UNIVERSIDAD DE LA LAGUNA

«Diseño de políticas de identificación y control de robots basadas en redes neuronales y sistemas neuro-fuzzy.»

Autor: Graciliano Nicolás Marichal Plasencia
Director: Leopoldo Acosta Sánchez

Departamento de Física Fundamental y Experimental, Electrónica y Sistemas

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

CERTIFICA:

Que D. Graciliano Nicolás Marichal Plasencia, Licenciado en Ciencias Físicas, ha realizado bajo mi dirección la presente Tesis Doctoral: DISEÑO DE POLÍTICAS DE IDENTIFICACIÓN Y CONTROL DE ROBOTS BASADAS EN REDES NEURONALES Y SISTEMAS NEURO-FUZZY, para optar al grado de Doctor en Informática.

Con esta fecha autorizo la presentación de la misma.

La Laguna, Marzo de 1999

El Director

Leopoldo Acosta Sánchez

A mi esposa
Angeles
Por su amor
y constante apoyo.

A mis padres
Con cariño.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento, en primer lugar, al Dr. D. Lorenzo Moreno Ruiz por haberme apoyado en todo momento en esta carrera como investigador y docente. Además quiero agradecer sus aportaciones y acertados puntos de vista sobre este trabajo que han ayudado a incrementar la solidez del mismo.

Al Dr. D. Leopoldo Acosta Sánchez a quien le debo gran parte del conocimiento y formación que he alcanzado durante estos años de trabajo, y a quien además he de agradecer la preocupación y motivación que ha puesto en este trabajo.

Al Dr. D. Francisco Mauricio Domínguez, Dr. D. José Peraza Hernández y Dr. Felix Herrera Cabello, cuyas enseñanzas y apoyo han sido un aliciente en la realización de este trabajo.

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

A D. José Julio Rodrigo Bello cuya ayuda resultó imprescindible en la parte final del trabajo.

En especial al Dr. D. Demetrio Piñeiro y a D. Roberto Luis Marichal Plasencia por su inestimable ayuda en el campo de las redes neuronales dinámicas, cuyos diferentes puntos de vista han resultado de gran interés para el trabajo.

En general estoy muy agradecido a mis padres, por haber puesto todo de su parte para que nunca me faltara nada en mi carrera como estudiante y luego como Profesor. Al igual que con todos los familiares que me han apoyado constantemente.

Quiero agradecer especialmente a mi esposa Angeles su constante apoyo en la realización de esta tesis.

ÍNDICE

Capítulo 4

Capítulo 5

Capítulo 6

Capítulo 7

Capítulo 8

Introducción

INTRODUCCIÓN

En la presente memoria presentamos un trabajo de investigación enmarcado dentro del campo de la robótica. Comenzamos en el capítulo 1 dando los fundamentos de las redes neuronales y estableciendo una clasificación de las mismas. Este capítulo constituye una base esencial para el seguimiento de la memoria dado que las redes neuronales se aplican a varios problemas de la robótica. En el capítulo 2 describimos algunos de los problemas que se presentan dentro del mundo de la robótica, haciendo especial hincapié en las soluciones dadas hasta nuestros días y las dificultades que éstas presentan. En el capítulo 3 proponemos una red neuronal dinámica con el objeto de determinar la dinámica de un robot manipulador, obviando el uso de las técnicas clásicas mostradas en el capítulo 2, dado que poseen una alta carga computacional y dan un modelo nominal. En este capítulo aplicaremos la red neuronal al problema de la determinación de la dinámica de un robot manipulador. Los resultados obtenidos nos llevarán a realizar una serie de modificaciones sobre el algoritmo de aprendizaje en pro de obtener mejores prestaciones. Debido al resultado infructuoso de las modificaciones realizadas en el capítulo 4 nos replanteamos el algoritmo de aprendizaje, llegando a una versión diferente del mismo en donde evitamos muchas de las suposiciones hechas en el algoritmo propuesto inicialmente. En este capítulo comprobamos como los resultados mejoran para el ejemplo tratado en el capítulo 3, recogiendo adicionalmente los resultados de otros ejemplos que confirman la bondad de la red neuronal dinámica propuesta. En el capítulo 5 en un afán por simplificar la red neuronal dinámica propuesta introducimos dependencia de unas neuronas respecto a otras de la misma capa, además de proponer otro mecanismo para obtener las salidas a partir de las entradas a la red. Esto nos lleva a una red de 2 capas, con un número de neuronas inferior al de la red neuronal dinámica propuesta inicialmente. Aplicaremos esta red al problema de determinar la dinámica directa de un robot PUMA, utilizando para ello los datos provenientes del modelo de Newton-Euler implementado en simulación.

En el capítulo 6 trataremos el problema de la cinemática inversa. Se aplicarán redes neuronales estáticas como primera forma de afrontarlo obteniendo resultados sobre algunos ejemplos. Como estrategia alternativa se estudia el entrenamiento de las redes neuronales estáticas mediante algoritmos genéticos en vez de emplear el algoritmo de "Backpropagation Standard". Veremos como en uno y otro caso se obtienen resultados similares. Es importante destacar que el problema de la cinemática inversa tiene un notable interés desde el punto de vista del control de los robots manipuladores dado que nos permite especificar de una manera cómoda las tareas a realizar por el robot manipulador determinando las trayectorias deseadas en función de variables más

convenientes desde el punto de vista del control. En el capítulo 7 abordamos el problema del control de los manipuladores partiendo de que podemos obtener los ángulos de las articulaciones correspondientes a la trayectoria deseada a través de la resolución de la cinemática inversa. En este capítulo presentamos una estrategia de control para los manipuladores robóticos basada en redes neuronales. Previamente exponemos algunas de las estrategias de control neuronal propuestas hasta la fecha. Es importante destacar que la determinación de la dinámica a través de redes neuronales dinámicas, mostradas en los capítulos anteriores permite aumentar la eficiencia del control neuronal propuesto, para determinadas situaciones. En el capítulo se presentan varias simulaciones que muestran la eficiencia del algoritmo de control. Por último en el capítulo 8 tratamos el problema del control de los robots móviles. Se comienza presentando una estrategia de control difuso clásica, tomando como ejemplo para exponer los conceptos básicos del mismo dos robots móviles (SULTAN I y RAP) desarrollados por el grupo de Computadoras y Control de la Universidad de La Laguna. Posteriormente se introduce una estrategia de control Neuro-Fuzzy que nos permite de forma automática la determinación de las reglas Fuzzy y de las funciones de pertenencia, así como la optimización del número de reglas a aplicar, todo ello a partir de la información suministrada por un operador. Por último presentaremos varias simulaciones en presencia de diferentes obstáculos para las cuales se obtienen resultados satisfactorios.

Capítulo 1
REDES NEURONALES

1

REDES NEURONALES

En este capítulo se describen los fundamentos de las redes neuronales. Comenzamos dando una breve introducción histórica de su nacimiento y las características más importantes que han hecho de las mismas una herramienta fundamental en la resolución de un gran número de problemas. Después presentamos las tres clases de redes neuronales en función de su arquitectura y su mecanismo de aprendizaje, que utilizaremos en el trabajo, haciendo especial hincapié en los aspectos más relevantes de cada una de ellas.

1.1 INTRODUCCIÓN.

En 1949 D.O. Hebb propuso una ley de aprendizaje que puede considerarse como el punto de partida de los algoritmos de entrenamiento de las redes neuronales, mientras que la primera estructura de red neuronal fue propuesta por Frank Rosenblatt en 1958 [ROS58]. Desde entonces el interés suscitado por las redes neuronales ha crecido enormemente, proponiéndose nuevos modelos que han venido a superar las limitaciones de las redes anteriores, dándoles mayores capacidades.

Las redes neuronales tienen su origen en la observación de los sistemas neuronales de los organismos vivientes, sin embargo, éstas no son un fiel reflejo de dichas redes biológicas.

Como características más relevantes de las redes neuronales artificiales citamos las siguientes:

1- La capacidad de aprendizaje. La naturaleza no lineal de las mismas les permite exhibir comportamientos verdaderamente complejos. Esta característica junto con algoritmos que extraen un mayor rendimiento de tales sistemas les confieren esa capacidad.

2- La posibilidad de procesamiento paralelo, aspecto que aprovechado en la implementación lleva a aumentar considerablemente la velocidad de procesamiento.

3- La significativa tolerancia a fallos, dado que un fallo en unas pocas conexiones locales, rara vez contribuye significativamente al funcionamiento global de la red.

Esta propiedad las hace convenientes en ciertas aplicaciones, donde, una avería puede constituir un serio peligro.

Estas características, entre otras, hacen de las redes neuronales una herramienta fundamental en varios campos del conocimiento.

Las redes neuronales generalmente han sido implementadas mediante la programación sobre computadoras digitales, sin embargo ya en nuestros días empiezan a proliferar implementaciones en hardware, ya sea puramente mediante montajes electrónicos o incorporando dispositivos ópticos. Tales ingenios, presentan un paso importante hacia la consecución de mayores velocidades de procesamiento, así como un camino hacia la popularización de las mismas, siendo ésta un área de investigación muy activa.

En nuestros días la ingeniería de control se enfrenta a sistemas cada vez más complejos, a requisitos de diseño más exigentes y al desconocimiento de las plantas y sus entornos, revitalizando este hecho la investigación en el campo de las redes neuronales.

En cuanto a la clasificación de las redes neuronales podríamos establecer una división de las mismas en función de su arquitectura y forma de procesar las señales en tres clases.

La primera clase de redes se conoce con el nombre de redes neuronales estáticas. En esta clase podemos encontrar la red Multicapa de Perceptrones, la red de Funciones de Base Radial, Red Neuronal Probabilística (PNN), etc. Todas ellas tienen como característica común el no poseer memoria, es decir, sólo son capaces de transformar un conjunto de entradas en un conjunto de salidas, de tal manera que una vez establecidos todos los parámetros de la red las salidas únicamente dependen de las entradas. En general la relación deseada de entradas y salidas se determina en este caso externamente mediante alguna forma de ajuste de los parámetros del sistema supervisado. Este tipo de redes se han empleado con éxito en muchos problemas de clasificación, como funciones lógicas, así como en el campo de la aproximación funcional.

Como segunda clase de redes neuronales debemos nombrar las redes neuronales dinámicas. Estas a diferencia de las anteriores permiten establecer una relación entre salidas y entradas y/o salidas y entradas previas. Esto añade cierta memoria a estas redes. Matemáticamente, esta memoria se traduce en la aparición de ecuaciones diferenciales o ecuaciones en diferencia formando parte del modelo de las mismas. Como ejemplos de este tipo de redes encontramos la red de Hopfield, la red de retardos en el tiempo (Time Delay Neural Network), la red de tiempo discreto (Time Discrete Neural Network), etc. Las redes neuronales dinámicas se han revelado útiles en problemas de modelización de la dinámica directa e inversa de sistemas complejos, tales como robots, cohetes, naves espaciales, etc., así como en la modelización de circuitos secuenciales y en la conversión de texto a voz.

Por último podemos destacar las denominadas redes auto-organizativas, en las cuales los nodos vecinos dentro de la red neuronal compiten en actividad por medio de las interacciones mutuas laterales, y evolucionan adaptativamente hacia detectores específicos de las diferentes señales de entrada. El aprendizaje en este caso se denomina aprendizaje competitivo, no supervisado o autoorganizativo. Estas se han aplicado con éxito en problemas de reconocimiento de patrones, control de procesos e incluso en el procesamiento de información semántica.

En las secciones siguientes comentaremos en más detalle cada una de las clases de redes nombradas en los párrafos anteriores.

1.2 REDES NEURONALES ESTÁTICAS.

Comencemos nuestra exposición de las redes estáticas dando el modelo de neurona más empleado para éstas. En esencia, responde al modelo de neurona introducido por Rosenblatt al que denominó perceptron [ROS58]. Dicha neurona se ilustra en la figura 1.1.

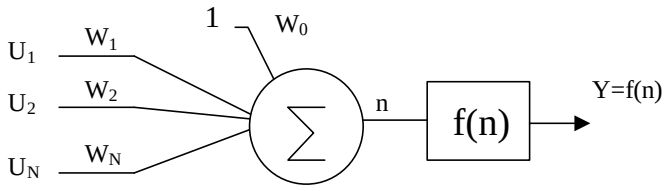


Figura 1.1 – Perceptron.

Como se puede ver en la Figura (1.1) las entradas se multiplican por una serie de pesos, siendo la suma de esos productos la variable independiente de una función f(x). Dicha función recibe el nombre de función de activación, en analogía con las neuronas biológicas que inspiraron su creación. Además suele ser una función no lineal. En el modelo propuesto originalmente se tomó la función salto unidad.

$$f(n) = \begin{cases} 1 & n > 0 \\ 0 & n \leq 0 \end{cases} \qquad (1.1)$$

El resultado de la aplicación de esa función a la suma de productos nos dará la salida de la neurona. Se suele adicionar un peso más que no es multiplicado por ninguna entrada y sirve como una entrada de referencia. Este nuevo peso contribuye generalmente a hacer el aprendizaje más rápido.
Una neurona de este estilo, por tanto, vendrá representada por las ecuaciones que se muestran a continuación:

$$n = \sum_{i=1}^{N} w_i u_i + w_0 \qquad (1.2)$$

$$Y = f(n) \qquad (1.3)$$

Donde :

N = número de entradas.
u_i = entrada i-ésima.
w_i = peso i-ésimo.
w_0 = entrada de referencia (peso bias)
f(n) = función de activación
Y = Salida de la neurona.

