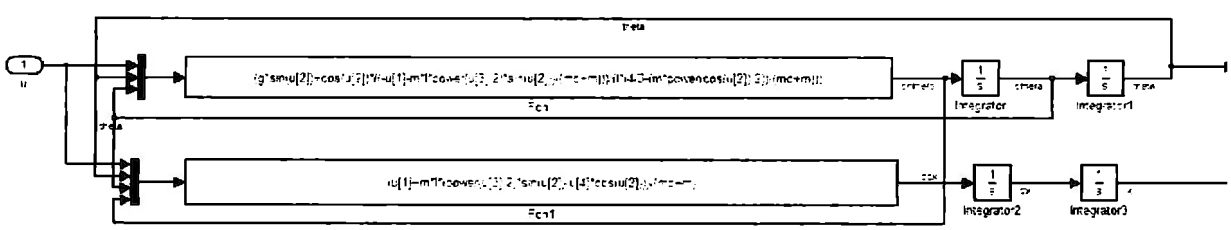


Figura 5. Diagrama de bloques del simulador que contiene las ecuaciones del sistema de péndulo invertido

3. Controladores

El presente capítulo incluye todos los controladores que se simularon para el péndulo invertido, que van desde técnicas de control tradicional, como el PID, y de control robusto (LQR), para luego introducir a los controladores inteligentes como el difuso, de redes neuronales, ANFIS y optimización con algoritmos genéticos. Para cada uno se presenta una descripción, detalle de sus características, y resultados.

3.1. Controlador PID

Aunque el sistema es muy complicado de controlar, es una buena idea utilizar un controlador sencillo que sirva como punto inicial para controladores más complejos. También, sabiendo que el control inteligente requiere de un experto que proporcione la lógica de control, se buscó crear controladores tradicionales que dieran pistas para un buen controlador inteligente. Con dichas intenciones, se buscaron por prueba y error las ganancias de el controlador PID para controlar tanto el ángulo como la posición. Para lograrlo, por simplicidad, se emplearon dos controladores PID's: uno para el ángulo y otro para la posición, y se sumaron las señales de control. Los valores del controlador para ángulo son Kp=18.8, Kd=0.011 y Ki=0.66, y para el controlador de posición Kp=0.6, Kd=0.0105 y Ki=0.2. También se utilizaron ganancias en la señal de error para dar prioridades al error y a las salidas de control.
En la figura 6 se muestra el sistema con dos señales de perturbación, y en la figura 7 los resultados encontrados.

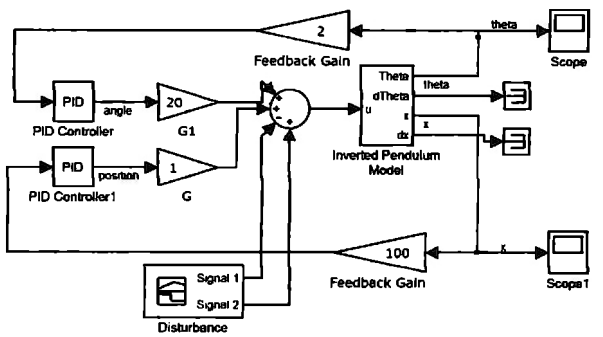


Figura 6. Diagrama de bloques del controlador PID

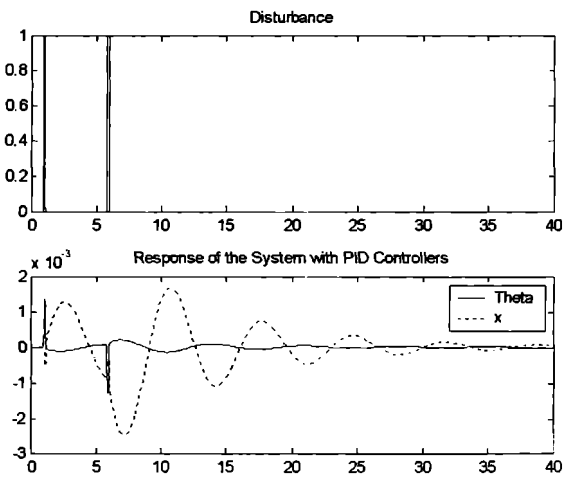


Figura 7. Resultados a perturbaciones de la simulación del controlador PID

Se muestra una muy buena respuesta a perturbaciones tipo impulso. Es importante mencionar que es muy difícil de sintonizar este tipo de controladores si no se desea (por supuesta complejidad) linealizar el sistema, y que sólo trabajan para el punto de operación al que fueron diseñados. Para el siguiente controlador sí se linealizó el modelo.

3.2. Controlador robusto LQR

Otra forma de emplear un controlador no inteligente, como se sugiere en una demostración del "Robust Control Toolbox" de MATLAB [36], consiste en emplear un PID para controlar la posición, y un LQR (Linear Quadratic Regulador) con un estimador discreto de estado para estabilizar el péndulo, como se muestra en la figura 8.

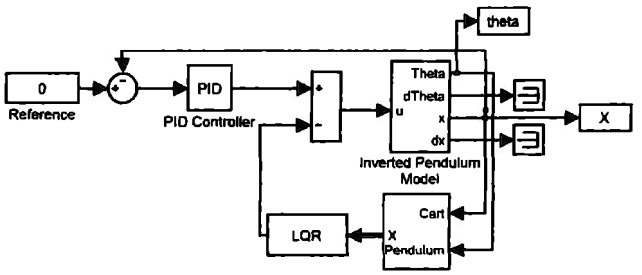


Figura 8. Diagrama de bloques del controlador robusto con LQR

Nuevamente, resulta muy difícil de sintonizar las ganancias del PID y LQR. Fue necesario linealizar el modelo (Apéndice 2) para encontrar las ganancias.

Para calcular los valores de K, [19], asumimos que tendremos retroalimentación de todo el estado X (cuatro variables), y se busca el vector K, que cumple la ley de retroalimentación de estado $u = -K*X$ minimizando según la función de costo $J = \int (X^T QX + u^T Ru)dt$.

Esto se ha hecho mediante la función *lqr*, que devuelve el controlador optimizado, permitiéndonos escoger dos parámetros: R y Q, que priorizan las entradas y la función de costo del estado que vamos a optimizar. Se eligieron:

$$Q = \begin{bmatrix} 5000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; R = [1], \text{ y se obtuvo } \quad K = \begin{bmatrix} 0 \\ -18 \\ -166.5 \\ -15.2 \end{bmatrix}$$

Se simuló con el diagrama de bloques anteriormente mostrado, con condiciones iniciales de 0.1 radianes y 0.1 m obteniendo una muy buena respuesta, como se muestra en la figura 9.

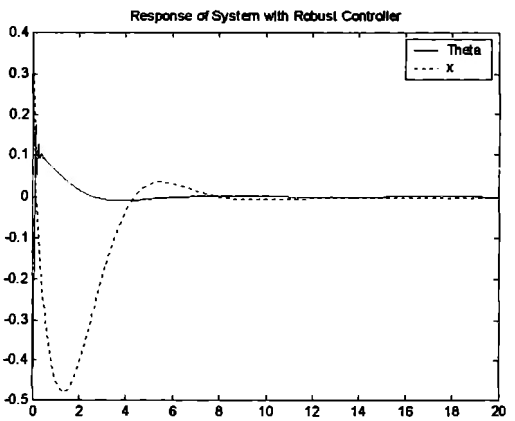


Figura 9. Respuesta del controlador robusto.

Posteriormente se utilizará éste método para entrenar las redes neuronales.

3.3. Controlador difuso.

Se utiliza el sistema de inferencia Mamdani, porque es más fácil de entender gráficamente la lógica del controlador. El conocimiento previo para controlar el sistema consiste en las siguientes reglas [20]:

1. Cuando el péndulo está cayendo de la vertical, y cambias la velocidad angular en la dirección opuesta en que está cayendo, el péndulo será forzado a moverse en la dirección de la velocidad angular.
2. Cuando el carro se mueve a una cierta distancia del centro de la vía y el péndulo está vertical, el péndulo debe tender a caer en dirección del centro de la vía.

Por simplicidad, buscando la aplicación de éste controlador difuso, aquí solo se buscará estabilizar el péndulo, sin importar la posición del carro.

Definimos las siguientes funciones de membresía:

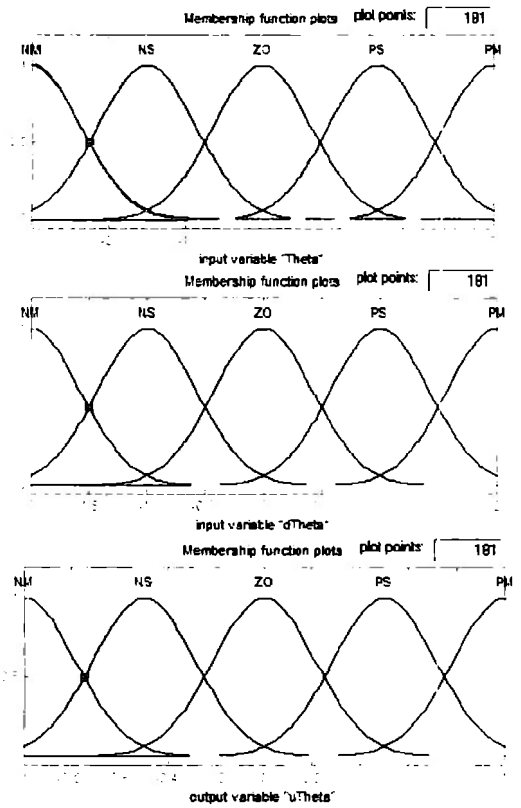


Figura 10. Funciones de membresia del controlador difuso

Y determinamos las siete reglas como sigue [21]:

Regla 1: Si θ= PM y Δθ=ZO, entonces u=PM,
Regla 2: Si θ= PS y Δθ=PS, entonces u=PS,
Regla 3: Si θ= PS y Δθ=NS, entonces u=ZO,
Regla 4: Si θ= NM y Δθ=ZO, entonces u=NM,
Regla 5: Si θ= NS y Δθ=NS, entonces u=NS,
Regla 6: Si θ= NS y Δθ=PS, entonces u=ZO,
Regla 7: Si θ= ZO y Δθ=ZO, entonces u=ZO,

Que puede ser resumido en la siguiente tabla, llamada matriz de asociación difusa, o tabla de conocimiento.

Tabla 2. Matriz de asociación difusa del controlador difuso

θ \ Δθ	NS	ZO	PS
NM		NM	
NS	NS		ZO
ZO		ZO	
PS	ZO		PS
PM		PM	

Se creó el Sistema de Inferencia Difuso (FIS), como se muestra en la figura 11, y en la 12 se observa la superficie de control.

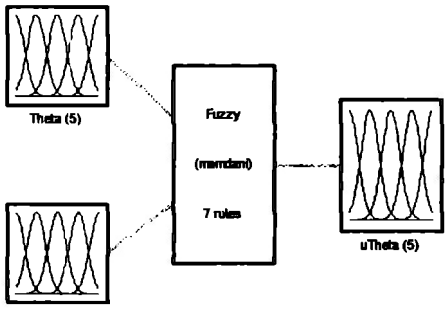


Figura 11. Sistema de Inferencia Difuso

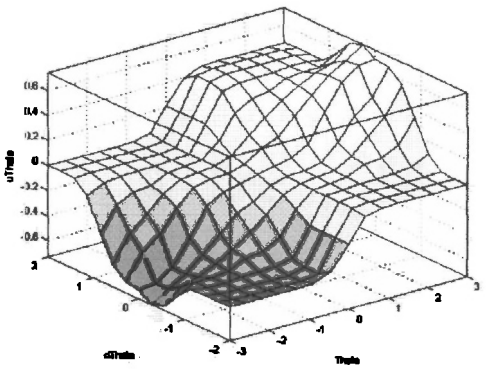


Figura 12. Superficie generada por el controlador difuso

Para emplear el controlador difuso, se construye el diagrama de bloques de la figura 13, con condiciones iniciales de 0.1 rad y 0.1rad/s. La respuesta se observa en la figura 14.

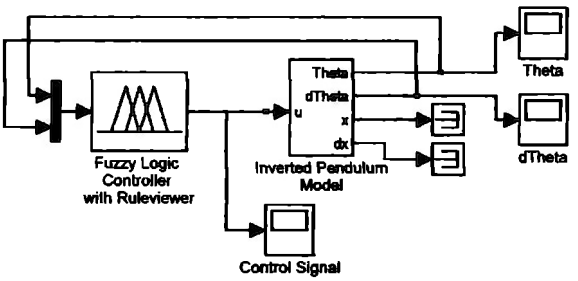


Figura 13. Diagrama de bloques del controlador difuso

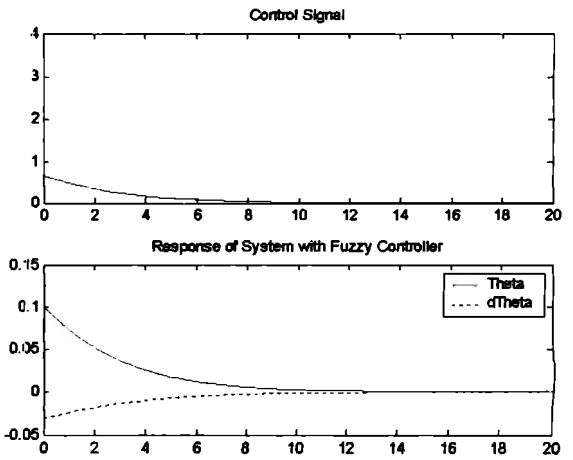


Figura 14. Respuesta del controlador difuso.

Tenemos una buena respuesta porque se estabiliza muy bien el péndulo, sin sobrepaso pero con lentitud. Así se demuestra que el controlador difuso funciona correctamente, pero también requiere cierta sintonización, especialmente al elegir los rangos y formas de las funciones de membresía y el universo de discurso.

3.4. Red neuronal como identificador del sistema.

Para demostrar una de las aplicaciones de las redes neuronales (NN) en los sistemas de control, aquí se modela el sistema utilizando las redes neuronales.
El primer paso, muy importante, consiste en hacer experimentos y adquirir puntos representativos del sistema. Para tener los datos del área de estabilización (posición vertical), se utilizó el controlador robusto de lazo cerrado descrito en la sección 3.2. Así, se excitaba el sistema en la región deseada, y se incluyeron perturbaciones para extender la región de los datos, como se muestra en el diagrama de bloques de la figura 15.

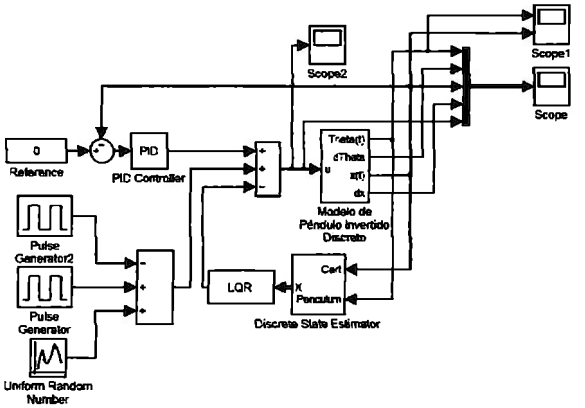


Figura 15. Diagrama de bloques del controlador robusto con perturbaciones

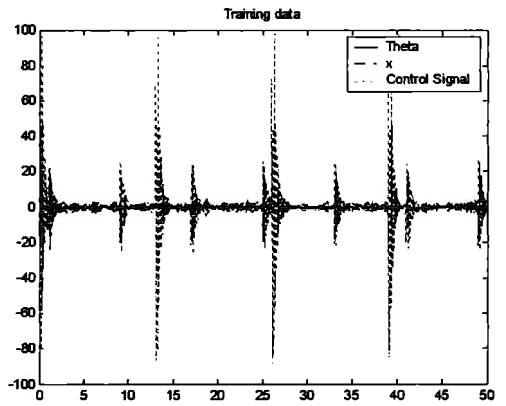


Figura 16. Datos de entrenamiento

Los datos de entrenamiento consiste de las variables de estado y la señal de control, como se muestra en la figura 16.
El siguiente paso consiste en elegir la mejor arquitectura de la red neuronal. La arquitectura depende básicamente de la complejidad del sistema en cuanto al número de entradas y salidas. Aquí se eligió utilizar el perceptrón multicapa (MLP), con 20 nodos escondidos con funciones tangente hiperbólico, y dos nodos lineales como salida (fig. 17).

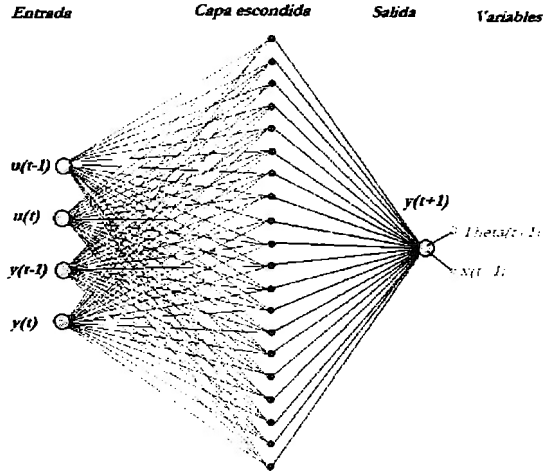


Figura 17. Arquitectura de la red neuronal

Nuestro MLP estimará el ángulo y posición siguientes, conociendo el estado anterior y la señal de control, siguiendo la función [22]:

$$y(t+1) = g[y(t), y(t-1), y(t-2), u(t), \ldots, u(t-1)] \quad (9)$$

Donde

$$y(t) = \begin{bmatrix} \theta(t) \\ x(t) \end{bmatrix}$$

Todo el código, donde se entrena la red, se encuentra en el apéndice 5.

El entrenamiento con Levenberg-Marquardt alcanza rápidamente el objetivo de error de $1x10^{-8}$, como se muestra en la figura 18, y se estiman muy bien los datos de entrenamiento (figura 19).

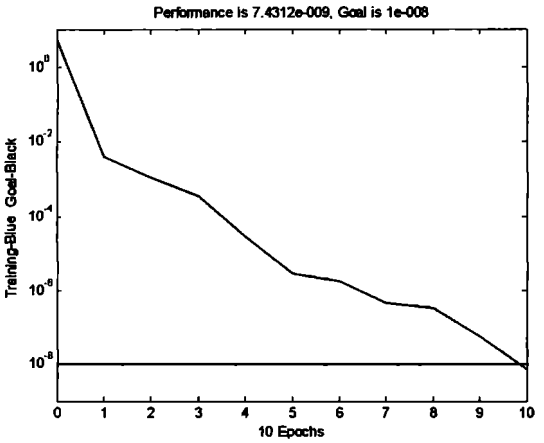


Figura 18. Aprendizaje de la red neuronal.

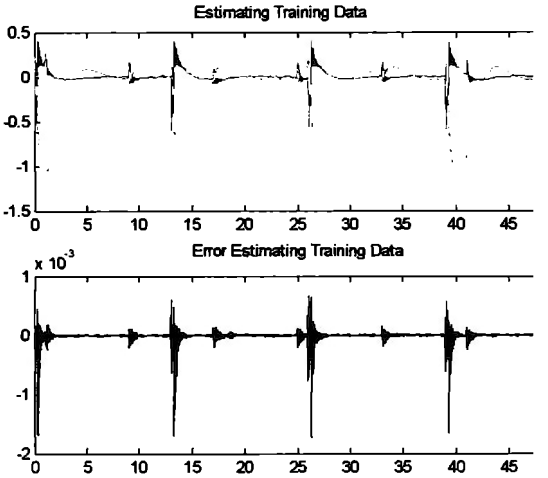


Figura 19. Estimación de datos y su error.

Las redes neuronales generalmente aprenden muy bien los datos de entrenamiento, pero en ocasiones lo hacen demasiado bien y aprenden solo para ésos datos. Para tener la certeza de que la NN realmente aprendió a generalizar el sistema, y no solo aprendió ésos puntos de operación, se generaron los llamados datos de validación, como vemos en la figura 20, y la estimación de la NN logra un error menor a 10^{-4}, como se muestra en la figura 21.

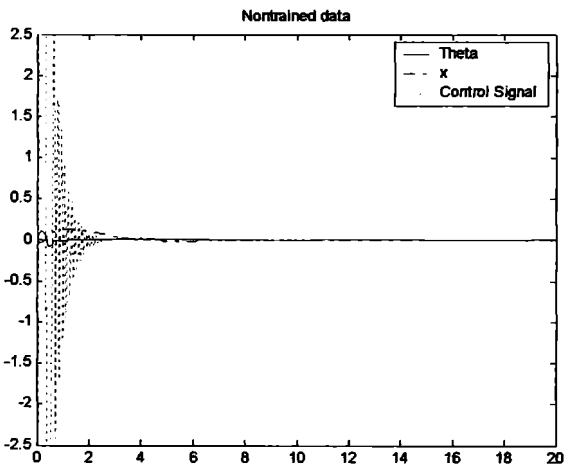


Figura 20. Datos no entrenados

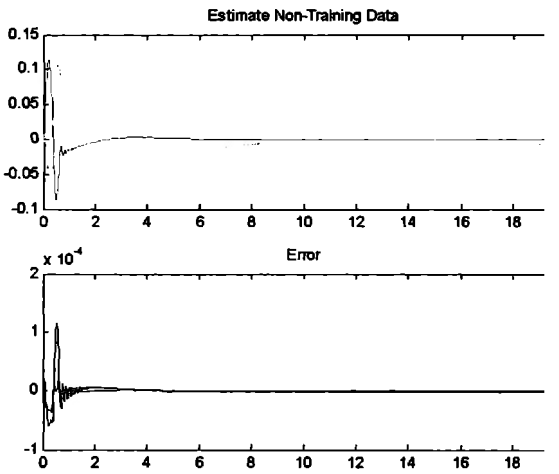


Figura 21. Respuesta a los datos no entrenados y su error.

Ésta es una excelente estimación que muestra que las redes neuronales funcionan excelentemente como identificadores de plantas, demostrando su capacidad de aprendizaje y generalización.

3.5. Controlador neuronal

En ésta sección se intenta demostrar la capacidad de las redes neuronales como controladores, para ello se muestran varias metodologías y sus resultados, presentando al final el mejor resultado. El detalle de las pruebas fallidas se encuentra en el apéndice 3.
El método más sencillo es llamado control directo inverso, "direct inverse control", específicamente del tipo entrenamiento general, "general training". [22]

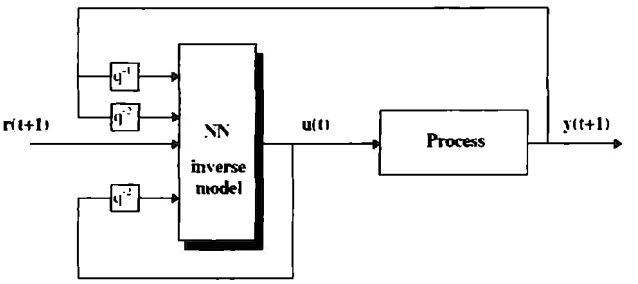


Figura 22. Diagrama esquemático del controlador con redes neuronales

Para realizarlo, se emplea el emulador (NN como identificador), y se busca el modelo inverso (aún sin demostrar si el sistema tiene modelo inverso). Dicho modelo inverso emplea exactamente los mismos datos que para el emulador, pero realiza la siguiente función:

$$\hat{u}(t) = g^{-1}[y(t+1), y(t), y(t-1), y(t-2), u(t-1), u(t-2)]$$

Donde

$$y(t) = \begin{bmatrix} \theta(t) \\ x(t) \end{bmatrix} \tag{10}$$

Al emplear el modelo inverso como controlador, la entrada $y(t+1)$ representa los valores de referencia (deseados).

3.5.1. Utilizando MATLAB.

Nuevamente, se empleó MLP como NN, en este caso con dos capas escondidas de funciones tangente hiperbólico (10 y 25 nodos, respectivamente) y solamente una capa de salida con función de activación lineal. El código completo puede encontrarse en el apéndice 6.
El entrenamiento alcanza el objetivo de $1x10^{-3}$ en 70 iteraciones, como se muestra en la figura 23. El error es muy pequeño comparado con la señal de control que emula (figura 24).

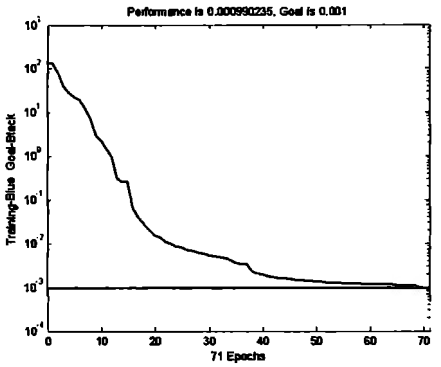


Figura 23. Entrenamiento de la red neuronal.

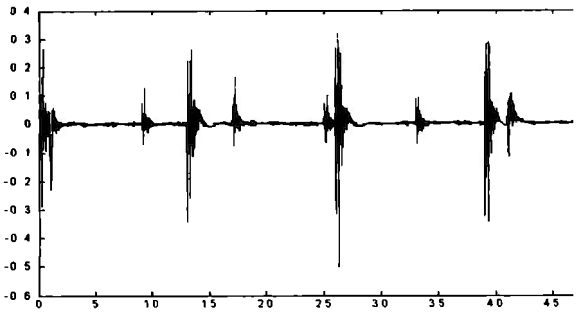


Figura 24. Error al simular la red neuronal

Posteriormente, se exporta la red neuronal a Simulink con la función *gensim*, y para implementar el control directo inverso, se utiliza el diagrama de bloques de la figura 25.

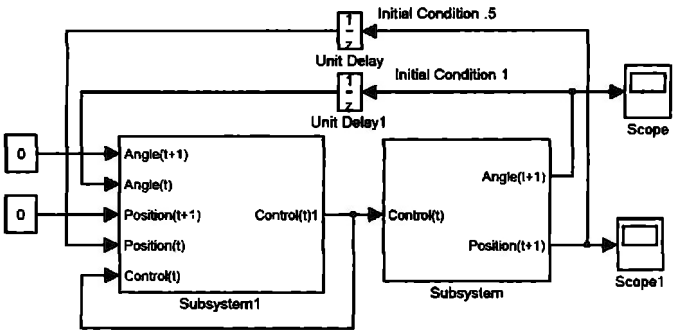


Figura 25. Diagrama esquemático de la red neuronal con estado en el tiempo

Donde los subsistemas están configurados para suministrar las entradas necesarias para la red neuronal (estados y señal de control anteriores, y referencia), como se observa en las figuras 26, 27.

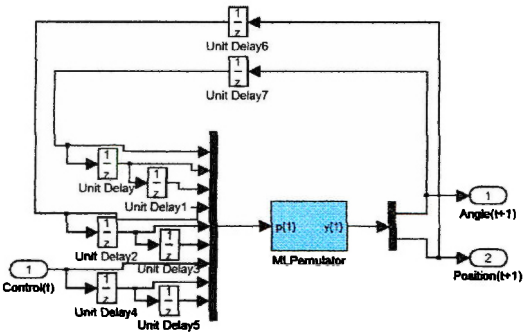


Figura 26. Diagrama de bloques del subsistema emulador.

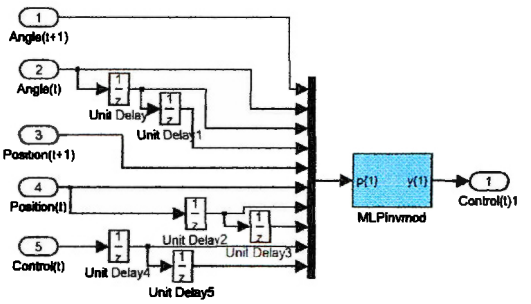


Figura 27. Diagrama de bloques del subsistema1 controlador.

Con todo esto, después de muchas pruebas de controladores, los resultados se muestran en la figura 28.

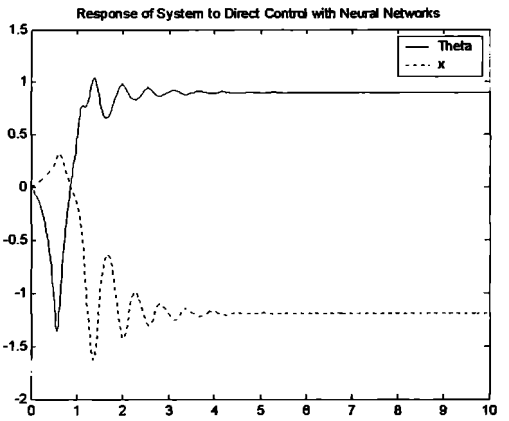


Figura 28. Respuesta del sistema con redes neuronales.

Es obvio que la respuesta del sistema no es satisfactoria. Después de muchos intentos, no se encontró la estructura adecuada del controlador con redes neuronales, aunque dichas configuraciones debieran funcionar para el sistema de péndulo invertido [23, 24, 25].
Como resumen, en MATLAB tres diferentes herramientas fueron utilizadas: la primera fue el "Neural Network Toolbox" en la línea de comando, que permite mayor flexibilidad pero no logró controlar el sistema; la segunda con el "NNSYSID Toolbox" (Neural Network System Identification) y "NNCTRL Toolbox" (Neural Network Control) [27], con menos flexibilidad y peores resultados; y la última con el "Neural Network Toolbox" desde Simulink, aún con menos flexibilidad y los peores resultados. Los desarrollos se ejemplifican en el apéndice 3.

3.5.2. Utilizando EXTEND.

Existen otros programas aparte de MATLAB que presentan muchas ventajas al trabajar con diagramas de bloques, con la posibilidad de simular controladores usando redes neuronales. A continuación se presenta el desarrollo de un controlador realizado en el programa EXTEND [38]. Se tiene un compensador con retroalimentación de estado (con los parámetros presentados anteriormente), que es complementado mediante redes neuronales para generalizar el punto de operación en que, como se muestra en la siguiente figura:

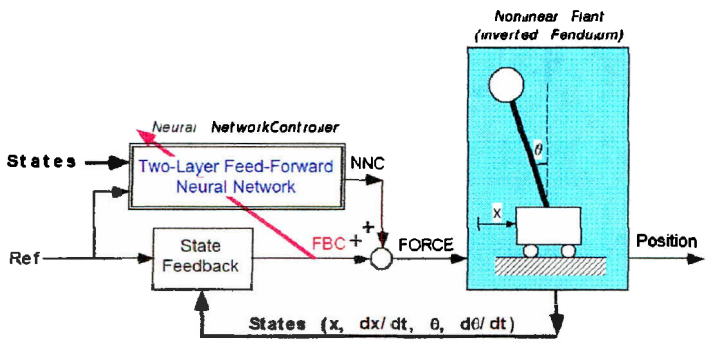


Figura 29. Configuración de control con redes neuronales con el programa EXTEND.

La red neuronal tiene como subsistema lo siguiente:

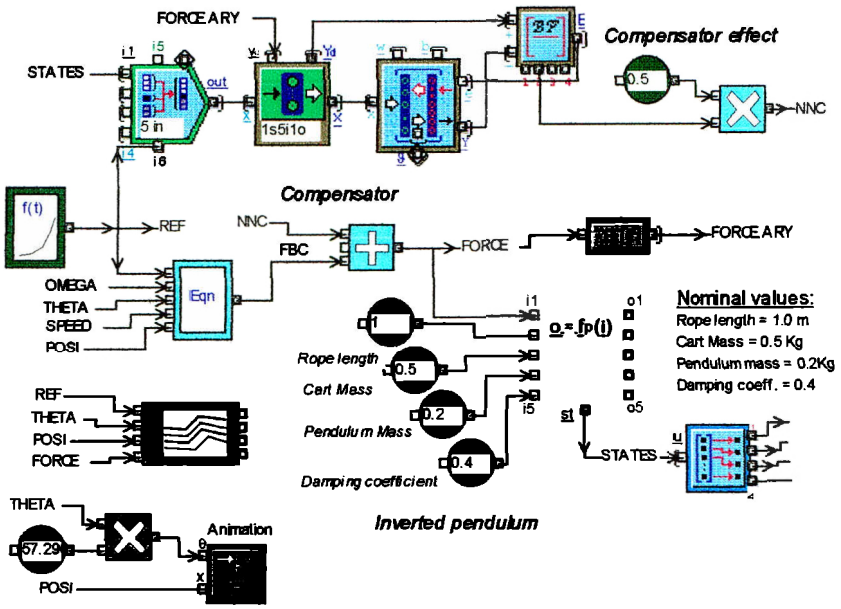


Figura 30. Detalle del sistema controlado por redes neuronales en el programa EXTEND.

Es decir, una red neuronal con dos capas escondidas, de 10 neuronas con función de activación tipo tangente hiperbólica cada una, y una salida lineal. Se entrena empleando retro-propagación simple con una tasa de aprendizaje de 0.025.
En la simulación, se busca que el péndulo esté balanceado, con condiciones iniciales de 0.5rad y 0.5m. La excelente respuesta se muestra en la figura 31:

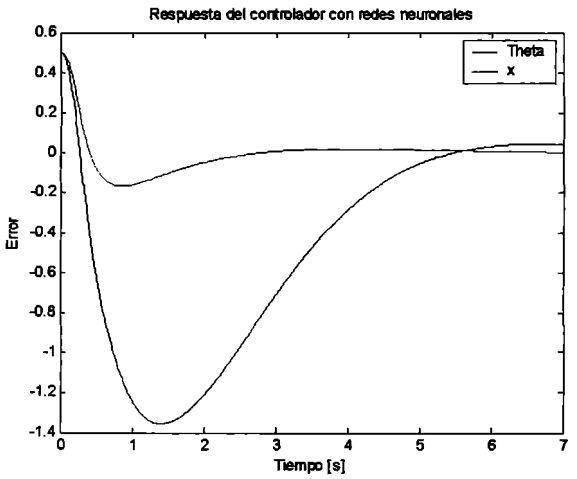


Figura 31. Respuesta del sistema con redes neuronales.

3.6. Controlador ANFIS

Se busca entrenar un controlador ANFIS [26] que cumpla su función correctamente, es decir, que estabilice el sistema en el menor tiempo posible para condiciones iniciales diferentes a cero (tanto positivas como negativas), tan sencillo como sea posible para posteriormente implementar algoritmos genéticos que lo optimicen.
Se probaron diversos tipos de controladores siguiendo con la idea del controlador inverso [27, 28] (que aprende del modelo), al entrenarse con los datos de salida del proceso, pero ejecutarse con las salidas anteriores como entradas y como salida controlan el proceso.
Se probaron porque las características del entrenamiento de la red pueden tener diferencias.
Se crearon varios controladores ANFIS, se entrenaron con diferentes datos, y se corrieron en el simulador.

3.6.1.ANFIS para ángulo y posición.

Se hará un controlador con entrenamiento de ángulo y posición en secuencia de tiempo de 8 entradas, donde el controlador aprende y actúa según las salidas anteriores del sistema y la señal anterior del mismo controlador. Se tendrán siete entradas en nuestro caso, teniendo ángulos, posiciones y señal de control anteriores. Al usar el controlador la señal de $r(t+1)$ consistirá en puros ceros, significando la señal de referencia del sistema.

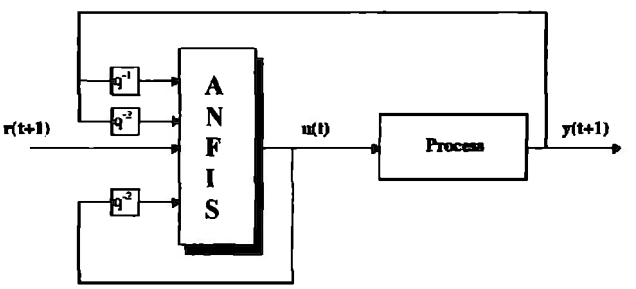


Figura 32. Diagrama esquemático del controlador con redes neuronales.

Se realizaron tres formas diferentes: entrenamiento aleatorio, sub agrupamiento y mallado.

3.6.1.1. Entrenamiento aleatorio

Los datos de entrenamiento fueron creados excitando el sistema del péndulo invertido con valores aleatorios, sin tener controladores PID sintonizados, buscando únicamente obtener la respuesta del sistema para diferentes tipos de entradas. Se utilizó un modelo del péndulo invertido en Simulink, con etapa gráfica, creado por el Dr. Ernesto Olguín [37]. Como se muestra en la figura 33.

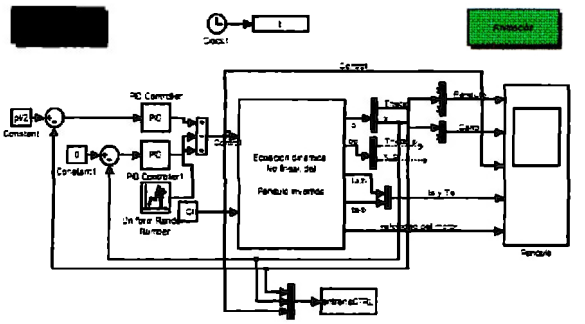


Figura 33. Simulador del péndulo invertido para obtener datos aleatorios

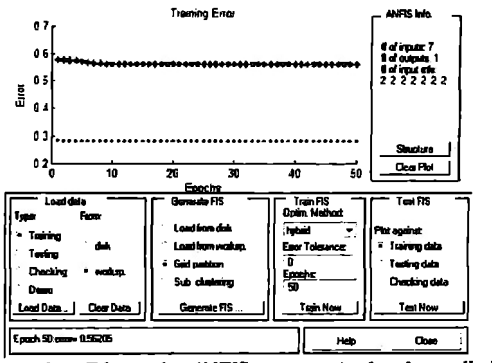


Figura 34. Editor de ANFIS para método de mallado.

El código necesario para generar datos con secuencia en el tiempo (estado actual, estado anterior, y señal de referencia) es el siguiente:

```
C=length(entrenaCTRL);
entrenaU=[entrenaCTRL(3:C,1:2), entrenaCTRL(2:(C-1),1:2), entrenaCTRL(1:(C-2),1:3),
entrenaCTRL(2:(C-1),3)];
TrainU=[entrenaU(2:(C-2)/2,:)];
TestU=[entrenaU((C-2)/2:3*(C-2)/4,:)];
CheckU=[entrenaU(3*(C-2)/4:(C-2),:)];
anfisedit
plot(entrenaU)
```

Se entrenó un controlador ANFIS de tipo mallado ("grid partition"), con 2 funciones de membresía tipo gaussianas por entrada, teniendo 7 entradas, con una salida constante, y 50 entrenamientos. El entrenamiento se generó según observamos en la figura 34.
Donde el error de 0.56205 es aceptable porque la señal de control alcanza valores de ±200, es decir, el error es de menos del 0.28%. Dicho error se compara con el valor de referencia, como vemos en la siguiente figura 35.

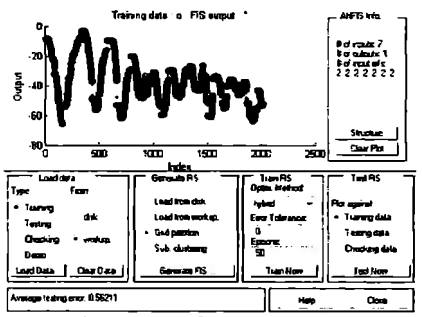


Figura 35. Editor de ANFIS mostrando datos entrenados

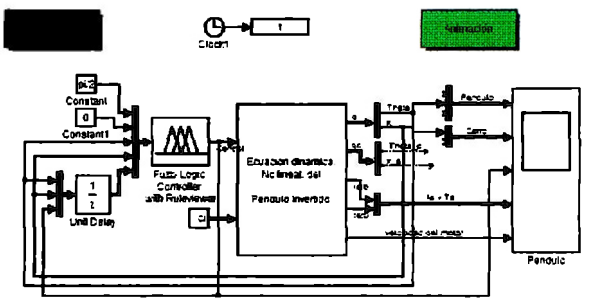


Figura 36. Diagrama de bloques del simulador, controlado por ANFIS.

Se observan brincos en los datos de entrenamiento, por lo que no cubrimos todas las posibilidades. Aún así, la red aprende suficientemente bien el sistema.
Para simular el controlador ANFIS se configuró un diagrama de bloques que alimenta las señales con secuencia en el tiempo, figura 36.

Y los resultados fueron poco satisfactorios. La figura 37 muestra que el sistema se mantiene en la zona deseada ($\pi/2$ para éste simulador) por décimas de segundo, pero continúa siendo inestable.

Se puede inferir que el controlador ANFIS si tendrá la capacidad de controlar el sistema, pero es necesario entrenar más en la zona de estabilización, y puesto que las señales aleatorias no cubren especialmente ésta área, es necesario realizar un controlador que lo mantenga en la zona deseada.

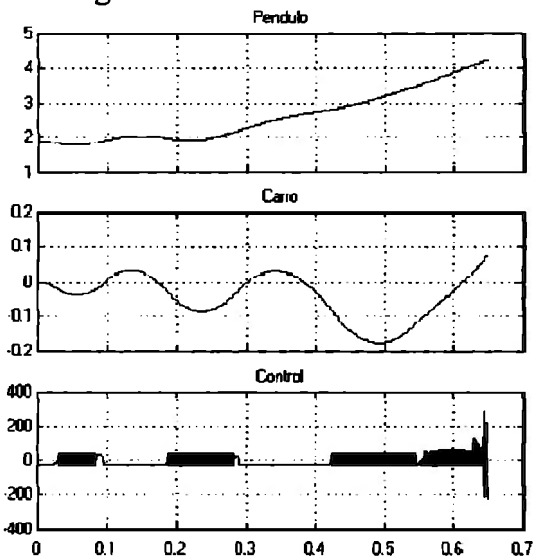


Figura 37. Resultados de la simulación ANFIS con mallado

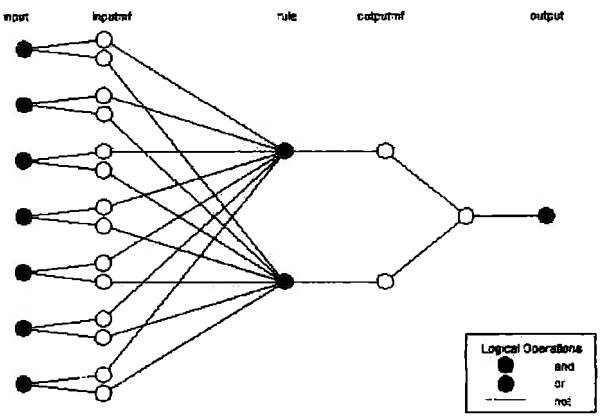


Figura 38. Arquitectura del ANFIS

3.6.1.2. Sub agrupamiento ("sub clustering")

Ésta forma de crear la red difusa consiste en dividir los datos de entrenamiento en diferentes zonas, y crear reglas de pertenencia entre ellas. Con los parámetros estándar:

- Rango de influencia: 0.5
- Factor de compresión: 1.5
- Tasa de aceptación: 0.5
- Tasa de rechazo: 1.5

Se creó una red como muestra la figura 35, que resulta mucho más sencilla que el caso anterior.

Se obtuvieron los datos con el sistema de control robusto, los valores iniciales son 0.5 radianes y 0.3 metros, para ángulo y posición respectivamente. Adicionalmente, se introducen tres perturbaciones tipo impulso que hacen cambio de referencia en la posición del carro.

La señal de control se satura en ± 20, obteniendo la figura 39. Posteriormente se adecua la señal, siguiendo la ecuación:

$$\hat{u}(t) = g^{-1}[y(t+1), y(t), \ldots, y(t-n+1), u(t-1), \ldots, u(t-m)] \qquad (11)$$

Se utilizaron los siguientes comandos:

```
entrenaCTRL=Ucontrollim;
C=length(entrenaCTRL);
entrenaU=[entrenaCTRL(3:C,1:2),
entrenaCTRL(2:(C-1),1:2),
entrenaCTRL(1:(C-2),1:3),
entrenaCTRL(2:(C-1),3)];
TrainU=[entrenaU(2:(C-2)/2,:)];
TestU=[entrenaU((C-2)/2:3*(C-2)/4,:)];
CheckU=[entrenaU(3*(C-2)/4:(C-2),:)];
```

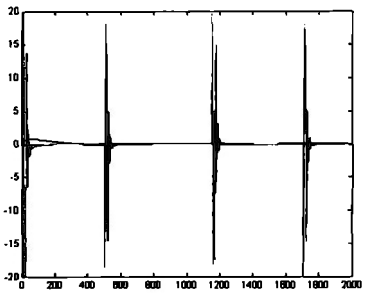


Figura 39. Señal del control ANFIS

El entrenamiento se realiza con la mitad de los datos, y se prueba y checa con la última mitad de ellos. Se entrena usando ANFISEDIT, usando sub agrupamiento, con los siguientes parámetros:

- Rango de influencia: 0.5
- Factor de compresión: 1.5
- Tasa de aceptación: 0.5
- Tasa de rechazo: 1.2

Se genera una red con 7 entradas, dos funciones de membresía por entrada, y una salida lineal, de la forma:

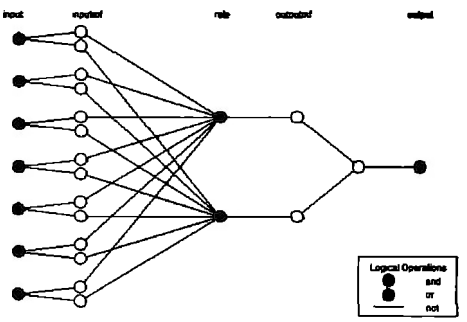


Figura 40. Arquitectura de la red ANFIS.

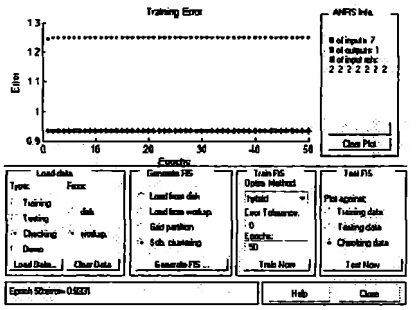


Figura 41. editor ANFIS entrenando sistema con sub agrupamiento

Se entrenó durante 50 iteraciones, con método de optimización híbrido.

Ya con la ANFIS entrenada, se simula en el diagrama de bloques siguiente:

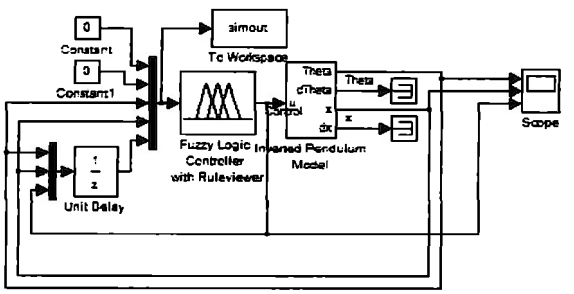


Figura 42. Diagrama de bloques del sistema con controlador ANFIS con estado a través del tiempo.

El sistema controla para condiciones iniciales de 0.1rad y 0.1m y también logra hacerlo hasta 0.3 y 0.1 respectivamente.

Pero nunca se estabiliza totalmente, pues oscila, pero al menos mantiene el sistema en la región deseada (vertical), según la figura 43.
También se probó moviendo la salida a constante, pero apenas son dos reglas, y se vuelve infuncional.

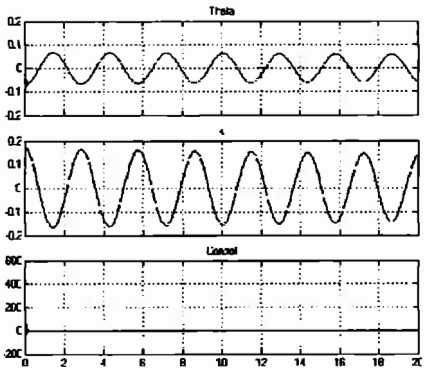


Figura 43. Resultados de la simulación con sub agrupamiento

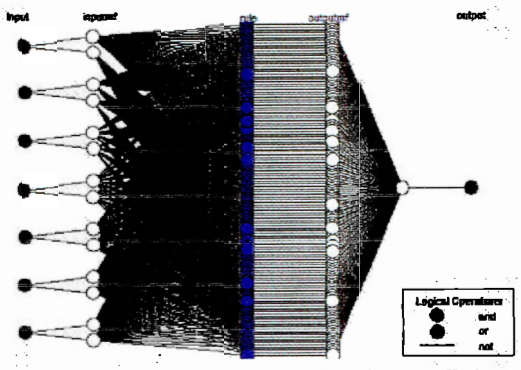


Figura 44. Arquitectura del ANFIS mediante mallado

3.6.1.3. Mallado ("grid partition")

Se obtuvieron datos de entrenamiento mediante el controlador robusto obtenido anteriormente. Se cuidó que el controlador obtuviera no sólo los datos en que se estabiliza, sino que se desestabilizara momentáneamente para obtener un mayor rango de respuesta.
Se creó entonces un sistema con dos funciones de membresía tipo gaussianas para las 7 entradas, y una salida constante. La red crea muchas reglas entre sí mismas, como se muestra en la siguiente figura 44:
Se entrenó con un error de 0.7335, como se observa en la figura 45. Pero la respuesta en simulación es inestable, figura 46.

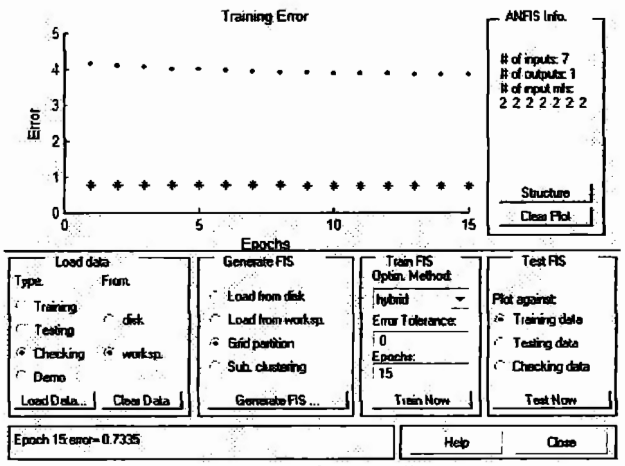


Figura 45. Editor ANFIS para mallado

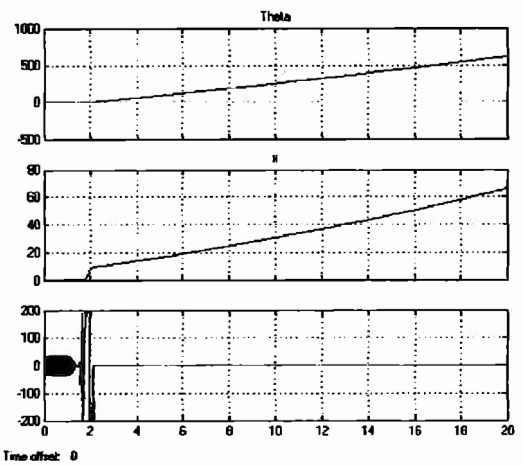


Figura 46. Resultados de ANFIS para mallado

La salida son 128 parámetros (128 reglas, tipo constante).

Buscando la mejora en este tipo de controladores, se probó nuevamente pero para un sistema más complejo. Los parámetros son los del caso anterior, pero con salida lineal. El entrenamiento tiene un error de 0.59917; pero el resultado es igualmente no satisfactorio. Las figuras 47 y 48 lo muestran.

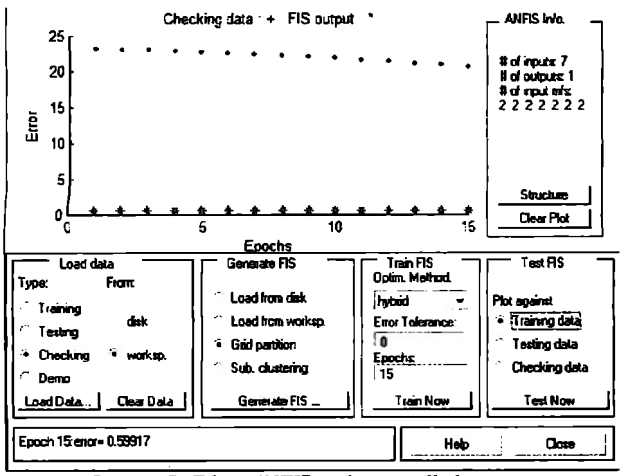


Figura 47. Editor ANFIS mediante mallado

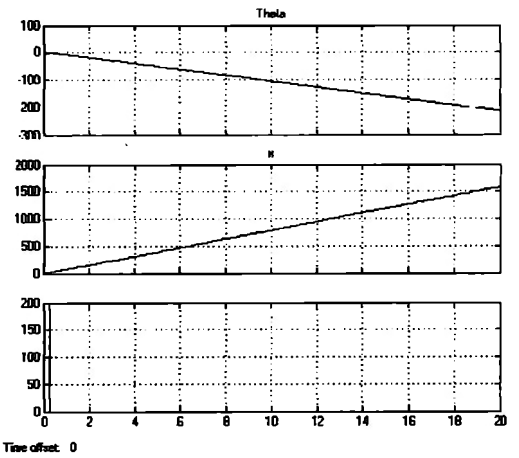


Figura 48. Resultados de la simulación con ANFIS mediante mallado

3.6.2.*ANFIS para todo el estado.*

Se hará un controlador ANFIS con entrenamiento de estado en secuencia de tiempo de 8 entradas, en que se entrena con la señal deseada y la salida anterior del sistema; al usar el controlador la señal deseada será la matriz de ceros. Se simplifica el aprendizaje, pero se tienen menos condiciones que muestren el estado del sistema.

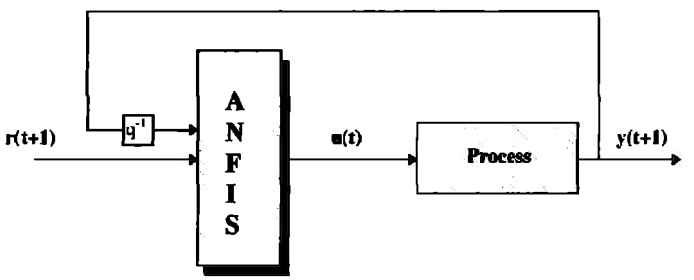


Figura 49. Diagrama esquemático del controlador mediante ANFIS.

Con el mismo controlador robusto se obtuvieron los datos de entrenamiento, y se reordenaron de tal forma que se contara con el estado actual (theta, dtheta, x, dx) y futuro, a fin de entrenar así el controlador. Se realizó con el siguiente código:

%8 ENTRADAS, ESTADO +1 (THETA, DTHETA, X, DX) Y ESTADO ACTUAL
entrenaXref=UcontrollimX;

```
D=length(entrenaXref);
entrenaUXref=[entrenaX(3:D,1),entrenaX(3:D,2),entrenaX(3:D,3),entrenaX(3:D,4),entrenaX(2:(D-
1),1),entrenaX(2:(D-1),2),entrenaX(2:(D-1),3),entrenaX(1:(D-2),4),entrenaX(2:(D-1),5)];
TrainUXref=[entrenaUXref(2:(D-2)/2,:)];
TestUXref=[entrenaUXref((D-2)/2:3*(D-2)/4,:)];
CheckUXref=[entrenaUXref(3*(D-2)/4:(D-2),:)];
```

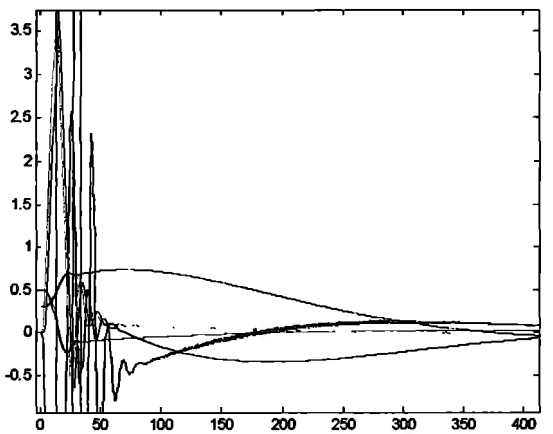


Figura 50. Datos de entrenamiento para ANFIS

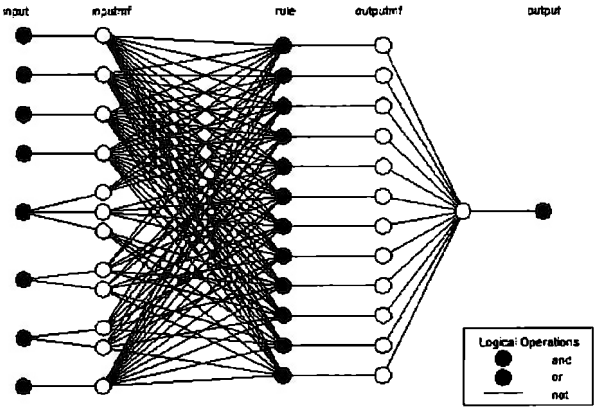


Figura 51. Arquitectura del ANFIS

Los datos se muestran en la figura 50, y posteriormente se construye una ANFIS con mallado (fig. 51)

Es decir, se buscó simplificar (tontamente) la red creada con mallado por medio de reducir las funciones de membresía de cada entrada, haciendo que se tenga una sola función de membresía para el estado futuro (que posteriormente se cambiará por puros ceros al usar como controlador) y 3 funciones para theta, 2 para dtheta y x, y una para dx; el entrenamiento, apenas con 3 iteraciones, se muestra en la figura 52.

Y como era de esperarse (pero valió la pena probarlo), no funcionó (fig. 53).

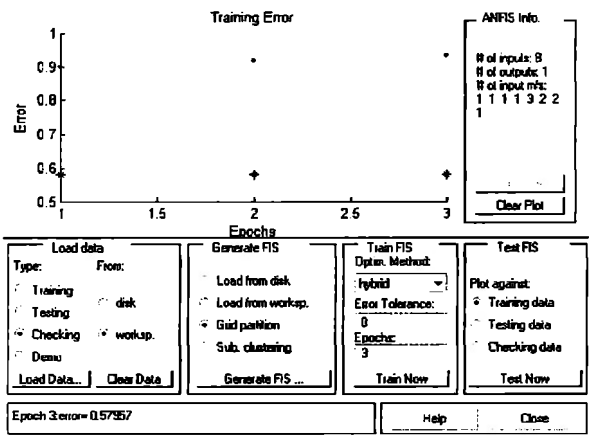


Figura 52. Editor ANFIS para mallado

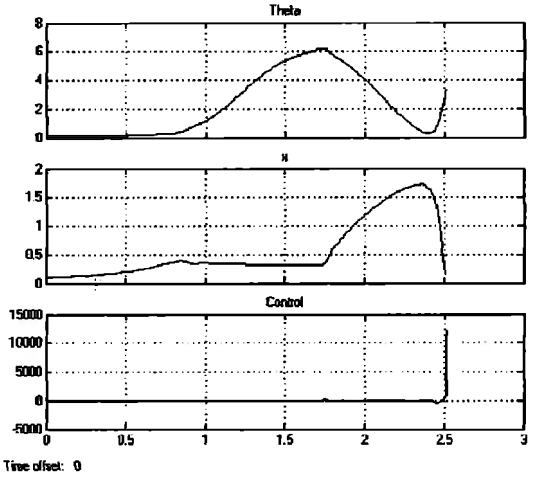


Figura 53. Respuesta no satisfactoria del controlador ANFIS

3.6.3.ANFIS sólo con el estado.

Se hará un controlador ANFIS con entrenamiento de estado de 4 entradas, que es el más sencillo de la familia, y en este caso se entrena y ejecuta únicamente con el estado del sistema. Se puede utilizar porque la referencia no varía con el tiempo. Es el método más sencillo, por lo que su implementación ayudaría a la siguiente etapa con algoritmos genéticos.

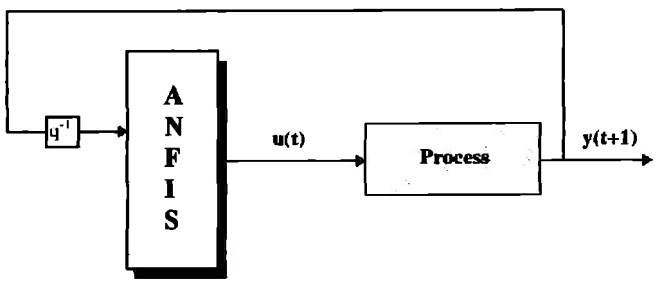


Figura 54. Diagrama esquemático del controlador ANFIS

Se realizaron tres formas diferentes, siendo la última la más eficaz. Fueron: Control diferencial en función de k, continuo con pocos datos de entrenamiento y continuo con múltiples datos de entrenamiento.
Se analizará tan sólo el mejor ejemplo de cada uno, y todas las pruebas que acercaron a dicho resultado se muestran en el apéndice 4.

3.6.3.1. Control diferencial en función de k

Debido a que el control funciona con los datos de estado en el tiempo K, se pensó discretizar los datos de entrenamiento, de forma que la derivada sería la diferencia *k* y *k+1*. Se realizó con el siguiente código:

```
entrenaX=Ucontrollim;
D=length(entrenaX);
entrenaUX=[entrenaX(2:(D-1),1),-
entrenaX(2:(D-1),1)+entrenaX(1:(D-
2),1),entrenaX(2:(D-1),2),-entrenaX(2:(D-
1),2)+entrenaX(1:(D-2),2),entrenaX(2:(D-1),3)];
TrainUX=[entrenaUX(2:(D-2)/2,:)];
TestUX=[entrenaUX((D-2)/2:3*(D-2)/4,:)];
CheckUX=[entrenaUX(3*(D-2)/4:(D-2),:)];
```

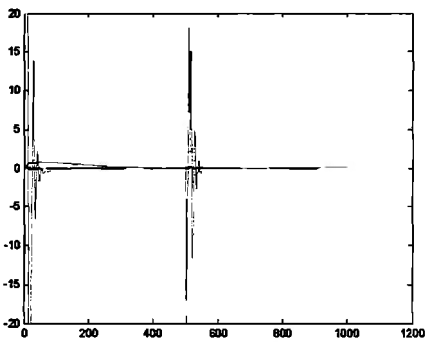


Figura 55. Datos de entrenamiento.

Se generaron los datos de entrenamiento, se entrenó de diferentes formas, y se simuló en el diagrama de bloques de la siguiente figura 56, que se simplifica porque la derivada debe ser igual al estado anterior y el actual, pero el actual es el deseado, que es cero, y por ello la derivada es el valor negativo de la salida anterior:

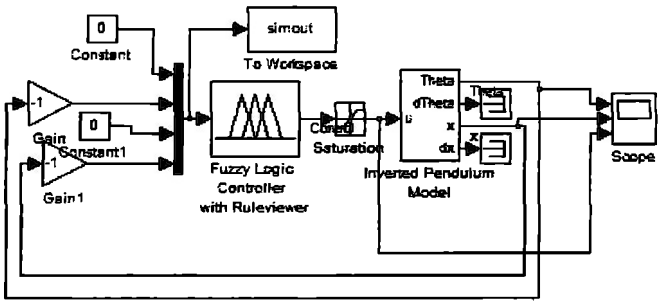


Figura 56. Diagrama de bloques del controlador ANFIS

A continuación analizaremos varias pruebas al configurar y entrenar el ANFIS.

- Sub agrupamiento

Con los siguientes parámetros:

- Rango de influencia: 0.3
- Factor de compresión: 1.5
- Tasa de aceptación: 0.5
- Tasa de rechazo: 1.5

El error de entrenamiento es de 2.0805, y el resultado se vuelve inestable después de unos segundos.

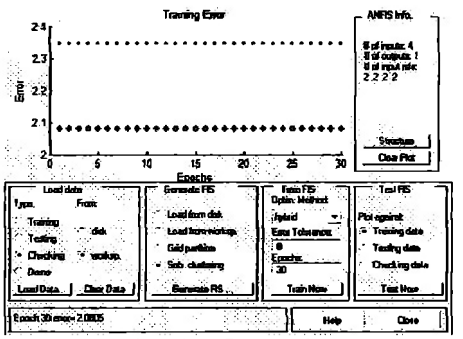


Figura 57. Editor ANFIS para sub agrupamiento

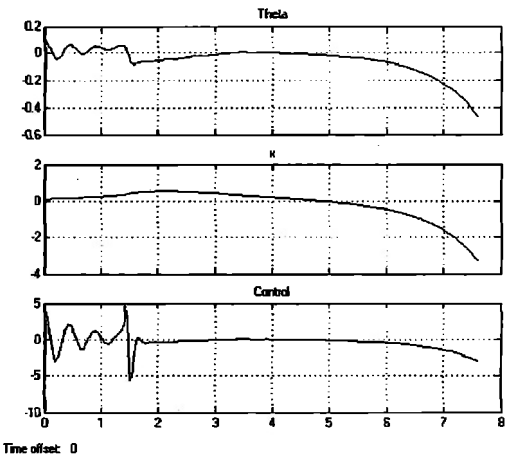


Figura 58. Resultados de la simulación del controlador ANFIS.

3.6.3.2. Continuo con pocos datos de entrenamiento.

Se aprovechó que por el modelo del sistema (continuo) tenemos acceso a las derivadas, y de forma directa se obtuvo el estado actual. Nuevamente se obtuvieron los datos del controlador robusto, siendo los que se muestran a continuación, figura 59.

Se generaron los datos de entrenamiento:

```
% 4 ENTRADAS, ESTADO THETA,
DTHETA, X, DX
entrenaX=UcontrollimX;
D=length(entrenaX);
entrenaUX=[entrenaX(2:(D-
1),1),entrenaX(2:(D-
1),2),entrenaX(2:(D-
1),3),entrenaX(1:(D-
2),4),entrenaX(2:(D-1),5)];
TrainUX=[entrenaUX(2:(D-2)/2,:)];
TestUX=[entrenaUX((D-2)/2:3*(D-
2)/4,:)];
CheckUX=[entrenaUX(3*(D-2)/4:(D-
2),:)];
```

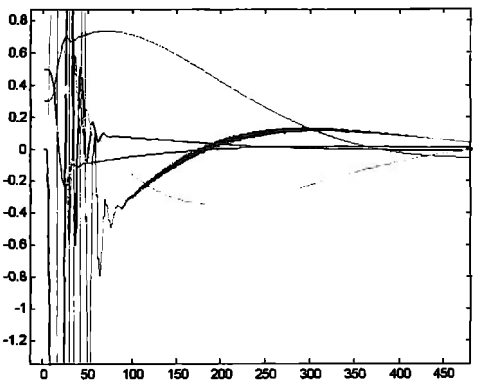


Figura 59. Datos de entrenamiento

Se entrenó el controlador ANFIS con mallado, 2 funciones de membresía x 4 entradas, 1 salida lineal. Las reglas se muestran a en la figura 61.

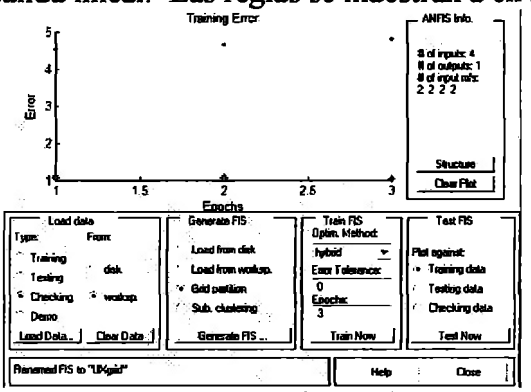


Figura 60. Editor ANFIS para mallado

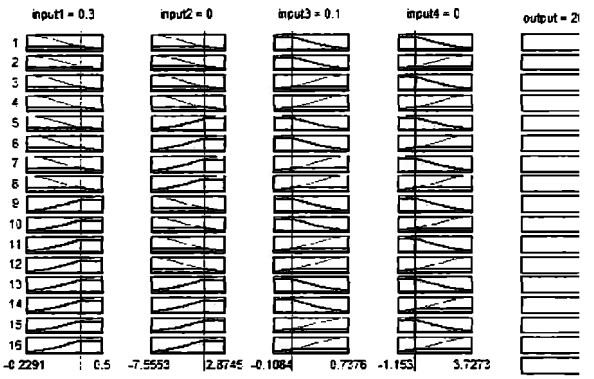


Figura 61. Evaluación de reglas difusas con ANFIS.

Se simuló de forma directa, donde la retroalimentación es el estado actual; nótese que el controlador no depende del error del sistema, sino de sus condiciones actuales, y según ello decide la nueva señal que haga que la planta se estabilice.

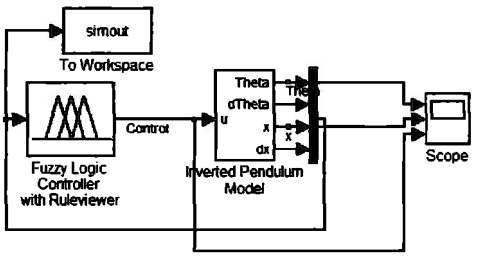


Figura 62. Diagrama de bloques del simulador de ANFIS.