A pesar de la simplicidad del perceptron, éste es capaz de simular diferentes funciones lógicas, aun cuando el número de las mismas está limitado, denominándose a este límite, umbral de funciones lógicas. También se ha empleado en problemas de

clasificación de patrones dentro de dos clases en donde la limitación fundamental se encuentra en que la frontera entre las dos clases puede ser únicamente un hiperplano.
Existen varios algoritmos de entrenamiento diseñados para el perceptron. Entre ellos podemos citar el algoritmo de los mínimos cuadrados (LMS) que tiene la peculiaridad de ser un caso particular del algoritmo backpropagation, algoritmo propuesto por Werbos en 1974 [WER74].
Tal como hemos observado el perceptron por si sólo tiene ciertas limitaciones, varios investigadores realizaron experimentos con una capa de perceptrones, al objeto de conseguir aprender ciertas funciones que un único perceptron no permitía aprender. El resultado de tales experimentos fue publicado por Marvin Minsky y Parpert en 1969 en su libro titulado "Perceptrons" [MIN69] donde ponían de manifiesto la incapacidad de una capa de perceptrones para aprender ciertas funciones, incluyendo la función lógica or-exclusiva. Este resultado abrió un período de pesimismo frente a las posibilidades de las redes neuronales estáticas. De hecho tales resultados llevaron al abandono de las redes neuronales durante dos décadas. Sin embargo gracias al esfuerzo de unos pocos investigadores tales como Teuvo Kohonen, Stephen Grossberg y James Anderson las redes neuronales volvieron a coger auge, dado que demostraron que utilizando varias capas de neuronas se podían solucionar problemas que con una capa eran irresolubles tales como el aprendizaje de una función or-exclusiva. A partir de entonces han surgido diferentes topologías de redes neuronales.
La topología más general de red neuronal, es aquella en donde un conjunto de neuronas se agrupan entre sí. Las salidas de unas neuronas hacen la función de entradas de otras, siendo algunas de las salidas escogidas como salidas de la red neuronal. Un cierto subconjunto de neuronas recibirán las entradas a la red. Este tipo de red neuronal recibe el nombre de red neuronal completamente conexionada, dado que contempla todas las posibilidades de uniones entre neuronas. Un esquema de dicha topología se puede observar en la siguiente figura.

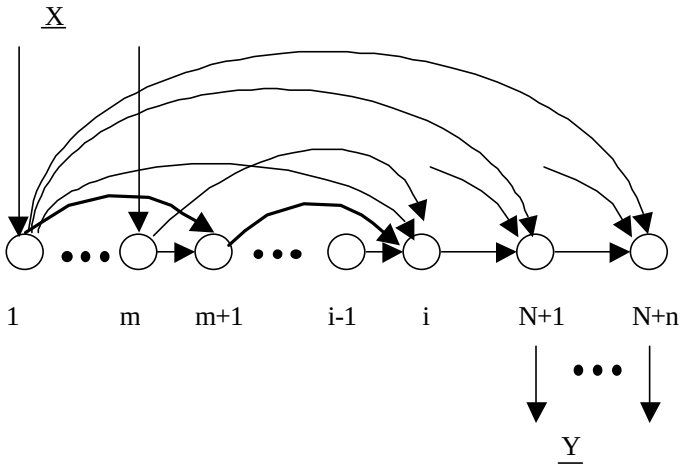


Figura 1.2 – Topología general de una red neuronal.

Donde :

m = número de entradas.
n = número de salidas.
N = indica el número de neuronas existentes sin contar las que dan las salidas de la red.
$\underline{X}$ = vector de entradas a la red.

$\underline{Y}$ = vector de salidas de la red.

Sin embargo la mayoría de investigadores prefieren una organización de la red en forma de capas, es decir las neuronas se posicionan en forma de capas unas detrás de otras, tal que las salidas de las neuronas de unas capas sean las entradas de las neuronas de la capa siguiente. A este tipo de redes en donde las neuronas se disponen en capas sin realimentaciones entre ellas se les da el nombre de redes neuronales feedforward. En estas redes estructuradas por capas se suele distinguir entre la capa de entrada, o sea la capa de neuronas que recibe las entradas a la red, la capa de salida, es decir la capa que contiene las neuronas cuyas salidas serán las que constituyan las salidas de la red y las capas ocultas que son las capas existentes entre las dos anteriores. En la figura 1.3 se muestra una red de este tipo constituida por tres capas.

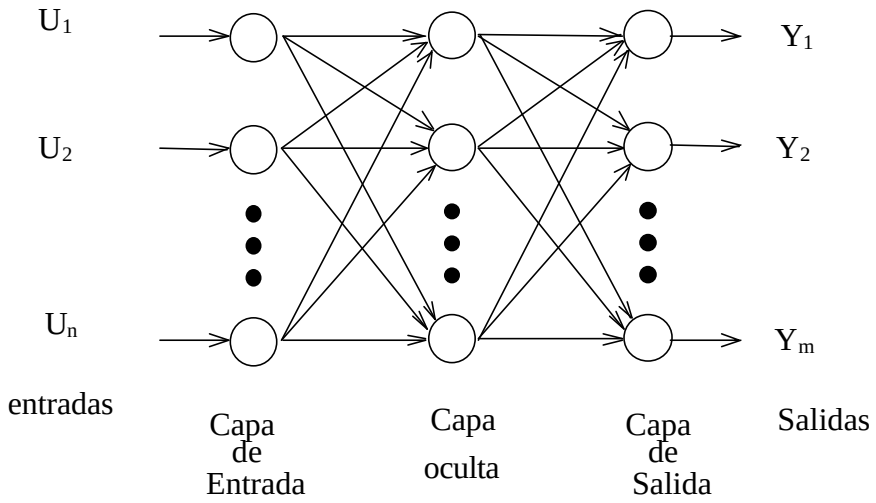


Figura 1.3 – Red neuronal de 3 capas.

Este es el caso de la red Multicapa de Perceptrones (MLP), en donde el modelo de neurona adoptado es el perceptron, ilustrado en los párrafos anteriores. Sin embargo en este caso la función de activación deja de ser la función salto unidad para convertirse en la función sigmoide, la tangente hiperbólica o cualquier otra función no lineal que sea continua, suave y monótonamente creciente. La función salto unidad presenta el inconveniente de no ser derivable, circunstancia ésta, que hace más difícil el diseño de algoritmos de entrenamiento basados en el gradiente.

En ocasiones se han utilizado funciones de activación lineales para las neuronas de la capa de salida, motivado por el hecho de facilitar el aprendizaje.

En general se han aplicado varios algoritmos de entrenamiento a la MLP, siendo el más usado el “backpropagation”. A continuación presentamos de forma resumida el pseudocódigo correspondiente a este algoritmo.

Procedimiento BACK_PROP /* procedimiento principal */
    Inicializar los pesos a valores pequeños al azar
    Repetir
        Elegir el próximo par de entrenamiento (x,d).
        Establecer las entradas de la primera capa como u0=x;
        FEED_FORWARD;
        COMPUTE_GRADIENT;

UPDATE_WEIGHTS;
Hasta que la condición de finalización sea alcanzada;
Final; (BACK_PROP)

Subrutina FEED_FORWARD /* calcula salidas de la red a partir de las entradas */
Desde *capa*=1 **hasta** *capa*=L
Desde *nodo*=1 **hasta** *nodo*= Ncapa

$$u_{capa,nodo} = f\left(\sum_{i=0}^{N_{capa-1}} w_{capa,nodo,i}\, u_{capa-1,i}\right)$$

Final bucle;
Final bucle;
Final Subrutina; (FEED_FORWARD)

Subrutina COMPUTE_GRADIENT /* calcula el gradiente */
Desde *capa*=L **hasta** *capa*=1
Desde *nodo*=1 **hasta** *nodo*=Ncapa
Si *capa*=L **entonces** $e_{L,nodo} = u_{L,nodo} - d_{nodo}$;
En cualquier otro caso

$$e_{capa,\,nodo} = \sum_{m=1}^{N_{capa-1}} e_{capa+1,m}\, u_{capa+1,\,m}\left(1-u_{capa+1,m}\right) w_{capa+1,m,nodo}\ ;$$

Final bucle;
Para todos los pesos en la capa *capa*

$$g_{capa,\,j,i} = e_{capa,\,j}\, u_{capa,\,j}\left(1-u_{capa,\,j}\right) u_{capa-1,\,i}\ ;$$

Final bucle;
Final bucle;
Final Subrutina; (COMPUTE_GRADIENT)

Subrutina UPDATE_WEIGHTS /* actualiza los pesos */
Para todos los pesos de la red

$$w_{l,\,j,i}(k+1) = w_{l,\,j,i}(k) - m\, g_{l,\,j,i}$$

Final bucle;
Final Subrutina; (UPDATE_WEIGHTS)

Obsérvese que μ es un número real cuyo valor determina la influencia del valor del gradiente en la actualización de los pesos, dándosele el nombre de velocidad de aprendizaje.
La MLP conjuntamente con el “backpropagation” se han empleado con éxito en ciertos problemas de clasificación, pudiendo generar fronteras entre las diferentes clases mucho más complejas que en el caso del perceptron. Investigadores tales como R. P. Lippman [LIP87], G. Cybenko [CYB89] o Chester [CHE90] demostraron de forma teórica que una red MLP de 2 capas es capaz de aproximarse a cualquier función no lineal continua, así como implementar cualquier frontera no lineal entre varias clases. Estas características han permitido introducir las redes neuronales en el mundo del control. Citamos a título de ejemplo las aplicaciones a problemas de optimización, en donde se ha revelado como estrategia alternativa a la programación dinámica o la posibilidad de implementar controladores basados en redes neuronales.

Sin embargo el "backpropagation" es un algoritmo inherentemente recursivo, propiedad que se manifiesta de forma negativa en la velocidad de procesamiento del mismo, aunque hemos de destacar que el hecho de ser un algoritmo susceptible de cierta paralelización contribuye a contrarrestar este efecto. Una de las dificultades más significativas de la aplicación de las redes neuronales es la no existencia de métodos para elegir el tamaño óptimo de la red según la aplicación, en general se deja en manos de la experiencia. Con el objeto de salvar esta dificultad ciertos algoritmos de entrenamiento toman en consideración el variar el tamaño de la red hasta obtener un resultado óptimo. Una característica que condiciona también el aprendizaje es la irregularidad de la superficie de error. Esto contribuye, por un lado a confundir puntos sobre la superficie de error como mínimos globales, mientras que por otro aparecen saltos abruptos que limitan considerablemente la velocidad de aprendizaje, dado que las velocidades de aprendizaje excesivas llevan a inestabilizar el algoritmo. Con el fin de resolver este problema los investigadores introducen términos adicionales en las fórmulas de corrección de los pesos, de tal forma que las actualizaciones sobre los pesos dependan de las actualizaciones previas. De igual forma se intenta evitar el problema de que el algoritmo de aprendizaje quede atrapado en un mínimo local por medio de métodos especiales de inicialización de los pesos [WES92] [DRA92], mejorando considerablemente los resultados.

En la figura 1.4 se ilustra como sería el aspecto de una superficie de error típica. Como se puede observar esta curva presenta un carácter irregular que dificulta el aprendizaje, contribuyendo a confundir mínimos locales con mínimos globales [HUS93].

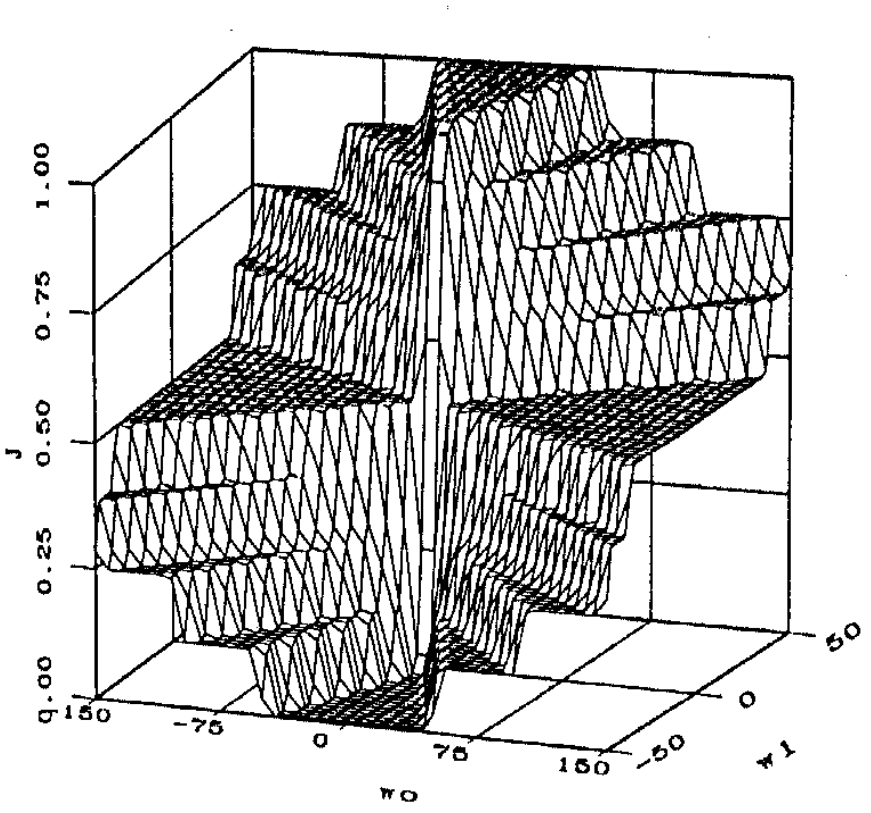


Figura 1.4 – Superficie de error típica.

Donde:

J = Función de costo, que da la bondad del ajuste.

W_0 y W_1 = dos pesos de la red.

Una vez comentados todos los problemas concernientes al entrenamiento, hemos de hacer un análisis sobre la bondad de los resultados. Una vez la red ha sido entrenada se ha de observar la generalización de los resultados. Una buena correspondencia en el caso particular de los pares de entrada/salida de entrenamiento no implica un buen comportamiento general. La generalización se encuentra ligada a varios factores, entre los que citamos: el tamaño de red, la relación subyacente a aprender y el número de muestras de que se disponga. Normalmente se influye sobre la generalización a través del número de muestras y el tamaño de la red, según las posibilidades que permita la aplicación, con lo que las acciones adoptadas en pro de una mejor generalización quedan limitadas a la aplicación particular.

Otra de las redes neuronales representativas de las redes neuronales estáticas es la red de funciones radiales, conocida como Radial Basis Function (RBF) en la terminología inglesa. Esta consiste en tres capas de neuronas, donde las neuronas de la capa oculta tienen una función de activación Gaussiana, mientras que las neuronas de la capa de salida son neuronas lineales. La utilización de funciones de activación gaussianas hace que la respuesta de las neuronas de la capa oculta sea distinta de cero dentro de una cierta región localizada del espacio de entrada. Debido a esta característica a este tipo de red se le conoce también con el nombre de red de campo receptivo localizado. En la figura 1.5 mostramos un esquema de la red.