Se realizó una prueba para condiciones iniciales de 0.1 radianes y 0.1m y en 6 segundos se estabiliza totalmente (fig. 63), pero para otras condiciones iniciales, con 0.3 radianes y 0.1m., el sistema se vuelve inestable (NaN en integrador), como se observa en la figura 64.

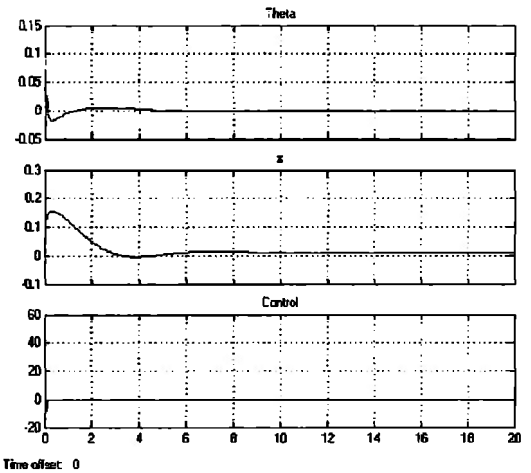


Figura 63. Resultados de la simulación con ANFIS

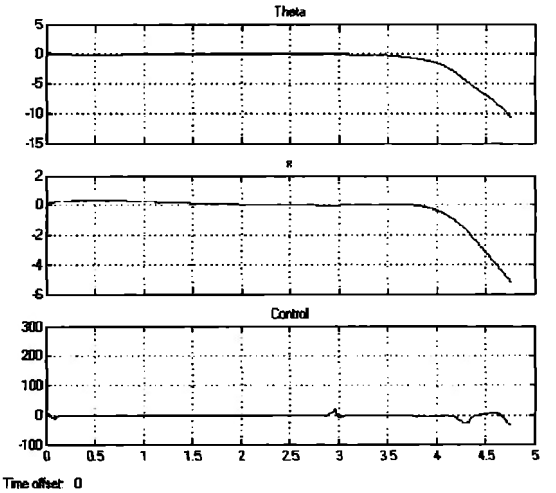


Figura 64. Resultados de la simulación con ANFIS otras condiciones iniciales

3.6.3.3. Continuo con múltiples datos de entrenamiento.

Por último, se probó también la retroalimentación directa del estado (dinámica del sistema), pero ahora con más datos de entrenamiento. Se utilizaron los mismos datos del controlador robusto, dividiéndolo en dos; la primer mitad para entrenar, y la segunda mitad dividida nuevamente en dos, una parte para probar y la última para checar.

En el apéndice 4 se presentan las otras condiciones realizadas (que son muchas), con su resultado, sin hacer mayor énfasis en cada uno.

Con la misma lógica que los controladores anteriores, pero ahora los datos se crearon tanto para condiciones iniciales positivas como negativas; se realizó el entrenamiento primero con la respuesta a condiciones iniciales positivas, y posteriormente algunos datos de la respuesta a condiciones iniciales negativas. Asimismo se realizó para la prueba y chequeo del controlador. Los datos se muestran en la figura 65.

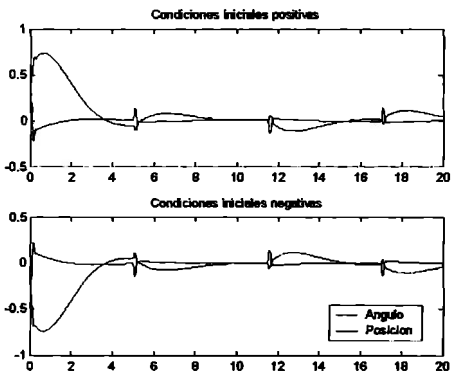


Figura 65. Respuesta a nuevas condiciones iniciales

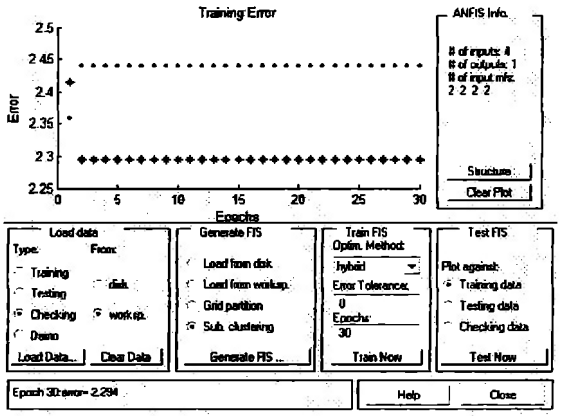


Figura 66. Editor ANFIS para sub agrupamiento

Con sub agrupamiento, 30 iteraciones y parámetros:

- Rango de influencia: 0.2
- Tasa de aceptación: 0.2
- Tasa de rechazo: 0.1

Se obtuvo una red con la estructura:

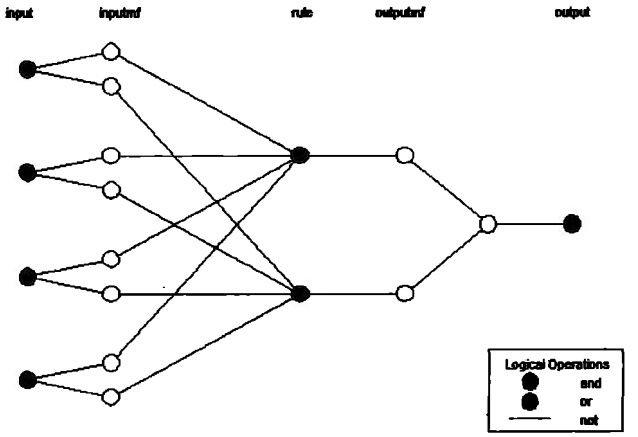


Figura 67. Arquitectura de ANFIS con sub agrupamiento

Se simuló para condiciones iniciales de 0.3 radianes y 0.3m. de condiciones iniciales, obteniendo los resultados de la figura 68. Luego con -0.3 rad y -0.3m (figura 69).

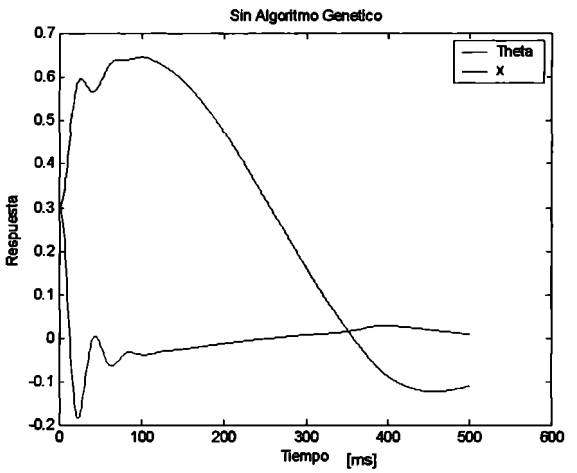


Figura 68. Respuesta del sistema a condiciones iniciales positivas

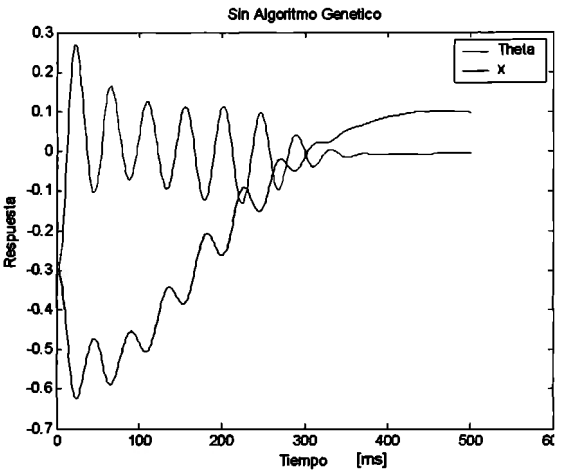


Figura 69. Respuesta del sistema a condiciones iniciales negativas

Analizando a detalle, el error de estado estable en *theta* es 0, y en x es $5x10^{-3}$.
Éste último controlador es el que mejor funcionó según los requerimientos establecidos, y se preparó para optimizar con algoritmos genéticos.
Para cada tipo de control se probaron diferentes parámetros en la forma de crear la red, y diferentes datos de entrenamiento. Esto hace que la búsqueda de un controlador eficiente puede ser muy compleja, aparte de que no se realizó con ninguna mitología establecida, sino por observación de los resultados al variar dichos parámetros.
Las simulaciones menos importantes, con sus procedimientos, resultados y análisis se muestran en el apéndice 4, por lo que se sugiere remitirse a él.

3.7. Controlador ANFIS optimizado

Los algoritmos genéticos sirven para optimizar un sistema sin emplear derivación, mediante un criterio definido, y con parámetros a definir como son la tasa de mutación, de cruzamiento, así como el método de codificación, de selección y de evaluación de bienestar de cada miembro de la población [29]. Para una mejor descripción teórica de los algoritmos genéticos, se sugiere revisar el apéndice 1.

Se parte del controlador ANFIS obtenido anteriormente. Se modifican únicamente los parámetros de las dos funciones lineales de salida, que son 5 para cada una.

El controlador es evaluado en el siguiente diagrama de bloques:

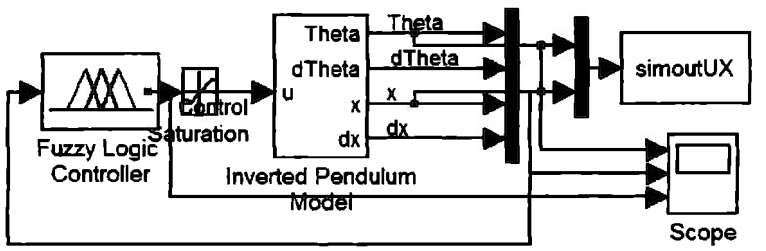


Figura 70. Sistema de bloques a optimizar con algoritmo genético, utilizando un bloque de saturación

Donde el modelo del péndulo invertido tiene sus condiciones iniciales.

Puesto que los algoritmos genéticos tienen miembros de población que son mutaciones, es decir, valores aleatorios, es posible que el sistema se haga inestable y la simulación se vaya a infinito, alentando la optimización, por ello se pusieron bloques de saturación para evitar que tanto el controlador como el estado del sistema salieran de una zona permisible. No se afecta el resultado, puesto que sólo limita cuando el controlador tiene un error muy grande.

El código completo del algoritmo genético [30] se encuentra en el apéndice 5, donde están todos los programas.

Los pasos más importantes del algoritmo genético son:

1. Codificar parámetros del sistema a optimizar.
2. Crear población, con elementos diferentes.
3. Probar cada elemento y asignar cuál es mejor (criterio de evaluación).
4. Mantener los mejores, modificar los peores (mutación y cruzamiento)... Evolucionar.
5. Volver a 3 y salir hasta cumplir criterio de paro.

El algoritmo genético se trabajó en todas las simulaciones con los siguientes parámetros:

- VTR – Valor a alcanzar ("Value To Reach"), es una condición de paro que el criterio de error establece; en nuestro caso, fue $1x10^{-3}$, sabiendo que nunca llegaría a dicho valor.
- Iteraciones máximas – Se modificaron las iteraciones máximas (generaciones), siendo éste el factor de paro del algoritmo.
- D – Número de parámetros a modificar para el sistema, en nuestro caso fueron 10 parámetros.
- XVmin – vector de límite inferior para la población inicial; fue -10% el valor de cada parámetro de salida del ANFIS inicial.

- XVmin – vector de límite superior para la población inicial; fue +10% el valor de cada parámetro de salida del ANFIS inicial.
- NP – Número de miembros de la población. Usamos 15*D, es decir, 150 miembros por cada población.
- F – Valor del escalón entre cada iteración, se sugiere un valor entre 0 y 2, por lo que tomamos 1.
- CR – probabilidad de cruzamiento, debe estar entre 0 y 1, por lo que tomamos 0.4
- Estrategia – El algoritmo permite muchas estrategias, pero la que más nos sirvió fue una codificación con valores exponencial (no binaria), con la lógica rand-to-best, que de los valores aleatorios refuerza los que mejor respuesta presentaron.
- Refresh – es un parámetro no importante para la optimización, pero que nos permitió observar el proceso; significa cada cuántas generaciones se presenta el resultado. Gracias a ello presentamos gráficas de optimización.

Se crearon varios programas de algoritmos genéticos, con diferentes sistemas y con diferentes iteraciones máximas, a saber:

Optimización 1. Sistema con condiciones iniciales -0.3rad y -0.3 m, a 100 iteraciones.

Optimización 2. Sistema con condiciones iniciales 0.3rad y 0.3 m, a 2000 iteraciones.

Optimización 3. Sistema con condiciones iniciales tanto positivas como negativas (±0.3 rad y ±0.3 m). Con 200 iteraciones.

Optimización 4. Sistema con condiciones iniciales tanto positivas como negativas (±0.5 rad y ±0.5 m). Con 2000 iteraciones.

Optimización 5. Sistema con condiciones iniciales tanto positivas como negativas (±0.5 rad y ±0.5 m). Con 1000 iteraciones. Pero el criterio de paro es ponderado.

Los primeros cuatro se presentan en el Apéndice 5.

El método de optimización que mejor funcionó fue el quinto, que optimizó al sistema con condiciones iniciales tanto positivas como negativas (±0.5 rad y ±0.5 m), con 1000 iteraciones. Especialmente se empleó un criterio de paro ponderado.

Buscando salir del transitorio lo antes posible, y eliminar cualquier error en estado estable, se cambió el criterio de evaluación por uno que pondera el error respecto al tiempo, es decir, linealmente, cada que el tiempo se incrementa un milisegundo, la ponderación aumenta en 0.1, iniciando con una ordenada al origen de 1 (para no descartar las condiciones iniciales).

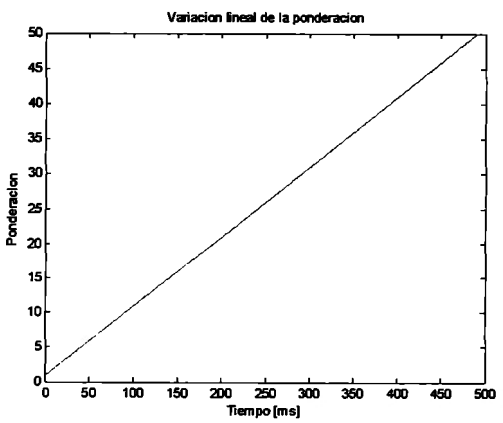


Figura 71. Gráfica que muestra la variación lineal de la ponderación.

Ello se logra con el comando:

ponderacion=1:.1:length(simout)/10;

donde *simout* son los valores de salida de la simulación.

Para el controlador ANFIS sin optimización con GA's, el error de *theta* será (fig 72).

*errorcuadradoponderado=power(simout(:,1).*ponderac',2);*

plot(ponderac,errorcuadradoponderado)

En resumen, se cumple el criterio de minimización:

$$criterio = 20*\sum pond*(\theta_{pos}+\theta_{neg})^2 + 10*\sum pond*(x_{pos}+x_{neg})^2 \qquad (14)$$

La optimización, con condiciones iniciales (± 0.5 rad y ± 0.5 m), y 1000 iteraciones (generaciones), el aprendizaje se muestra en la figura 72:

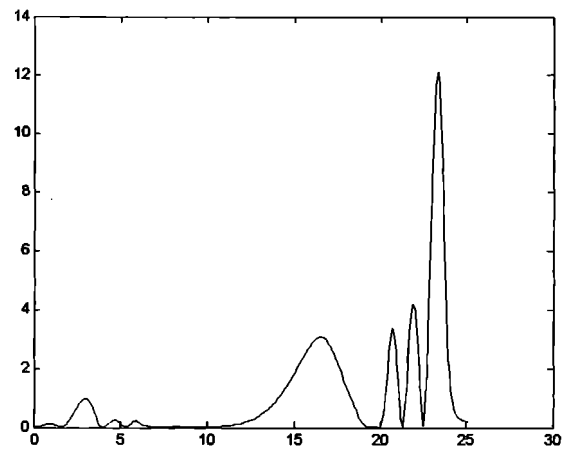


Figura 71. Error del sistema según nuevo criterio cuadrático ponderado

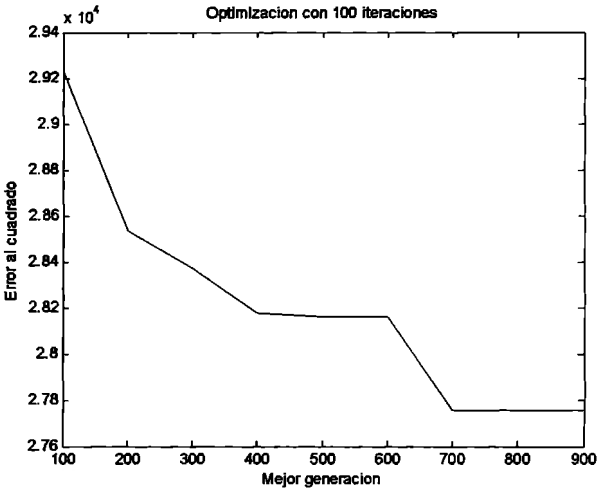


Figura 72. Optimización del sistema con 1000 generaciones.

Dicha optimización tomó 4 horas, ejecutando 300 000 veces la simulación, y se obtuvo una optimización para condiciones iniciales críticas:

best(1) = 41.116217
best(2) = 6.484782
best(3) = 11.042188
best(4) = 8.654519
best(5) = -0.795826
best(6) = 340.727888
best(7) = 79.851994
best(8) = 53.532277
best(9) = 31.549543
best(10) = 1.838144

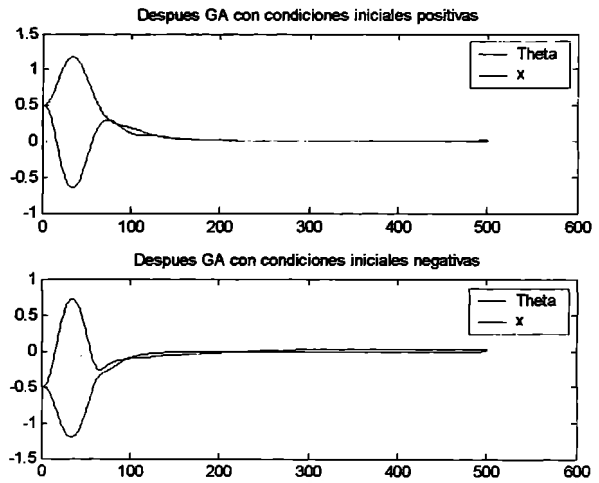


Figura 73. Respuesta del sistema a dos condiciones iniciales diferentes.

4. Implementación

En ésta sección se describe el procedimiento que se llevó a cabo para implementar un controlador con lógica difusa para un sistema de péndulo invertido realizado con las partes de una impresora descompuesta.

4.1. Sistema de péndulo invertido

Se desarmaron dos impresoras descompuestas, hasta tener acceso al motor y al cabezal deslizable. Se analizó el motor, y ya que ambos eran de pasos, se eligió el que tuviera pasos más pequeños, para mejorar la definición en el movimiento. Por último, se consideró que el cabezal pudiera removerse e instalarle el sensor angular (potenciómetro). Se eligió la impresora con mejores condiciones mecánicas, es decir, menor fricción, mejor engranaje y facilidad de manejo, que fue la Lexmark Z12, con un motor a pasos de 7.5Ω y 7.5° por paso.

Al notar que las impresoras no tenían ninguna retroalimentación de posición, o sensores de inicio o fin de riel, se decidió que tampoco se pondría un sensor de posición, pues se haría mediante programación al mandar los pasos al motor.

El siguiente paso consistió en la elección del sensor angular, que por practicidad fue un potenciómetro. Se compró un potenciómetro con poca fricción en la perilla, para asemejar al mecanismo libre de fricción. Se tomaron nueve ángulos diferentes y se observó su resistencia para determinar su linealidad, como se muestra en la siguiente tabla y figura (74):

Tabla 3. Linealidad del sensor angular

Ángulo	Resistencia [kΩ]
60	7.13
70	6.56
80	5.81
90	5.16
100	4.53
110	3.89
120	3.21
130	2.53

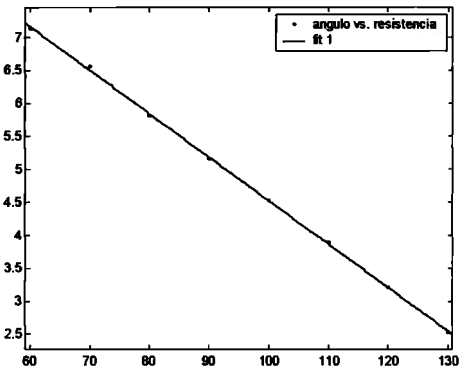


Figura 74. Linealidad del sensor angular (potenciómetro)

Se tiene una correlación de 0.9996, y encontramos que la ecuación que lo describe es:

$$y(x) = -0.06588*x + 11.11$$

Se montó el potenciómetro al cabezal deslizable, dejando que la varilla girara libremente, y atornillando la base del potenciómetro dentro del carro. Se realizó el cableado flexible para tener acceso a las tres terminales del potenciómetro.

El siguiente punto importante fue la elección del péndulo propiamente. Para ello, de la otra impresora descompuesta se tomó una varilla con un diámetro apenas mayor al de la perilla, y con una longitud de 30 cm. Para unir la perilla con la varilla se desbastó la segunda, hasta empotrar en la ranura del potenciómetro. En éste paso se rompió un potenciómetro, por lo que hubo que volver a ajustar el actual.

Ya que el motor a pasos es de tipo bipolar, es decir, cuenta con cuatro cables de control, se eligió el driver MC3479 de Motorola, que tiene una configuración como en la figura 75.

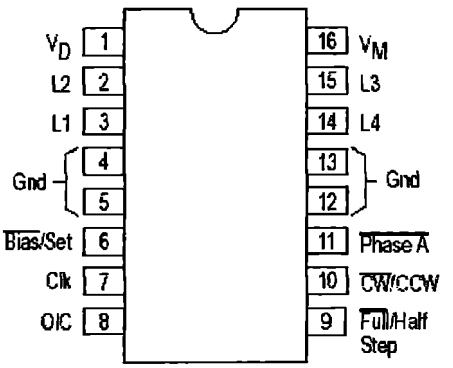


Figura 75. Descripción de pines del MC3479

Lo siguiente, y más complejo, fue la integración de sensor y actuador mediante un microcontrolador. Después de analizar varias opciones, se optó por emplear el BasicATOM, de BasicMicro Inc., que es un microcontrolador que utiliza lenguaje avanzado de programación similar a Basic, pero aún con más poder, como un ICD (In Circuit Debugger). Su set de instrucciones incluye los del microcontrolador BasicStamp, y algunos otros comandos integrados que son de gran utilidad.

El BasicATOM incluye un microcontrolador PIC 16F876, con 384 bytes de RAM y 8k de FLASH/espacio programable. Tiene un regulador de 5 volts y 16 puertos entrada/salida. Se eligió también por tener tres convertidores análogo-digital (ADC), y por poder utilizar interrupciones con programación simple.

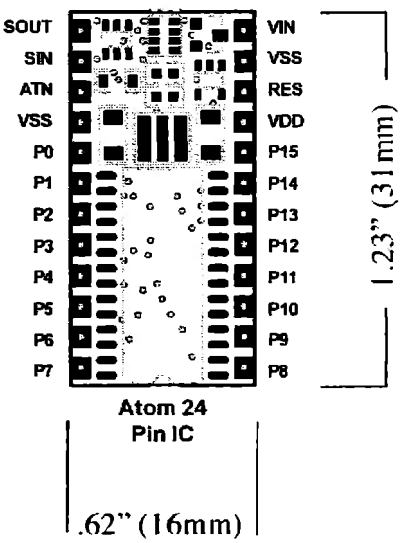


Figura 76. Puertos del BasicATOM

Se realizó el cableado, conectando el puerto 0 como salida de reloj, el puerto 1 para determinar el sentido del motor, y el puerto 2 para iluminarse cuando el péndulo esté vertical. Se conectó el potenciómetro a ±5V en sus extremos, y la escobilla móvil conectada al puerto AX0 para su conversión análogo-digital. El resto consistió en la programación, que se describe en la siguiente sección.

4.2. Controlador difuso

El control difuso presenta muchas ventajas, la principal por la que se decidió implementar es porque no requiere un conocimiento matemático del sistema, sino la ayuda de un experto para elegir los rangos de las entradas y salidas, así como sus funciones de membresía y las reglas lógicas si-entonces. Además, se eligió un controlador difuso tipo Takagi-Sugeno por la factibilidad de implementación empleando un microcontrolador.

Se realizó tan solo con tres funciones de membresía triangulares por entrada, siendo una entrada para el ángulo, otra para la derivada del ángulo y una tercera para la posición. En la figura 77 se muestra la fuzzificación, específicamente para el ángulo y velocidad angular (derivada de Euler).

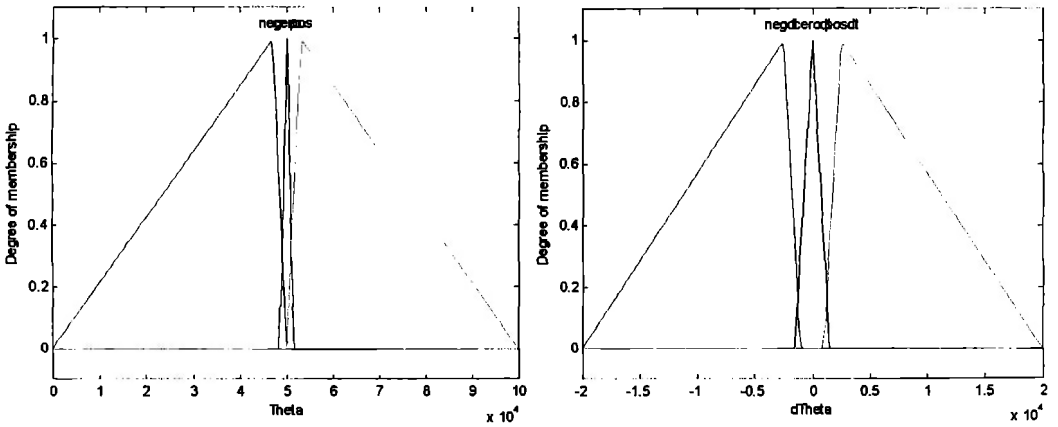


Figura 77. Funciones de membresia para las entradas Theta y Dtheta, del controlador difuso

Ya que las entradas se volvieron valores difusos, se evalúan las siguientes reglas, que corresponden a un PID difuso para el control de ángulo y velocidad angular:

[1] Si Error=P y d(Error)=P entonces Salida=P
[2] Si Error=P y d(Error)=Z entonces Salida=P
[3] Si Error=P y d(Error)=N entonces Salida=P
[4] Si Error=Z y d(Error)=P entonces Salida=N
[5] Si Error=Z y d(Error)=Z entonces Salida=Z
[6] Si Error=Z y d(Error)=N entonces Salida=P
[7] Si Error=N y d(Error)=P entonces Salida=N
[8] Si Error=N y d(Error)=Z entonces Salida=N
[9] Si Error=N y d(Error)=N entonces Salida=N

La Matriz de Asociación, para tres funciones de membresía por entrada, resulta:

Tabla 4. Tabla de asociación del controlador difuso tipo PD.

$\dot{e}$ \ e	N	Z	P
N	N	N	P
Z	N	Z	P
P	N	P	P

La salida se encuentra utilizando un método de inferencia min-max. Por último, se defuzzifica, y como es un sistema difuso de tipo Takagi-Sugeno de primer orden, simplemente se realiza un promedio entre la salida positiva (constante=50), cero (constante=0) y negativa (constante=-50).

Como se controla un motor de pasos, el sentido del motor estará determinado como positivo si la salida difusa de positivo es mayor a la negativa, o viceversa. Se hizo un valor absoluto del valor de salida (ya no importa el signo) y mientras mayor fuera la salida, se debía hacer más rápido el movimiento.

4.3. Resultados

En la imagen 78 se muestra el sistema físico de péndulo invertido.
En el apéndice 6 se encuentra el programa en BasicMicro con la lógica difusa. También, como anexo al documento digital, se encuentra un video que demuestra cómo el péndulo invertido es balanceado y llevado a la posición cero de la vía, tal como se propuso en el objetivo de control.
La velocidad del microcontrolador es crítica para sistemas de naturaleza inestable, y aunque el BasicATOM no es muy rápido, logra hacer la conversión análogo-digital y el procesamiento inteligente en un tiempo aproximado de 40 ms, de tal forma que logra estabilizar al sistema.
Lamentablemente el motor de pasos no logra alcanzar una velocidad muy rápida, como para lograr estabilizar el sistema ante grandes perturbaciones; pero sí demuestra su lógica de control ante pequeñas perturbaciones o condiciones iniciales.
La figura 79 muestra los resultados de la implementación, obtenidos del microcontrolador mediante el puerto serial, cuestión que lo hacía un poco más lento. Se alcanza a observar cómo trabaja a la velocidad máxima del motor, y logra estabilizar el péndulo.

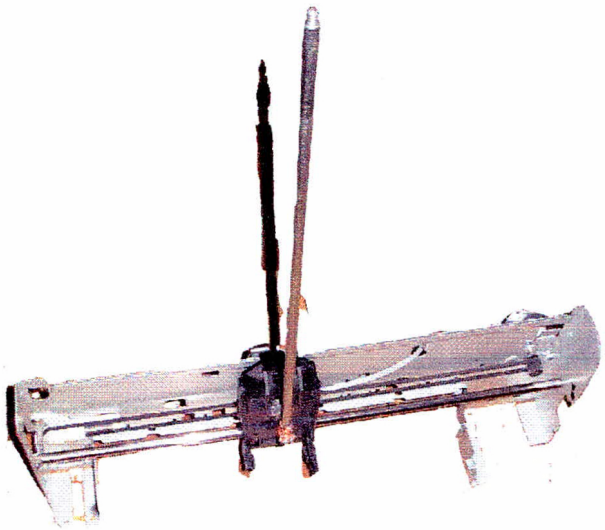


Figura 78. Sistema del péndulo invertido a partir de una impresora

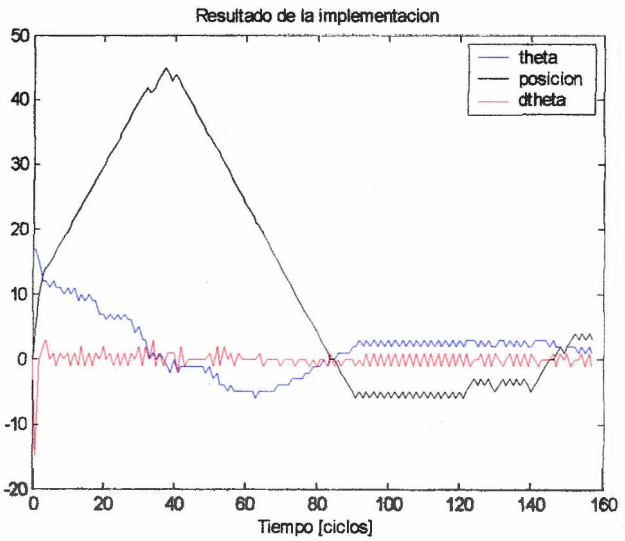


Figura 79. Resultado de la implementación

5. Conclusiones

El control inteligente presenta una nueva área para solucionar problemas de control, sus ventajas de deben a la integración de las computadoras para realizar algoritmos inteligentes y adaptivos, o mediante lógica más humana que con el control tradicional.
A lo largo de éste documento se definió el problema del péndulo invertido, modelándolo y realizando simulaciones. También, sin afán de comparación sino de aplicación, se realizaron diferentes controladores, desde unos tradicionales hasta otros inteligentes y optimizados. Todos ellos lograron controlar el péndulo invertido, y todos son mejorables, pero demostraron la aplicación y características principales de cada método empleado.
El controlador PID tiene la ventaja de ser simple y requerir pocos parámetros, pero requiere de un modelo lineal bien definido, y trabaja sólo para un punto de operación. De manera similar que el control con retroalimentación de estado, que funciona mejor pues tiene mejor control sobre todo el sistema, pero es aún más complejo de sintonizar.
Por su parte, la gran ventaja de todos los tipos de controladores inteligentes es que no requieren necesariamente de un modelo definido ni trabajan sólo en un punto de operación, pues son generalizables o trabajan en un rango más amplio. Se ha probado el potencial del control inteligente mediante simulación, al implementar en el péndulo invertido sus principales componentes, como son los controladores difusos, las redes neuronales y los algoritmos genéticos, así como una interacciones entre todos ellos con el controlador ANFIS-genético.
En el caso del controlador difuso, tiene la ventaja de utilizar lógica parecida a la humana, pues con tan solo un experto se puede realizar un controlador, sin realizar ninguna modelación matemática, pero tiene como desventaja una gran cantidad de parámetros a definir.
Por su parte, las redes neuronales permiten un aprendizaje y optimización al variar los pesos de cada neurona, y tienen la propiedad de generalización, por lo que algunos datos del sistema nos permiten identificarlo totalmente, el inconveniente es que no todas las estructuras convergen, y las configuraciones no aplican para todos los sistemas, por lo que requieren un complejo estudio de estabilidad para tener certeza de que son buenos controladores.
Conociendo las desventajas de los sistemas difusos y las redes neuronales, aparece ANFIS como una integración de ambas técnicas. Logran optimizar los pesos de una red neuronal que funciona con lógica difusa, pero aún así es necesario definir muchos parámetros, y el tiempo de entrenamiento es considerable. Aún así, considero que ANFIS es un método bastante confiable y elegante, que necesita únicamente algunos datos del sistema para lograr controlarlo correctamente.
Por último, al integrar los algoritmos genéticos con el controlador ANFIS, aunque el tiempo de entrenamiento es una gran desventaja, se obtienen controladores que realizan su tarea de la mejor forma posible.
El presente trabajo logró sobradamente sus objetivos, partiendo de un aprendizaje completo en técnicas de control inteligente, logrando demostrar en simulación que los controladores

funcionaban correctamente, y llegando aún más allá al demostrarlo con una implementación física del péndulo invertido mediante un controlador difuso.
Un trabajo a futuro para la continuación de éste proyecto consiste en la mejora de cada uno de los métodos propuestos, sintonizando correctamente los parámetros, para hacerlos sistemas más rápidos, con mayor adaptabilidad, que reaccionen mejor ante perturbaciones o cambios de planta, y con tiempos de entrenamiento menores y mejor validados teóricamente.
También se sugiere realizar la implementación de cada uno de los métodos, con un sistema mecánico más rápido, por ejemplo con un motor de corriente directa, y un microcontrolador que permita tener un tiempo de ciclo mucho menor, para realizar un muestreo y ejecución más rápidos.
La implementación de control inteligente tiene un campo de desarrollo muy amplio, cada vez observamos más microcontroladores con lógica difusa, o aplicaciones de redes neuronales en más áreas de investigación. Está trabajándose en la interrelación de las metodologías para tomar sus ventajas y buscar controladores que no dependan de un conocimiento pleno del sistema, y que sean adaptables ante cambios en él, y hasta que se optimicen en tiempo real.
Estoy convencido de que en pocos años se encontrará una metodología de controlador inteligente que sea totalmente adaptable y se pueda implementar en cualquier planta sin importar sus características, y aprender de ella con el tiempo, eliminando toda necesidad de modelación de sistemas.