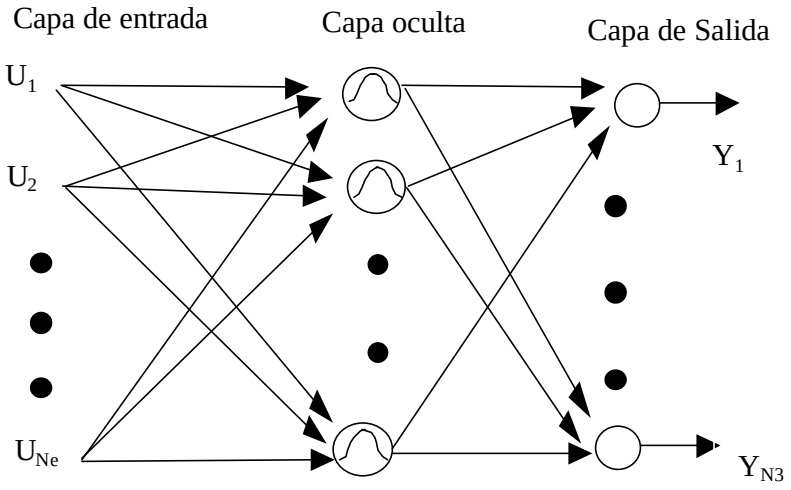


Figura 1.5 – Red neuronal de funciones base radiales.

Teniendo en cuenta los comentarios realizados en el párrafo anterior, las ecuaciones son:

$$u_{1j} = \exp\left(-\frac{\left(\mathbf{U}-\mathbf{w}1_j\right)^T \left(\mathbf{U}-\mathbf{w}1_j\right)}{2\,\sigma_{ij}^{\,2}} \right) \qquad \text{j} = 1,2,...,\text{N1} \qquad (1.4a)$$

$$y_j = \mathbf{w}_{2,j}^T \mathbf{u}_1 \qquad \text{j} = 1,2,...,\text{Ne} \qquad (1.4b)$$

Donde:

u_{1j} = Salida de la neurona j de la capa oculta.
y_j = Salida j-ésima de la red.
U = Vector de entradas a la red.
$\mathbf{u}_1$ = Vector de salidas de la capa oculta cuyas componentes son u_{1j}.
$\mathbf{w}_{1j}$ = Vector de pesos correspondiente a la neurona j de la capa oculta.
$\mathbf{w}_{2j}$ = Vector de pesos correspondiente a la neurona j de la capa de salida.
N1 = Número de neuronas en la capa oculta.
Ne = Número de salidas de la red.
s_{ij} = Parámetro de normalización de la neurona j de la capa oculta.

Las redes neuronales de base radial se han utilizado en problemas de aproximación funcional, en donde las funciones se aproximan por una combinación lineal de funciones gaussianas. Sin embargo la RBF realiza esta aproximación combinando funciones gaussianas que poseen una respuesta localizada en una cierta región del espacio de entradas. Esto nos lleva a considerar una MLP o una RBF según el problema sea abordable mediante la combinación de uno u otro tipo de funciones de activación. En general la RBF se comporta mejor en problemas de clasificación, sobre todo cuando la dimensión del problema aumenta, mientras que en problemas de aproximación funcional parece más conveniente la utilización de la MLP.

En cuanto a los algoritmos de aprendizaje para la RBF, el más común es dividir el proceso de aprendizaje en dos fases. Primero mediante algún algoritmo no supervisado ajustamos los parámetros de la capa oculta, para en una segunda fase del entrenamiento ajustar los parámetros de la capa de salida, sirviéndonos de algún tipo de algoritmo supervisado. En ocasiones el proceso de aprendizaje se completa aplicando simultáneamente un algoritmo de aprendizaje supervisado a la capa oculta y la capa de salida, con objeto de refinar los parámetros obtenidos en el proceso de aprendizaje previo.

El problema de la generalización en las redes RBF presenta las mismas características que en el caso de las redes MLP, estando ligado a la aplicación concreta en la que se utilizan.

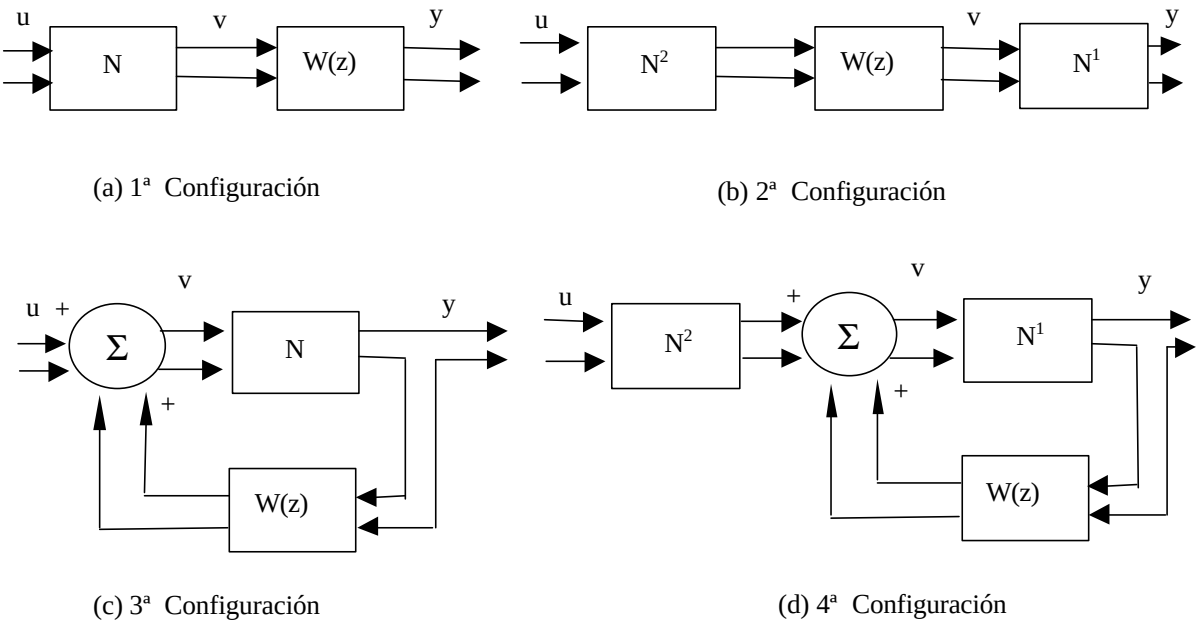
Existen otras arquitecturas de redes neuronales estáticas, sin embargo las mostradas nos dan una idea clara de las características más importantes de esta clase de redes. Como se ha visto las redes neuronales estáticas vienen caracterizadas por la ausencia de cualquier comportamiento dinámico, no existen dependencias respecto a instantes pasados. Sin embargo existen aplicaciones donde esta característica es de suma importancia, tales son el aprendizaje del comportamiento de los sistemas dinámicos o la emulación de circuitos secuenciales. Esta es una de las motivaciones que llevaron a la introducción de las redes neuronales dinámicas. En la sección siguiente daremos una descripción más detallada de las mismas.

1.2 REDES NEURONALES DINÁMICAS.

Las redes neuronales dinámicas se caracterizan por tener una dependencia de sucesos ocurridos en instantes pasados [NER93]. Desde un punto de vista matemático esto se refleja en el hecho de que en las ecuaciones que las definen aparecen ecuaciones diferenciales o ecuaciones en diferencia según sean redes continuas o discretas. Son éstas las que le confieren una cierta memoria, que resulta imprescindible en problemas tales como la identificación de sistemas dinámicos. La principal atracción de este tipo

de redes es que tienen memoria interna y por lo tanto son en sí mismas un sistema dinámico.

Como primer intento de incorporar un comportamiento dinámico en las redes neuronales se diseñaron redes neuronales dinámicas constituidas por una combinación entre redes neuronales estáticas y sistemas dinámicos. En la figura 1.6 representamos cuatro configuraciones conectadas en cascada y con realimentaciones que representan las unidades básicas con las cuales se construye cualquier red dinámica de este tipo[NAR91]. Obsérvese que W(z) hace referencia a la matriz de transferencia del sistema dinámico y N es la red neuronal estática o dado que podemos ver las redes neuronales estáticas como operadores no lineales, también se puede considerar como un operador no lineal. Cuando en las configuraciones de la figura 1.6 utilizamos varias redes neuronales estáticas denotamos cada una de ellas mediante un superíndice. Mediante estas configuraciones, considerando también el operador multiplicación por una constante, podemos construir una gran variedad de redes dinámicas discretas. Hemos de comentar que las redes consideradas son discretas, sin embargo el análisis se puede extender a las redes dinámicas continuas sin más que tomar matrices de transferencia en el dominio s (W(s)) correspondientes a sistemas dinámicos continuos. Lo que se traduce en el dominio temporal en sustituir los operadores retardo por integradores.



(a) 1ª Configuración

(b) 2ª Configuración

(c) 3ª Configuración

(d) 4ª Configuración

Figura 1.6 – Configuraciones básicas de redes neuronales dinámicas.

Dentro de esta clase de redes encontramos como aproximación más sencilla la red de retardos en el tiempo o Time Delay Neural Network (TDNN) [LAP87] [LAN90] [HER91]. En esta red el sistema dinámico se introduce entre las entradas y la red estática, de tal forma que el sistema dinámico actúa retardando las entradas. Esta red neuronal ha sido aplicada con éxito en problemas de conversión de texto a voz, así como en la predicción de series temporales no lineales. Otra red neuronal que ha suscitado un gran interés es la red de Narendra, en donde además de retrasar las entradas, realimentamos las salidas de la red a través de un sistema dinámico. Esta arquitectura corresponde a la combinación de la primera y tercera configuración de la

figura 1.6, atribuyéndose el nombre genérico de redes con realimentación de las salidas a las redes que realimentan las salidas con el objeto de introducir un comportamiento dinámico. En teoría este tipo de redes son capaces de modelizar cualquier sistema dinámico cuya relación entre la entrada y salida se pueda expresar como:

$$y(k) = F\left[u(k), u(k-1), ..., u(k-n), y(k-1), y(k-2), ..., y(k-m)\right] \qquad (1.5)$$

Donde:

y = salidas de la red.
u = entradas a la red.
m = número de salidas previas tomadas.
n = número de entradas previas tomadas.

Otro tipo de redes dinámicas son las denominadas redes con realimentación de estados. Estas consisten en una capa de neuronas interconectadas entre sí como se muestra en la figura 1.7. En general muchos autores denominan a las redes dinámicas donde hay alguna clase de realimentación redes neuronales recurrentes [MAR98] [WIL89]. La introducción de realimentaciones favorece el aprendizaje. Este hecho se basa en que un sistema con realimentación es equivalente a un gran sistema feedforward, posiblemente con un número infinito de unidades. Así es bien conocido que un circuito secuencial es equivalente a un circuito con un número infinito de puertas lógicas colocadas en forma de capas unas detrás de las otras.

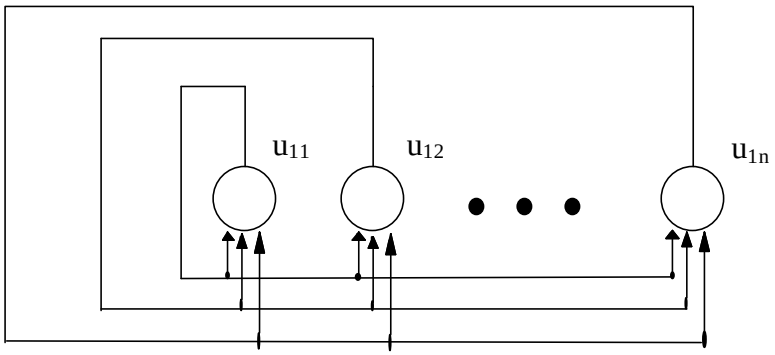


Figura 1.7 – Redes con realimentación de estados.

Una de las primeras redes propuestas de este tipo es la red de Hopfield [HOP82] [HOP84]. Debido a este hecho, ha sido una de las más estudiadas. Según tomemos el tiempo de forma continua o a intervalos regulares tenemos la red de Hopfield de tiempo continuo o la red de Hopfield de tiempo discreto. Aquí únicamente consideraremos la red de Hopfield de tiempo discreto ilustrando las ideas implícitas a este modelo de red.
Los nodos de esta red vienen caracterizados por las ecuaciones en diferencia:

$$y_i(k) = \sum_{j=1}^{N} w_{ij}\, u_j(k) + \theta_i \qquad (1.6a)$$

$$u_i(k+1) = f(y_i(k)) \qquad (1.6b)$$

Donde:

u_j = Salida de la neurona j.
ν_i = Entrada externa a la neurona i.
N = Número de neuronas en la primera capa.

Hemos de destacar que para la red de Hopfield se han empleado diferentes funciones de activación. En problemas de reconocimiento de patrones la función mas comúnmente utilizada ha sido la función salto unidad. Se ha hecho un estudio detallado de sus propiedades de estabilidad y convergencia. Mediante la aplicación del segundo método de Lyapunov, se ha demostrado como condición suficiente para la estabilidad el que la matriz de pesos sea simétrica y definida positiva. Son estos principios los que se utilizan en la denominada memoria de Hopfield asociativa, una de las aplicaciones más frecuentes de la red de Hopfield, en donde se asignan los pesos de tal forma que el sistema posea los puntos de equilibrio adecuados a la aplicación. Aunque la aplicación más común de las redes de Hopfield es en problemas de memoria asociativa, también se ha empleado en otros problemas tales como la descomposición de señales en un conjunto específico de funciones base. Esta propiedad ha sido utilizada en la identificación de sistemas en el dominio frecuencial.