6. Bibliografía

[1] Jang. Self Learning Fuzzy Controllers Based on Temporal Back-Propagation

[2] Kent Lundeberg. The Inverted Pendulum System. 1994-2002.

[3] Williams, V.; Matsuoka, K.; Learning to balance the inverted pendulum using neural networks. Neural Networks, 1991. 1991 IEEE International Joint Conference on , 18-21 Nov. 1991. Page(s): 214 -219 vol.1

[4] M.I. Jordan and R.A. Jacobs. Learning piecewise control strategies in a modular neural network architecture. Systems, Man and cybernetics, IEEE Transactions on, Volume: 23, Issue: 2, March-April 1993, pages 337-345.

[5] S. Kitamulra and M. Saitoh: Stability Learning Control of the inverted pendulum using neural networks (in Japanese). Knowledge and Intelligent systems symposium. Pp. 61-61. March, 1990.

[6] Kouda, N.; Matsui, N.; Nishimura, H. Control for swing-up of an inverted pendulum using qubit neural network. ; SICE 2002. Proceedings of the 41st SICE Annual Conference , Volume: 2 , 5-7 Aug. 2002. Pages:765 - 770 vol.2

[7] Sazonov, E.S.; Klinkhachorn, P.; Klein, R.L.; Hybrid LQG-neural controller for inverted pendulum system. System Theory, 2003. Proceedings of the 35th Southeastern Symposium on , 16-18 March 2003. Pages:206 – 210

[8] Balancing and position tracking control of an inverted pendulum on an x-y plane using decentralized neural networks. Hyun Taek Cho; Seul Jung;Advanced Intelligent Mechatronics, 2003. AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on , Volume: 1 , July 20 - July 24, 2003. Pages:181 – 186

[9] Inoue, H.; Matsuo, K.; Hatase, K.; Kamei, K.; Tsukamoto, M.; Miyasaka, K.;A fuzzy classifier system using hyper-cone membership functions and its application to inverted pendulum control. Systems, Man and Cybernetics, 2002 IEEE International Conference on , Volume: 6 , 6-9 Oct. 2002. Pages:6 pp. vol.6

[10] Harrison, R.F.; Asymptotically optimal stabilising quadratic control of an inverted pendulum. Control Theory and Applications, IEE Proceedings- , Volume: 150 , Issue: 1 , Jan. 2003. Pages:7 – 16

[11] Shing-Chia Chen; Wen-Liang Chen; Output regulation of nonlinear uncertain system with nonminimum phase via enhanced RBFN controller. Systems, Man and Cybernetics, Part A, IEEE Transactions on , Volume: 33 , Issue: 2 , March 2003. Pages:265 – 270

[12] Pal, T.; Pal, N.R.; SOGARG: A self-organized genetic algorithm-based rule generation scheme for fuzzy controllers. Evolutionary Computation, IEEE Transactions on , Volume: 7 , Issue: 4 , Aug. 2003. Pages:397 – 415

[13] Lam, H.K.; Leung, F.H.; Tam, P.K.S.; Design and stability analysis of fuzzy model-based nonlinear controller for nonlinear systems using genetic algorithm. Systems, Man and Cybernetics, Part B, IEEE Transactions on , Volume: 33 , Issue: 2 , April 2003. Pages:250 – 257

[14] Mohanlal, P.P.; Kaimal, M.R.; Exact fuzzy modeling and optimal control of the inverted pendulum on cart. Decision and Control, 2002, Proceedings of the 41st IEEE Conference on , Volume: 3 , 10-13 Dec. 2002. Pages:3255 - 3260 vol.3

[15] Yang Gao; Meng Joo Er; Online adaptive fuzzy neural identification and control of a class of MIMO nonlinear systems. Fuzzy Systems, IEEE Transactions on , Volume: 11 , Issue: 4 , Aug. 2003. Pages:462 - 477

[16] Ernesto Olguín. Modelado matemático del péndulo invertido.

[17] Jang,Shing. Self-Learning Fuzzy Controllers Based on Temporal Back Propagation. Neural Networks, IEEE Transactions on , Volume: 3 , Issue: 5 , Sept. 1992 Pages:714 – 723.

[18] Chen Wei Ji; Fang Lei; Lei Kam Kin. Fuzzy Logic Controller for An Inverted Pendulum System. Intelligent Processing Systems, 1997. ICIPS '97. 1997 IEEE International Conference on , Volume: 1 , 28-31 Oct. 1997 Pages:185 - 189 vol.1

[19] Control tutorials for MATLAB and Simulink. Department of Electrical and computer engineering. Utah State University, 2003. http://mechatronics.ece.usu.edu/2003/lab/ctms/examples/pend/invss.htm#lqr

[20] Williams, V.; Matsuoka, K.; Learning to balance the inverted pendulum using neural networks. Neural Networks, 1991. 1991 IEEE International Joint Conference on , 18-21 Nov. 1991. Page(s): 214 -219 vol.1

[21] Omatu, S.-Ide, T. Stabilization of Inverted Pendulum by Neuro-Control. Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on, Volume: 4, 27 June-2 July 1994. Pages 2367-3272 vol. 4

[22] Nørgaard, Ravn, Poulsen, Hansen. Neural Networks for Modelling and control of Dynamic Systems. Ed. Springer. 2nd Printing 2001. ISBN 1-85233-227-1.

[23] Jang-Sun-Mizutani: Neuro-Fuzzy and Soft Computing, Prentice-Hall, 1997. ISBN: 0132610663FAS

[24] Riedmiller, M. Controlling an inverted pendulum by neural plant identification. Systems, Man and Cybernetics, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on, 17-20 Oct. 1993. Pages 473-478 vol. 4

[25] Omatu, S.;Fujinaka, T.;Yoshioka, M. Neuro-PID control for inverted single and double pendulums. Systems, Man, and Cybernetics, 2000 IEEE International Conference on, Volume: 4, 8-11 Oct. 2000. Pages 3685-2690 vol. 4

[26] J. Jang. (1993). ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans. Man and Cybernetics Systems. pp. 665-685.

[27] M. Nørgaard: "Neural Network Based Control System Design Toolkit, ver. 2" Tech. Report. 00-E-892, Department of Automation, Technical University of Denmark, 2000).

[28] M.Nørgaard:"Neural Network Based System Identification Toolbox," Tech. Report. 00-E-891, Department of Automation, Technical University of Denmark, 2000.

[29] Omatu S.;Deris S.; Kitagawa, K. Stabilization of inverted pendulum by the genetic algorithm. Systems, Man and Cybernetics, 1995. 'Intelligent Systems for the 21st Century'. Pages 4372-4377 vol. 5

[30] Kenneth Price and Rainer Storn, Differential Evolution (DE). http://www.icsi.berkeley.edu/~storn/code.html . Price, K. and Storn, R., "Differential Evolution: Numerical Optimization Made Easy", Dr. Dobb's Journal, April 97, pp. 18 - 24.

[31] Mirza, Ashab; Dr. Sarfraz Hussain. Inverted Pendulum. Journal of AMSE France, Dec 2000, Vol. 55, No 3,4. http://www.ewh.ieee.org/sb/iiee/pip.pdf

[32] Messner, Bill and Tilbury, Dawn. Control Tutorials for Matlab. Prentice Hall. ISBN 0-201-47700-9. http://www.engin.umich.edu/group/ctm/index.html

[33] Dorf R. C.;Bishop, R.H.. Teaching modern control system design. Decision and Control, 1999. Proceedings of the 38th IEEE Conference on, Volume: 1, 7-10 Dec. 1999. Pages 364-369 vol. 1

[34] T. Takagi and M Sugeno. Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. Systems, Man and Cybernetics, SMC-15:116-132, 1985.

[35] Jang-Sun-Mizutani; Neuro-Fuzzy and Soft Computing, Prentice-Hall, 1997. ISBN: 0132610663.

[36] MATLAB, the language of technical computing. De MathWorks. Versión 6.5.0 R-13., Imagen: http://www.microrobotna.com/pendulum.htm

[37] Ernesto Olguín. Simulación en MATLAB de péndulo invertido, con etapa gráfica. Abril 1, 2003.

[38] EXTEND, "performance modeling for decision support". De Imagine That. Versión 3.2.1

[39] Eugenio Álvarez. Soft computing with application on Inverted Pendulum system. Tampere University of Techynology, Finlandia.

A. Apéndices

A.1. "Soft Computing".

En este anexo se presentan en inglés los conocimientos teóricos básicos sobre control inteligente, buscando una mejor comprensión de los desarrollos de este proyecto. El texto se presenta en inglés porque fue realizado como parte de proyectos de ingeniería I, en un programa de intercambio internacional en la Tampere University of Technology, en Finlandia[39].

A.1.1. Introduction

Soft computing (SC), an innovative approach to constructing computationally intelligent systems, has just come into the limelight. It is now realized that complex real-world problems require intelligent systems that combine knowledge, techniques, and methodologies from various sources. These intelligent systems are supposed to possess humanlike expertise within a specific domain, adapt themselves and learn to do better in changing environments, and explain how they make decisions or take actions.
Neuro-fuzzy modeling, together with a new driving force from stochastic, gradient-free optimization techniques such as genetic algorithms and simulated annealing, forms the constituents of so-called soft computing, which is aimed at solving real-world decision-making, modeling and control problems. These problems are usually imprecisely defined and require human intervention. Thus, neuro-fuzzy and soft computing, with their ability to incorporate human knowledge and to adapt their knowledge base via new optimization techniques, are likely to play increasingly important roles in the conception and design of hybrid intelligent systems.
The next table shows the soft computing constituents and their main strength's.

Table 5 Soft computing constituents (the first three items) and conventional artificial intelligence.

Methodology	**Strength**
Neural Network	Learning and adaptation
Fuzzy set theory	Knowledge representation by fuzzy if-then rules
Genetic algorithm & simulated annealing	Systematic random search
Conventional AI	Symbolic manipulation

With all this methodology's as tools, soft computing is able to help in several ways; the characteristics of soft computing can be summarized as follows:

Human expertise. Soft computing utilized human expertise in the form of fuzzy if-then rules, as well as in conventional knowledge representations, to solve practical problems.
Biological inspired computing models. Inspired by biological neural networks, artificial neural networks are employed extensively in soft computing to deal with Perceptron, pattern recognition, and nonlinear regression and classification problems.
New optimization techniques. Soft computing applies innovative optimization methods arising from various sources; they are genetic algorithms (inspired by the evolution and selection process), simulated annealing (motivated by thermodynamics), the random search method, and the downhill Simplex method. These optimization methods do not require the gradient vector of an objective function, so they are more flexible in dealing with complex optimization problems.
Numerical computation. Unlike symbolic AI, soft computing relies mainly on numerical computation. Incorporation of symbolic techniques in soft computing is an active research area within this field.
New application domains. Because of its numerical computation, soft computing has found a number of new application domains besides that of AI approaches. These application domains are mostly computation intensive and include adaptive signal processing, adaptive control, nonlinear system identification, nonlinear regression, and pattern recognition.
Model-free learning. Neural networks and adaptive fuzzy inference systems have the ability to construct models using only target system sample data. Detailed insight into the target system helps set up the initial model structure, but it is not mandatory.
Intensive computation. Without assuming too much background knowledge of the problem being solved, neuro-fuzzy and soft computing rely heavily on high-speed number-crunching computation to find rules or regularity in data sets. This is a common feature of all areas of computational intelligence.
Fault tolerance. Both neural networks and fuzzy inference systems exhibit fault tolerance. The deletion of a neuron in a neural network, or a rule in a fuzzy inference system, does not necessarily destroy the system. Instead, the system continues performing because of its parallel and redundant architecture, although performance quality gradually deteriorates.
Goal driven characteristics. Neuro-fuzzy and soft computing are goal driven; the path leading from the current state to the solution does not really matter as long as we are moving toward the goal in the long run. This is particularly true when used with derivative-free optimization schemes, such as genetic algorithms, simulated annealing, and the random search method. Domain-specific knowledge helps reduces the amount of computation and search time, but it is not a requirement.
Real-world applications. Most real-world problems are large scale and inevitably incorporate built-in uncertainties; this precludes using conventional approaches that require detailed description of the problem being solved. Soft computing is an integrated approach that can usually utilize specific techniques within subtasks to construct generally satisfactory solutions to real-world problems.
In the following chapters, I will give an introduction of some of the methodologies of soft computing.

A.1.2. Fuzzy Theory

A classical set is a set with crisp boundary. For example, mathematically we can express the set of tall persons as a collection of persons whose height is more than 1.5m, like:

$$A = \{x | x > 1.5\} \tag{2.1}$$

If we let A="tall person" and x="height". This is an unnatural and inadequate way of representing our usual concept of "tall person". For one thing, the dichotomous nature of the classical set would classify a person of 1.5001m tall as a tall person, but not a person 1.4999 m tall. This distinction is intuitively unreasonable.

In contrast to a classical set, a fuzzy set, as the name implies, is a set without a crisp boundary. That is, the transition from "belong to a set" to "not belong to a set" is gradual, and this smooth transition is characterized by membership functions that give fuzzy sets flexibility in modeling commonly used linguistic expressions, such as "the water is hot" or "the temperature is high".

A few definitions will follow. If X is a collection of objects denoted generically by x, then a fuzzy set A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) | x \in X\} \tag{2.2}$$

Where $\mu_A(x)$ is called the membership function for the fuzzy set A. The MF maps each element of X to a membership grade or membership value) between 0 and 1.

A fuzzy if-then rule (also known as fuzzy rule, fuzzy implication, or fuzzy conditional statement) assumes the form

$$\text{If x is A then y is B,} \tag{2.3}$$

Where A and B are linguistic values (like "hot" or "negative") defined by fuzzy sets on universes of discourse (the range) X and Y, respectively. Often "x is A" is called the antecedent or premise, while "y is B" is called the consequence or conclusion. For example, if pressure is high, then volume is small.

Fuzzy reasoning, also known as approximate reasoning, is an inference procedure that derives conclusions from a set of fuzzy if-then rules and known facts. In summary, it can be divided into four steps:

1. Degree of compatibility. Compare the known facts with the antecedents of fuzzy rules to find the degrees of compatibility with respect to each antecedent MF.
2. Firing strength. Combine degrees of compatibility with respect to antecedent MFs in a rule using fuzzy AND or OR operators to form a firing strength that indicates the degree to which the antecedent part of the rule is satisfied.
3. Qualified (induced) consequent MFs. Apply the firing strength to the consequent MF of a rule to generate a qualified consequent MF.
4. Overall output MF. Aggregate all the qualified consequent MFs to obtain an overall output MF.

Sometimes it is necessary to have a crisp output, especially in a situation where a fuzzy inference system is used as a controller. Therefore, we need a method of defuzzification to extract a crisp value that best represents a fuzzy set. A fuzzy inference system with a crisp output is shown in the next figure.

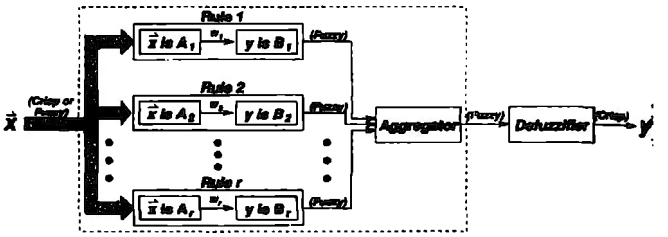


Figure 80 Block diagram for a Fuzzy Inference System

In this work we will discuss two fuzzy models: Mandani and Sugeno.

A.1.2.1. Mandani Fuzzy Model

The next figure is an illustration of how a two-rule Mandani fuzzy inference system derives the overall output z when subjected to two inputs x and y, using the min-max criteria (there are also the min-product or sum-product criteria's).

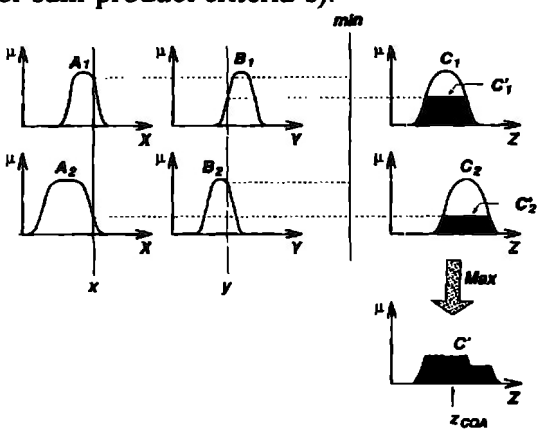


Figure 81 Mandani Fuzzy Inference System

Defuzzification refers to the way a crisp value is extracted from a fuzzy set as a representative value. In general, there are five methods, as shown in the figure.

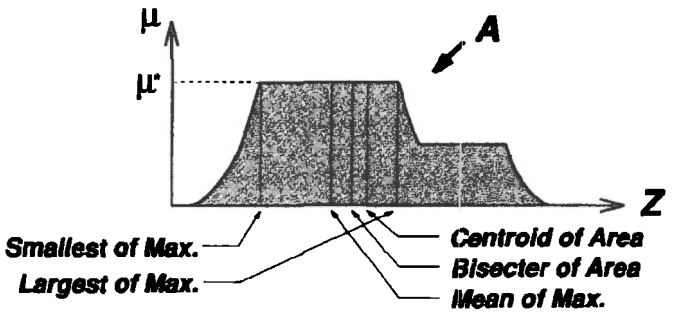


Figure 82 Various defuzzification schemes for obtaining a crisp output

A.1.2.2. Sugeno Fuzzy Model

The Sugeno Fuzzy model (also known as the TSK fuzzy model) was proposed by Takagi, Sugeno, and Kang in an effort to develop a systematic approach to generating fuzzy rules from a given input-output data set. A typical fuzzy rule in a Sugeno fuzzy model has the form:

$$\text{If x is A and y is B then } z = f(x,y), \tag{2.4}$$

Where A and B are fuzzy sets in the antecedent, while z=f (x,y) is a crisp function in the consequent, as shown in the next figure.

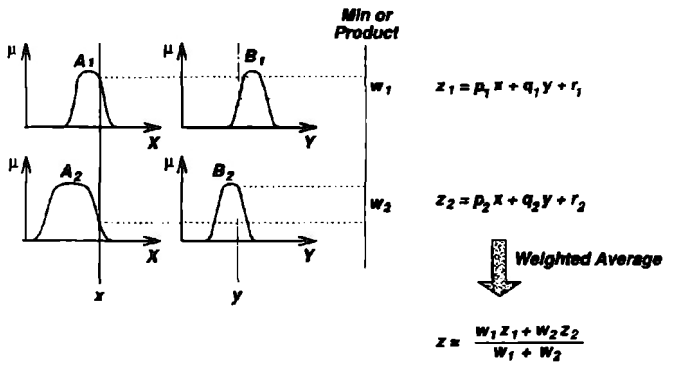


Figure 83 The Sugeno Fuzzy Model

Since each rule has a crisp output, the overall output is obtained via weighted average, thus avoiding the time-consuming process of defuzzification required in Mandani model. In practice, the weighted average operator is sometimes replaced with the weighted sum operator (that is, $z=w_1z_1+ w_2z_2$ in the figure) to reduce computation.

Unlike the Mandani fuzzy model, the Sugeno fuzzy model cannot follow the compositional rule of inference strictly in its fuzzy reasoning mechanism, to formalize an inference procedure upon a set of fuzzy if-then rules.

A.1.3. *Regression Theory*

The problem of determining a mathematical model for an unknown system (also referred to as the target system) by observing its input-output data pairs is generally referred to as system identification.

System identification generally involves two top-down steps: Structure identification need to apply a priori knowledge about the target system to determine a class of models within which the search for the most suitable model is to be conducted, usually this class of models is denoted by a parameterized function $y=f(u; \theta)$, where y is the model's output, u is the input vector, and θ is the parameter vector. On Parameter identification, the structure of the model is known and all we need to do is apply optimization techniques to determine the

parameter vector $\theta = \hat{\theta}$ such that the resulting model $\hat{y} = f(u;\hat{\theta})$ can describe the system appropriately.
The least square methods provide us with mathematical procedures by which a linear model can achieve a best fit to experimental data in the sense of least-squared error. Nonlinear models that are intrinsically linear can also take advantage of the least-squares methods.
We need some basic definitions.

Gradient of a scalar function. Let $\mathbf{x}=[x_1,\ldots,x_n]^T$ and let $f(x)$ be a scalar function of $\mathbf{x}$. Then the derivative of $f(x)$ with respect to $\mathbf{x}$, is a column vector denoted by

$$\nabla f(x) = \begin{bmatrix} \partial f(x)/\partial x_1 \\ \vdots \\ \partial f(x)/\partial x_n \end{bmatrix} \tag{3.1}$$

Jacobian of a vector function. Let $\mathbf{x}=[x_1,\ldots,x_n]^T$ and let $\mathbf{f(x)}$ be a vector function of $\mathbf{x}$, denoted by $\mathbf{f(x)}=[f_1(\mathbf{x}),\ldots,f_m(\mathbf{x})]^T$. Then the derivative of $\mathbf{f(x)}$ with respect to $\mathbf{x}$, is an $m \times n$ matrix denoted by

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_1} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1(x) \\ \vdots \\ \nabla^T f_1(x) \end{bmatrix} = \begin{bmatrix} g_{f_1}^T \\ \vdots \\ g_{f_1}^T \end{bmatrix} \tag{3.2}$$

Hessian of a scalar function. Let $\mathbf{x}=[x_1,\ldots,x_n]^T$ and let $f(\mathbf{x})$ be a scalar function of $\mathbf{x}$, then the second derivative of $\mathbf{f(x)}$ with respect to $\mathbf{x}$, is an $n \times n$ matrix denoted by

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{3.3}$$

A.1.3.1. Least-Squares Estimator (LSE)

In the general least-squares problem, the output of a linear model y is given by the linearly parameterized expression

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \ldots + \theta_n f_n(u) \tag{3.4}$$

Where $\mathbf{u}=[u_1,\ldots,u_p]^T$ is the model's input vector, $f_1,\ldots,f_n$ are known functions of $\mathbf{u}$, and $\theta_1,\ldots,\theta_n$ are unknown parameters to be estimated. In statistics, the task of fitting data using

a linear model is referred to as linear regression. Thus, this is also called regression function, and the θ_i's are called the regression coefficients.
To identify the unknown parameters, θ_i, usually we have to perform experiments to obtain a training data set composed of data pairs $\{(\mathbf{u}_i; y_i), i=1,\ldots,m\}$; they represent desired input-output pairs of the target system to be modeled. Substituting each data pair, yields have a set of m linear equations:

$$\begin{cases} f_1(u_1)\theta_1 + f_2(u_1)\theta_2 + \ldots + f_n(u_1)\theta_n = y_1 \\ f_1(u_2)\theta_1 + f_2(u_2)\theta_2 + \ldots + f_n(u_2)\theta_n = y_2 \\ \vdots \\ f_1(u_m)\theta_1 + f_2(u_m)\theta_2 + \ldots + f_n(u_m)\theta_n = y_m \end{cases} \tag{3.5}$$

Using matrix notation, we can rewrite the preceding equations in a concise form:

$$A\,\theta = y, \tag{3.6}$$

Where **A** is an $m \times n$ matrix (sometimes called the design matrix):

$$A = \begin{bmatrix} f_1(u_1) & \cdots & f_n(u_1) \\ \vdots & \vdots & \vdots \\ f_1(u_m) & \cdots & f_n(u_m) \end{bmatrix} \tag{3.7}$$

θ is an $n \times 1$ unknown parameter vector:

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \tag{3.8}$$

And **y** is an $m \times 1$ output vector

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \tag{3.9}$$

We incorporate an error vector to account for random noise or modeling error, and instead of finding the exact solution, we want to search for $\theta = \hat{\theta}$ which minimizes the sum of squared error, defined by

$$E(\theta) = \sum_{i=1}^{m} (y_i - a_i^T \theta)^2 = e^T e = (y - A\theta)^T (y - A\theta) \tag{3.10}$$

Where $\mathbf{e}=\mathbf{y}-\mathbf{A}\,\boldsymbol{\theta}$ is the error vector produced by a specific choice of $\boldsymbol{\theta}$. Note that $E(\boldsymbol{\theta})$ is in quadratic form and has a unique minimum at $\theta = \hat{\theta}$.

A.1.3.2. Steepest Descent Method

The following reviews a fundamental class of gradient-based optimization techniques, capable of determining search directions according to an objective function's derivative information.

In the rest of the chapter, we focus on minimizing a real-valued objective function E defined on an n-dimensional input space $\theta = [\theta_1, \theta_2, ..., \theta_n]^T$. Finding a (possibly local) minimum point $\theta = \theta^*$ that minimized $E(\theta)$ is of primary concern.
In iterative descent methods, the next point θ_{next} is determined by a step down from the current point θ_{now} in a direction vector **d**:

$$\theta_{next} = \theta_{now} + \eta d \tag{3.11}$$

Where η is some positive step size regulating to what extent to proceed in that direction. For convenience, we alternatively use the following formula:

$$\theta_{k+1} = \theta_k + \eta_k d_k \quad (k=1,2,3,\ldots), \tag{3.12}$$

Where k denotes the current iteration number.
The iterative descent methods compute the kth step $\eta_k d_k$ through two procedures: firs determining direction **d**, and then calculating step size η. The next point θ_{next} should satisfy the following inequality:

$$E(\theta_{next}) = E(\theta_{now} + \eta d) < E(\theta_{now}) \tag{3.13}$$

A.1.3.3. Levenberg-Marquardt Method

The Hessian can be altered by adding a positive definite matrix **P** to **H** to make **H** positive definite. Levenberg and Marquardt introduced this notion in least-squares problems. This method can handle well ill-conditioned matrices $\mathbf{J}^T\mathbf{J}$ by altering the descent equation to

$$\theta_{next} = \theta_{now} - (\mathbf{J}^T\mathbf{J} + \lambda \mathbf{I})^{-1} g_k \tag{3.14}$$

Where λ is some nonnegative value and $\mathbf{g}_h \equiv 1/2\ \mathbf{g}$ for simplicity.
The search direction found from a Gauss-Newton method need not be particularly optimal. The direction is determined via an approximation of the criterion, $L^{(i)}(\theta)$, which can be expected to be valid only in a neighborhood around the current iterate. If the minimum $L^{(i)}(\theta)$ is far from the current iterate, $\theta^{(i)}$, a poor search direction may be obtained. Intuitively, it makes more sense to search for the minimum of $L^{(i)}(\theta)$ only inside some neighborhood around the current iterate. Selecting this neighborhood as a ball with radius $\delta^{(i)}$, the minimization problem can be formulated as

$$\theta^{(i+1)} = \arg\min_{\theta} L^{(i)}(\theta)$$

Subject to $$\left|\theta - \theta^{(i)}\right| \le \delta^{(i)} \tag{3.15}$$

The update rule appearing when solving the constrained optimization problem can be found in Marquardt (1963):

$$\theta^{(i+1)} = \theta^{(i)} + f^{(i)} \tag{3.16}$$

$$\left[R(\theta^{(i)}) + \lambda^{(i)} I\right] f^{(i)} = -G(\theta^{(i)}) \tag{3.17}$$

The Levenberg-Marquardt direction is an intermediate between the Gauss-Newton direction ($\lambda \to 0$) and the steepest descent direction ($\lambda \to \infty$).

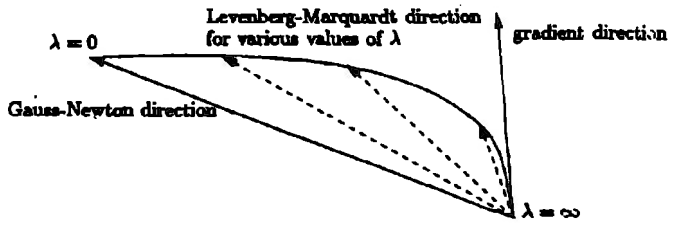


Figure 84 The Levenberg-Marquardt direction for various values of λ

A.1.4. Neural Networks

When scanning the literature on neural networks there does not seem to be one definition of a neural network with which everyone agrees, but most types of neural networks can be covered by the definition: A system of simple processing elements, neurons, that are connected into a network by a set of (synaptic) weights. The function of the network is determined by the architecture of the network, the magnitude of the weights and the processing element's mode of operation.

An adaptive network is a network structure whose overall input-output behavior is determined by a collection of modifiable parameters. Specifically, the configuration of an adaptive network is composed of a set of nodes connected by directed links, where each node performs a static node function on its incoming signals to generate a single node output and each link specifies the direction of signal flow from one node to another. Usually a node function is a parameterized function with modifiable parameters; by changing these parameters, we change the node function as well as the overall behavior of the adaptive network.

The parameters of an adaptive network are distributed into its nodes, so each node has a local parameter set. The union of these local parameter sets in the network's overall parameter set. If a node's parameter set is not empty, then its node function depends on the parameter values; we use a square to represent this kind of adaptive node. On the other hand, if a node has an empty parameter set, then its function is fixed; we use a circle to denote this type of fixed node. Each adaptive node can be decomposed into a fixed node plus one or several parameter nodes.

Adaptive networks are generally classified into two categories on the basis of the type of connections they have: feedforward and recurrent. The adaptive network is feedforward when the output of each node propagates from the input side (left) to the output side (right) unanimously. If there is a feedback link that forms a circular path in a network, then the network is recurrent.

In the layered representation, there are no links between nodes in the same layer, and outputs of nodes in a specific layer are always connected to nodes in succeeding layers. This representation is usually preferred because of its modularity, in that nodes in the same layer have the same functionality or generate the same level of abstraction about input vectors.

Another representation is the topological ordering representation, which labels the nodes in an ordered sequence 1,2,3,..., such that there are no links from node i to node j whenever $i \leq j$. The representation is less modular, but it facilitates the formulation of learning rules. The basic learning rule of the adaptive network is the well-known steepest descent method, in which the gradient vector is derived by successive invocations of the chain rule.

A.1.4.1. Back propagation Learning.

The procedure of finding a gradient vector in a network structure is generally referred to as back propagation because the gradient vector is calculated in the direction opposite to the flow of the output of each node. Once the gradient is obtained, a number of derivative-based optimization and regression techniques are available for updating the parameters. In particular, if we use the gradient vector in a simple steepest descent method, the resulting learning paradigm is often referred to as the back propagation learning rule.

When networks with more than one layer of nonlinear activation functions are considered, the expressions for the elements in the partial derivative matrix, $\phi(t)$, obviously become more complex. In particular, one should be careful not to carry out the same computations more than once. Due to the simple structure, it is possible to derive a very nice algorithm for computing the gradient of the criterion for general n-layer feedforward networks with arbitrary activation functions. The algorithm is shown below for a two-layer network. The basic algorithm is restricted to pure feedforward networks; however, the algorithm is quite easily modified to give only the partial derivative matrix, $\phi(t)$.

It is recalled that the gradient of the least squares criterion takes the form (for completeness the multi-output case is now considered)

$$G(\theta^{(i)}) = V_N'\left(\theta^{(i)}, Z^N\right) = \frac{1}{N}\sum_{t=1}^{N} \frac{\partial \varepsilon^T(t,\theta)}{\partial \theta}\varepsilon(t,\theta)\Big|_{\theta=\theta^{(i)}}$$

$$= -\frac{1}{N}\sum_{t=1}^{N} \frac{\partial \hat{y}^T(t|\theta)}{\partial \theta}\left[y(t) - \hat{y}(t|\theta)\right]\Big|_{\theta=\theta^{(i)}} \qquad (4.1)$$

The output of the kth unit of a general two-layer network can be expressed as

$$\hat{y}(t|\theta) = F_k\left[\sum_{j=0}^{n_h} w_{k,j} h_j(t)\right] = F_k\left[\sum_{j=0}^{n_h} w_{k,j} f_j\left(\sum_{l=0}^{n_\varphi} w_{j,l}\varphi_l(t)\right) + w_{k,0}\right] \qquad (4.2)$$

f_j specifies the activation function for unit j in the hidden layer and F_k specifies the activation function for output k. For simplicity, the biases are regarded as additional weights.

The partial derivatives of the network output with respect to the weights in each of the two layers are determined by

$$\varphi_{j,k}^{(w)} = \frac{\partial \hat{y}_k(t|\theta)}{\partial w_{k,j}} = h_j(t) F_k'\left[\sum_{j=0}^{n_h} w_{k,j} h_j(t)\right] \qquad (4.3)$$

And

$$\varphi_{j,k,l}^{(w)} = \frac{\partial \hat{y}_k(t|\theta)}{\partial w_{k,l}} = \varphi_l(t) f'_j \left[\sum_{j=0}^{n_\varphi} w_{j,l} \varphi_l(t) \right] w_{k,j} F'_k \left[\sum_{j=0}^{n_h} w_{k,j} h_j(t) \right] \tag{4.4}$$

This leads to the following expression for the gradient in the hidden-to-output layer

$$G(w_{k,j}) = \sum_{t=1}^{N} h_j(t) F'_k \left[\sum_{j=0}^{n_h} w_{k,j} h_j(t) \right] \left(y_k(t) - \hat{y}_k(t|\theta) \right) = \sum_{t=1}^{N} h_j(t) \delta_k^{(w)}(t) \tag{4.5}$$

Where the quantity $\delta_k^{(w)}(t)$ has been introduced

$$\delta_k^{(w)}(t) = F'_k \left[\sum_{j=0}^{n_h} w_{k,j} h_j(t) \right] \left(y_k(t) - \hat{y}_k(t|\theta) \right) \tag{4.6}$$

Similarly, the gradient for an input-to-hidden layer weights is given by

$$G(w_{k,l}) = \sum_{t=1}^{N} \varphi_l(t) f'_j \left[\sum_{l=0}^{n_\varphi} w_{j,l} \varphi_l(t) \right] \sum_{k=1}^{n_y} w_{k,j} \delta_k^{(w)}(t) = \sum_{t=1}^{N} \varphi_l(t) \delta_j^{(w)}(t) \tag{4.7}$$

Where

$$\delta_j^{(w)}(t) = f'_j \left[\sum_{l=0}^{n_\varphi} w_{j,l} \varphi_l(t) \right] \sum_{k=1}^{n_y} w_{k,j} \delta_k^{(w)}(t) \tag{4.8}$$

It is straightforward to generalize the algorithm to networks containing more than two layers. The procedure is simply continued backwards layer by layer while back-propagating "deltas".

A.1.4.2. Online-Offline Learning

There are two types of learning paradigms that are available to suit the needs for various applications. In off-line learning the update formula for parameter α is based on the equation:

$$\frac{\partial^+ E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial^+ E_p}{\partial \alpha} \tag{4.9}$$

And the update action takes place only after the whole training data set has been presented –that is, only after each epoch or sweep. On the other hand, in on-line learning, the parameters are updated immediately after each input-output pair has been presented, and the update formula is based on the equation:

$$\frac{\partial^+ E_p}{\partial \alpha} = \frac{\partial^+ E_p}{\partial x_{l,i}} \frac{\partial f_{l,i}}{\partial \alpha} = \varepsilon_{l,i} \frac{\partial f_{l,i}}{\partial \alpha} \tag{4.10}$$

A.1.4.3. Multi Layer Perceptron

The Perceptron represents one of the early attempts to build intelligent and self-learning systems using simple components. It was derived from a biological brain neuron model introduced by McCulloch and Pitts in 1943. Later, Rosenblatt designed the Perceptron with

a view toward explaining and modeling pattern-recognition abilities of biological visual systems.
Units can be combined into a network in numerous fashions. Beyond any doubt, the most common of these is the Multilayer Perceptron (MLP) network. The basic MLP-network is constructed by ordering the units of layers, letting each unit in a layer take as input only the outputs of units in the previous layer or external inputs. If the network has two such layers of units, it is referred to as a two layer network, if in has three layers it is called a three layer network and so on.

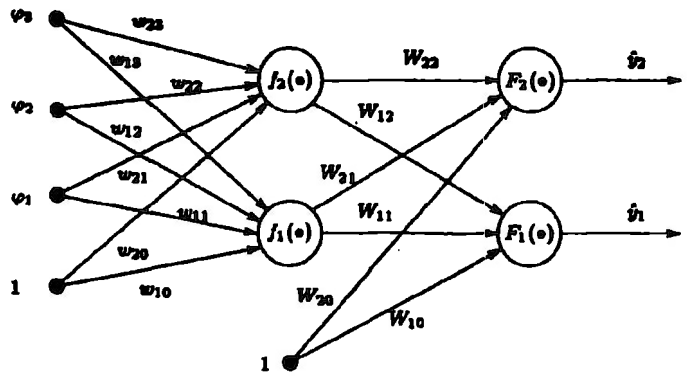


Figure 85 A fully connected two-layer feedforward network with three inputs, two hidden units and two outputs.

In the figure, the second layer is called the output layer referring to the fact that it produces the output of the network. The first layer is known as the hidden layer since it is in some sense hidden between the external inputs and the output layer. The depicted network is said to be fully connected since all inputs/all units in one layer are connected to all units in the following layer.
The mathematical formula expressing what is going on in the MLP-network takes the form

$$\hat{y}_i(t) = g_i[\varphi,\theta] = F_i\left[\sum_{j=1}^{n_h} w_{j,1} f_i\left(\sum_{l=1}^{n_\varphi} w_{j,l}\varphi_l + w_{j,0}\right) + w_{i,0}\right] \tag{4.11}$$

A.1.4.4. Why use Neural Networks?

Partly because it simplifies the modeling process itself, and also because it will enable implementation of generic tools for control system designs. When searching for a single technique tat in most cases of practical interest performs reasonably well, certain types of neural network appear to be an excellent choice. In particular, the Multilayer Perceptron Network has gained an immense popularity.

A.1.5. ANFIS

ANFIS stands for adaptive-network-based fuzzy inference system [3], is a hybrid method that uses the Sugeno and Tsukamoto fuzzy models with an adaptive network by a hybrid-learning rule.

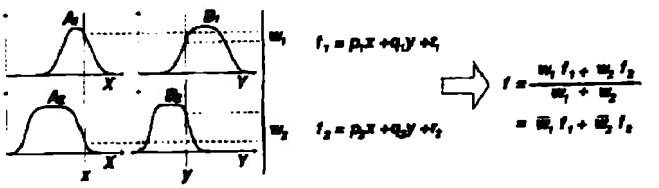
A.1.5.1. ANFIS Architecture

For simplicity, we assume that the fuzzy inference system under consideration has two inputs x and y and one output z. For a first-order Sugeno fuzzy model, a common rule set with two fuzzy if-then rules is the following:

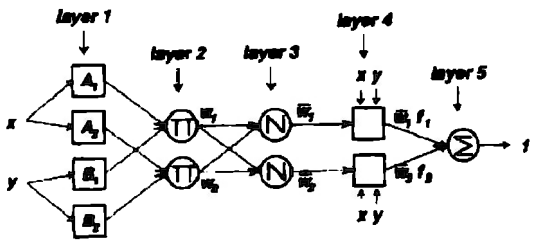
Rule 1: If x is A_1 and y is B_1, then $f_1=p_1x+q_1y+r_1$,

Rule 2: If x is A_1 and y is B_1, then $f_1=p_1x+q_1y+r_1$,

The next figure illustrates the corresponding equivalent ANFIS architecture of a Sugeno model, where nodes of the same layer have similar functions, as described next:



(a)



(b)

Layer 1. Every node i in this layer is an adaptive node with a node function

$$O_i^1 = \mu_{A_i}(x) \tag{5.1}$$

Where x is the input to node i and A_i is the linguistic label (small, large, etc.) associated with this node function. In other words, O_i^1 is the membership function of A_i and it specifies the degree to which the given x satisfies the quantifier A_i. Usually we choose $\mu_{A_i}(x)$ to be bell-shaped with maximum equal to 1 and minimum equal to 0, such as

$$\mu_{A_i}(x) = \frac{1}{1+\left[\left(\frac{x-c_i}{a_i}\right)^2\right]^{b_i}} \tag{5.2}$$

Or

$$\mu_{A_i}(x) = \exp\left\{-\left(\frac{x-c_i}{a_i}\right)^2\right\} \tag{5.3}$$

Where $\{a_i, b_i, c_i\}$ is the parameter set. As the values of these parameters change, the bell-shaped functions vary accordingly, thus exhibiting various forms of membership functions on linguistic label Ai. In fact, any continuous and piecewise differentiable functions, such as commonly used trapezoidal or triangular-shaped membership functions, are also qualified candidates for node functions in this layer. Parameters in this layer are referred to as premise parameters.

Layer 2. Every node in this layer is a circle node labeled Π, which multiplies the incoming signals and sends the product out. For instance,

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \quad i=1,2 \tag{5.4}$$

Each node output represents the firing strength of a rule.

Layer 3. Every node in this layer is a circle node labeled N. The *i*th node calculates the ratio of the *i*th rule's firing strength to the sum of all rule's firing strengths:

$$\overline{w}_i = \frac{w_i}{w_1 + w_2}, \quad i=1,2 \tag{5.5}$$

For convenience, outputs of this layer will be called normalized firing strengths.

Layer 4. Every node *i* in this layer is a square node with a node function

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i) \tag{5.6}$$

Where $\overline{w}_i$ is the output of layer 3, and $\{p_i, q_i, r_i\}$ is the parameter set. Parameters in this layer will be referred to as consequent parameters.

Layer 5. The single node in this layer is a circle node labeled Σ that computes the overall output as the summation of all incoming signals, i.e.,

$$O_i^5 = overall\_output = \sum \overline{w}_i f_i = \frac{\sum w_i f_i}{\sum w_i} \tag{5.7}$$

Thus, we have constructed an adaptive network, which is functionally equivalent to a fuzzy inference system.

A.1.5.2. Hybrid Learning Algorithm

From proposed architecture, it is observed that given the values of premise parameters, the overall output can be expressed as linear combinations of the consequent parameters. More precisely, the output *f* can be rewritten as:

$$\begin{aligned} f &= \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2 = \overline{w}_1 f_1 + \overline{w}_2 f_2 \\ &= (\overline{w}_1 x) p_1 + (\overline{w}_1 y) q_1 + (\overline{w}_1) r_1 + (\overline{w}_2 x) p_2 + (\overline{w}_2 y) q_2 + (\overline{w}_2) r_2 \end{aligned} \tag{5.8}$$

Which is linear in the consequent parameters (p_1, q_1, r_1, p_2, q_2 and r_2).

Then, we can apply back propagation or steepest descent to identify the parameters in an adaptive network. We may observe that an adaptive network's output (assuming there is only one) is linear in some of the network's parameters; thus, we can identify these linear parameters by the linear least-squares method. This approach leads to a hybrid-learning

rule, which combines steepest descent and the least-squares estimator for fast identification of parameters.

A.1.6. System Identification with Neural Networks

System identification is the task of inferring a mathematical description, a model, of a dynamic system from a series of measurements on the system. There can be several motives for establishing mathematical descriptions of dynamic systems. Typical applications encompass simulation, prediction, fault detection, and control system design. When attempting to identify a model of a dynamical system it is common to practice to follow the procedure depicted in the next figure.

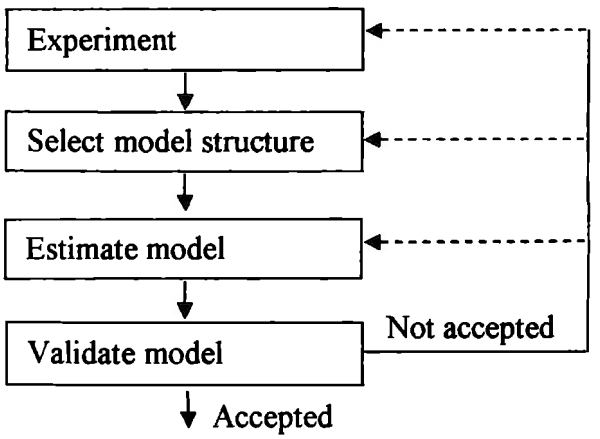


Figure 87 System Identification Process

Some model structures are:
FIR (Finite Impulse Response model structure)/NNFIR: past control inputs
ARX (AutoRegressive, external input)/NNARX: past control inputs and observed outputs.
OE (Output Error model structure)/NNOE: past control inputs and output predictions.
ARMAX (AutoRegressive, Moving Average, external input)/NNARMAX: past control inputs, output predictions, and residuals.
SSIF (State Space Innovations Form)/NNSSIF: past input, state estimate, and residual.

As a rule of thumb, the model structure should be chosen so that the ratio between the number of training data and the weights in the network, N/p, is between 2 and 7.

A.1.7. Control Design with Neural Networks

Control of nonlinear systems is a major application area for neural networks. The control design problem will be approached in two ways: direct and indirect design methods.
Direct design means that a neural network directly implements the controller. A neural network controller is often advantageous when real-time platform available prohibits complicated solutions. The implementation is simple while the design and tuning are

difficult implying a retraining of the network every time a design parameter is modified. Often this training has to be performed according to an on-line scheme. There are:

- Direct inverse control (including training of inverse models).
- Internal Mode Control
- Feedback linearization
- Feedforward with inverse models.
- Optimal control.

Later on, we will analyze the Direct inverse control and something of Internal Mode Control.

Indirect methods are based on a neural network model of the system to be controlled, in this case the controller is not itself a neural network. The model is then employed in a more "conventional" controller design. The model is typically trained in advance, but the controller is designed on-line. As it will appear, the indirect design is very flexible; thus, it is the most appropriate for the majority of common control problems. There are:

- Approximate pole placement, minimum variance, and predictive control.
- Nonlinear predictive control.

Here, we won't analyze these indirect methods.

A.1.7.1. Direct Inverse Control

The basic principle is as follows:
Assuming that the system to be controlled can be described by

$$y(t+1) = g[y(t),...,y(t-n+1),u(t),...,u(t-m)] \tag{7.1}$$

The desired network is then the one that isolates the most recent control input, *u(t)*,

$$\hat{u}(t) = g^{-1}[y(t+1),y(t),...,y(t-n+1),u(t-1),...,u(t-m)] \tag{7.2}$$

Assuming such a network has somehow been obtained, it can be used for controlling the system by substituting the output at time t+1 by the desired output, the reference, *r(t+1)*. If the network represents the exact inverse, the control input produced by it will thus drive the system output at time *t+1* to *r(t+1)*. As illustrate:

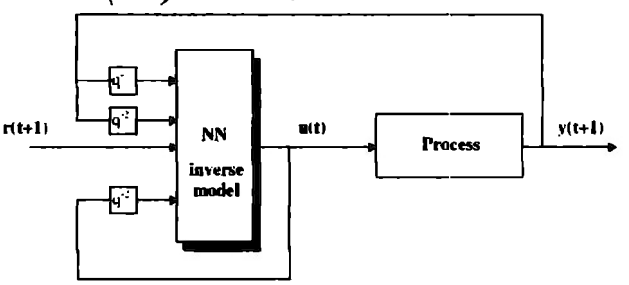


Figure 88 Direct Inverse Control

If the inverse model is unstable, one must anticipate that the closed-loop system becomes unstable. Unfortunately, this situation occurs quite frequently in practice. Another problem with the design arises when the system to be controlled is not one-to-one since then a unique inverse model does not exist. Most often, however, one will end up with useless inverse model. There is also a problem in relation to training the network on

realistic signals, basically the experiment data, this is referred to as identification for control problem, where a priori it is unclear what realistic data means.
As a characteristic of this control, although it manages to control the system quite well, the control signal appears to be extremely active and assumes very large values; a simple low-pass filtering of the reference solves this.
The Direct Inverse Control method can be obtained by General Training, that just train the network as the inverse of a system, and Specialized Training, where it learn of more than just the model, as shown in the next figure.

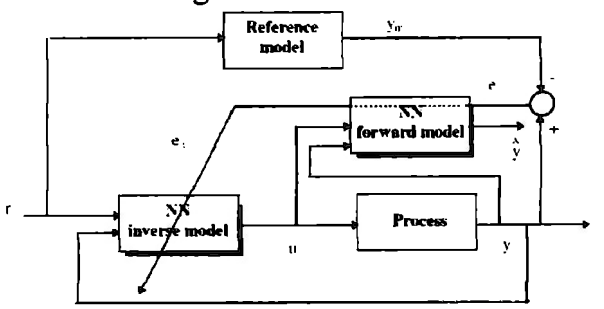


Figure 89 Specialized training principle

A.1.7.2. Internal Model Control

Internal model control is a control design closely connected to direct inverse control. An internal model controller requires a forward model as well as a model of the inverse of the system to be controlled. As shown in the figure.

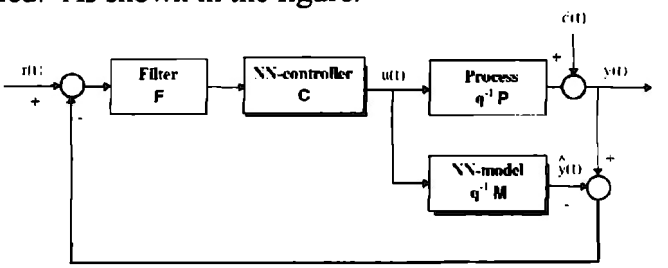


Figure 90 Internal Model Control

In contrast to direct inverse control, the feedback is not composed directly of the system's output. Instead, the error between system output and model output is feedback.
A condition for global stability of the closed-loop system is that the system to be controlled and the inverse model are both stable. Then a restrictive requirement is that the system must be open-loop stable.

A.1.8. Genetic Algorithm Theory

Genetic Algorithm are derivative-free stochastic optimization methods based loosely on the concepts of natural selection and evolutionary processes. They were first proposed and investigated by John Holland at the University of Michigan in 1975.

GAs encode each point in a parameter (or solution) space into a binary bit string called a chromosome, and each point is associated with a "fitness" value that, for maximization, is usually equal to the objective function evaluated at the point. Instead of a single point, GAs usually keep a set of points as a population, which is then evolved repeatedly toward a better overall fitness value. In each generation, the GA constructs a new population using genetic operators, such as crossover and mutation; members with higher fitness values are more likely to survive and to participate in mating (crossover) operation. After a number of generations, the population contains members with better fitness values.

Major components of GAs include encoding schemes, fitness evaluations, parent selection, crossover operators, and mutation operators; these are explained next.

Encoding schemes. These transform points in a parameter space into bit string representations. For instance, a point (11,6,9) in a three-dimensional parameter space can be represented as a concatenated binary string:

1011 0110 1001
11 6 9

Fitness evaluation. The first step after creating a generation is to calculate the fitness value of each member in the population. For a maximization problem, the fitness value f_i of the ith member is usually the objective function evaluated at this member (or point).

Selection. After evaluation, we have to create a new population from the current generation. The selection operation determines which parents participate in producing offspring for the next generation, and it is analogous to survival of the fittest in natural selection.

Crossover. To exploit the potential of the current gene pool, we use crossover operators to generate new chromosomes that we hope will retain good features from the previous generation. Crossover is usually applied to selected pairs of parents with a probability equal to a given crossover rate.

Mutation. Crossover exploits current gene potentials, but if the population does not contain all the encoded information to solve a particular problem, no amount of gene mixing can produce satisfactory solution. For this reason, a mutation operator capable of spontaneously generating new chromosomes is included. The most common way of implementing mutation is to flip a bit with a probability equal to a very low given mutation rate. It can prevent the population from converging and stagnating at any local optima.

The next figure shows the procedure of producing the next generation in GAs.

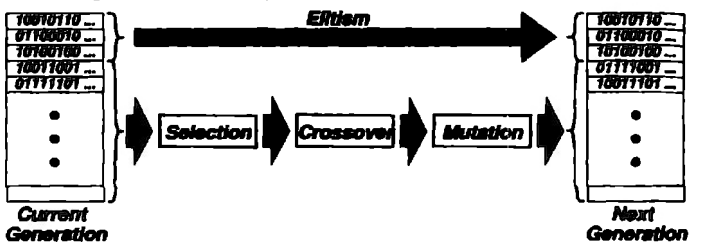


Figure 91 Producing the next generation in GAs.

These techniques rely on modern high-speed computers; require a significant amount of computation when compared with derivative-based approaches. However, these derivative-free approaches are more flexible in terms of incorporating intuitive guidelines and forming sophisticated objective functions.

A.2. "Modelo del sistema".

A.2.1. Modelo no lineal del sistema

Ecuación Euler - Lagrange

$$\frac{\partial}{\partial t}\frac{\partial \lambda}{\partial \dot{q}} - \frac{\partial \lambda}{\partial q} = \tau - Fricción$$

$\lambda = k_{TOTAL} - T_{TOTAL}$ (Diferencia de Energías)

$k_{TOTAL} = k_1 + k_2$ $\qquad$ $T_{TOTAL} = T_1 + T_2$

$k_1 = \frac{1}{2} m_1 v_1^{\,2} = \frac{1}{2} m_1 \dot{x}^2$ $\qquad$ $T_1 = mgl$

$k_2 = \frac{1}{2} m_2 v_2^{\,2} + \frac{1}{2} I\omega^2$ $\qquad$ $T_2 = m_2 g(l + l\cos\theta)$

(Por movimiento general de cuerpos rígidos)

$CM = (x + l_2 \sin\theta)i + (l_2 \cos\theta)j$ $\qquad$ $T_{TOTAL} = m_1 gl + m_2 g(l + l_2 \cos\theta)$

$v_2 = (\dot{x} - \dot{\theta} l_2 \cos\theta)i + (l_2 \dot{\theta} \sin\theta)j$

$v_2^{\,2} = \dot{x}^2 - 2\dot{x}\dot{\theta} l_2 \cos\theta + \dot{\theta}^2 l_2^{\,2} \cos^2\theta + l_2^{\,2}\dot{\theta}^2 \sin^2\theta$

$v_2^{\,2} = \dot{x}^2 - 2\dot{x}\dot{\theta} l_2 \cos\theta + \dot{\theta}^2 l_2^{\,2}$

$k_2 = \frac{1}{2} m_2 \dot{x}^2 - m_2 \dot{x}\dot{\theta} l_2 \cos\theta + \frac{1}{2}\dot{\theta}^2 l_2^{\,2} + \frac{1}{2} I\dot{\theta}^2$

$k_{TOTAL} = \frac{1}{2} m_1 \dot{x}_2^{\,2} + \frac{1}{2} m_2 \dot{x}^2 - m_2 \dot{x}\dot{\theta} l_2 \cos\theta + \frac{1}{2}\dot{\theta}^2 l_2^{\,2} + \frac{1}{2} I\dot{\theta}^2$

$\lambda = \frac{1}{2} m_1 \dot{x}^2 + \frac{1}{2} m_2 \dot{x}^2 - m_2 \dot{x}\dot{\theta} l_2 \cos\theta + \frac{1}{2} m_2 \dot{\theta}^2 l_2^{\,2} + \frac{1}{2} I\dot{\theta}^2 - m_1 gl - m_2 gl - m_2 gl_2 \cos\theta$

$\frac{\partial \lambda}{\partial \theta} = -m_2 l_2 \dot{x}\dot{\theta}\cos\theta - m_2 gl_2 \cos\theta$

$\frac{\partial \lambda}{\partial x} = 0$

$\frac{\partial \lambda}{\partial \dot{\theta}} = m_2 l_2 \dot{x}\cos\theta + m_2 l_2 \dot{\theta} + I\dot{\theta}$

$\frac{\partial \lambda}{\partial \dot{\theta}} = -m_2 \dot{x} + m_2 \dot{x} + m_2 l_2 \dot{\theta}\cos\theta$

$\frac{\partial}{\partial t}\frac{\partial \lambda}{\partial \dot{\theta}} = -m_2 \dot{x} l_2 \cos\theta + m_2 l_2^2 \dot{\theta} + I\dot{\theta}$

$\frac{\partial}{\partial t}\frac{\partial \lambda}{\partial \dot{x}} = m_1 \dot{x} + m_2 \dot{x} - m_2 l_2 \ddot{\theta}\cos\theta$

$$\frac{\partial}{\partial t}\frac{\partial \lambda}{\partial q}-\frac{\partial \lambda}{\partial q}=\begin{bmatrix}-m_2 \ddot{x} l_2 \cos\theta + m_2 l_2^2 \ddot{\theta} + I\ddot{\theta} + m_2 \ddot{x} l_2 \theta \sin\theta + m_2 g l_2 \sin\theta \\ m_1 \ddot{x} + m_2 \ddot{x} - m_2 l_2 \cos\theta\end{bmatrix}$$

$$\frac{\partial}{\partial t}\frac{\partial \lambda}{\partial q}-\frac{\partial \lambda}{\partial q}=\begin{bmatrix}-m_2 \ddot{x} l_2 \cos\theta + (m_2 l_2^2 + I)\ddot{\theta} + m_2 l_2 \sin\theta(\dot{x}\dot{\theta} + g) \\ (m_1 + m_2)\ddot{x} - m_2 l_2 \cos\theta\ddot{\theta}\end{bmatrix}$$

$$\tau - Fricción = \begin{vmatrix} v_1 - \beta_1 \dot{\theta} \\ v_2 \quad - \beta_2 \dot{x}\end{vmatrix}$$

$$\begin{bmatrix}-m_2 \ddot{x} l_2 \cos\theta + (m_2 l_2^2 + I)\ddot{\theta} + m_2 l_2 \sin\theta(\dot{x}\dot{\theta} + g) \\ (m_1 + m_2)\ddot{x} - m_2 l_2 \cos\theta\ddot{\theta}\end{bmatrix} = \begin{vmatrix} 0 - \beta_1 \dot{\theta} \\ v_2 \quad - \beta_2 \dot{x}\end{vmatrix}$$

$$\begin{bmatrix}-m_2 \ddot{x} l_2 \cos\theta + (m_2 l_2^2 + I)\ddot{\theta} + m_2 l_2 \sin\theta(\dot{x}\dot{\theta} + g) + \beta_1 \dot{\theta} \\ (m_1 + m_2)\ddot{x} - m_2 l_2 \cos\theta\ddot{\theta} + \beta_2 \dot{x}\end{bmatrix} = \begin{vmatrix} 0 \\ v_2 \end{vmatrix}$$

Matriz de Inercia:

$$\begin{bmatrix} m_2 l_2^2 + I & -m_2 l_2 \cos\theta \\ -m_2 l_2 \cos\theta & (m_1 + m_2)\end{bmatrix}$$

Matriz de Fricción (β)

$$\beta = \begin{bmatrix}\beta_1 & 0 \\ 0 & \beta_2\end{bmatrix}$$

Término de Gravedad

$m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g)$

La suma de la matriz B y g es la matriz h

$$M(q)\ddot{q} + q\beta + g(q) = \begin{bmatrix} 0 \\ v_2 \end{bmatrix}$$

Incluyendo la del motor

$v_2 = k_e r V_{(t)} - J\ddot{x} - \beta_3 \dot{x}$

$$M(q)\ddot{q} + q\beta + g(q) = \begin{bmatrix} 0 \\ k_e r V_{(t)} - J\ddot{x} - \beta_3 \dot{x}\end{bmatrix}$$

$$\left[M(q) + \begin{bmatrix}0 & 0 \\ 0 & j\end{bmatrix}\right]\ddot{q} + \left[\beta + \begin{bmatrix}0 & 0 \\ 0 & \beta_3\end{bmatrix}\right]q + g(q) = k_e r \begin{bmatrix}0 \\ V\end{bmatrix}$$

Resolviendo:

$$\begin{bmatrix}(m_2 l_2^2 + I)\ddot{\theta} - m_2 l_2 \ddot{x} \sin\theta + m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g) + \beta_1 \dot{\theta} \\ -m_2 l_2 \ddot{\theta} \sin\theta + (m_1 + m_2 + J)\ddot{x} + (\beta_2 + \beta_3)\dot{x}\end{bmatrix} = \begin{bmatrix} 0 \\ k_e r V\end{bmatrix}$$

Ahora simplificaremos, porque se asume que no hay fricción y no se incluye la modelación del motor.

Partiendo de:

$$M(q)\ddot{q} + q\beta + g(q) = \begin{bmatrix} 0 \\ v_2 \end{bmatrix}$$

$$\begin{bmatrix} m_2 l_2^2 + I & -m_2 l_2 \cos\theta \\ -m_2 l_2 \cos\theta & (m_1 + m_2) \end{bmatrix} \ddot{q} + m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g) = \tau$$

Se despeja para $\ddot{q}$, y se obtiene:

$$\ddot{q} = \begin{pmatrix} \ddot{\theta} \\ \ddot{x} \end{pmatrix} = \begin{bmatrix} (m_1 + m_2) & m_2 l_2 \cos\theta \\ m_2 l_2 \cos\theta & m_2 l_2^2 + I \end{bmatrix} \frac{1}{(m_1 + m_2) * (m_2 l_2^2 + I) - m_2 l_2 \cos\theta} (\tau - m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g))$$

Y encontramos:

$$\ddot{\theta} = \frac{-(m_1 + m_2) g \sin\theta + \cos\theta(\tau + m_2 l \dot{\theta}^2 \sin\theta)}{l[m_2 \cos\ \theta - (m_1 + m_2)]}$$

$$\ddot{x} = \frac{m_2 g \sin\theta \cos\theta - (\tau + m_2 l \dot{\theta} \sin\theta)}{m_2 \cos^2\theta - (m_1 + m_2)}$$

Y reordenando, encontramos las ecuaciones presentadas en la sección 2:

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta\left(\frac{-\tau - m_2 l \dot{\theta}^2 \sin\theta}{m_1 + m_2}\right)}{l\left(\frac{4}{3} - \frac{m_2 \cos^2\theta}{m_1 + m_2}\right)}$$

$$\ddot{x} = \frac{\tau + m_2 l(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)}{m_1 + m_2}$$

A.2.2. Modelo lineal del sistema

Linealizamos para un punto de operación, que en este caso está dado en 0° para el ángulo y 0m. para la posición.

Ec. De Estado:

$$x = \begin{pmatrix} q \\ \dot{q} \end{pmatrix} \rightarrow x = \begin{pmatrix} \theta \\ x \\ \dot{\theta} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Ec. Espacio Estado

$$\dot{x} = \begin{pmatrix} \dot{q} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} x_3 \\ x_4 \\ M^{-1}(\bar{u} - \bar{h}) \end{pmatrix}$$

Donde M es la matriz de Incidencia:

$$M = \begin{bmatrix} m_2(l_2^2 + I) & -m_2 l_2 \sin\theta \\ -m_2 l_2 \sin & (m_1 + m_2 + J) \end{bmatrix}$$

$\bar{u}$ es el voltaje de entrada motriz

$$\bar{u} = \begin{pmatrix} 0 \\ k_e r V \end{pmatrix}$$

$\bar{h}$ (Término de Coriollis y Gravedad)

$$\bar{h} = \begin{bmatrix} \beta_1\dot{\theta} + m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g) \\ (\beta_2 + \beta_3)\dot{x} \end{bmatrix}$$

$$(\bar{u} - \bar{h}) = \begin{bmatrix} -\beta_1\dot{\theta} - m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g) \\ k_e r V - (\beta_2 + \beta_3)\dot{x} \end{bmatrix}$$

$$M^{-1} = \frac{adj(M)}{(M)} \qquad adj(M) = \begin{bmatrix} (m_1 + m_2 + J) & -m_2 l_2 \sin\theta \\ -m_2 l_2 \sin\theta & (m_2 l_2^2 + I) \end{bmatrix}$$

$$(M) = (m_2 l_2^2 + I)(m_1 m_2 + J) - (m_2 l_2 \sin\theta)^2$$

$$M^{-1}(\bar{u} - \bar{h}) = \begin{bmatrix} \frac{(m_1 + m_2 + J)(-\beta_1\dot{\theta} - m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g)}{(m_2 l_2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} + \frac{(-m_2 l_2 \sin\theta)(k_e r V - (\beta_2 + \beta_3)\dot{x})}{(m_2 l_2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} \\ \frac{(-m_2 l_2 \sin\theta)(-\beta_1\dot{\theta} - m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g)}{(m_2 l_2^2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} + \frac{(m_2 l_2^2 + I)(k_e r V - (\beta_2 + \beta_3)\dot{x})}{(m_2 l_2^2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} \end{bmatrix}$$

$$f(x)\dot{x} = \begin{bmatrix} x_3(\dot{\theta}) \\ x_4(\dot{x}) \\ \frac{(m_1 + m_2 + J)(-\beta_1\dot{\theta} - m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g))}{(m_2 l_2^2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} + \frac{(-m_2 l_2 \sin\theta)(k_e r V - (\beta_2 + \beta_3)\dot{x})}{(m_2 l_2^2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} \\ \frac{(-m_2 l_2 \sin\theta)(-\beta_1\dot{\theta} - m_2 l_2 \cos\theta(\dot{x}\dot{\theta} + g))}{(m_2 l_2^2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} + \frac{(m_2 l_2^2 + I)(k_e r V - (\beta_2 + \beta_3)\dot{x})}{(m_2 l_2^2 + I)(m_1 + m_2 + J) - (m_2 l_2 \sin\theta)^2} \end{bmatrix}$$

Linealización tomando como referencia el punto de equilibrio $q^* = \begin{pmatrix} 0 \\ x \end{pmatrix}$

el cual lo usamos como origen de nuestro marco de referencia $(z, \dot{z})$ donde:

$z = x - x^*$

$\dot{z} = \dot{x} - \dot{x}^*$

Utilizamos series de Taylor como herramienta de linealización

$$\dot{x} = f(x, v) = f(x^*, V_0) + \left.\frac{\partial f}{\partial v}\right|_{x^*} (x - x^*) + \left.\frac{\partial f}{\partial V}\right|_{V_0} (V - V_0)$$

$f(x^*, V_0) = 0$ por condiciones iniciales igual a 0

Deteniendo:

$$A = \left.\frac{\partial f}{\partial x}\right|_{x^*} ; B = \left.\frac{\partial f}{\partial V}\right|_{V_0}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{(m_1+m_2+J)(-gm_2l_2)}{a-(m_2l_2)^2} & 0 & \frac{-\beta_1(m_1+m_2+J)}{a-(m_2l_2)^2} & \frac{m_2l_2(\beta_2+\beta_3)}{a-(m_2l_2)^2} \\ 0 & 0 & \frac{\beta_1 m_2 l_2}{a-(m_2l_2)^2} & \frac{-(m_2l_2^2+I)(\beta_2+\beta_3)}{a-(m_2l_2)^2} \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{-m_2l_2K_e r}{a-(m_2l_2)^2} \\ 0 & \frac{(m_2l_2^2+I)K_e r}{a-(m_2l_2)^2} \end{bmatrix}$$ dónde

$a = (m_1 + m_2 + J)(m_2 l_2^2 + I)$

$C = [1 \quad 0 \quad 0 \quad 0]$ Matriz de salida

$D = [0 \quad 0]$ Matriz de acoplamiento

A.3. "NNCTRL".

En este anexo se presentan en inglés los intentos de control usando redes neuronales, presentando algunos resultados fallidos pero haciendo énfasis en las capacidades y uso de diferentes herramientas para controlar con redes neuronales. El texto se presenta en inglés porque fue realizado como parte de proyectos de ingeniería I, en un programa de intercambio internacional en la Tampere University of Technology, en Finlandia [39].

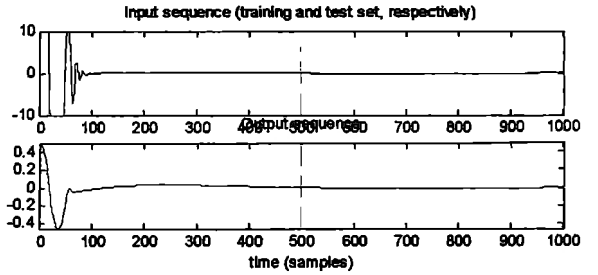
A.3.1. NNSYSID and NNCTRL Toolbox [11,12]

To reduce the risk that the error was on the way of creating the Neural Networks, or the experiment, or the Direct Control methodology, or any other mistake, I decided to try with the NNSYSID and NNCTROL Toolbox.