Otra red de características similares a la red neuronal de Hopfield es la red neuronal recurrente de tiempo discreto [PIN88] [WIL89]. Sus nodos vienen descritos por las ecuaciones mostradas a continuación:

$$u_i(k+1) = f\left(\sum_{j=0}^{M+N} w_{ij} u_j(k)\right) \qquad (1.7a)$$

Donde:

$$u_i(k) = \begin{cases} 1 & i = 0 \\ u_i(k) & i = 1,2,...,N \\ \boldsymbol{n}_{i-N}(k) & i = N+1,...,N+M \end{cases} \qquad (1.7b)$$

Obsérvese que:

N = Número de salidas.
M = Número de entradas externas.
u_i = Salida de la neurona i.
ν_{i-N} = Salida i-N.

Esta red en la terminología inglesa se conoce con el nombre de “Discrete-Time Recurrent Neural Network” y de forma abreviada DTRNN. Ha sido aplicada con éxito en el aprendizaje de autómatas deterministas finitos.

En cuanto a los algoritmos de aprendizaje, estos son más complejos que los algoritmos empleados en el caso de las redes neuronales estáticas. En general existen dos aproximaciones a la hora de elaborar nuevos algoritmos de aprendizaje para redes dinámicas. Por un lado, existen algoritmos que tratan de desdoblar a lo largo del tiempo la red neuronal de tal forma que podamos aplicar las ideas vistas para las redes estáticas, así uno de tales algoritmos es el algoritmo de aprendizaje “Backpropagation Through Time” propuesto por Werbos [WER90], donde se propaga el error en vez de a través de las capas como en el caso de la MLP a través del tiempo. Como una simplificación de este algoritmo surge el algoritmo “Truncated Back-propagation Through Time”, donde se toman únicamente un cierto número de instantes a la hora de calcular el gradiente. Por otra parte existen otros algoritmos de aprendizaje basados en calcular de forma recursiva el gradiente, uno de ellos es el “Real Time Recurrent” Learning (RTRL)

[WIL89]. Desarrollemos este algoritmo en este punto dado que será utilizado a lo largo de esta memoria.

Teniendo en cuenta la función de costo:

$$J(w) = \sum_{p=1}^{P} J_p(w) \qquad (1.8)$$

Donde:

J(w) = Función de costo global.
$J_P(w)$ = Función de costo parcial, para la secuencia p.
P = Número de secuencia de entrenamiento.

Cada una de estas funciones de costo parciales vienen definidas de la siguiente forma, en concreto para la secuencia p-ésima:

$$J_p(w) = \frac{1}{2}\sum_{k=1}^{k_p}\sum_{j\in\Omega}\left(d_j(k) - y_j(k)\right)^2 \qquad (1.9)$$

Donde:

k_P = Número de pares de entrenamiento en la secuencia p-ésima.
$d_j(k)$ = Salida deseada j-ésima correspondiente al par de entrenamiento k-ésimo de la secuencia p-ésima.
$y_j(k)$ = Salida j-ésima correspondiente al par de entrenamiento k-ésimo de la secuencia p-ésima.
Ω = Conjunto de salidas de la red.

La fórmula de actualización de los pesos vendrá dada por:

$$w_{ji}(m+1) = w_{ji}(m) - m\sum_{p=1}^{P}\frac{\partial J_p(w)}{\partial w_{ji}}\Big|_{w(m)} \qquad (1.10)$$

Siendo:

m = Velocidad de aprendizaje.

Obsérvese que el índice m hace referencia al tiempo de actualización de los pesos, mientras k, hace referencia al tiempo en la adquisición de los pares de entrenamiento. El problema que se nos plantea ahora es como calcular la derivada parcial de la función de costo respecto a los pesos, es decir el gradiente de la misma respecto a los pesos. Estas derivadas parciales pueden escribirse en la forma siguiente:

$$\frac{\partial J_p(w)}{\partial w_{ji}} = -\sum_{k=1}^{Kp}\sum_{h\in\Omega}\left(d_h(k) - u_h(k)\right)p_{ji}^{h}(k) \qquad (1.11)$$

Donde:

$$p_{ji}^{h}(k) = \frac{\partial u_h(w)}{\partial w_{ji}} \qquad (1.12)$$

Desarrollando esa expresión queda:

$$p_{ji}^{h}(k) = f'\left(\sum_{a=0}^{N+M} w_{h,a}\, y_a(k-1)\right)\left[\delta_{hj}\, y_i(k-1) + \sum_{b=0}^{N} w_{h,b}\, p_{ji}^{b}(k-1)\right] \quad (1.13)$$

Aquí δ_{hj} es la función delta de Kronecker. Sustituyendo los resultados obtenidos en la ecuación (1.10) tendremos la siguiente fórmula de actualización para los pesos:

$$w_{ji}(m+1) = w_{ji}(m) - \eta \sum_{p=1}^{p} \sum_{k=1}^{k_p} \sum_{h\in\Omega} \left(d_h(k) - y_h(k)\right)\, p_{ji}^{h}(k) \quad (1.14)$$

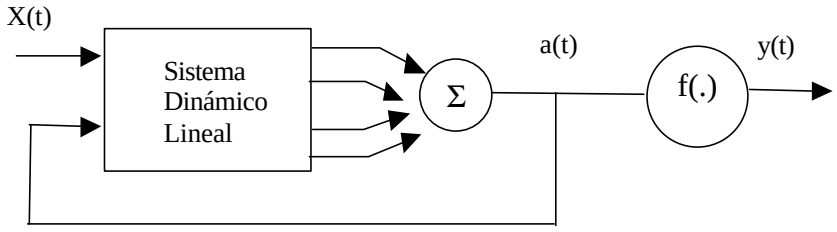
La expresión (1.14) se suele simplificar, tomando únicamente la derivada de una de las J_p en cada instante. Tomando en consideración esta aproximación, la ecuación de actualización de pesos quedaría como:

$$w_{ji}(m+1) = w_{ji}(m) - \eta \frac{\partial J_{m \bmod p}(w)}{\partial w_{ji}}\Bigg|_{w(m)} \quad (1.15)$$

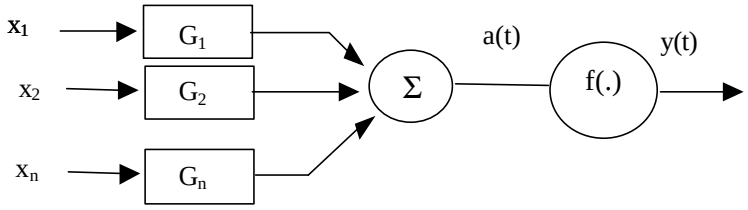
Obsérvese que mod es la operación que divide los operandos enteros y nos da el resto.

Como se puede observar, en el RTRL se vuelve a encontrar la filosofía del backpropagation, en donde a partir de una señal de error de las salidas, se elaboran las correcciones de los pesos. Sin embargo al tratarse de una red de una capa este algoritmo no genera la propagación de los errores de las salidas hacia las capas previas.

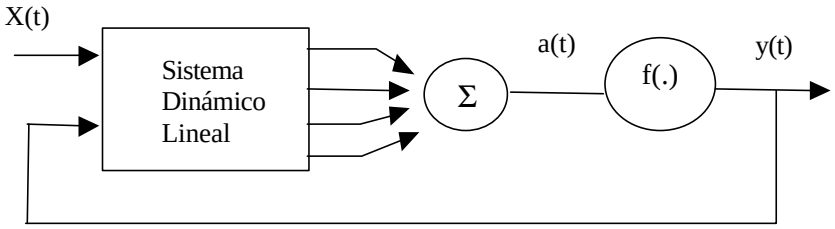
En contraposición a las redes mostradas hasta ahora aparecen las denominadas redes localmente recurrentes globalmente feedforward, abreviadamente se las conoce como redes LRGF (Locally Recurrent Globally Feedforward Neural Neworks) [TSO94] [SUY97]. Estas redes se encuentran entre las redes estáticas feedforward y las redes recurrentes en donde todas las neuronas están interconectadas entre sí. Por lo tanto constituyen una nueva forma de incorporar información del pasado. En las mismas las realimentaciones se producen a nivel del modelo de neurona, es decir, las realimentaciones son locales a cada neurona. Según donde se produzca esta realimentación dentro de las neuronas estas se dividen en tres grupos: de realimentación de la activación, de realimentación de la sinapsis y de realimentación de la salida. En la figura 1.9 presentamos esquemáticamente cada uno de los modelos de neuronas.



(a) neurona con realimentación de la activación



(b) neurona con realimentación de la sinapsis.



(c) neurona con realimentación de la salida

Figura 1.9 – Modelos de neuronas para las redes LRGF.

Debemos de destacar que las redes LRGF con realimentación en la activación son un caso particular de las redes LRGF con realimentación de la sinapsis. Existen varias redes neuronales que responden a una arquitectura LRGF. En el caso de la redes LRGF con realimentación en la activación encontramos la red de Frasconi-Gori-Soda [GOR90], mientras que redes LRGF con realimentación de sinapsis son la propia red de Frasconi-Gori-Soda, la red de Back-Tsoi, la red de De Vries-Principe [VIR92], y por último redes LRGF con realimentación de la salida son la red de Frasconi-Gori-Soda, la red de Poddar-Unnikrishan [POD91a] [POD92b] o la red de neuronas de Memoria, abreviadamente conocida como MNN (Memory Neuron Network) [SAS94]. Las arquitecturas de todas estas redes se pueden ver como casos particulares de la denominada red LRGF general propuesta por Back-Tsoi. En la figura 1.9 presentamos el esquema de la misma.

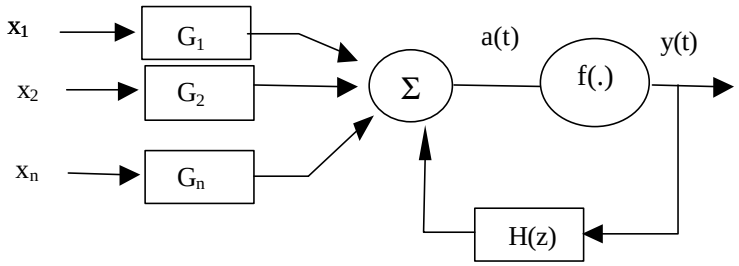


Figura 1.9 – Red LRGF general.

Las redes LRGF se han mostrado convenientes en la resolución de ciertos problemas, así la red De Vries-Principe se ha presentado bastante eficiente en la modelización de señales encefalográficas, mientras que la red de Frasconi-Gori-Soda presenta un buen comportamiento en el procesamiento de señales acústicas. Mediante la combinación de redes neuronales estáticas y sistemas dinámicos tal como se señala en la figura 1.6 se pueden construir un gran número de sistemas, sin embargo se necesita un buen

conocimiento de los sistemas dinámicos para decidir que combinación de redes neuronales estáticas y sistemas dinámicos es adecuada para el problema particular. Como principal problema hemos de determinar a priori el orden de los sistemas dinámicos que intervienen en esa combinación, sin embargo en las redes LRGF no hay que hacer suposiciones a priori sobre el orden de los sistemas. Esta característica es bastante deseable en los métodos de control dado que en los métodos de control empleados habitualmente como el regulador auto-sintonizable (STR) o el control adaptativo por modelo de referencia [NAR89] [GOO84] [AST80] se exhibe dependencia frente al modelo. Otra de las características que suscita más interés de las redes LRGF es que son más simples que las redes recurrentes.

1.4 REDES AUTO-ORGANIZATIVAS.

Las redes auto-organizativas se entienden como una alternativa a las arquitecturas de redes neuronales presentadas en los apartados anteriores. En 1980 Shun-Ichi Amari [AMA80] publicó un trabajo que sentaba las bases del aprendizaje de este tipo de redes, siendo propiamente desarrollado el aprendizaje auto-organizativo a principios de 1981 por Teuvo Kohonen [KOH87]. En este tipo de redes los nodos se ajustan específicamente a varios conjuntos de señales o clases de patrones mediante un proceso de aprendizaje no supervisado. En la versión más básica, solamente un nodo o un grupo local de ellos responde al mismo tiempo a la entrada actual. La localización de los nodos dentro de la red se corresponde con un dominio particular del espacio de entradas. Los nodos se van ordenando como si hubiese un sistema de coordenadas para las diferentes características de las entradas a lo largo de la red. Cada nodo o conjunto de los mismos actúa como un decodificador individual frente a las entradas.

Aunque las redes auto-organizativas se han desarrollado más desde un punto de vista de la ingeniería que desde un estudio puramente neurológico, las representaciones internas de la información en el cerebro están generalmente organizadas espacialmente. Así se ha observado que determinadas funciones se encuentran localizadas en determinadas áreas del cerebro. Esta característica es común a las redes de aprendizaje auto-organizativo, siendo gobernadas éstas últimas por principios similares a los que rigen el cerebro. Esta apreciación justifica el hecho de tomarlas en este capítulo como redes neuronales artificiales.

El algoritmo más comúnmente utilizado en estas redes es el algoritmo de aprendizaje auto-organizativo de Kohonen. En resumen éste se basa en la concentración espacial de la actividad de la red sobre los nodos que mejor se ajustan a la entrada actual y en el ajuste adicional del nodo más cercano y sus vecinos a la entrada actual. Este algoritmo ha sido aplicado con éxito en problemas de clasificación de patrones, en diferentes problemas del campo de la robótica, en el control de procesos, etc. En el caso de los problemas de clasificación, con objeto de aumentar la precisión a la hora de clasificar se han empleado adicionalmente otros algoritmos basados en aprendizaje supervisado como el LVQ (learning vector quantization) [LIN80] [GRA84] [KOH90]. Antes de concluir este apartado sería necesario especificar que aun cuando este tipo de redes es diferente de los presentados en las secciones anteriores carece al igual que en las redes estáticas de comportamiento dinámico. Este hecho hace que algunos autores las incluyan dentro de las redes estáticas, aunque en esta memoria hemos preferido considerarlas como un tipo de redes diferente.

Capítulo 2
ROBOTS MANIPULADORES

2

ROBOTS MANIPULADORES

Aunque los contenidos de un capítulo como éste aparecen desde hace tiempo en los libros de texto, sin embargo, por la notación y por la claridad expositiva los presentamos aquí, haciendo especial hincapié en los problemas que surgen desde el punto de vista del control. Empezaremos describiendo el problema de la cinemática directa e inversa, desarrollando el denominado modelo geométrico. Posteriormente, analizaremos el problema de la cinemática directa e inversa para las velocidades y aceleraciones teniendo especial interés en la determinación de trayectorias que incluyen aspectos relativos a las velocidades y aceleraciones. Y finalmente, describiremos algunos de los modelos dinámicos existentes, concretamente el modelo de Euler-Lagrange y el de Newton-Euler, siendo estos de especial interés de cara a tener una caracterización completa de los robots manipuladores. Hemos de destacar que los problemas presentados en este capítulo serán tratados en los siguientes capítulos mediante la aplicación de otras técnicas alternativas.