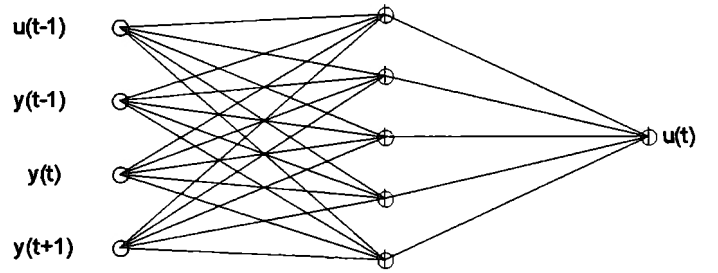
Magnus Norgaard, of the Department of Automation-Mathematical Modeling, of the Technical University of Denmark, created this Toolbox. Is free on the Internet, and consists on several script files and templates based on Neural Networks, especially MLP. The main problem is that is mostly dedicated to Single Output system's.

After learning how to use this Toolbox, I made the Direct Control approach for the Inverted Pendulum, modifying the invtest.m, invinit.m and invinit2.m files as seen on the Appendix 3.

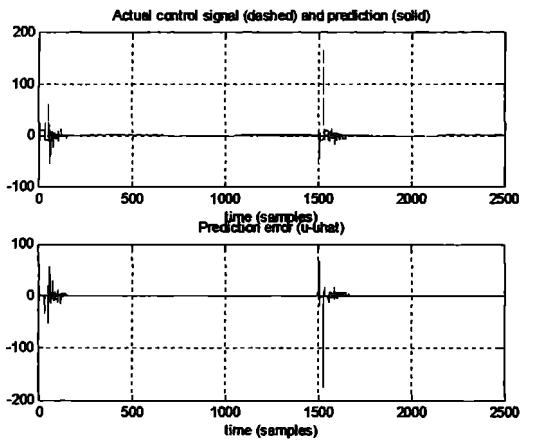
We began by importing the set of data of the experiment, based on the Robust Controller, as shown before. The data is:

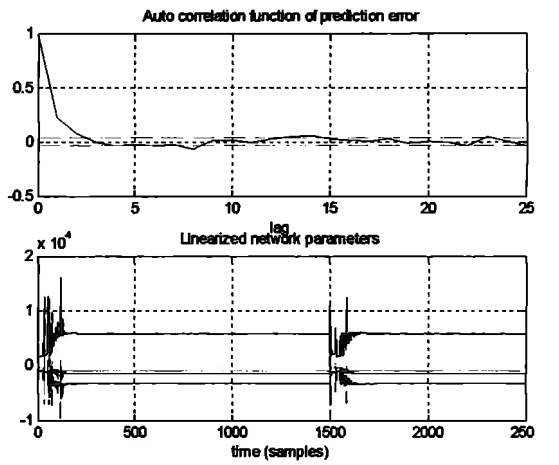


We will now train an inverse neural network model of the process by general training. Since it's a 2nd order process, we will use the regressor structure shown in the figure, with 5 hidden 'tanh' units and one linear output unit. The function 'general' of the Toolbox implements general training with a Levenberg-Marquardt algorithm.

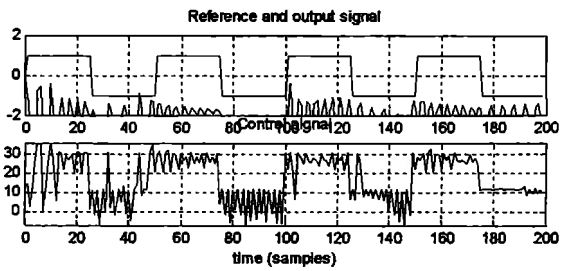


The network is then trained and then validated with the last data set (2500 data were used for training and the last 2500 for validation).

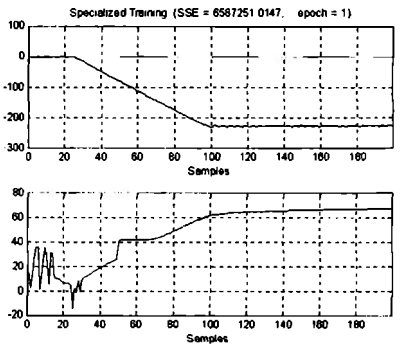




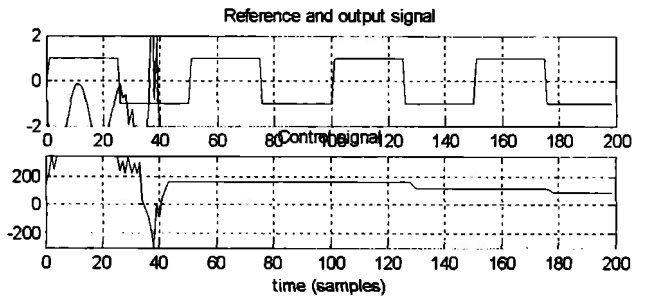
The auto-correlation shows pretty well results. Then we save the trained network to a file, so that it can be used as a controller in a Direct Inverse control scheme.
With the NNCTRL Toolbox, we realize this Direct Inverse Control, obtaining:



From the plot we can see that the output signal is far from the reference, and also that the control signal is very active and with bad accuracy on the final response. To try to fine-tune the network directly on the square wave reference trajectory we will use specialized training.
Before continuing with specialized training, it is necessary to identify a "forward" model of the process. The specialized training algorithm requires this model. We used the function "nnarx" from the NNSYSID-toolbox for this. The model is inferred from the same data set as was used for training the inverse model. The function "special2" was used for performing the specialized training. We choose a recursive Gauss-Newton algorithm with exponential forgetting for updating the weights.
The reference trajectory (consisting of 200 samples) was repeated eight times while updating the weights.



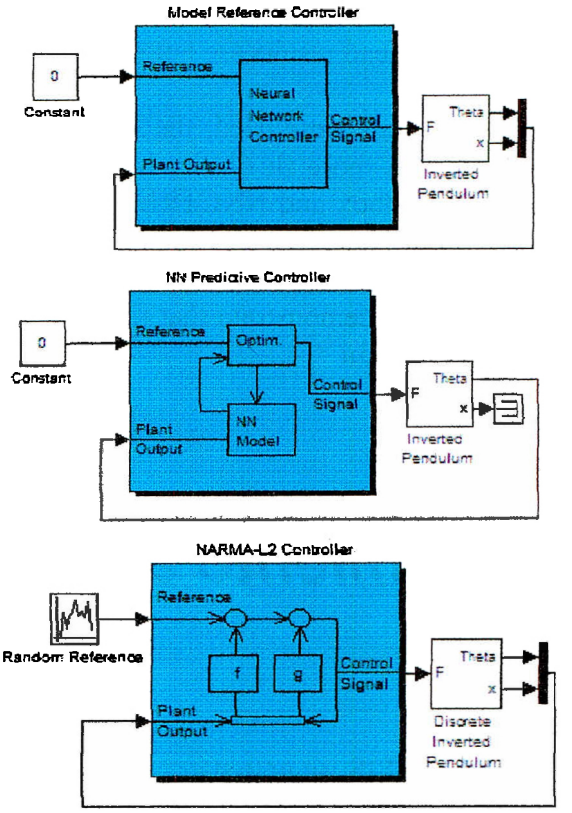
The plot shows the training, it's supposed that should be close to zero. The error is very high. To finalize this failed attempt, we will simulate the entire closed-loop system one more time.



Again, for unknown causes, and after lot of other tries, this methodology didn't work as it should.

A.3.2. Neural Network Toolbox in Simulink

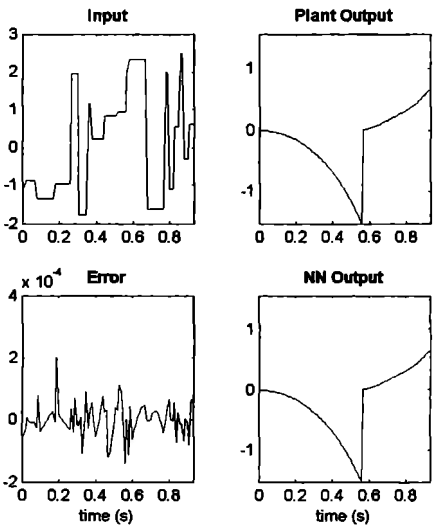
Finally, I also tried with the Neural Network Toolbox in Simulink, with the following controllers:



In all methodologies first we have to obtain the Plant Model, here we used the same training data.

Then we have to choose the size of the Hidden layer, the Simulink Plant model (with inport and outport blocks), de number of delayed controller and plant outputs, training function, import training data, normalize or no the training data, and decide if we want to use the validation/Testing data for training.

In all the methodologies, I obtained good plant identification, for example:



But although I tried lots of different options to have a good controller, all the results were terrific. To avoid bad feelings, those bad results are not showed here.

As a summary, three different systems were used; the first one with the Neural Network Toolbox on the command line, with more flexibility but not controlling the system; the second one with the NNSYSID Toolbox and NNCTRL Toolbox, with less flexibility and bad results; and the last was the Neural Network Toolbox from Simulink, with even less flexibility, and worse results.
Despite I couldn't, several articles do demonstrate that the application of Neural Networks as a controller is possible and very useful.

A.4. "Entrenamiento a controlador ANFIS".

Se crearon varios controladores ANFIS, se entrenaron con diferentes datos, y se corrieron en el simulador.
La organización con que se presentan en el artículo es la siguiente:

1. Control con entrenamiento de ángulo y posición en asecuencia de tiempo de 8 entradas.
    a. Entrenamiento aleatorio
    b. Sub agrupamiento ("sub clustering")
    c. Mallado ("grid partition")
2. Control con entrenamiento de estado en secuencia de tiempo de 8 entradas.
3. Control con entrenamiento de estado de 4 entradas.
    a. Control diferencial en función de k
    b. Continuo con pocos datos de entrenamiento.
    c. Continuo con múltiples datos de entrenamiento.

En éste anexo se presentan las pruebas no relevantes ni significativas de los controladores faltantes. Específicamente, se cubre el tipo de control con entrenamiento de estado de 4 entradas.

A.4.1. Control con entrenamiento de estado de 4 entradas.

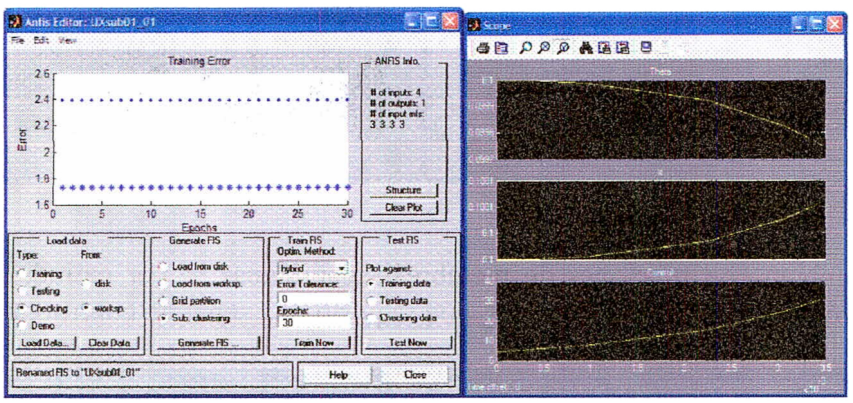
Es el control más sencillo, pues se entrena únicamente con el estado actual, y el controlador aprende de los datos la forma en que se debe estabilizar el sistema.

A.4.1.1. Control diferencial en función de k.

Se probó otro sistema con los siguientes parámetros:

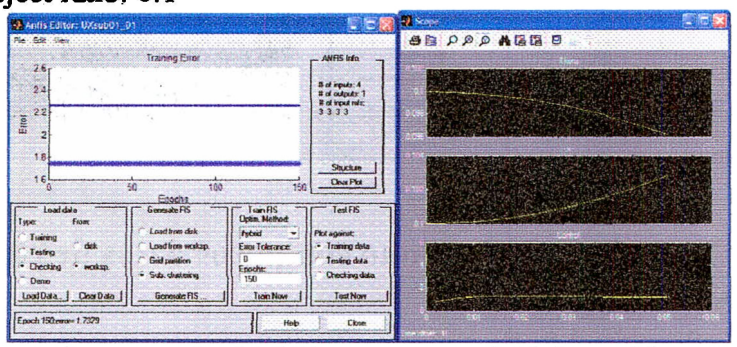
- Rango de influencia: 0.1
- Squash factor: 1.5
- Accept ratio: 0.5
- Reject ratio: 0.1

El error de entrenamiento es de 1.73, pero se volvió inestable y marcó error en la simulación



Por último, se intentó con los siguientes parámetros:

- Rango de influencia: 0.2
- Squash factor: 1.05
- Accept ratio: 0.2
- Reject ratio: 0.1

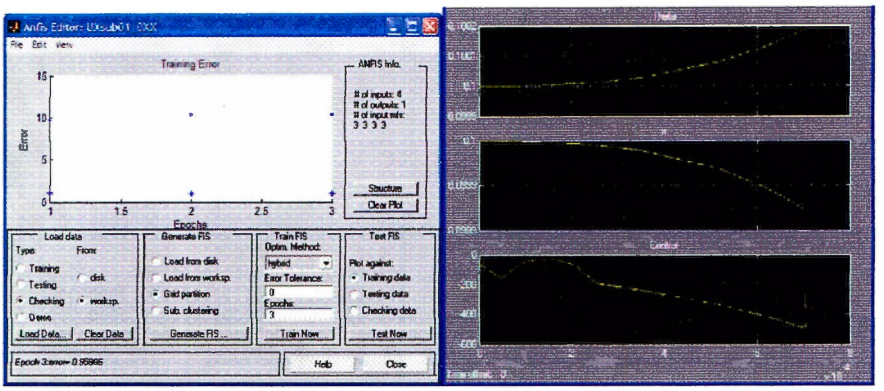


El error de entrenamiento es de 1.7379, pero se volvió inestable y marcó error en la simulación

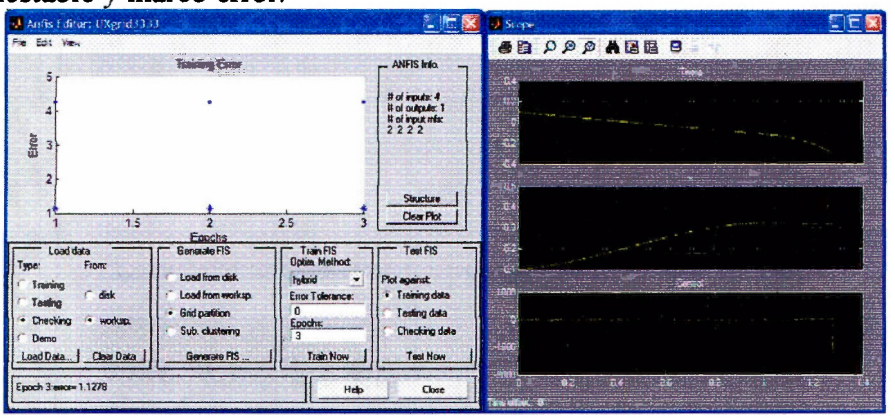
- Mallado

Se probó después de algunos entrenamientos, que con realizar 3 iteraciones se alcanzaba el mismo error, y aunque no se tiene la certeza de que se trabaje en un mínimo local, sí es representativo del controlador para menos de 50 iteraciones (que en mi computadora significa un tiempo de entrenamiento de 7 horas, dependiendo de la complejidad del sistema). Se presentan algunas pruebas realizadas.

Con funciones de membresía gaussianas, 3 por entrada, y una salida lineal, se entrenó con un error de 0.95806 . El sistema se volvió inestable y marcó error en la simulación



Se intentó también con funciones de membresía gaussianas, 2 por entrada, y una salida lineal, se entrenó con un error de 1.1278, que requirió una señal de control tan elevada que se volvió inestable y marcó error.

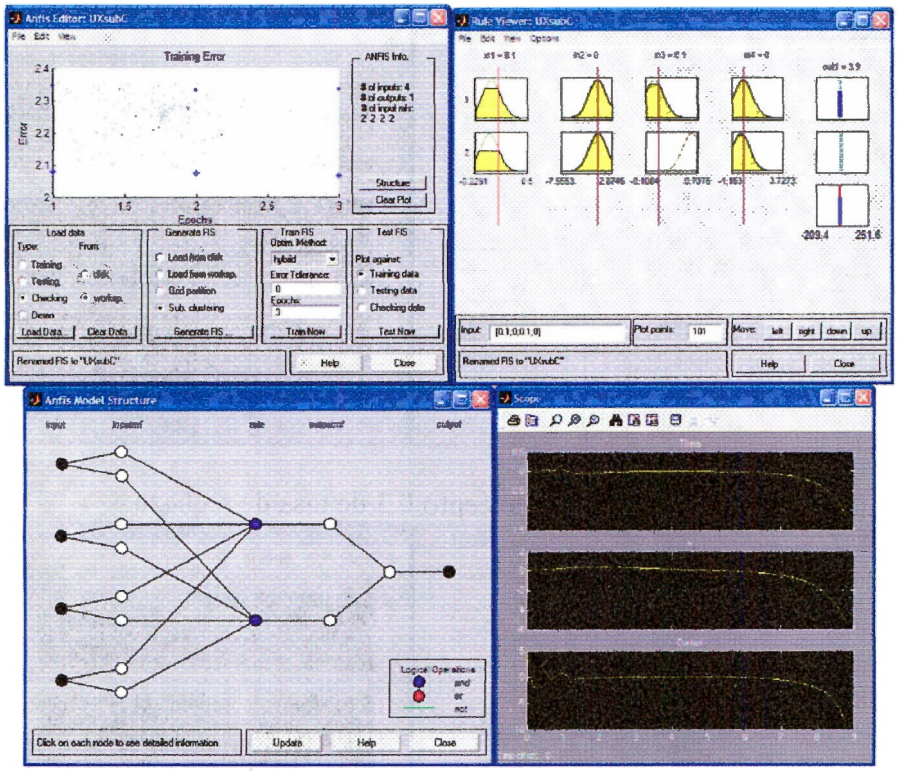


A.4.1.2. ESTADO Continuo con pocos datos de entrenamiento.

Se intentó con Sub Clustering, con

- Rango de influencia: 0.5
- Squash factor: 1.5
- Accept ratio: 0.5
- Reject ratio: 1.5

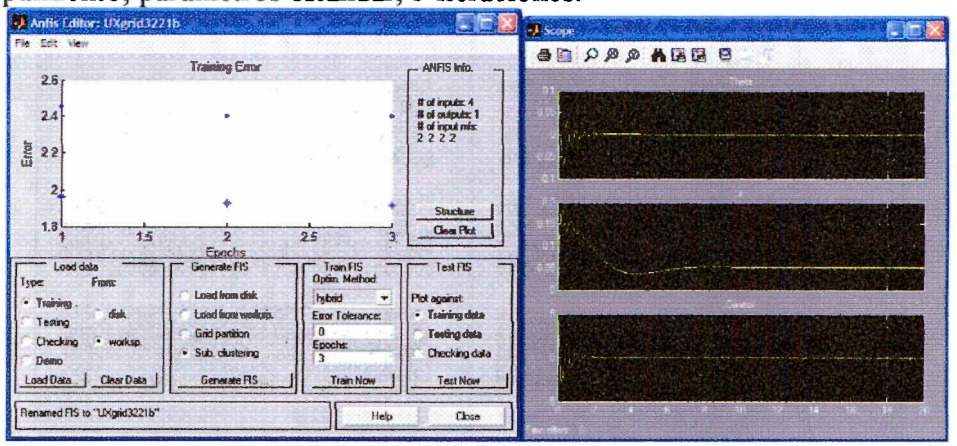
Y tres iteraciones; se genera una red como se muestra en las siguientes figuras:



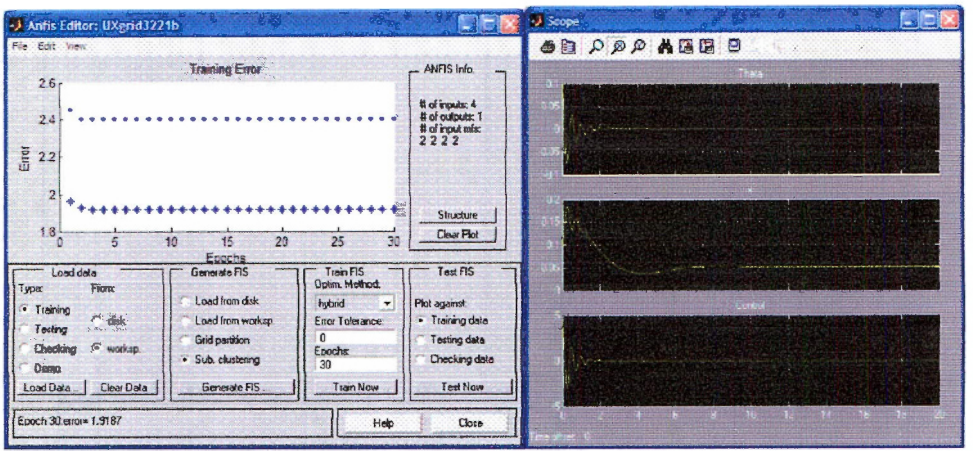
Pero no funcionó como debiera, pues aunque inicialmente sí controla, no es completamente estable y se descontrola después de 8 segundos.

A.4.1.3. Continuo con múltiples datos de entrenamiento.

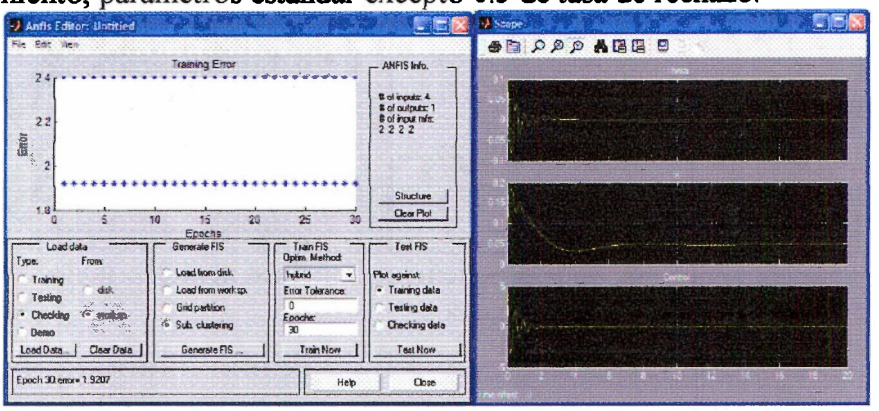
Sub agrupamiento, parámetros estándar, 3 iteraciones.



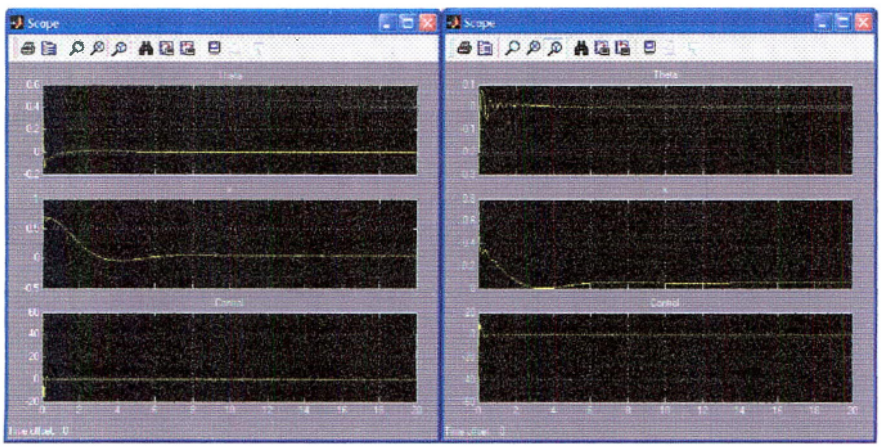
Sub agrupamiento, parámetros estándar y 30 iteraciones.



Sub agrupamiento, parámetros estándar excepto 0.3 de tasa de rechazo.

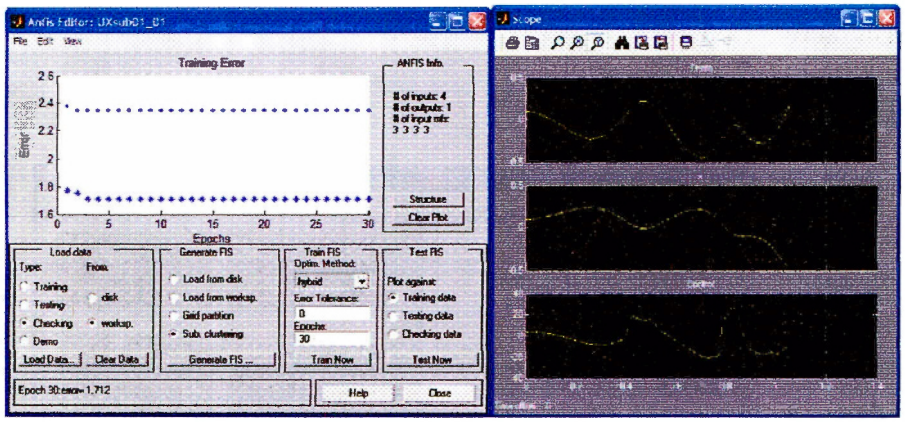


Se variaron las condiciones iniciales, ahora con 0.3 rad y 0.5 m., con el mismo control anterior.

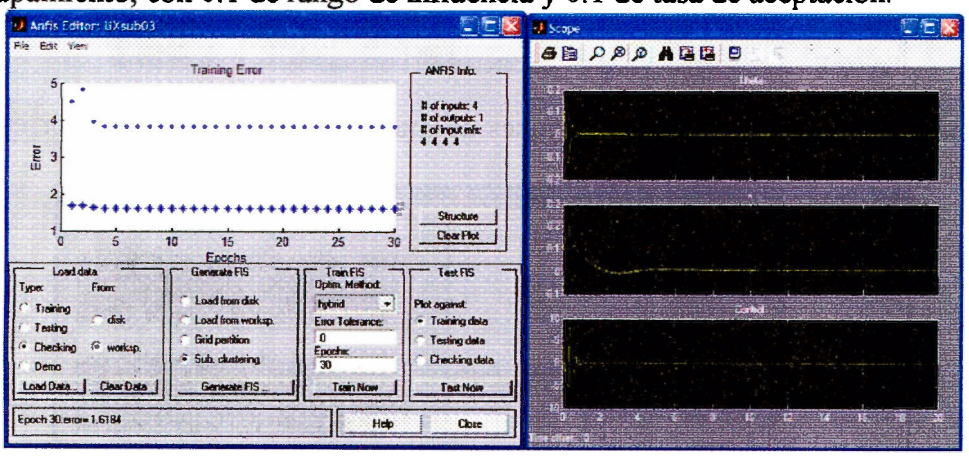


Se probó también para condiciones iniciales negativas, de -0.3 radianes y 0.5m.
En otros casos no converge a cero, sino a otro valor, con error de estado estable.

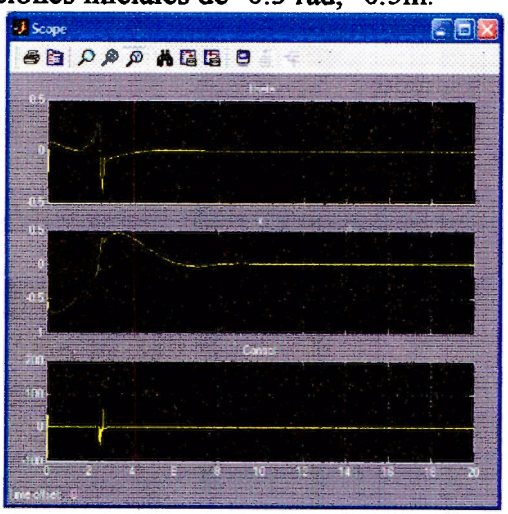
Nuevamente se probó sub agrupamiento, ahora con 0.1 de rango de influencia y 0.1 de tasa de rechazo.



Sub agrupamiento, con 0.1 de rango de influencia y 0.1 de tasa de aceptación.

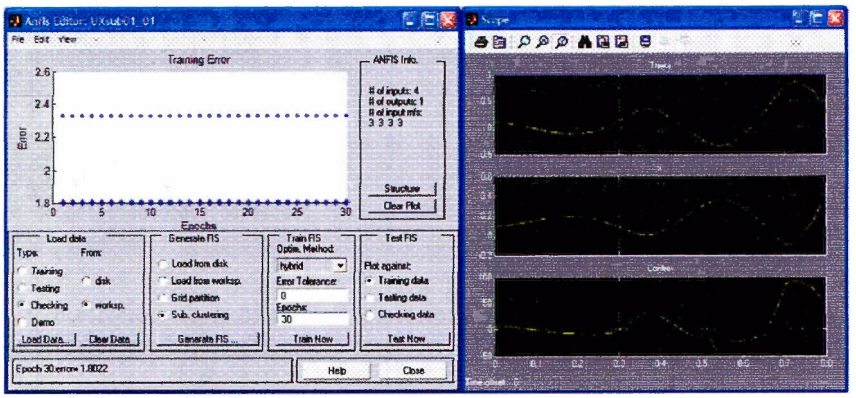


Se probó para otras condiciones iniciales de -0.3 rad, -0.5m.

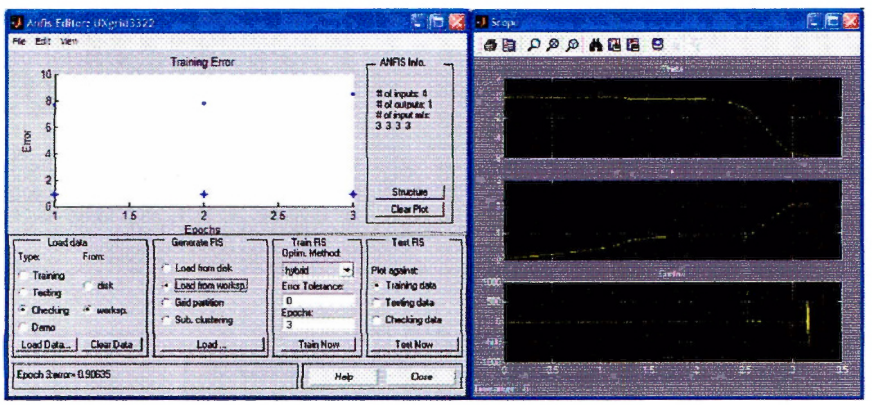


Sub agrupamiento con

- Rango de influencia: 0.2
- Squash factor: 1.05
- Accept ratio: 0.2
- Reject ratio: 0.1

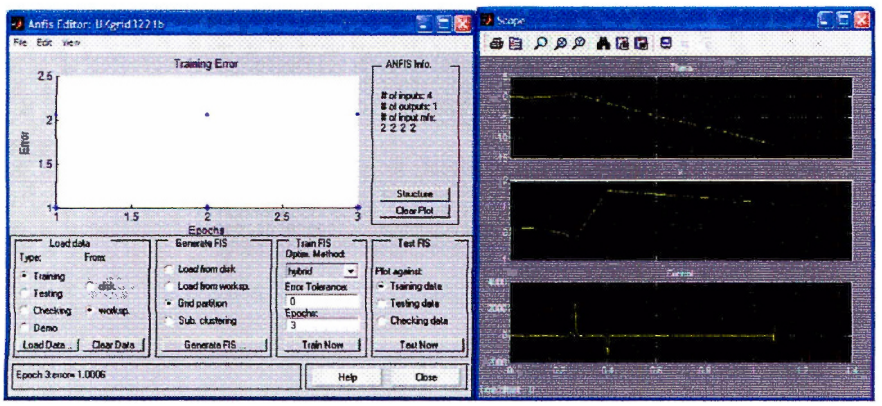


Con el método de mallado, con 3 funciones de membresía tipo gaussiana por entrada (4 entradas).

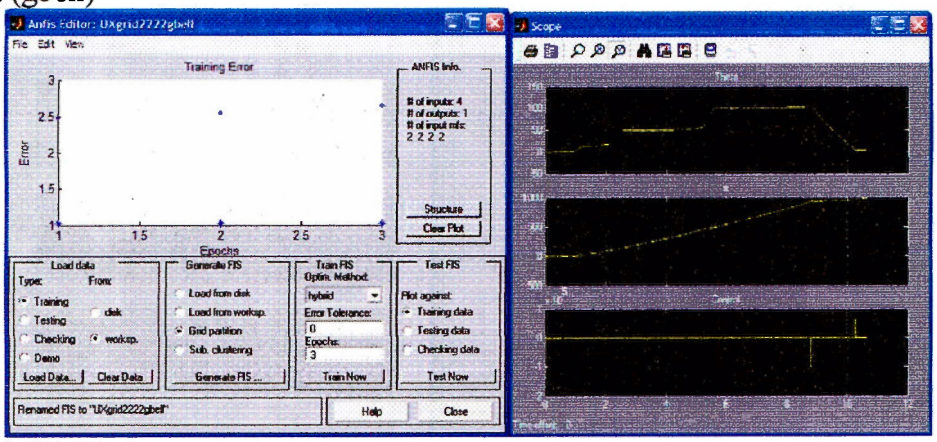


Marcó error después de 3.2 segundos.

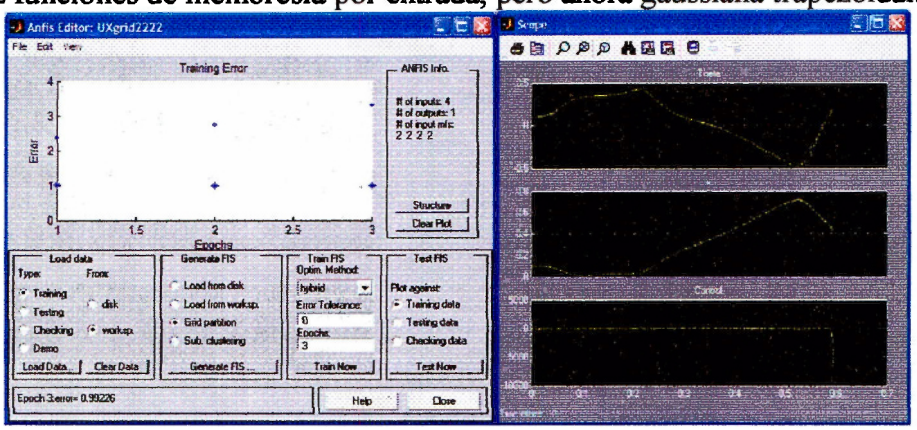
Nuevamente con mallado, ahora con 2 funciones de membresía gaussianas por cada entrada.