2.1 INTRODUCCIÓN.

Los robots se pueden definir como sistemas manipuladores diseñados para mover materiales, piezas, herramientas, etc. o en una concepción más general como dispositivos especiales que a través de movimientos programados pueden realizar una gran variedad de tareas. En general estos sistemas muestran como características más sobresalientes, por un lado, su versatilidad, que les permite realizar muchas tareas y por otro, su auto-adaptabilidad al medio, que les permite realizar las tareas aun con un medio cambiante.

Estas características han influido en que los robots se apliquen cada vez más. Así los encontramos frecuentemente en operaciones de exploración, en donde el medio explorado se presenta hostil al ser humano. También se han empleado ampliamente en el campo de la industria, en donde son idóneos en tareas tales como pintura, soldadura de piezas, ensamblaje, etc.

Además recientemente se han aplicado en el campo de la medicina, en donde, se presentan como elementos auxiliares para superar las limitaciones físicas de personas que presentan alguna clase de discapacidad o en la realización de operaciones quirúrgicas a distancia.

La estructura típica de un robot se compone de varias partes donde la columna vertebral del mismo la constituye su estructura mecánica, formada por una serie de articulaciones ligadas entre sí. Unidos a esta estructura mecánica existen una serie de dispositivos actuadores, en general motores, que posibilitan el movimiento de los robots. Adicionalmente, se encuentran los denominados sistemas de transmisión, cuyas funciones son diversas, entre las que podemos citar la conversión de tipos de movimientos, la amplificación de los pares aplicados sobre las articulaciones o el contribuir a la reducción de ciertos movimientos. Por último, tenemos los sensores, elementos esenciales en cualquier sistema de control. Los sensores nos permiten hacer mediciones de magnitudes propias del robot tales como la posición y velocidad de las articulaciones, la presión del efector, etc. Sin embargo también nos permiten hacer mediciones del entorno, de tal forma que estas mediciones puedan ser utilizadas por el robot para su adaptabilidad respecto al medio.

En la figura (2.1) mostramos ejemplos típicos de robots, concretamente se muestran un robot PUMA (Manipulador universal programable para tareas de ensamblaje), que imita al brazo humano y un SCARA de dos articulaciones de revolución en un plano y una prismática normal a este plano, pensado para trabajar con objetos que se encuentran en una mesa.

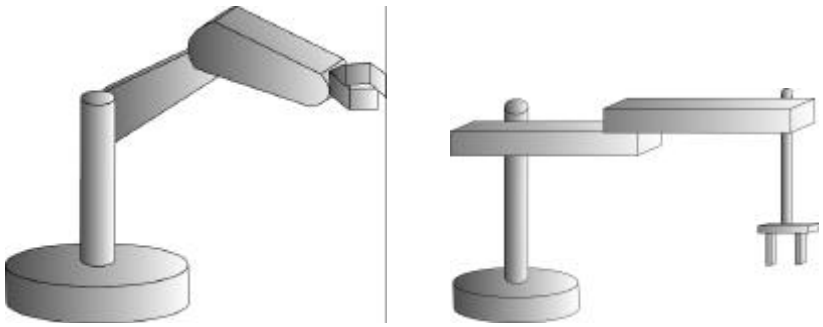


Figura 2.1 - Robots PUMA y SCARA.

Destaquemos que la pinza o efector se diseña de acuerdo a la tarea específica, sin embargo es observable, que variando el efector se pueden realizar otras tareas sin cambio adicional en sus componentes. Este hecho contribuye a la versatilidad que comentábamos en los párrafos anteriores.

Tal y como se ha visto en los párrafos anteriores el robot constituye un sistema de control. Este posee una serie de elementos actuadores, que nos permiten mover las articulaciones y una serie de dispositivos sensores que nos permiten determinar el estado del robot y en ocasiones hacernos una idea del entorno. Sin embargo no basta con tener estos elementos, es necesario disponer de una cierta estrategia de control que le permita realizar al robot la tarea para la que ha sido diseñado en la forma más óptima. En el diagrama siguiente se presenta un esquema general de todos los elementos necesarios para el control del robot.

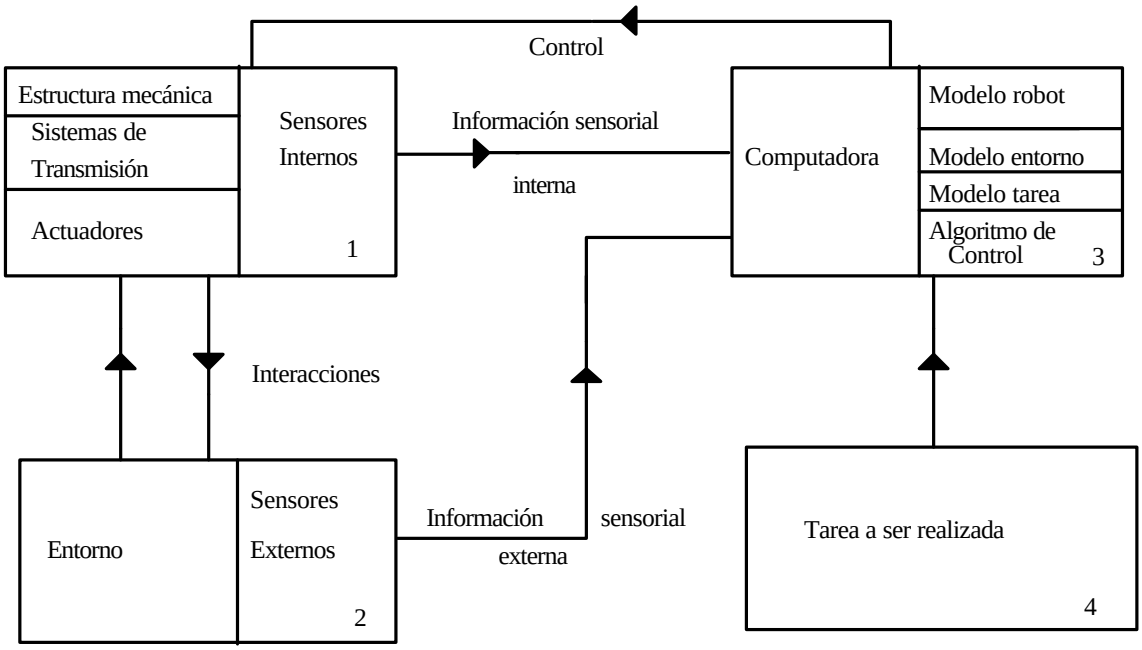


Figura 2.2

En este esquema se observan todos los elementos que se han de considerar de cara al control de un robot. Comentemos que el módulo 4 constituye un módulo de comunicación con el operador. El operador por medio de algún lenguaje le comunica la tarea a realizar. Los robots en donde está presente este módulo se dice que realizan un control textual. El módulo más importante de cara al control es el 3, en donde, a partir de toda la información recogida de los diversos sensores se generan los comandos adecuados a la tarea. Los comandos serán las variables de entrada a los actuadores. De cara a generar comandos adecuados es necesario tener diversos modelos que en general dependen de la tarea específica a realizar. Así para determinadas tareas es necesario tener un modelo del entorno en el que se encuentra inmerso el robot, además, en general se hace necesario tener un modelo de la tarea a realizar. Sin embargo el modelo más importante de cara a la realización del control es el modelo del robot. La mayoría de algoritmos de control se basan en este modelo, de tal forma que en estos casos una adecuada política de control se encuentra ligada a la obtención de un modelo fidedigno del robot.

En las próximas secciones describiremos algunos de los problemas con los que nos encontramos desde el punto de vista del control en los robots manipuladores [SPO88] [FU87]. Comenzaremos hablando del modelo geométrico relacionado con el denominado problema de la cinemática directa e inversa, en donde únicamente se incluyen aspectos relativos a las transformaciones de coordenadas. Seguiremos con la descripción del problema de la cinemática directa e inversa para las velocidades y aceleraciones, en los cuales se consideran aspectos relativos a las velocidades y aceleraciones, para por último presentar el modelo dinámico, que intenta dar una imagen más completa del robot, en donde se incluyen aspectos dinámicos relativos a fuerzas y pares intentando profundizar en las causas que producen el movimiento.

2.2 MODELO GEOMÉTRICO.

El modelo geométrico se basa en intentar buscar la relación entre la orientación y posición del efector del robot y las coordenadas generalizadas relacionadas con cada una de las articulaciones. Por lo tanto el problema se reduce a encontrar una relación como la mostrada a continuación:

$$X = F(\boldsymbol{q}) \tag{2.1}$$

Donde:

X = Posición del centro del efector y de la orientación del sistema de referencia ligado al efector.
θ = Coordenadas generalizadas correspondientes a las articulaciones.

Con objeto de determinar tal relación definamos lo que se entiende por coordenadas homogéneas y matriz de transformación de estas coordenadas. En general la representación de un vector de N componentes se puede presentar en la forma de un vector de N+1 componentes siendo esta su representación en coordenadas homogéneas. Así un vector:

$$P = \left[p_x, p_y, p_z\right]^t \tag{2.2}$$

se convierte en:

$$P_H = \left[w\,p_x, w p_y, w p_z, w\right]^t \tag{2.3}$$

El vector N dimensional se obtendrá del vector de coordenadas homogéneas dividiendo por el valor de la componente N+1 todas las demás componentes. El interés de estas coordenadas se encuentra en el hecho de que proporcionan una matriz de transformación entre sistemas de referencia para rotaciones, traslaciones, escalado y cambio de perspectiva.

Esta matriz se compone a su vez de submatrices que en definitiva representan alguna de las operaciones señaladas anteriormente, quedando tal matriz como se muestra a continuación:

$$T = \begin{bmatrix} \text{Matriz de rotación} & \text{vector de traslación} \\ \text{Transformación de perspectiva} & \text{factor de escalado} \end{bmatrix} \tag{2.5}$$

En el caso particular de un robot, sólo nos encontraremos con rotaciones y traslaciones por lo que la matriz quedará reducida a la siguiente:

$$T = \begin{bmatrix} \text{R} & \text{P} \\ 0\,0\,0 & 1 \end{bmatrix} \tag{2.6}$$

Donde:

R es la Matriz de rotación.

P es el vector de traslación.

Para definir los diferentes puntos que toma el centro del efector en el espacio respecto a un sistema de referencia fijo en la base, se disponen sistemas de referencia ortogonales sobre cada una de las articulaciones moviéndose cada uno de ellos con la articulación correspondiente. La matriz de transformación homogénea entre dos de tales sistemas nos permite obtener las coordenadas del punto dadas en un sistema respecto al otro. Así tendremos:

$$[x_i, y_i, z_i, 1]^t = T_{i,i+1}[x_{i+1}, y_{i+1}, z_{i+1}, 1]^t \qquad (2.7)$$

Donde:

$$T_{i,i+1}(q_i) = \begin{bmatrix} R_{i,i+1}(q_i) & P_{i,i+1}(q_i) \\ 0 & 1 \end{bmatrix}$$

Obsérvese que $T_{i,i+1}$ depende únicamente de q_i que es la coordenada generalizada correspondiente a la unión entre las articulaciones i e i-1.

Señalemos por otro lado que la relación entre dos sistemas de referencia situados entre dos articulaciones no adyacentes puede ser expresada como el producto de las matrices de transformación homogéneas que relacionan sistemas de referencia de articulaciones adyacentes. Esta propiedad nos permite construir una matriz de transformación que nos relacione de una forma simple los puntos dados en el sistema de referencia situado en el efector y el sistema situado en la base de la estructura mecánica del robot. A esta matriz se le denomina matriz de transformación homogénea generalizada encontrándose en la literatura con el nombre abreviado de HTM. Seguidamente mostramos como quedaría esta matriz en función de las matrices homogéneas elementales que relacionan las coordenadas de sistemas adyacentes:

$$T_{1,n+1} = T_{1,2}(q_1)T_{2,3}(q_2)T_{3,4}(q_3)\dots T_{n,n+1}(q_n) \qquad (2.8)$$

Obsérvese que $T_{1,n+1}$ dependerá de un conjunto de variables generalizadas. Dada la matriz homogénea generalizada la relación de coordenadas entre sistemas queda:

$$[x_1, y_1, z_1, 1]^t = T_{1,n+1}[x_{n+1}, y_{n+1}, z_{n+1}, 1]^t \qquad (2.9)$$

Esta relación nos permite obtener la ecuación (2.1) sin más que dar las coordenadas del centro del efector respecto al sistema de referencia ligado a la misma, de esta forma ya tenemos la posición del centro del efector en función de una serie de variables generalizadas. Sin embargo en algunas aplicaciones no basta con obtener la posición del efector respecto a un sistema de referencia fijo en la base, en ocasiones se necesita también la orientación de la misma. Esta relación se puede conseguir a partir de la ecuación (2.9) simplemente tomando las coordenadas de los vectores unitarios que definen el sistema de referencia en el efector y transformándolos.

De la forma indicada en el párrafo anterior se pueden obtener una serie de ecuaciones que nos darán la posición y orientación del efector en función de las variables generalizadas. Sin embargo respecto a las ecuaciones que nos dan la orientación hemos de considerar que algunas son redundantes, dado que por un lado los vectores unitarios tienen norma unidad y por otro el sistema de referencia elegido es ortogonal, lo que determina una relación de ortogonalidad entre los vectores unitarios que definen el mismo. Por otra parte puede resultar que el número de coordenadas generalizadas sea menor que el número de ecuaciones, aumentando la redundancia. Este hecho se traduce en la posibilidad de que existan varias soluciones.

Destaquemos que existe un método standard para sistematizar la elección de los sistemas de referencia unidos a las articulaciones, así como la elección de los parámetros que caracterizan al robot respecto al establecimiento de las relaciones del modelo geométrico, denominados estos últimos parámetros de Denavit-Hartenberg. Estos parámetros corresponden a las siguientes distancias y ángulos:

θ_i = ángulo de giro de una articulación o ángulo de la articulación del eje x_{i-1} al eje x_i respecto al eje z_{i-1} utilizando la regla de la mano derecha.

d_i = distancia desde el origen del sistema $i-1$ hasta la intersección del eje z_{i-1} con el eje x_i a lo largo de z_{i-1}

a_i = distancia de separación desde la intersección del eje z_{i-1} con el eje x_i hasta el origen del sistema iésimo a lo largo del eje x_i (o distancia más corta entre z_{i-1} y z_i)

α_i = ángulo de separación del eje z_{i-1} al eje z_i respecto del eje x_i

En la figura (2.3) mostramos como se toman estos parámetros para un robot PUMA. En general el algoritmo de asignación de sistemas de coordenadas para un robot de n grados de libertad consta de los siguientes pasos:

1. Establecer el sistema de coordenadas de la base dextrógiro (regla de la mano derecha), siendo el sistema $(x_0 y_0 z_0)$ con z_0 el eje de giro de la articulación si es de rotación, o eje de movimiento si es de traslación. Debe apuntar hacia fuera del hombro del brazo del robot. Los ejes x_0 e y_0 se pueden tomar convenientemente siempre que cumplan la regla de la mano derecha.
2. Inicializar y repetir para i = 1,...,n-1, realizar los pasos del 3 al 6.
3. Establecer los ejes de la articulación. Alinear z_i con el eje de movimiento de la articulación i+1.
4. Establecer el origen del sistema de coordenadas i-ésimo. Localizar el origen del sistema de coordenadas i-ésimo en la intersección de los ejes z_i y z_{i-1} o en la intersección de las normales comunes entre estos ejes.
5. Establecer el eje x_i en una normal común entre los ejes z_i e z_{i-1} cuando estos son paralelos.
6. Establecer y_i para completar el sistema de coordenadas dextrógiro.
7. Establecer el sistema de coordenadas de la mano. Normalmente la articulación n-ésima es de tipo giratorio así que se establece z_n a lo largo de la dirección del eje z_{n-1} y apuntando hacia fuera del robot. Eligiendo los otros dos ejes como se ha dicho anteriormente.
8. Establecer los parámetros de Denavit-Hartenberg d, a, θ, α.

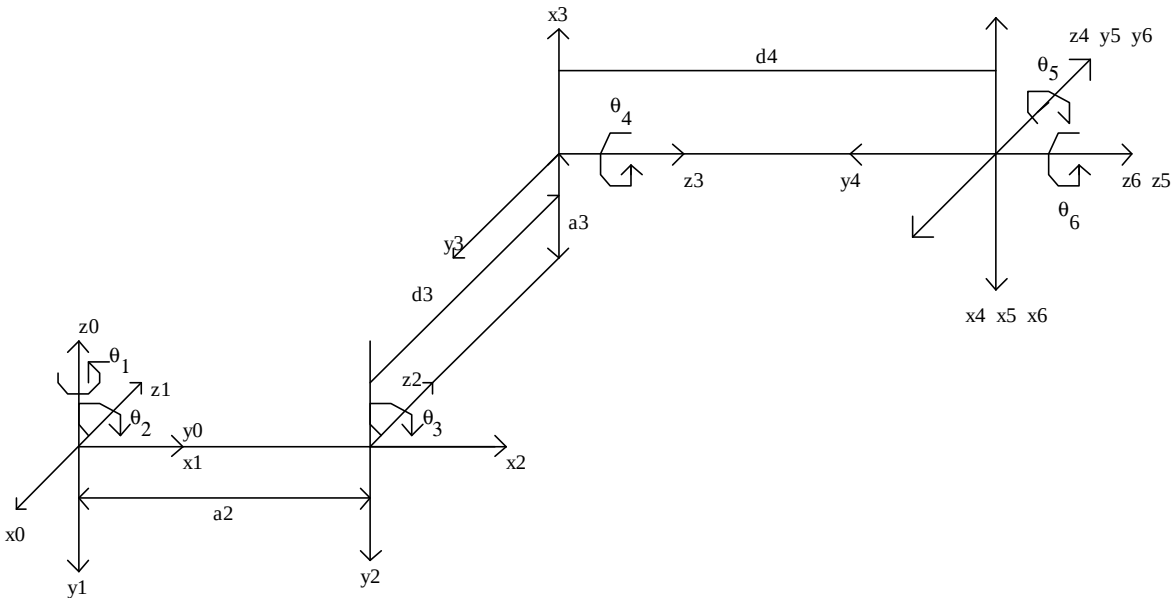


Figura 2.3 – Elección de parámetros de Denavit-Hartenberg para un robot PUMA.

El modelo geométrico tal y como se presenta a través de la ecuación (2.1) nos resuelve el problema de la cinemática directa, es decir, podemos obtener las coordenadas cartesianas del efector a partir de las coordenadas generalizadas de las articulaciones. Este problema se reduce a pasar de unos sistemas de coordenadas a otros, de forma iterativa, pasando finalmente del sistema del efector al sistema de la base. Realizando la asignación de ejes tal como se describió en los párrafos anteriores podemos expresar la ecuación (2.1) en función de los parámetros de Denavit-Hartenberg. La obtención de la matriz de transformación que nos relaciona el sistema i-ésimo con el i-1 ésimo se puede calcular, teniendo en cuenta los siguientes pasos:

1. Girar respecto del eje z_{i-1} un ángulo θ_i para alinear el eje x_{i-1} con el eje x_i (el eje x_{i-1} es paralelo a x_i y apunta en la misma dirección).
2. Trasladar a lo largo del eje z_{i-1} una distancia d_i para llevar en coincidencia los ejes x_{i-1} y x_i.
3. Trasladar a lo largo del eje x_i una distancia de a_i para traer en coincidencia los dos orígenes de los ejes x.
4. Girar respecto del eje x_i un ángulo α_i para traer en coincidencia a los sistemas de coordenadas.

Siguiendo los pasos anteriores la matriz de transformación que pasa del sistema i-ésimo al (i-1)-ésimo estará definida como una matriz 4x4 en la que la última fila es 0 0 0 1. Los elementos a11, a21, y a31 corresponden a las coordenadas del vector de la base del eje x del sistema i respecto a la base del sistema i-1. Los elementos a12, a22, a32 corresponden a las coordenadas del vector de la base del eje y del sistema i respecto a la base del sistema i-1. Los elementos a31, a32, a33 son las coordenadas del vector de la base del eje z del sistema i respecto a la base del sistema i-1. Los elementos a41, a42, a43 son las coordenadas de un vector que va del origen del sistema i-1 al sistema i respecto de la base del sistema i-1. Esta construcción equivale a la multiplicación de cuatro matrices de transformación que darán como resultado la matriz de transformación ${}^{i-1}A_i$.

$$ {}^{i-1}A_i = T_{z,d}T_{z,\theta}T_{x,a}T_{x,\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \cos\theta_i & -\operatorname{sen}\theta_i & 0 & 0 \\ \operatorname{sen}\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * $$

$$ * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\operatorname{sen}\alpha_i & 0 \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_i & -\cos\alpha_i * \operatorname{sen}\theta_i & \operatorname{sen}\alpha_i * \operatorname{sen}\theta_i & a_i * \cos\theta_i \\ \operatorname{sen}\theta_i & \cos\alpha_i * \cos\theta_i & -\operatorname{sen}\alpha_i * \cos\theta_i & a_i * \operatorname{sen}\theta_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

(2.10)

Teniendo en cuenta la matriz de transformación (2.10) el cálculo se reduce a una simple multiplicación de matrices, tal como se señalaba en la ecuación (2.8). De forma que conocidos los ángulos de giro θ de las articulaciones de revolución y los desplazamientos d de las articulaciones prismáticas, ya se pueden definir las matrices de transformación ${}^0A_1, {}^1A_2, {}^2A_3, {}^3A_4, {}^4A_5$ y ${}^5A_6$ con lo que el punto del centro del efector (px,py,pz) se obtiene de la siguiente forma:

$$ \begin{pmatrix} px \\ py \\ pz \\ 1 \end{pmatrix} = {}^0A_1 * {}^1A_2 * {}^2A_3 * {}^3A_4 * {}^4A_5 * {}^5A_6 * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad (2.11) $$

Mediante la resolución de la cinemática directa podemos obtener las posiciones del efector a través de los ángulos y desplazamientos de las articulaciones, sin embargo desde el punto de vista de la especificación de tareas es más adecuado especificarlas en coordenadas cartesianas del efector, mientras que de cara al control es conveniente tener los ángulos y desplazamientos de las articulaciones necesarios para obtener esa posición del efector. Esto nos lleva al problema de la cinemática inversa, que consiste en la obtención de los ángulos y desplazamientos de las articulaciones que corresponden a una posición del efector. Desde un punto de vista matemático este problema se reduce a la obtención de la siguiente ecuación:

$$ \boldsymbol{q} = F^{-1}(X) \qquad (2.12) $$

Donde:

F^{-1} = es la inversa de F.

Sin embargo existen ciertas dificultades en la obtención de las variables generalizadas a partir del modelo. La primera dificultad surge cuando la posición a ser alcanzada es imposible de alcanzar físicamente por la estructura mecánica, siendo en este caso irresoluble el problema. En segundo lugar en el modelo no se han impuesto limitaciones a las variables generalizadas, pero sin embargo éstas representan generalmente rotaciones o traslaciones acotadas entre ciertos valores, lo que implica que ciertas

soluciones teóricas del modelo no se corresponden con soluciones reales. Y en tercer lugar, las ecuaciones que surgen del modelo geométrico son ecuaciones no lineales siendo propiamente este hecho una dificultad matemática adicional.

Destaquemos que el problema de la cinemática inversa es un problema básico en el seguimiento de trayectorias, ya que se pretenden obtener posiciones determinadas del efector a través de la elección de ciertos ángulos y desplazamientos de las articulaciones. Sin embargo, resulta un problema más complejo de resolver dada la no linealidad del mismo.

2.3 PROBLEMA DE LA CINEMÁTICA INVERSA DE LAS VELOCIDADES Y ACELERACIONES.

Al igual que en la sección anterior se planteaba el problema de la cinemática directa e inversa [SPO88] [FU87], resolviéndose éste mediante el establecimiento del modelo geométrico, en esta sección trataremos de resolver este problema para el caso de las velocidades y aceleraciones del efector, dado que la mayoría de algoritmos de control, así como los modelos de los robots que contemplan aspectos dinámicos necesitan resolver este problema previamente. En resumen se trata de buscar las relaciones entre las velocidades y aceleraciones del efector y las velocidades y aceleraciones generalizadas correspondientes a las articulaciones. Partiendo del modelo geométrico ya expuesto y teniendo en cuenta únicamente pequeños desplazamientos podemos obtener esta relación. Realicemos sobre la ecuación (2.1) que define el modelo geométrico, un desarrollo de Taylor de la función F en torno a un cierto punto X_0. Tomando en consideración desplazamientos pequeños de las variables generalizadas nos podemos quedar con el primer término del desarrollo. De tal forma que del modelo geométrico deducimos que:

$$\Delta X = [J]\Delta\theta \qquad (2.13)$$

Donde:

ΔX = Desplazamiento de las coordenadas del efector respecto a X_0.
$\Delta\theta$ = Desplazamiento de las coordenadas generalizadas del efector respecto a θ_0.
$[J]$ = Matriz Jacobiana respecto a las variables generalizadas.

Dividiendo en la expresión (2.13) por el incremento de tiempo Δt, se puede deducir la siguiente expresión que relaciona la primera derivada del vector de coordenadas del efector y la primera derivada del vector de coordenadas generalizadas para la posición X_0 del efector en el instante de tiempo considerado:

$$\dot{X}_0 = [J]\dot{\theta} \qquad (2.14)$$

Ahora derivando la ecuación (2.14) podemos obtener una expresión que relacione la aceleración del efector con las aceleraciones de las articulaciones. Esta viene dada como:

$$\ddot{X}_0 = [J]\ddot{\theta} + \frac{d}{dt}[J]\dot{\theta} \qquad (2.14)$$

Debemos remarcar que la relación de la velocidad del efector con las velocidades generalizadas es lineal, lo que contribuye notablemente a simplificar el tratamiento matemático. De cara a realizar un control se hace necesario tener una relación que nos dé las velocidades generalizadas a partir de la velocidad del efector, a este problema se le conoce con el nombre de determinación de la cinemática inversa de las velocidades. Esta relación se consigue simplemente invirtiendo la matriz jacobiana. En el caso de que la matriz Jacobiana no sea cuadrada habremos de recurrir a la teoría generalizada de inversión matricial [FUE93]. Sin embargo en el caso de ser cuadrada, se aplicaría el método standard de inversión. En este último caso, para asegurar que existe matriz inversa el determinante de la matriz Jacobiana ha de ser distinto de cero en el punto de coordenadas generalizadas de trabajo. En caso contrario decimos que nos encontramos en una singularidad. Las configuraciones para las que ocurre una singularidad se denominan configuraciones singulares y a los robots que las poseen se los denomina robots singulares. Estas singularidades pueden surgir de diferentes formas. En ocasiones dichas singularidades no tienen correspondencia física, tal es el caso cuando surgen para traslaciones negativas, sin embargo en otros casos tales singularidades nos llevan a restringir los posibles movimientos, dado que dan lugar a dependencias entre las coordenadas a acceder. En tales casos es necesario aplicar métodos de resolución particulares al problema concreto. Cuando el Jacobiano (determinante de la matriz Jacobiana) es nulo para todo el espacio de trabajo, esto implica dependencias entre las ecuaciones del modelo. En este caso la solución más común es elegir otro conjunto de variables que sean independientes. En cuanto a la determinación de las aceleraciones generalizadas a partir de la aceleración del efector, éstas son determinadas a través de la ecuación (2.14), siendo este problema conocido como determinación de la cinemática inversa de las aceleraciones. Obsérvese que este caso es más complejo que el problema de la cinemática inversa de las velocidades dado que además de la inversión de la matriz Jacobiana interviene su derivada primera con lo que presenta una mayor carga computacional.