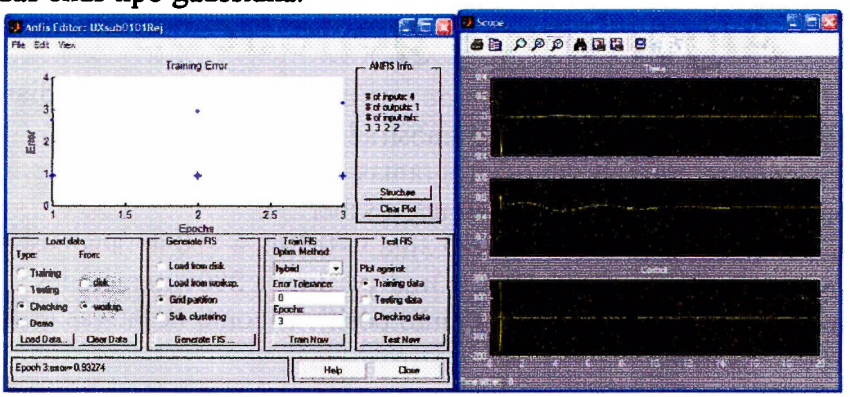
Con mallado, igualmente 2 funciones de membresía por entrada, pero ahora tipo campana gaussiana (gbell)



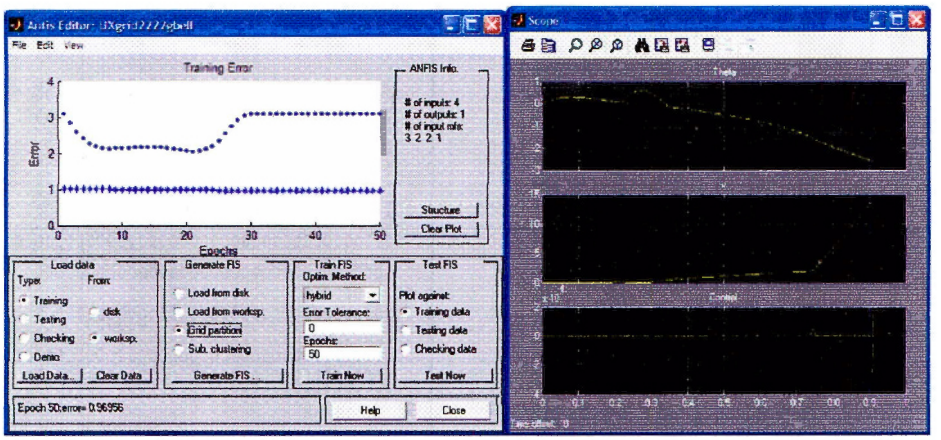
Mallado, 2 funciones de membresía por entrada, pero ahora gaussiana trapezoidal.



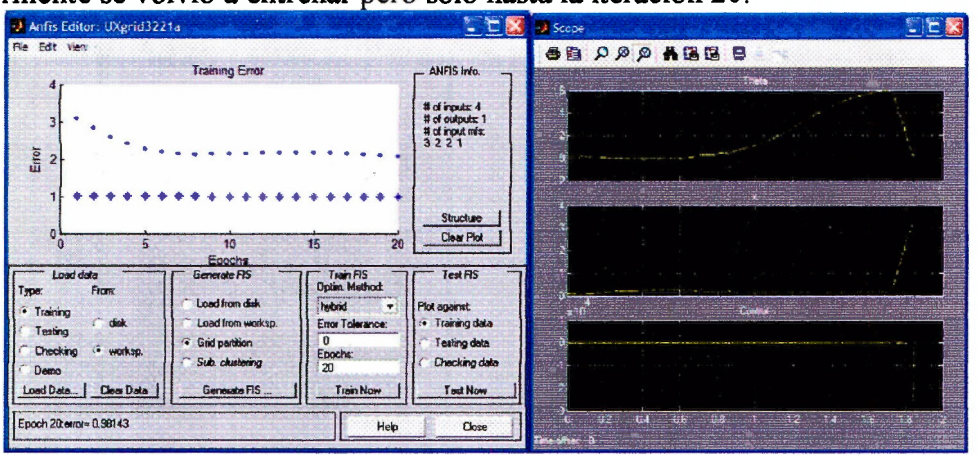
Mallado, 3 funciones de membresía para ángulo y posición, y tan solo 2 funciones para las derivadas, todas ellas tipo gaussiana.



Mallado, 3 funciones de membresía para ángulo, 2 funciones para la posición y la velocidad angular, y una sola para la velocidad lineal, todas ellas tipo gaussiana. Se probó con 50 iteraciones.



Posteriormente se volvió a entrenar pero sólo hasta la iteración 20:



A.5. "Algoritmos genéticos"

Se presentan las pruebas de optimización de una controlador ANFIS mediante algoritmos genéticos, que precedieron a los resultados mostrados en la sección 8.

Como se mencionó, se crearon varios programas de algoritmos genéticos, con diferentes sistemas y con diferentes iteraciones máximas, a saber:

1. Sistema con condiciones iniciales -0.3rad y -0.3 m, a 100 iteraciones.
2. Sistema con condiciones iniciales 0.3rad y 0.3 m, a 2000 iteraciones.
3. Sistema con condiciones iniciales tanto positivas como negativas (±.3 rad y ±.3 m). Con 200 iteraciones.
4. Sistema con condiciones iniciales tanto positivas como negativas (±.5 rad y ±.5 m). Con 2000 iteraciones.
5. Sistema con condiciones iniciales tanto positivas como negativas (±.5 rad y ±.5 m). Con 1000 iteraciones. Pero el criterio de paro es ponderado.

Los primeros cuatro se analizarán a continuación:

A.5.1. Sistema con condiciones iniciales -0.3rad y -0.3 m, a 100 iteraciones.

Con el sistema en condiciones iniciales de -0.3 radianes y -0.3 m, se busca optimizar la siguiente función:

$$criterio = 20 * \sum theta^2 + 10 * \sum x^2 \qquad (12)$$

donde theta y x son los valores de salida en cada tiempo de simulación. Se da una ponderación del doble del error para el ángulo respecto a la posición, para hacer al controlador más activo en dicha variable.
Para ello, se corre la simulación en Simulink y se exportan en una matriz los valores de theta y x.
Se optimiza para 100 iteraciones:

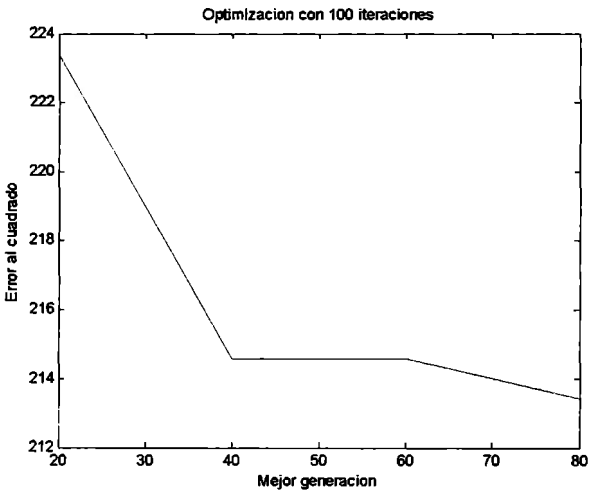


Figura 92. Optimización con algoritmo genético en 100 iteraciones.

Y los parámetros optimizados, que son las constantes de las ecuaciones lineales de salida del controlador Sugeno, son:

best(1) = 34.995418
best(2) = 3.882741
best(3) = 3.441896
best(4) = 4.816650
best(5) = 0.074609
best(6) = 18.771968
best(7) = 5.367916
best(8) = 21.764996
best(9) = 10.569450
best(10) = -0.256071

Al correr en la simulación, con condiciones -0.3rad y -0.3m. obtenemos:

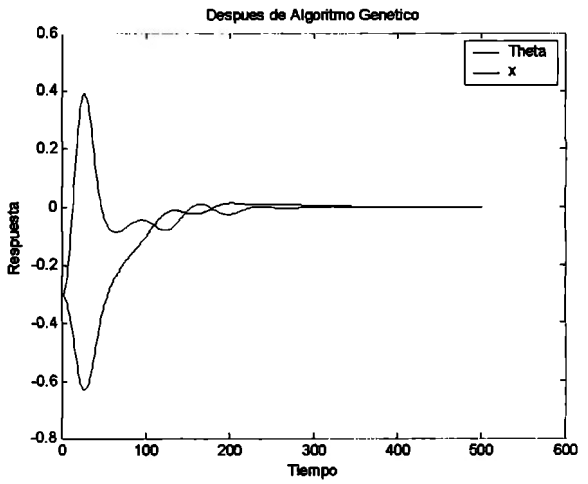


Figura 93. Respuesta del sistema optimizado, en condiciones negativas

Y con condiciones iniciales positivas de 0.3 rad y 0.3m, tenemos:

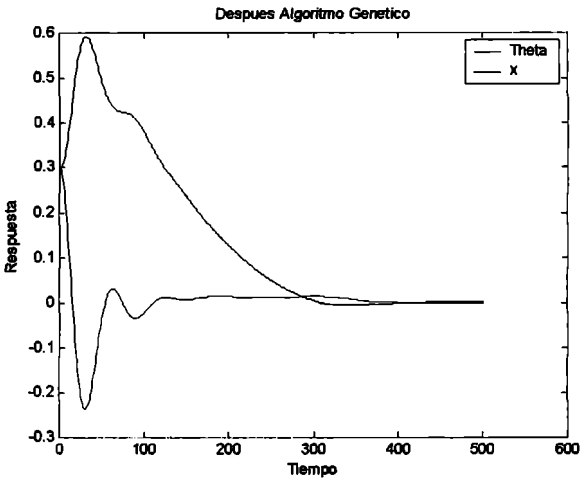


Figura 94. Respuesta del sistema a condiciones iniciales positivas

A.5.2. Sistema con condiciones iniciales 0.3rad y 0.3 m, a 2000 iteraciones.

Se realiza igual que en la optimización anterior, pero ahora con condiciones iniciales positivas (0.3 rad y 0.3 m), hasta 2000 iteraciones (generaciones), como observamos:

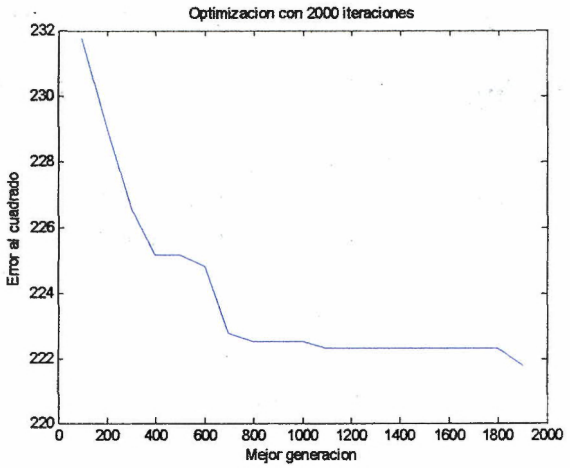


Figura 95. Optimización del sistema en 2000 generaciones

Con ello, el algoritmo tardó 19 horas en terminar, ejecutando 300 000 evaluaciones al simulador (diagrama de bloques). Los parámetros obtenidos fueron:

best(1) = 46.415248
best(2) = 0.443056
best(3) = 24.635170
best(4) = -1.163018
best(5) = -7.951112
best(6) = 7975.784165
best(7) = 297.404686
best(8) = 3420.556959
best(9) = 364.341177
best(10) = -1.014442

Obtenemos:

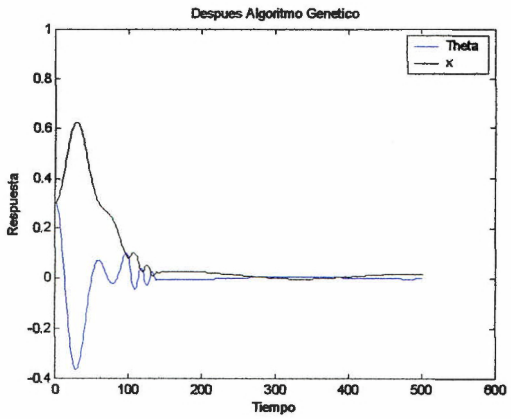


Figura 96. Respuesta del sistema a condiciones iniciales positivas

A.5.3. Sistema con condiciones iniciales tanto positivas como negativas (±0.3 rad y ±0.3 m). Con 200 iteraciones.

Se realizó de forma similar que lo anterior, pero cuidando que la optimización tenga al mismo tiempo una evaluación con condiciones iniciales positivas y negativas. Ello se logró mediante correr dos simulaciones de 5 segundos para condiciones iniciales positivas y 5 segundos para las negativas. Ambas ejecutadas cada vez que se evalúe el criterio a optimizar, y exportando ambos arreglos para cumplir con el criterio:

$$criterio = 20 * \sum (theta_{pos} + theta_{neg})^2 + 10 * \sum (x_{pos} + x_{neg})^2 \qquad (13)$$

Se corrió con 200 iteraciones (generaciones), tardando 4 horas en terminar.

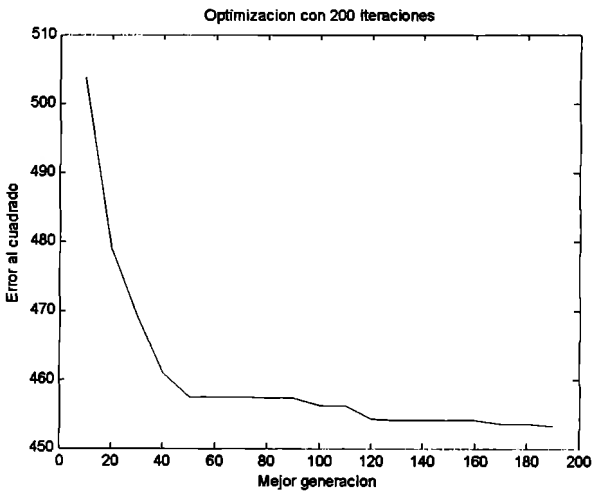


Figura 97. Optimización del sistema con 200 generaciones

Los parámetros optimizados son:

best(1) = 38.348628
best(2) = 4.352005
best(3) = 6.595191
best(4) = 6.084092
best(5) = 0.315752
best(6) = 21.473104
best(7) = 6.225609
best(8) = 20.732177
best(9) = 10.852273
best(10) = -0.450915

Y se obtiene:

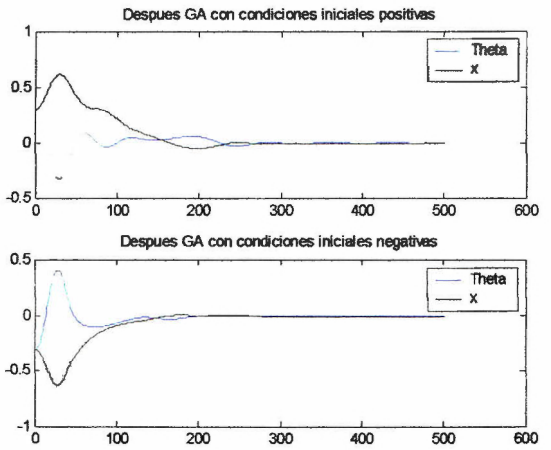


Figura 98. Respuesta del sistema a dos condiciones iniciales diferentes

A.5.4. Sistema con condiciones iniciales tanto positivas como negativas (±0.5 rad y ±0.5 m). Con 2000 iteraciones.

Con condiciones iniciales positivas y negativas, pero mucho más críticas (± 0.5 rad y ± 0.5 m), se entrenó con 2000 iteraciones:

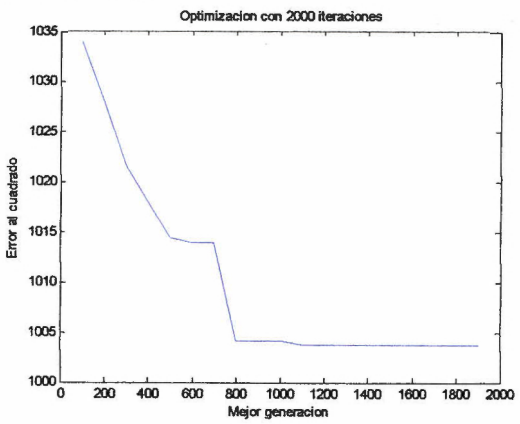


Figura 99. Optimización del sistema con 2000 generaciones

Se evaluó el sistema en 600 000 ocasiones, y se obtuvieron los siguientes valores:

best(1) = 838.174723
best(2) = 77.954881
best(3) = 31.878815
best(4) = 69.917063
best(5) = 0.114835
best(6) = 402.254374
best(7) = 5.614562
best(8) = 190.474634

best(9) = 33.375531
best(10) = 30.360559

Y al ejecutarlo, resulta no satisfactorio.

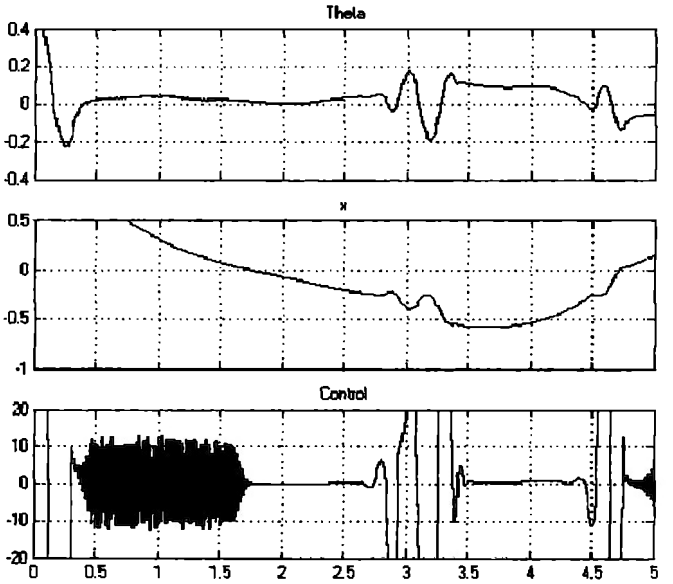


Figura 100. Respuesta del sistema a condiciones iniciales positivas

A.6. "Programas".

A.6.1. MLPemulator.m

```
%Create the data arrays P and T.
Q=length(data);
P(1,:)=[0;data(1:Q-1,1)]';    %Angle (t)
P(2,:)=[0;0;data(1:Q-2,1)]';  %Angle (t-1)
P(3,:)=[0;0;0;data(1:Q-3,1)]'; %Angle (t-2)
P(4,:)=[0;data(1:Q-1,2)]';    %Position (t)
P(5,:)=[0;0;data(1:Q-2,2)]';  %Position (t-1)
P(6,:)=[0;0;0;data(1:Q-3,2)]'; %Position (t-2)
P(7,:)=[0;data(1:Q-1,3)]';    %Control (t)
P(8,:)=[0;0;data(1:Q-2,3)]';  %Control (t-1)
P(9,:)=[0;0;0;data(1:Q-3,3)]'; %Control (t-2)
T=[datos(1:Q,1) datos(1:Q,2)]';
% Create the network ranges, with 20% more than the train data.
RA=1.2*max([abs(min(data(:,1))) abs(max(data(:,1)))]);
RP=1.2*max([abs(min(data(:,2))) abs(max(data(:,2)))]);
```

```
RC=1.2*max([abs(min(data(:,3))) abs(max(data(:,3)))]);
RangeAngle=[-1*RA RA;-1*RA RA;-1*RA RA];
RangePosition=[-1*RP RP;-1*RP RP;-1*RP RP];
RangeControl=[-1*RC RC;-1*RC RC;-1*RC RC];
%Create the network architecture
net=newff([RangeAngle; RangePosition;RangeControl],[20,2],{'tansig','purelin'},'trainlm');
%Define train parameters
net.trainParam.show = 100;
net.trainParam.lr = 0.01;
net.trainParam.epochs = 1000;
net.trainParam.goal = 1e-8;
%Train network
MLPemulator = train(net, P, T);
t=0:.01:50;
subplot(2,1,1),plot(t,sim(MLPemulator,P),t,T)
title('Estimate Training Data');
subplot(2,1,2),plot(t,sim(MLPemulator,P)-T)
title('Error');
figure,
t2=0:.01:20;
T2=[other_data(:,1) other_data(:,2)]';
subplot(2,1,1),plot(t2,sim(MLPemulator,P2),t2,T2)
title('Estimate Non-Training Data');
subplot(2,1,2),plot(t2,sim(MLPemulator,P2)-T2)
title('Error');
```

A.6.2. MLPinvmod.m

```
%Create the data arrays P and T
Q=length(data);
P(1,:)=[data(2:Q,1)]';    %Angle (t+1)
P(2,:)=[data(1:Q-1,1)]';  %Angle (t)
P(3,:)=[0;data(1:Q-2,1)]';  %Angle (t-1)
P(4,:)=[0;0;data(1:Q-3,1)]';%Angle (t-2)
P(5,:)=[data(2:Q,2)]';    %Position(t+1)
P(6,:)=[data(1:Q-1,2)]';  %Position (t)
P(7,:)=[0;data(1:Q-2,2)]';  %Position (t-1)
P(8,:)=[0;0;data(1:Q-3,1)]'; %Position (t-2)
P(9,:)=[0;data(1:Q-2,3)]';  %Control (t-1)
T=[data(1:Q-1,3)]';       %Control (t)
% Create the network ranges, with 20% more than the train data.
RA=1.2*max([abs(min(data(:,1))) abs(max(data(:,1)))]);
RP=1.2*max([abs(min(data(:,2))) abs(max(data(:,2)))]);
RC=1.2*max([abs(min(data(:,3))) abs(max(data(:,3)))]);
```

```
RangeAngle=[-1*RA RA;-1*RA RA;-1*RA RA;-1*RA RA];
RangePosition=[-1*RP RP;-1*RP RP;-1*RP RP;-1*RP RP];
RangeControl=[-1*RC RC];
% Create the network architecture
net=newff([RangeAngle;RangePosition;RangeControl],[10,25,1],{'tansig','tansig','purelin'},
'trainlm');
%Define parameters
net.trainParam.show =10;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 100;
net.trainParam.goal = 1e-8;
%Train network
MLPinvmod = train(net, P, T);
```

A.6.3. Invtest.m

```
% PROGRAM DEMONTRATION OF DIRECT INVERSE TRAINING
%
% Programmed by Magnus Norgaard, IAU/IMM, Technical Univ. of Denmark
% LastEditDate: Jan. 23, 2000
close all
StopDemo=0;
figure
guihand=gcf;
for k=1:1, %dummy loop
 % >>>>>>> BUILD GUI INTERFACE <<<<<<<<<<
 [guihand,edmulti,contbut,quitbut]=pmnshow;
 set(guihand,'Name','Direct inverse control demo');
 % >>>>>>>>>>>>>>> SCREEN 1 <<<<<<<<<<<<<<<<
 s0='1';
 s1='The purpose of this demo is to show different methods';
 s2='for training a neural network to act as the inverse';
 s3='of a nonlinear process. Subsequently the inverse';
 s4='model is used for direct inverse control.';
 s5='  The process under consideration is a spring-mass-';
 s6='damper system with a hardening spring:';
 s7=' y"(t) + y"(t) + y(t) + y(t)^{3} = u(t)';
 smat=str2mat(s0,s1,s2,s3,s4,s5,s6,s7);
 pmnshow(smat,guihand,edmulti,contbut,quitbut);
 if StopDemo==1, close all, break; end
% >>>>>>>>>>>>>>> SCREEN 2 <<<<<<<<<<<<<<<<
 % -- Generate data --
 figure('Units','Centimeters','Position',[1.5 1.5 10 1.5]);
 load expdata;close%[Y,U]=experim;close
```

```
N2=length(U);
N1=floor(N2/2);
Y1 = Y(1:N1)';
U1 = U(1:N1)';
Y2 = datosfin(1:2501,1)';%Y(N1+1:N2)';
U2 = datosfin(1:2501,5)';U(N1+1:N2)';
s0='2';
s1='We began by generating a set of data by performing a';
s2='small experiment. The data set is then split into two';
s3='portions: one for training and one for testing.';
s4='The function "experim" was used for simulating the';
s5='experiment. The design parameters has been initialized in';
s6='the file "expinit".';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6);

subplot(411)
plot(U); hold on
plot([N1 N1],[min(U) max(U)],'m--'); hold off
axis([0 N2 min(U) max(U)])
title('Input sequence (training and test set, respectively)')
subplot(412)
plot(Y); hold on
plot([N1 N1],[min(Y) max(Y)],'m--'); hold off
axis([0 N2 min(Y) max(Y)])
title('Output sequence')
xlabel('time (samples)')
drawnow
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end

% >>>>>>>>>>>>>> SCREEN 3 <<<<<<<<<<<<<<<<<<
subplot(411);delete(gca);subplot(412);delete(gca)
subplot('position',[0.1 0.5 0.6 0.5])
NetDefi  = ['HHHHHHHHHH';'L---------'];
NN       = [3 3 1];
drawnet(ones(5,5),ones(1,6),eps,{'y(t+1)' '  y(t)' 'y(t-1)' 'u(t-1)'},{'u(t)'});
title('Network architecture')
s0='3';
s1='We will now train an inverse neural network model of the';
s2='process by general training. Since it"s a 2nd order process';
s3='we will use the regressor structure shown above. The';
s4='network has 5 hidden "tanh" units and one linear output';
s5='unit. The function "general" implements general';
s6='training with a Levenberg-Marquardt algorithm.';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6);
```

```
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end
% >>>>>>>>>>>>>> SCREEN 4 <<<<<<<<<<<<<<<<
% ----- Train network -----
s0='4';
s1=[];
s2='   >> Training process in action!! <<';
s3=[];
s4=[];
s5='We run up to 500 iterations so you may have to';
s6='wait for a while.';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6);
set(edmulti,'String',smat);
drawnow
[W1i,W2i,PI_vector,iteration,lambda]=general(NetDefi,NN,[],[],[],Y1,U1);
% >>>>>>>>>>>>>> SCREEN 5 <<<<<<<<<<<<<<<<
% ----- Evaluate network -----
s0='5';
s1='The network has now been trained and we will then';
s2='validate it on a fresh data set.';
smat=str2mat(s0,s1,s2);
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end
[Yhat,PI] = invsim(NetDefi,NN,W1i,W2i,Y2,U2);
figure(1)
% >>>>>>>>>>>>>> SCREEN 6 <<<<<<<<<<<<<<<<
s0='6';
s1='Let''s just say we''re satisfied with the network';
s2='model and let''s save architecture definition ';
s3='and weights in a file.';
s4=[];
s5='The network can now be used as controller in a';
s6='"Direct inverse control" concept';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6);
drawnet(W1i,W2i,eps,{'y(t+1)' ' y(t)' 'y(t-1)' 'u(t-1)'},{'u(t)'});
title('Network after training')
save inverse W1i W2i NetDefi NN
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end
% >>>>>>>>>>>>>> SCREEN 7 <<<<<<<<<<<<<<<<
close(2); close(3);
figure('Units','Centimeters','Position',[1.5 1.5 10 1.5]);
invcon;
close
subplot(411)
```

```
plot([0:samples-1],[ref_data y_data]); grid
axis([0 samples -2 2])
title('Reference and output signal')
subplot(412)
plot([0:samples-1],u_data);
axis([0 samples min(u_data) max(u_data)]); grid
title('Control signal')
xlabel('time (samples)')
drawnow
s0='7';
s1='Reasonable model-following has been achieved but';
s2='there is obviously a small overshoot when the reference';
s3='changes. From the plot showing the control signal it is';
s4='evident that the model is used outside the region in';
s5='which it was operated during the initial experiment. To';
s6='fine-tune the network directly on the square wave';
s7='reference trajectory we will use specialized training.';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6,s7);
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end
 % >>>>>>>>>>>>>> SCREEN 8 <<<<<<<<<<<<<<<<<<
s0='8';
s1='Before continuing with specialized training it is necessary';
s2='to identify a "forward" model of the process. This model is';
s3='required by the specialized training algorithm. We will use';
s4='the function "nnarx" from the NNSYSID-toolbox for this. The';
s5='model is inferred from the same data set as was used for';
s6='training the inverse model.';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6);
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end
s8='   >> Training process in action!! <<';
smat=str2mat(s0,s1,s2,s3,s4,s5,s6,[],s8);
set(edmulti,'String',smat);
trparms = settrain;
trparms = settrain(trparms,'maxiter',200);
NetDeff = ['HHHHH';'L----'];

[W1f,W2f,PI_vector,iteration,lambda]=nnarx(NetDeff,NN,[],[],trparms,Y1,U1);
save forward W1f W2f NetDeff NN
 % >>>>>>>>>>>>>> SCREEN 9 <<<<<<<<<<<<<<<<<<
s0='9';
s1='The function "special2" will be used for performing the';
s2='specialized training. We choose a recursive Gauss-Newton';
s3='algorithm with exponential forgetting for updating the weights.';
```

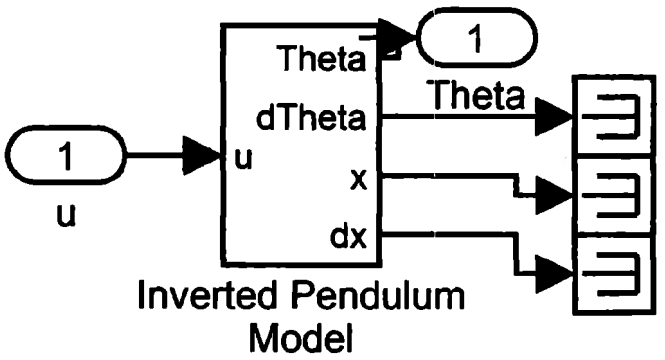
```
s4='The reference trajectory (consisting of 200 samples) will be';
s5='repeated eight times while updating the weights.';
smat=str2mat(s0,s1,s2,s3,s4,s5);
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end
s7='   >> Specialized training process in action!! <<';
smat=str2mat(s0,s1,s2,s3,s4,s5,[],s7);
set(edmulti,'String',smat);
figure(2)
special2
close(2)
save inverse W1i W2i NetDefi NN

% >>>>>>>>>>>>> SCREEN 10 <<<<<<<<<<<<<<<<
s0='10';
s1='The training is now finished and the new weights are saved';
s2='in a file. To conclude the session we will simulate the entire';
s3='closed-loop system again.';
smat=str2mat(s0,s1,s2,s3);
pmnshow(smat,guihand,edmulti,contbut,quitbut);
if StopDemo==1, close all, break; end

figure('Units','Centimeters','Position',[1.5 1.5 10 1.5]);
invcon
close(2)
subplot(411)
plot([0:samples-1],[ref_data y_data]); grid
axis([0 samples -2 2])
title('Reference and output signal')
subplot(412)
plot([0:samples-1],u_data);
axis([0 samples min(u_data) max(u_data)]); grid
title('Control signal')
xlabel('time (samples)')
drawnow

% >>>>>>>>>>> SCREEN 11 <<<<<<<<<<<<<<<<
s1=[];
s2='              >> THE END <<';
smat=str2mat(s1,s1,s1,s1,s2);
set(edmulti,'String',smat);
drawnow
end
```

A.6.3.1. spm1.mdl



A.6.3.2. invinit.m

```
% ----------------------> INVINIT.M <----------------------
% ---------- Switches ----------
regty    ='dic';        % Controller type (dic, pid, none)
refty    ='siggener';   % Reference signal (siggener/<var. name>)
simul    ='simulink';   % Control object spec. (simulink/matlab/nnet)
if exist('simulink')~=5,
  simul    ='matlab';   % Simulink not present
end

% ------ General Initializations -------
Ts = 0.20;              % Sampling period (in seconds)
samples = 200 ;         % Number of samples to be simulated
u_0 = 0;                % Initial control input
y_0 = 0;                % Initial output
ulim_min = -Inf;        % Minimum control input
ulim_max = Inf;         % Maximum control input

% -- System to be Controlled (SIMULINK) --
integrator= 'ode45';    % Name of dif. eq. solver (f. ex. ode45 or ode15s)
sim_model = 'spm1';     % Name of SIMULINK model

% --- System to be Controlled (MATLAB) --
mat_model = 'springm';  % Name of MATLAB model
model_out = 'smout';    % Output equation (function of the states)
x0        = [0;0];      % Initial states

% ----- Inverse Network Specification ------
```

```
% The file must contain the variables:
% NN, NetDefi, W1i, W2i
% (i.e. regressor structure, architecture definition, and weight matrices)
nninv = 'inverse';          % Name of file

% ----- Forward Network Specification ------
% Only used if simul='nnet' (no Simulink or Matlab model available)
% The file must contain: NN, NetDeff, W1f, W2f
nnforw = 'forward';         % Name of file

% ---------- Reference Filter ----------
Am = [1 -0.7];              % Filter denominator
Bm = [0.3];                 % Filter numerator
Am=1;Bm=1;

% ---------- Reference signal ----------
dc     = 0;                 % DC-level
sq_amp  = 1;                % Amplitude of square signals (row vector)
sq_freq = 0.1;              % Frequency of square signals (column vector)
sin_amp = [0];              % Amplitude of sine signals  (row vector)
sin_freq= [0]';             % Frequency of sine signals   (column vector)
Nvar  = 0';                 % Variance of white noise signal

% -------- Linear Controller Parameters ---------
K=8;                        % PID parameters
Td=0.8;                     % PID
alf=0.1;                    % PID
Wi=0.2;                     % PID (1/Ti)

% ------- Specify data vectors to plot --------
% plot_a and plot_b must be cell structures containing the vector names in strings
plot_a = {'ref_data','y_data'};
plot_b = {'u_data'};
```

A.6.3.3. invinit2.m

```
% ---------------------> INVINIT2.M <-------------------------
% Initialization file for the program "special2"
% ---------      Switches      ----------
simul     = 'simulink';   % System specification (simulink/matlab/nnet)
method    = 'ff';         % Training algorithm (ff/ct/efra)
refty     = 'siggener';   % Reference signal (siggener/<variable name>)
if exist('simulink')~=5,
```

```
 simul    ='matlab';    % Simulink not present
end

% ------  General Initializations  ------
Ts = 0.20;              % Sampling period (in seconds)
samples = 200 ;         % Number of samples in each epoch
u_0 = 0;                % Initial control input
y_0 = 0;                % Initial output
ulim_min = -Inf;        % Minimum control input
ulim_max = Inf;         % Maximum control input

% --  System to be Controlled (SIMULINK) --
integrator= 'ode45';    % Name of dif. eq. solver (f. ex. ode45 or ode15s)
sim_model = 'spm1';     % Name of SIMULINK model

% ---  System to be Controlled (MATLAB)  --
mat_model = 'springm';  % Name of MATLAB model
model_out = 'smout';    % Output equation (function of the states)
x0      = [0;0];        % Initial states

% -----  Neural Network Specification  ------
% The "forward model file" must contain the following variables which together
% define a NNARX-model:
% NN, NetDeff, W1f, W2f
% and the "inverse model file" must contain
% NN, NetDefi, W1i, W2i
% (i.e. regressor structure, architecture definition, and weight matrices)
nnforw = 'forward';     % Name of file containing forward model
nninv  = 'inverse';     % Name of file containing inverse model

% ----------  Reference Model  ----------
Am = [1 -0.7];          % Model denominator
Bm = [0.3];             % Model numerator (starts in z^-1)
Am=1;Bm=1;

% ----------  Training parameters  ----------
maxiter = 8;

% ---  Forgetting factor algorithm (ff)  ---
% trparms = [lambda p0]
%   lambda = forgetting factor (suggested value 0.995)
%   p0     = Covariance matrix diagonal (1-10)
%
% ---  Constant trace algorithm (ct)  ---
% trparms = [lambda alpha_max alpha_min]
```

```
%   lambda = forgetting factor (suggested value 0.995)
%   alpha_max = Max. eigenvalue of covariance matrix (100)
%   alpha_min = Min. eigenvaule of covariance matrix (0.001)
%
% --- Exponential Forgetting and Restting Algorithm (efra) ---
% trparms = [alpha beta delta lambda]
%   Suggested values:
%   alpha = 0.5-1
%   beta = 0.001
%   delta = 0.001
%   lambda = 0.98
trparms = [0.995 10];
%trparms = [0.995 100 0.001];
%trparms = [1 0.001 0.001 0.98];

% ---------- Reference signal ----------
% Reference generated by the signal generator
dc     = 0;              % DC-level
sq_amp  = 1;              % Amplitude of square signals (row vector)
sq_freq = 0.1;            % Frequency of square signals (column vector)
sin_amp = [0];            % Amplitude of sine signals  (row vector)
sin_freq= [0]';           % Frequency of sine signals   (column vector)
Nvar  = 0;                % Variance of white noise signal

% ------- Specify data vectors to plot --------
% plot_a and plot_b must be cell structures containing the vector names in strings
plot_a = {'ref_data','y_data','ym_data','yhat_data'};
plot_b = {'u_data'};
```

A.6.4. genetic.m

```
%function [bestmem,bestval,nfeval] =
%devec3();%(fname,VTR,D,XVmin,XVmax,y,NP,itermax,F,CR,strategy,refresh);
% minimization of a user-supplied function with respect to x(1:D),
% using the differential evolution (DE) algorithm of Rainer Storn
% (http://www.icsi.berkeley.edu/~storn/code.html)
%
% Special thanks go to Ken Price (kprice@solano.community.net) and
% Arnold Neumaier (http://solon.cma.univie.ac.at/~neum/) for their
% valuable contributions to improve the code.
%
% Strategies with exponential crossover, further input variable
% tests, and arbitrary function name implemented by Jim Van Zandt
```

```
% <jrv@vanzandt.mv.com>, 12/97.
%
% Output arguments:
% ----------------
% bestmem       parameter vector with best solution
% bestval       best objective function value
% nfeval        number of function evaluations
%
% Input arguments:
% ---------------
%
% fname         string naming a function f(x,y) to minimize
% VTR           "Value To Reach". devec3 will stop its minimization
%               if either the maximum number of iterations "itermax"
%               is reached or the best parameter vector "bestmem"
%               has found a value f(bestmem,y) <= VTR.
% D             number of parameters of the objective function
% XVmin         vector of lower bounds XVmin(1) ... XVmin(D)
%               of initial population
%               *** note: these are not bound constraints!! ***
% XVmax         vector of upper bounds XVmax(1) ... XVmax(D)
%               of initial population
% y                     problem data vector (must remain fixed during the
%               minimization)
% NP            number of population members
% itermax       maximum number of iterations (generations)
% F             DE-stepsize F from interval [0, 2]
% CR            crossover probability constant from interval [0, 1]
% strategy      1 --> DE/best/1/exp          6 --> DE/best/1/bin
%               2 --> DE/rand/1/exp          7 --> DE/rand/1/bin
%               3 --> DE/rand-to-best/1/exp  8 --> DE/rand-to-best/1/bin
%               4 --> DE/best/2/exp          9 --> DE/best/2/bin
%               5 --> DE/rand/2/exp          else  DE/rand/2/bin
%               Experiments suggest that /bin likes to have a slightly
%               larger CR than /exp.
% refresh       intermediate output will be produced after "refresh"
%               iterations. No intermediate output will be produced
%               if refresh is < 1
%
%       The first four arguments are essential (though they have
%       default values, too). In particular, the algorithm seems to
%       work well only if [XVmin,XVmax] covers the region where the
%       global minimum is expected. DE is also somewhat sensitive to
%       the choice of the stepsize F. A good initial guess is to
%       choose F from interval [0.5, 1], e.g. 0.8. CR, the crossover
```

```
%       probability constant from interval [0, 1] helps to maintain
%       the diversity of the population and is rather uncritical. The
%       number of population members NP is also not very critical. A
%       good initial guess is 10*D. Depending on the difficulty of the
%       problem NP can be lower than 10*D or must be higher than 10*D
%       to achieve convergence.
%       If the parameters are correlated, high values of CR work better.
%       The reverse is true for no correlation.
%
% default values in case of missing input arguments:
%       VTR = 1.e-6;
%       D = 2;
%       XVmin = [-2 -2];
%       XVmax = [2 2];
%       y=[];
%       NP = 10*D;
%       itermax = 200;
%       F = 0.8;
%       CR = 0.5;
%       strategy = 7;
%       refresh = 10;
%
% Cost function:        function result = f(x,y);
%                       has to be defined by the user and is minimized
%                       w.r. to  x(1:D).
%
% Example to find the minimum of the Rosenbrock saddle:
% ---------------------------------------------------
% Define f.m as:
%               function result = f(x,y);
%               result = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
%               end
% Then type:
%
%       VTR = 1.e-6;
%       D = 2;
%       XVmin = [-2 -2];
%       XVmax = [2 2];
%       [bestmem,bestval,nfeval] = devec3("f",VTR,D,XVmin,XVmax);
%
% The same example with a more complete argument list is handled in
% run1.m
%
% About devec3.m
% --------------
```

```
% Differential Evolution for MATLAB
% Copyright (C) 1996, 1997 R. Storn
% International Computer Science Institute (ICSI)
% 1947 Center Street, Suite 600
% Berkeley, CA 94704
% E-mail: storn@icsi.berkeley.edu
% WWW:    http://http.icsi.berkeley.edu/~storn
%
% devec is a vectorized variant of DE which, however, has a
% propertiy which differs from the original version of DE:
% 1) The random selection of vectors is performed by shuffling the
%    population array. Hence a certain vector can't be chosen twice
%    in the same term of the perturbation expression.
%
% Due to the vectorized expressions devec3 executes fairly fast
% in MATLAB's interpreter environment.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 1, or (at your option)
% any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details. A copy of the GNU
% General Public License can be obtained from the
% Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

%-----Check input variables---------------------------------

TRNDATA1=[datosCondInic(:,1) datosCondInic(:,3) datosCondInic(:,5)]; %Se toman solo
dos entradas para facilitar entrenamiento, Se entrena Theta y dTheta
fis=genfis1(TRNDATA1,[2 2]);
fis=anfis(TRNDATA1,fis,500); %entrena ANFIS con los datos anteriores  Checar el
numero de epochs

vx1=fis.input(1,1).mf(1,1).params; %Rangos actuales
vx2=fis.input(1,1).mf(1,2).params;
vx3=fis.input(1,2).mf(1,1).params;
vx4=fis.input(1,2).mf(1,2).params;
vx5=fis.output(1,1).mf(1,1).params;
vx6=fis.output(1,1).mf(1,2).params;
vx7=fis.output(1,1).mf(1,3).params;
vx8=fis.output(1,1).mf(1,4).params;
```

```
vxmax=[vx1+.2*abs(vx1) vx2+.2*abs(vx2) vx3+.2*abs(vx3) vx4+.2*abs(vx4)
vx5+.2*abs(vx5) vx6+.2*abs(vx6) vx7+.2*abs(vx7) vx8+.2*abs(vx8)];
vxmin=[vx1-.2*abs(vx1) vx2-.2*abs(vx2) vx3-.2*abs(vx3) vx4-.2*abs(vx4) vx5-
.2*abs(vx5) vx6-.2*abs(vx6) vx7-.2*abs(vx7) vx8-.2*abs(vx8)];

%Defaul variables and values
VTR = 1.e-6;
        D = 2;
        XVmin = [-2 -2];
        XVmax = [2 2];
        y=[];
        NP = 10*D;
        itermax =200 ;
        F = 0.8;
        CR = 0.5;
        strategy = 7;
        refresh = 10;

VTR = 1.e-03;
D = 24;
NP=15*D;
XVmin = vxmin;
XVmax = vxmax;
itermax=2000;%DEBEN SER MUCHAS
refresh=50;
%fname='ftest';

if (NP < 5)
  NP=5;
  fprintf(1,' NP increased to minimal value 5\n');
end
if ((CR < 0) | (CR > 1))
  CR=0.5;
  fprintf(1,'CR should be from interval [0,1]; set to default value 0.5\n');
end
if (itermax <= 0)
  itermax = 200;
  fprintf(1,'itermax should be > 0; set to default value 200\n');
end
refresh = floor(refresh);

%-----Initialize population and some arrays-------------------

pop = zeros(NP,D); %initialize pop to gain speed
```

```
%----pop is a matrix of size NPxD. It will be initialized----------
%----with random values between the min and max values of the parameters----------
------------------------------

for i=1:NP
  pop(i,:) = XVmin + rand(1,D).*(XVmax - XVmin);
end

popold   = zeros(size(pop));    % toggle population
val      = zeros(1,NP);         % create and reset the "cost array"
bestmem  = zeros(1,D);          % best population member ever
bestmemit = zeros(1,D);         % best population member in iteration
nfeval   = 0;                   % number of function evaluations

%------Evaluate the best member after initialization----------

ibest   = 1;                    % start with first population member
x=pop(ibest,:);

%result=simout;
%simout=feval(fname,pop(ibest,:),y)';

   %IN GENETIC ALGORITHM
   fis.input(1,1).mf(1,1).params=[x(1) x(2) x(3)];
   fis.input(1,1).mf(1,2).params=[x(4) x(5) x(6)];
   fis.input(1,2).mf(1,1).params=[x(7) x(8) x(9)];
   fis.input(1,2).mf(1,2).params=[x(10) x(11) x(12)];
   fis.output(1,1).mf(1,1).params=[x(13) x(14) x(15)];
   fis.output(1,1).mf(1,2).params=[x(16) x(17) x(18)];
   fis.output(1,1).mf(1,3).params=[x(19) x(20) x(21)];
   fis.output(1,1).mf(1,4).params=[x(22) x(23) x(24)];
   sim('controlleranfis',[0:.01:5]); %number of data
   error=2*sum(power(errori(:,1),2))+sum(power(errori(:,2),2));%errori es la salida de la
simulacion... el error en Theta y x
val(1)=error;

bestval = val(1);               % best objective function value so far
nfeval  = nfeval + 1;
for i=2:NP                      % check the remaining members

   x=pop(ibest,:);

%result=simout;
%simout=feval(fname,pop(ibest,:),y)';
```

```
   %IN GENETIC ALGORITHM
   fis.input(1,1).mf(1,1).params=[x(1) x(2) x(3)];
   fis.input(1,1).mf(1,2).params=[x(4) x(5) x(6)];
   fis.input(1,2).mf(1,1).params=[x(7) x(8) x(9)];
   fis.input(1,2).mf(1,2).params=[x(10) x(11) x(12)];
   fis.output(1,1).mf(1,1).params=[x(13) x(14) x(15)];
   fis.output(1,1).mf(1,2).params=[x(16) x(17) x(18)];
   fis.output(1,1).mf(1,3).params=[x(19) x(20) x(21)];
   fis.output(1,1).mf(1,4).params=[x(22) x(23) x(24)];
   sim('controlleranfis',[0:.01:5]); %number of data
   error=2*sum(power(errori(:,1),2))+sum(power(errori(:,2),2));%errori es la salida de la
simulacion... el error en Theta y x
val(i)=error;

 %val(i) = feval(fname,x,y);
 nfeval  = nfeval + 1;
 if (val(i) < bestval)        % if member is better
   ibest   = i;              % save its location
   bestval = val(i);
 end
end
bestmemit = pop(ibest,:);       % best member of current iteration
bestvalit = bestval;            % best value of current iteration

bestmem = bestmemit;            % best member ever

%------DE-Minimization-----------------------------------------
%------popold is the population which has to compete. It is--------
%------static through one iteration. pop is the newly----------
%------emerging population.-----------------------------------

pm1 = zeros(NP,D);              % initialize population matrix 1
pm2 = zeros(NP,D);              % initialize population matrix 2
pm3 = zeros(NP,D);              % initialize population matrix 3
pm4 = zeros(NP,D);              % initialize population matrix 4
pm5 = zeros(NP,D);              % initialize population matrix 5
bm  = zeros(NP,D);              % initialize bestmember  matrix
ui  = zeros(NP,D);              % intermediate population of perturbed vectors
mui = zeros(NP,D);              % mask for intermediate population
mpo = zeros(NP,D);              % mask for old population
rot = (0:1:NP-1);               % rotating index array (size NP)
rotd= (0:1:D-1);                % rotating index array (size D)
rt  = zeros(NP);                % another rotating index array
rtd = zeros(D);                 % rotating index array for exponential crossover
```

```
a1  = zeros(NP);          % index array
a2  = zeros(NP);          % index array
a3  = zeros(NP);          % index array
a4  = zeros(NP);          % index array
a5  = zeros(NP);          % index array
ind = zeros(4);

iter = 1;
while ((iter < itermax) & (bestval > VTR))
  popold = pop;           % save the old population

  ind = randperm(4);      % index pointer array

  a1  = randperm(NP);     % shuffle locations of vectors
  rt = rem(rot+ind(1),NP); % rotate indices by ind(1) positions
  a2  = a1(rt+1);         % rotate vector locations
  rt = rem(rot+ind(2),NP);
  a3  = a2(rt+1);
  rt = rem(rot+ind(3),NP);
  a4  = a3(rt+1);
  rt = rem(rot+ind(4),NP);
  a5  = a4(rt+1);

  pm1 = popold(a1,:);     % shuffled population 1
  pm2 = popold(a2,:);     % shuffled population 2
  pm3 = popold(a3,:);     % shuffled population 3
  pm4 = popold(a4,:);     % shuffled population 4
  pm5 = popold(a5,:);     % shuffled population 5

  for i=1:NP              % population filled with the best member
    bm(i,:) = bestmemit;  % of the last iteration
  end

  mui = rand(NP,D) < CR;  % all random numbers < CR are 1, 0 otherwise

  if (strategy > 5)
    st = strategy-5;      % binomial crossover
  else
    st = strategy;        % exponential crossover
    mui=sort(mui');       % transpose, collect 1's in each column
    for i=1:NP
      n=floor(rand*D);
      if n > 0
        rtd = rem(rotd+n,D);
        mui(:,i) = mui(rtd+1,i); %rotate column i by n
```

```
    end
   end
   mui = mui';                    % transpose back
  end
  mpo = mui < 0.5;               % inverse mask to mui

  if (st == 1)                   % DE/best/1
   ui = bm + F*(pm1 - pm2);      % differential variation
   ui = popold.*mpo + ui.*mui;   % crossover
  elseif (st == 2)               % DE/rand/1
   ui = pm3 + F*(pm1 - pm2);     % differential variation
   ui = popold.*mpo + ui.*mui;   % crossover
  elseif (st == 3)               % DE/rand-to-best/1
   ui = popold + F*(bm-popold) + F*(pm1 - pm2);
   ui = popold.*mpo + ui.*mui;   % crossover
  elseif (st == 4)               % DE/best/2
   ui = bm + F*(pm1 - pm2 + pm3 - pm4); % differential variation
   ui = popold.*mpo + ui.*mui;          % crossover
  elseif (st == 5)               % DE/rand/2
   ui = pm5 + F*(pm1 - pm2 + pm3 - pm4); % differential variation
   ui = popold.*mpo + ui.*mui;          % crossover
  end

%-----Select which vectors are allowed to enter the new population----------
  for i=1:NP

    x=ui(i,:);

%result=simout;
%simout=feval(fname,pop(ibest,:),y)';

    %IN GENETIC ALGORITHM
    fis.input(1,1).mf(1,1).params=[x(1) x(2) x(3)];
    fis.input(1,1).mf(1,2).params=[x(4) x(5) x(6)];
    fis.input(1,2).mf(1,1).params=[x(7) x(8) x(9)];
    fis.input(1,2).mf(1,2).params=[x(10) x(11) x(12)];
    fis.output(1,1).mf(1,1).params=[x(13) x(14) x(15)];
    fis.output(1,1).mf(1,2).params=[x(16) x(17) x(18)];
    fis.output(1,1).mf(1,3).params=[x(19) x(20) x(21)];
    fis.output(1,1).mf(1,4).params=[x(22) x(23) x(24)];
    sim('controlleranfis',[0:.01:5]); %number of data
    error=2*sum(power(errori(:,1),2))+sum(power(errori(:,2),2));%errori es la salida de la
simulacion... el error en Theta y x
    tempval=error;
```

```
  %tempval = feval(fname,ui(i,:),y);  % check cost of competitor
  nfeval  = nfeval + 1;
  if (tempval <= val(i))  % if competitor is better than value in "cost array"
    pop(i,:) = ui(i,:);  % replace old vector with new one (for new iteration)
    val(i)   = tempval;  % save value in "cost array"

    %----we update bestval only in case of success to save time----------
    if (tempval < bestval)     % if competitor better than the best one ever
      bestval = tempval;      % new best value
      bestmem = ui(i,:);      % new best parameter vector ever
    end
  end
 end %---end for imember=1:NP

 bestmemit = bestmem;       % freeze the best member of this iteration for the coming
                            % iteration. This is needed for some of the strategies.

%----Output section----------------------------------------------------

 if (refresh > 0)
  if (rem(iter,refresh) == 0)
    fprintf(1,'Iteration: %d,  Best: %f,  F: %f,  CR: %f,  NP: %d\n',iter,bestval,F,CR,NP);
    for n=1:D
      fprintf(1,'best(%d) = %f\n',n,bestmem(n));
    end
  end
 end

 iter = iter + 1;
end %---end while ((iter < itermax) ...

%------------
```

A.6.5. animacion.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Animation of the Inverted pendulum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ernesto Olguín Díaz, adaptado por Eugenio Alvarez
% April 1, 2003. Modificacion Febrero 2004.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% points des articulations du bras
```

```
% bb1=[0,0];                                   Joint 1 (base point)
% bb2=[bb13(:,1),bb13(:,2)];Joint 2
% bb3=[bb13(:,3),bb13(:,4)];Joint 3
% bb4=[bb13(:,5),bb13(:,6)];End-effector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a2=0.500;      %longeur du corp 2
a3=0.300;      %longeur du corp 3
n=10;
l3=.61/2;%0.4202;    %distace du joint 3 au centre de masses du corp 3
q=qEug;
q(:,1)=pi/2+q(:,1);
q(:,2)=-q(:,2);
%global l2 l3
bb=0.315;      %Anchura de carro base
a0=bb/2;
hb=0.13;       %Altura del carro base
h0=0.1;    %altura del punto de giro del pendulo
r2=0.027;      %diametre moyen du pendule
r3=0.06;       %diametre moyen du corp 3
n=size(t,1);
w=0.1;                 %angular step for semicirclular plottings
% simulacion dinamica del manipulador
figure(2);
clf;
axis([-1.0 1.0 -1.0 1.0])%([-1.5 1.5 -1.5 1.5])
axis('equal')
film=round(0.08/(t(2)-t(1)));
for i=1:film:n;
line([-1 1],[0 0]);              %line de suelo
line([0,0],[0 0.25]);
%Points of the base
bb0=[0+q(i,2),0];                %centro del carro base
bbe1=( bb0' + [-a0;0] )';        %esquina inferior izquierda
bbe2=( bb0' + [-a0;hb] )';       %esquina superior izquierda
bbe3=( bb0' + [a0;hb] )';        %esquina inferior derecha
bbe4=( bb0' + [a0;0] )';         %esquina superior derecha
pg =( bb0' + [0;h0] )';          %punto de giro del pendulo
%Points of pendulum (mass 2)
R1=[cos(q(i,1)) -sin(q(i,1))
sin(q(i,1)) cos(q(i,1))];  %matriz de rotacion para el calculo de los puntos del pendulo
ppf= ( pg' + R1*([a2; 0]) )';     %punto final del pendulo (masa 2)
%curves du link
phi1= q(i,1) + pi/2;
phi2= q(i,1) - pi/2;
for j=0:abs(pi/w)
```

```
xr11(j+1,:)=pg + [r2*cos(w*j+phi1),r2*sin(w*j+phi1)];
xr11(j+2,:)=pg + [r2*cos(pi+phi1),r2*sin(pi+phi1)];
xr12(j+1,:)=ppf + [r2*cos(w*j+phi2),r2*sin(w*j+phi2)];
xr12(j+2,:)=ppf + [r2*cos(pi+phi2),r2*sin(pi+phi2)];
end
%Points of pendulum (mass 3)
pp3e1= ( pg' + R1*([l3-r3;r3]) )';
pp3e2= ( pg' + R1*([l3+r3;r3]) )';
pp3e3= ( pg' + R1*([l3+r3;-r3]) )';
pp3e4= ( pg' + R1*([l3-r3;-r3]) )';
base=[bbe1;bbe2;bbe3;bbe4];
pendul=[xr11;xr12];
masa3=[pp3e1;pp3e2;pp3e3;pp3e4];
if i==1
%arm drawing
l00=patch(base(:,1),base(:,2),'b');
l01=patch(pendul(:,1),pendul(:,2),'g');
l02=patch(masa3(:,1),masa3(:,2),'y');
set(findobj(gca, 'type', 'patch'), 'erasemode', 'xor')
else
set(l00,'XData',base(:,1),'YData',base(:,2));
set(l01,'XData',pendul(:,1),'YData',pendul(:,2));
set(l02,'XData',masa3(:,1),'YData',masa3(:,2));
end     %end of if
drawnow;
pause(0.01)
end     %end of for
```

A.6.6. Implementacion_ difusa.bas

```
valor var word
errortviejo var long
derrort var long
errort var long
salida var long
sumaresta var long
contx var long
espera var word

negta con 0
negtb con 47000
negtc con 50000
cerota con 48500
```

```
cerotb con 50000
cerotc con 51500
posta con 50000
postb con 53000
postc con 100000
negdta con -20000
negdtb con -2500
negdtc con -1000
cerodta con -1500
cerodtb con 0
cerodtc con 1500
posdta con 1000
posdtb con 2500
posdtc con 20000

t1 var word
t2 var word
t3 var word
dt1 var word
dt2 var word
dt3 var word
pos1 var word
pos2 var word
pos3 var word
pos4 var word
cero var word
neg1 var word
neg2 var word
neg3 var word
neg4 var word
pos var word
neg var word

CLK con 2

defuzzifica var word
defuzzifica2 var word

Inicializa
        contx=0
        salida=0
        errortviejo=50000
        errort=0
        derrort=0
        sumaresta=1
```

```
;EMPIEZA PROGRAMA
Main
        if contx>=400 or contx<=-400 then
                stop
        endif
        Adin AX0,CLK,AD_RON,valor
        if salida>=4900 then
                contx=contx+sumaresta;
                pulsout P0, 30
        endif
        errort=valor*100;(valor-setpoint)*100
        debug ["e ",dec errort, 10]
        derrort=errort-errortviejo
        debug ["de ",dec derrort, 10]

        if errort>=53000 then
                errort=53000
        elseif errort<47000
                errort=47000
        endif
        errortviejo=errort;

;// FUZZIFICA
        if errort<=negta OR errort>=negtc then
                t1=0
        elseif errort>negta AND errort<negtb
                t1=(errort-negta)/((negtb-negta)/100);//x(index)-a)/(b-a)
        else
                t1=(negtc-errort)/((negtc-negtb)/100); //(c-x(index))/(c-b)
        endif
        if errort<=cerota OR errort>=cerotc then
                t2=0
        elseif errort>cerota AND errort<cerotb
                t2=(errort-cerota)/((cerotb-cerota)/100);//x(index)-a)/(b-a)
        else
                t2=(cerotc-errort)/((cerotc-cerotb)/100); //(c-x(index))/(c-b)
        endif
        if errort<=posta OR errort>=postc then
                t3=0
        elseif errort>posta AND errort<postb
                t3=(errort-posta)/((postb-posta)/100);//x(index)-a)/(b-a)
        else
                t3=(postc-errort)/((postc-postb)/100); //(c-x(index))/(c-b)
        endif
```

```
        if salida>=3000 then
                contx=contx+sumaresta;
                pulsout P0, 30
        endif
        if derrort<=negdta OR derrort>=negdtc then
                dt1=0
        elseif derrort>negdta AND derrort<negdtb
                dt1=(derrort-negdta)*100/(negdtb-negdta);  //x(index)-a)/(b-a)
        else
                dt1=(negdtc-derrort)*100/(negdtc-negdtb); //(c-x(index))/(c-b)
        endif
        if derrort<=cerodta OR derrort>=cerodtc then
                dt2=0
        elseif derrort>cerodta AND derrort<cerodtb
                dt2=(derrort-cerodta)*100/(cerodtb-cerodta);//x(index)-a)/(b-a)
        else
                dt2=(cerodtc-derrort)*100/(cerodtc-cerodtb); //(c-x(index))/(c-b)
        endif
        if derrort<=posdta OR derrort>=posdtc then
                dt3=0
        elseif derrort>posdta AND derrort<posdtb
                dt3=(derrort-posdta)*100/(posdtb-posdta);  //x(index)-a)/(b-a)
        else
                dt3=(posdtc-derrort)*100/(posdtc-posdtb); //(c-x(index))/(c-b)
        endif
        if salida>=4900 then
                contx=contx+sumaresta;
                pulsout P0, 30
        endif

;//EVALUA REGLAS con min-max
        pos1=t3 max dt3 ; Regla 1
        pos2=t3 max dt2 ; Regla 2
        pos3=t3 max dt1 ; Regla 3
        neg1=t2 max dt3 ; Regla 4
        cero=t2 max dt2 ; Regla 5
        pos4=t2 max dt1 ; Regla 6
        neg2=t1 max dt3 ; Regla 7
        neg3=t1 max dt2 ; Regla 8
        neg4=t1 max dt1 ; Regla 9
;[1]    If Error=P and d(Error)=P then Voltaje=P
;[2]    If Error=P and d(Error)=Z then Voltaje=P
;[3]    If Error=P and d(Error)=N then Voltaje=P
;[4]    If Error=Z and d(Error)=P then Voltaje=N
;[5]    If Error=Z and d(Error)=Z then Voltaje=Z
```

```
;[6]    If Error=Z and d(Error)=N then Voltaje=P
;[7]    If Error=N and d(Error)=P then Voltaje=N
;[8]    If Error=N and d(Error)=Z then Voltaje=N
;[9]    If Error=N and d(Error)=N then Voltaje=N
       defuzzifica=neg1 min neg2
       defuzzifica2=neg3 min neg4
       neg=defuzzifica min defuzzifica2 ; max(neg1, neg2, neg3, neg4)
       defuzzifica=pos1 min pos2
       defuzzifica2=pos3 min pos4
       pos=defuzzifica min defuzzifica2 ; max(pos1, pos2, pos3, pos4)

;//DEFUZZIFICA
       if salida>=3000 then
              contx=contx+sumaresta;
              pulsout P0, 30
       endif
       if cero>=85 then
              High P2
              if contx>=0 then
                     pos=pos+contx/3
              else
                     neg=neg-contx/3
              endif
       else
              Low P2
       endif
       if neg>=pos then
              salida=((5000*(neg-pos))/(pos+neg+cero))
              Low P1
              sumaresta=-1;
       else
              salida=((5000*(pos-neg))/(pos+neg+cero))
              High P1
              sumaresta=1;
       endif
       debug ["salida", dec salida, 10]
       contx=contx+sumaresta;
       espera=((5001-salida)*10) min 30
       pulsout P0, espera
goto Main
```

A.7. PÓSTER

Control inteligente con aplicación al péndulo invertido

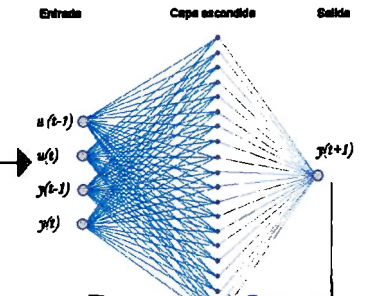
TECNOLÓGICO DE MONTERREY.

Eugenio Álvarez Ibargüengoitia
925188

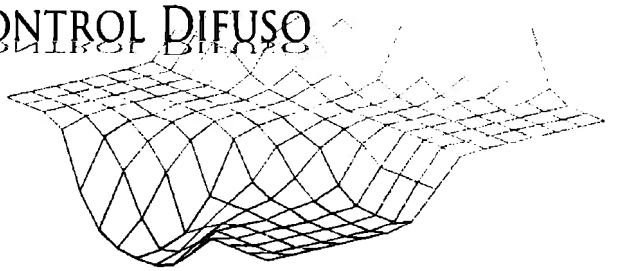
Asesor: Dr. Pedro Ponce Cruz
Profesor: Dr. Ernesto Olguín Díaz

Objetivo

Aplicación de control inteligente al problema del péndulo invertido, empleando lógica difusa, redes neuronales, algoritmos genéticos, y sus integraciones.
Implementar el sistema empleando un controlador PD difuso, que mantenga el péndulo en posición vertical y en el centro de la vía.

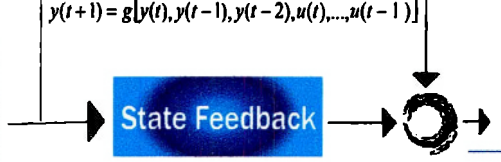


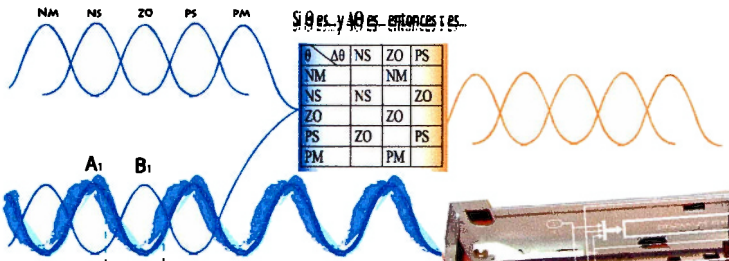
CONTROL DIFUSO



REDES NEURONALES

$$y(t+1) = g[y(t), y(t-1), y(t-2), u(t), \ldots, u(t-1)]$$

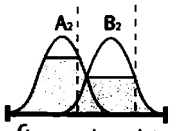




$$\theta = \frac{g\sin\theta + \cos\theta\left(\frac{-\tau - ml\theta^2\sin\theta}{M+m}\right)}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{M+m}\right)}$$

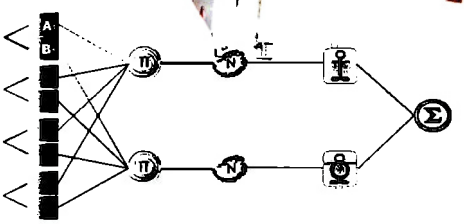
$$x = \frac{\tau + ml\left(\theta^2\sin\theta - \dot{\theta}\cos\theta\right)}{M+m}$$

$$q = \begin{pmatrix}\theta \\ x\end{pmatrix} = H(q)^{-1}\left(\tau - C(q,\dot{q})\dot{q} - g(q)\right)$$



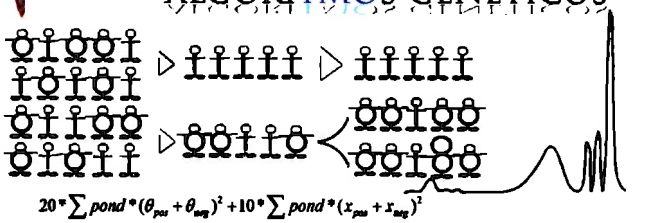
$$f_i = p_i x + q_i y + r_i$$

$$f = f_1\bar{w}_1 + f_2\bar{w}_2$$



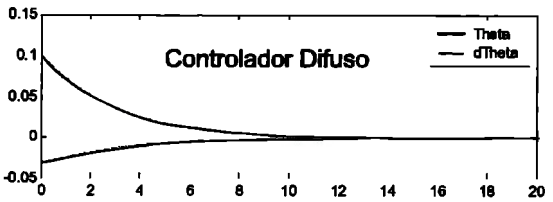
ANFIS

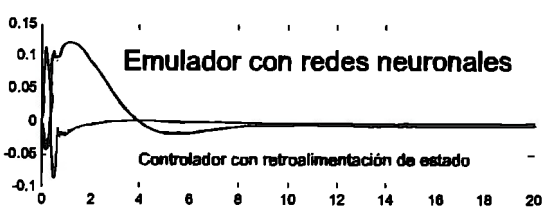
ALGORITMOS GENÉTICOS

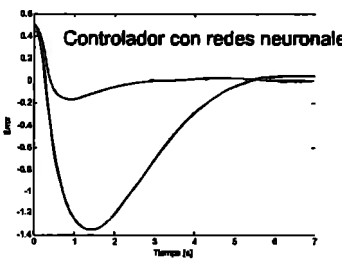


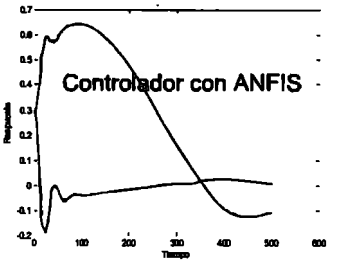
$$20 * \sum pond * (\theta_{pos} + \theta_{neg})^2 + 10 * \sum pond * (x_{pos} + x_{neg})^2$$

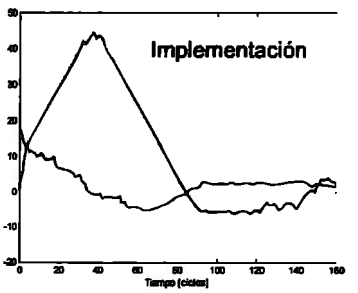
RESULTADOS

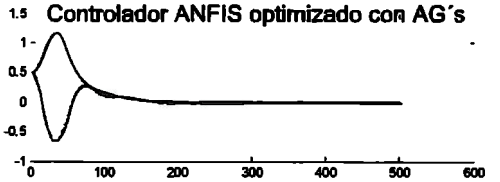












A.8. CONTENIDO DEL CD

- Documento en formato Word
- Video de implementación en formato WMV
- Video de ejemplo de aplicación en formato WMV
- Póster en formato PDF
- Simuladores en formato MDL