Concluyamos esta sección haciendo un balance de los problemas resueltos en esta y la sección anterior. Estos inciden sobre todo en aspectos relativos a la transformación de coordenadas, velocidades y aceleraciones. Aunque se pueden tomar como base para la implementación de algoritmos de control, tienen como mayor desventaja el que no consideran características dinámicas del robot tales como: masas, inercias, fricciones, etc. Por lo tanto en la sección siguiente abordaremos los modelos existentes en donde se tienen en cuenta las características dinámicas. Dichos modelos se pueden tomar de base para la implementación de estrategias de control eficientes.

2.4 MODELOS DINÁMICOS.

Hasta ahora en las secciones anteriores únicamente hemos considerado el problema de la transformación de coordenadas, velocidades y aceleraciones del efector, sin embargo en ninguno de ellos se han considerado las características dinámicas del movimiento. En general los robots se encuentran constituidos por articulaciones con formas y masas determinadas que influyen en el movimiento. Existen varias fuerzas que influyen en el robot, así tenemos la fuerza de la gravedad que influye sobre la misma estructura mecánica, así como sobre las piezas que desplaza el efector del robot si éste es utilizado

en tareas que impliquen transporte. Por otro lado existen fuerzas aplicadas por los motores a las articulaciones. Además derivado del movimiento de las masas aparecen fuerzas tales como las fuerzas de inercia o centrífugas. Por último en todo robot surgen fuerzas propias del rozamiento entre los diversos componentes del mismo que han de ser incluidas de alguna forma, si se quiere que éste sea un fiel reflejo de toda la dinámica presente en el mismo.

De forma general todos los modelos que tienen en consideración los efectos dinámicos citados tratan de obtener una relación entre las fuerzas generalizadas aplicadas sobre las articulaciones y los estados de estas articulaciones, viniendo estos últimos dados por las correspondientes coordenadas generalizadas, además de sus velocidades y aceleraciones. Tal relación tendría la siguiente forma:

$$\Gamma = \Gamma\left(q, \dot{q}, \ddot{q}, \boldsymbol{q}\right) \qquad (2.15)$$

Donde:

Γ = Vector de fuerzas generalizadas aplicadas a las articulaciones.

$\left(q, \dot{q}, \ddot{q}\right)$ = Vector de estado de todas las articulaciones.

θ = Vector de parámetros propios del robot (dimensiones, masas, ...)

A esta relación es a la que se denomina modelo dinámico directo, sin embargo en ocasiones nos interesa una relación inversa, es decir, que a partir de las fuerzas generalizadas podamos obtener los estados de las articulaciones. A estos modelos se les denomina modelos dinámicos inversos y responden a la siguiente relación:

$$\ddot{q} = f\left(\Gamma, q, \dot{q}, \boldsymbol{q}\right) \qquad (2.16)$$

Para obtener los modelos citados tradicionalmente se han aplicado diferentes métodos. Es interesante destacar que todos los formalismos utilizados representan la dinámica de los robots, sin embargo conducen a diferentes niveles de complejidad en los procesos de cálculo empleados. Los requerimientos que se persiguen en estos modelos son por un lado que presenten tiempos de cálculo mínimos y por otro que requieran el menor espacio de memoria permitiendo así controlar el robot en tiempo real. Se han aplicado diferentes formalismos utilizando las ecuaciones de Newton en el equilibrio, el principio de D'Alembert, o las ecuaciones de Lagrange del movimiento.

2.4.1 MODELO DE LAGRANGE-EULER.

El modelo de Lagrange-Euler se basa en la aplicación de las ecuaciones de Lagrange del movimiento. El elemento clave en el formalismo Lagrangiano es la obtención del Lagrangiano que en nuestro caso se puede expresar como:

$$L(q,\dot{q}) = T(q,\dot{q}) - U(q) \qquad (2.17)$$

Donde:

$L(q,\dot{q})$ = Lagrangiano del robot.

$T(q,\dot{q})$ = Energía cinética del robot.

$U(q)$ = Energía potencial del robot.

Las Fuerzas generalizadas vendrán dadas por:

$$\Gamma_i = \frac{d}{dt}\left\{\frac{\partial L(q,\dot{q})}{\partial \dot{q}_i}\right\} - \frac{\partial L(q,\dot{q})}{\partial q_i} + P \qquad i = 1,....,n \qquad (2.18)$$

Donde:

Γ_i = Fuerza generalizada correspondiente a q_i (igual al par o torque en el caso de una articulación de revolución o a la fuerza en el caso de una articulación prismática).

P = Término correspondiente a las fuerzas no conservativas.

Ahora únicamente hemos de determinar el Lagrangiano a partir del cálculo de la energía cinética y energía potencial. Comencemos realizando el cálculo de la expresión para la energía cinética. Con objeto de facilitar el seguimiento de los razonamientos que se expondrán presentamos en la figura (2.4) el esquema típico de un robot general, donde las elipses representan las diferentes articulaciones de longitudes respectivas l_n.

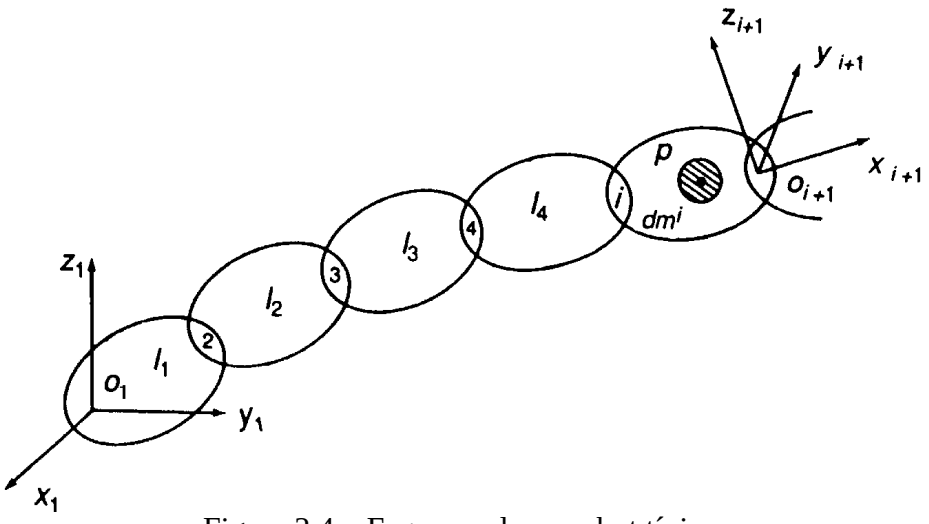


Figura 2.4 – Esquema de un robot típico

Es importante destacar que en la figura se presentan sendos sistemas de referencia uno en la base del robot y otro en la articulación i con orígenes denotados respectivamente por O_1 y O_{i+1}. Cualquier punto genérico P de cualquier articulación i se localizará respecto al sistema de referencia fijo en esa articulación que en nuestro caso se trata del

sistema de referencia de origen O_{i+1} y dada la relación entre este sistema de referencia y el situado en la base este mismo punto se podrá expresar en función del sistema de referencia de la base. Este hecho contribuye a simplificar notablemente los cálculos tal como se muestra en las siguientes líneas. La relación citada vendrá dada por la siguiente expresión en donde aparece la matriz de transformación homogénea entre los dos sistemas:

$$[x_1, y_1, z_1, 1]^t = T_{1,i+1}[x_{i+1}, y_{i+1}, z_{i+1}, 1]^t \qquad (2.19)$$

Ahora derivando respecto al tiempo utilizando la regla de derivación de un producto matricial y teniendo en cuenta que el punto P no varía a lo largo del tiempo respecto al sistema de referencia fijo en la articulación i, tendríamos la relación de transformación de las velocidades:

$$\begin{bmatrix} \dot{x}_1 & \dot{y}_1 & \dot{z}_1 & 0 \end{bmatrix}^t = \dot{T}_{1,i+1}\begin{bmatrix} x_{i+1} & y_{i+1} & z_{i+1} & 1 \end{bmatrix}^t \qquad (2.20)$$

Ahora tomando la masa infinitesimal dm^i en torno al punto P correspondiente a la articulación i tendremos la energía cinética infinitesimal:

$$dT^i(q, \dot{q}) = \frac{1}{2} dm^i (\dot{x}^2_1 + \dot{y}^2_1 + \dot{z}^2_1) \qquad (2.21)$$

Que se puede expresar en función de la traza de una matriz como:

$$dT^i(q, \dot{q}) = \frac{1}{2} Tr\left(\begin{bmatrix} \dot{x}_1 & \dot{y}_1 & \dot{z}_1 & 0 \end{bmatrix}^t \begin{bmatrix} \dot{x}_1 & \dot{y}_1 & \dot{z}_1 & 0 \end{bmatrix} \right) dm^i \qquad (2.22)$$

Donde:

Tr = Operador traza.

$\begin{bmatrix} \dot{x}_1 & \dot{y}_1 & \dot{z}_1 & 0 \end{bmatrix}$ = Vector velocidad respecto del sistema de referencia en la base del robot.

Sustituyendo la ecuación (2.20) en (2.22) tenemos:

$$dT^i(q, \dot{q}) = \frac{1}{2} Tr\left(\dot{T}_{1,i+1}\begin{bmatrix} x_{i+1} & y_{i+1} & z_{i+1} & 1 \end{bmatrix}^t \begin{bmatrix} x_{i+1} & y_{i+1} & z_{i+1} & 1 \end{bmatrix} \dot{T}^t_{1,i+1} \right) dm^i \qquad (2.23)$$

Ahora expresando $\dot{T}_{1,i+1}$ en función de las variables generalizadas tal como se muestra a continuación:

$$\dot{T}_{1,i+1} = \sum_{j=1}^{i} \frac{\partial \dot{T}_{1,i+1}}{\partial q_j} \dot{q}_j \qquad (2.24)$$

Sustituyendo esta expresión en (2.23) y sacando los sumatorios del operador traza llegamos a:

$$dT^i(q,\dot{q}) = \frac{1}{2} \sum_{j=1}^{i} \sum_{k=1}^{i} Tr\left\{ \frac{\partial T_{1,i+1}}{\partial q_j} \begin{bmatrix} x_{i+1} y_{i+1} z_{i+1} 1 \end{bmatrix}^t \begin{bmatrix} x_{i+1} y_{i+1} z_{i+1} 1 \end{bmatrix} \frac{\partial T_{1,i+1}^t}{\partial q_k} \right\} \dot{q}_j \dot{q}_k dm^i \qquad (2.25)$$

Ahora realizando el producto $\begin{bmatrix} x_{i+1} & y_{i+1} & z_{i+1} \end{bmatrix}^t \begin{bmatrix} x_{i+1} & y_{i+1} & z_{i+1} \end{bmatrix}$ obteniendo de esta forma una matriz de dimensión 4x4, e integrando cada componente de la matriz para toda la distribución de masa correspondiente a la articulación i dado que los otros términos pertenecientes a la expresión (2.25) no presentan dependencia de la masa, la energía cinética para esta articulación quedará de la forma:

$$T^i(q,\dot{q}) = \frac{1}{2} \sum_{j=1}^{i} \sum_{k=1}^{i} Tr\left\{ \frac{\partial T_{1,i+1}}{\partial q_j} J^i \frac{\partial T_{1,i+1}^t}{\partial q_k} \right\} \dot{q}_j \dot{q}_k \qquad (2.26)$$

Donde:

$$J^i = \begin{bmatrix} \int x_{i+1}^2 dm^i & \int x_{i+1} y_{i+1} dm^i & \int x_{i+1} z_{i+1} dm^i & \int x_{i+1} dm^i \\ \int x_{i+1} y_{i+1} dm^i & \int y_{i+1}^2 dm^i & \int y_{i+1} z_{i+1} dm^i & \int y_{i+1} dm^i \\ \int x_{i+1} z_{i+1} dm^i & \int y_{i+1} z_{i+1} dm^i & \int z_{i+1}^2 dm^i & \int z_{i+1} dm^i \\ \int x_{i+1} dm^i & \int y_{i+1} dm^i & \int z_{i+1} dm^i & \int dm^i \end{bmatrix} \qquad (2.27)$$

Siendo J^i la pseudo-matriz de inercia de la articulación i. Es importante destacar que la pseudo-matriz de inercia J^i depende exclusivamente de las características propias de la articulación.

Para obtener la energía cinética total hemos únicamente de sumar sobre el índice i a todas las articulaciones, quedando:

$$T(q,\dot{q}) = \sum_{i=1}^{n} T^i(q,\dot{q}) \qquad (2.28)$$

Donde:

n = número de articulaciones.

Si desarrollamos la expresión (2.28) se puede llegar a expresar la energía cinética total como una forma cuadrática en el espacio de los vectores de velocidades generalizadas correspondientes a todas las articulaciones. Dicha forma cuadrática viene definida por una matriz A(q) definida positiva y no singular. Así la energía cinética queda en la forma:

$$T(\mathrm{q},\dot{\mathrm{q}}) = \frac{1}{2}\dot{\mathrm{q}}^{\mathrm{t}} A(\mathrm{q})\ \ \dot{\mathrm{q}} \qquad (2.29)$$

Obtenida la energía cinética total centrémonos ahora en el cálculo de la energía potencial total. Comencemos recordando como se puede expresar la energía potencial gravitatoria de una partícula con objeto de llegar gradualmente a expresar de forma conveniente la energía potencial gravitatoria para la articulación i. Sea un conjunto de partículas de masa m_i su energía potencial gravitatoria viene expresada como:

$$U^i(q) = -\vec{g}\ \sum_{i=1}^{n} \vec{r}_i m^i \qquad (2.30)$$

Donde:

$\vec{r}_i$ = vector de posición de la partícula i.

$\vec{g}$ =vector aceleración de la gravedad.

m^i = masa de la partícula i.

n = número de partículas.

Definiendo el vector de posición del centro de masas como:

$$\vec{R} = \frac{\sum_{i=1}^{n} \vec{r}_i m^i}{M} \qquad (2.31)$$

Donde:

M = masa total del sistema de partículas.

La energía potencial se puede expresar de forma más compacta como:

$$U^i(q) = -\vec{g}\ M\vec{R} \qquad (2.32)$$

En conclusión, la energía potencial de un sistema de partículas es la misma que se tendría si el cuerpo estuviera concentrado en su centro de masas, de tal forma que para determinar la energía potencial correspondiente a la articulación i habremos únicamente de calcular el vector de posición del centro de masas correspondiente a la misma. Obsérvese que aunque las articulaciones son medios continuos los razonamientos permanecen válidos sin más que sustituir los sumatorios por integrales. En nuestro caso calcularemos la energía potencial dando todos los vectores implicados en el problema respecto al sistema de referencia situado en la base del robot. Sin embargo resulta más fácil el calcular los vectores de posición de los centros de masas de las articulaciones respecto a los respectivos sistemas de referencia fijos en los mismos y utilizando la relación (2.19) expresarlos en función del sistema de referencia en la base, por lo que abordaremos los cálculos de esta forma. Es importante destacar que en nuestro caso tomaremos vectores de posición y velocidad homogéneos a diferencia de los vectores tomados en la expresión (2.32). Sin embargo esto no representa ninguna incorrección dado que la energía potencial queda indefinida en una constante. En base a lo anterior, la energía potencial se puede expresar como:

$$U^i(q) = -g^t T_{1,i+1} m^i\ \boldsymbol{r}_i \qquad (2.33)$$

Donde:

ρ_i = es el vector homogéneo del centro de masas de la i-ésima articulación.
g^t = es el vector de aceleración de la gravedad. (Obsérvese que en este caso el vector se ha colocado como una matriz columna).

Sumando las respectivas energías potenciales para cada una de las articulaciones podemos obtener la energía potencial total. Así:

$$U(q) = \sum_{i=1}^{n} U^i(q) \qquad (2.34)$$

De la expresión (2.34) se puede averiguar la fuerza debida a la gravedad en la articulación i (G^i), tan solo derivando respecto a la coordenada generalizada correspondiente:

$$G^i = -\frac{\partial U(q)}{\partial q_i} \qquad (2.35)$$

Ahora aplicando las ecuaciones (2.18) podemos obtener las fuerzas generalizadas sin más que formar el Lagrangiano tal como se indica en (2.17):

$$\Gamma^i = \frac{d}{dt}\left\{\frac{\partial}{\partial \dot{q}_i}\left(\frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} a^{jk}\,\dot{q}_j\dot{q}_k\right)\right\} - \frac{\partial}{\partial q_i}\left(\frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} a^{jk}\dot{q}_j\dot{q}_k\right) - G^i + P^i \qquad (2.36)$$

Donde:

a^{jk} = Elementos de la matriz A(q) correspondientes a la energía cinética.
G^i = Fuerzas generalizadas correspondientes al potencial gravitatorio.
P^i = Fuerzas generalizadas correspondientes a todas las fuerzas no conservativas.

Teniendo en cuenta la definición de los símbolos de Christofell sobre los elementos de A(q):

$$b^{i,jk} = \frac{1}{2}\left(\partial a^{ij}/\partial q_k + \partial a^{ik}/\partial q_j - \partial a^{jk}/\partial q_i\right) \qquad (2.37)$$

Y teniendo en cuenta las siguientes propiedades de los símbolos de Christofell:

$$\begin{aligned} &b^{i,kj} = b^{i,jk} \\ &b^{k,ji} = -b^{i,j,k} && \text{Para } j \le i \text{ y } i < k \\ &b^{i,ji} = 0 && \text{Para } j \le i \end{aligned} \qquad (2.38)$$

Podemos simplificar la expresión (2.36) tal como se indica a continuación:

$$\Gamma^i = \sum_{j=1}^{n} a^{ij}\ddot{q}_j + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} b^{i,jk}\dot{q}_j\dot{q}_k + \sum_{j=1}^{n} b^{i,jj}\dot{q}_j^{\,2} - G^i + P^i \qquad (2.39)$$

Donde:

i=1,...,n n=número de articulaciones.

El conjunto de ecuaciones del tipo (2.39) se puede agrupar en forma matricial simplemente definiendo matrices rectangulares y vectores. Así definiremos las siguientes matrices y vectores.

$A(q) =$ Matriz definida por los elementos a^{ij}.

$B(q) =$ Matriz definida por los elementos $b^{i,jk}$ correspondientes al segundo término .

$C(q) =$ Matriz definida por los elementos $b^{i,jj}$ correspondientes al tercer término .

$$\ddot{q} = [\ \ddot{q}_1\ \ddot{q}_2\ \ldots\ \ddot{q}_n\ ]^t$$

$$\dot{q}^{\,2} = \left[\dot{q}_1^2\ \dot{q}_2^2\ \ldots\ \dot{q}_n^2\right]^t$$

$$\dot{q}\dot{q} = \left[\dot{q}_1\dot{q}_2\ \ \dot{q}_1\dot{q}_3\ \ \cdots\ \ \dot{q}_1\dot{q}_n\ ,\ \dot{q}_2\dot{q}_3\ \ \dot{q}_2\dot{q}_4\ \ \cdots\ \ \dot{q}_2\dot{q}_n\ ,\ \ldots\ ,\ \dot{q}_{n-1}\dot{q}_n\ \right]^t$$

$$\Gamma = \left[\Gamma^1\Gamma^2\ldots\Gamma^n\right]$$

$$G(q) = \left[G^1G^2\ldots G^n\right]$$

$$P = \left[P^1P^2\ldots P^n\right]$$

Con estas definiciones el modelo dinámico quedaría condensado en la siguiente ecuación:

$$\Gamma = A(q)\ddot{q} + 2B(q)\dot{q}\dot{q} + C(q)\dot{q}^{\,2} - G(q) + P \qquad (2.40)$$

La dinámica directa correspondería a obtener q(t) a partir de las fuerzas generalizadas Γ(t). Por lo tanto sería necesario resolver la ecuación diferencial. En cambio la dinámica inversa se obtendría sin más que sustituir los valores de q(t) (y de $\dot{q}(t), \ddot{q}(t)$, que se pueden obtener de q(t)) para obtener las fuerzas generalizadas Γ(t).

Para resolver las ecuaciones del modelo dinámico presentado es necesario como primer punto conocer los valores de las matrices A(q), B(q), C(q) y el vector G(q). Además el vector P que representa las fuerzas de fricción y perturbaciones a que se haya sujeto el robot, ha de ser medido o estimado. Esto no es siempre posible en condiciones de

trabajo arbitrarias para el robot. Nótese que es un problema de identificación de un sistema no lineal, con un término P de forma desconocida.

Las matrices A(q), B(q) y C(q) tienen la siguiente forma explícita:

$$A(q) = \begin{bmatrix} a^{11} & a^{12} & \cdots & a^{1n} \\ a^{21} & a^{22} & \cdots & a^{2n} \\ a^{311} & a^{32} & \cdots & a^{3n} \\ \vdots & \vdots & \cdots & \vdots \\ a^{n1} & a^{n2} & & a^{nn} \end{bmatrix} ; \; C(q) = \begin{bmatrix} 0 & b^{1,22} & \cdots & b^{1,nn} \\ -b^{1,12} & 0 & \cdots & b^{2,nn} \\ -b^{1,13} & -b^{2,23} & \cdots & b^{3,nn} \\ \vdots & \vdots & \cdots & \vdots \\ -b^{1,1n} & b^{2,2n} & & 0 \end{bmatrix}$$

$$B^t(\mathbf{q}) = \begin{bmatrix} b^{1,12} & 0 & -b^{2,13} \cdots & -b^{2,1n-1} & -b^{2,1n} \\ b^{1,13} & b^{2,13} & 0 \cdots & -b^{3,1n-1} & -b^{3,1n} \\ \vdots & \vdots & \vdots \quad \vdots & \vdots & \vdots \\ b^{1,1n} & b^{2,1n} & b^{3,1n} \cdots & b^{n-1,1n} & 0 \\ \hline b^{1,23} & b^{2,23} & 0 \cdots & -b^{3,2n-1} & -b^{3,2n} \\ b^{1,24} & b^{2,24} & b^{3,24} \cdots & -b^{4,2n-1} & -b^{4,2n} \\ \vdots & \vdots & \vdots \quad \vdots & \vdots & \vdots \\ b^{1,2n} & b^{2,2n} & b^{3,2n} \cdots & b^{n-1,2n} & 0 \\ \hline \vdots & \vdots & \vdots \quad \vdots & \vdots & \vdots \\ \hline b^{1,n-2n-1} & b^{2,n-2n-1} & b^{3,n-2n-1} \cdots 0 & & -b^{n-1,n-2n} \\ b^{1,n-2n} & b^{2,n-2n} & b^{3,n-2n} \cdots & b^{n-1,n-2n} & 0 \\ \hline b^{1,n-1n} & b^{2,n-1n} & b^{3,n-1n} \cdots & b^{n-1,n-1n} & 0 \end{bmatrix}$$

En segundo lugar si se desea resolver la dinámica inversa es necesario calcular las fuerzas generalizadas. Existen diferentes procedimientos para su cálculo en orden a eliminar en todo lo posible operaciones redundantes. El escollo más importante con el que se encuentra este modelo en tiempo real está en el excesivo tiempo de cálculo. Como se aprecia de las ecuaciones el modelo resulta notablemente complejo. Con el objetivo de simplificar las ecuaciones desde el punto de vista del cálculo se puede abordar el problema de la obtención del modelo dinámico a partir de la segunda ley de Newton. Esta es la aproximación que se plantea en la siguiente sección.

2.4.2 MODELO DE NEWTON-EULER.

En la sección anterior hemos llegado a un conjunto de ecuaciones diferenciales a partir de la formulación de Lagrange-Euler para describir el comportamiento dinámico del

robot. El uso de estas ecuaciones para calcular los pares o fuerzas a aplicar a las articulaciones en tiempo real, supone una gran carga computacional.

Sin embargo, para obtener ecuaciones de movimiento más eficientes, se puede recurrir a la segunda ley de Newton desarrollando un nuevo conjunto de ecuaciones de movimiento denominado, modelo de Newton-Euler. Esta formulación consiste en un conjunto de ecuaciones hacia delante y hacia atrás basadas en la realización de productos vectoriales.

Antes de desarrollar la formulación establezcamos la notación que se utilizará:

(x_o, y_o, z_o) es el sistema de coordenadas de la base.
$(x_{i-1}, y_{i-1}, z_{i-1})$ es el sistema de coordenadas fijo i-1 con origen en O^*.
(x_i, y_i, z_i) es el sistema de coordenadas fijo i con origen en O'.
p_i es el vector que va al origen O' desde el sistema de coordenadas de la base.
p_{i-1} es el vector que va al origen O^* desde el sistema de coordenadas de la base.
p_i^* es el vector que va al origen O' desde el sistema de coordenadas $(x_{i-1}, y_{i-1}, z_{i-1})$ que tiene como origen O^*.

En la figura (2.5) mostramos los sistemas de coordenadas y los vectores indicados:

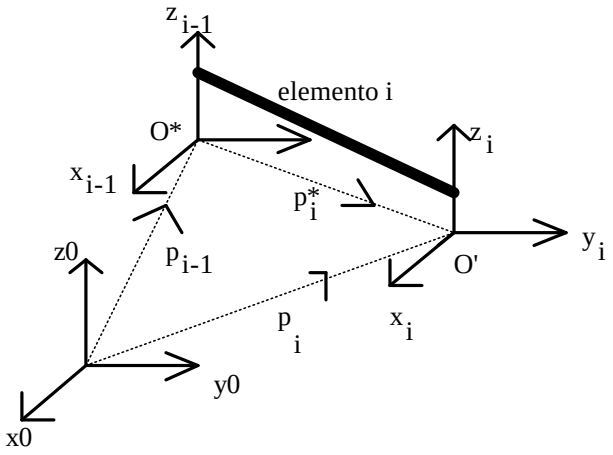


Figura 2.5 – Vectores y Sistemas de coordenadas considerados.

La base de esta formulación radica en establecer la relación entre los sistemas de coordenadas móviles con origen en O* y O' situados respectivamente en las articulaciones i-1 e i (en los extremos del elemento rígido que une las articulaciones i e i-1 y el sistema fijo o inercial situado en la base del robot. Es decir, habremos de buscar las relaciones entre las variables relativas a sistemas de coordenadas móviles con estas mismas variables relativas al sistema inercial situado en la base. Las variables a considerar no solo son las coordenadas de posición sino también las velocidades y las aceleraciones. A partir de las aceleraciones también tendremos las relaciones para los pares o torques y las fuerzas sin más que tener presente las ecuaciones de la mecánica que ligan dichas magnitudes. Seguidamente mostramos las variables de interés así como la notación que será utilizada de cara a expresar matemáticamente las relaciones entre las diferentes variables.

w_i^* = velocidad angular del sistema con origen O' respecto al O*.

w_i = velocidad angular del sistema con origen O' respecto al sistema de la base.

v_i^* = velocidad lineal del sistema con origen O' respecto al O*.

v_i = velocidad lineal del sistema con origen O' respecto al sistema de la base.

$\dot{w}_i^*$ = aceleración angular del sistema con origen O' respecto al O*.

$\dot{w}_i$ = aceleración angular del sistema con origen O' respecto al sistema de la base.

$\dot{v}_i^*$ = aceleración lineal del sistema con origen O' respecto al O*.

$\dot{v}_i$ = aceleración lineal del sistema con origen O' respecto al sistema de la base.

${}^{i-1}R_i$ = Matriz de Rotación. Sus columnas son los vectores de la base del sistema i expresados en función de los vectores de la base del sistema i-1. Recordemos que corresponde a una matriz 3x3 que cumple la siguiente propiedad ${}^{i}R_{i-1} = ({}^{i-1}R_i)^{-1} = ({}^{i-1}R_i)^T$.

m_i = masa del elemento i.

$\vec{s}_i$ = posición del centro de masas del elemento i desde el origen del sistema (x_i, y_i, z_i).

$\vec{r}_i$ = posición del centro de masas del elemento i desde el origen del sistema de la base.

$\vec{v}_i = \frac{d\vec{r}_i}{dt}$ es la velocidad lineal del centro de masas del elemento i.

$\vec{a}_i = \frac{d\vec{v}_i}{dt}$ es la aceleración lineal del centro de masas del elemento i.

F_i = Fuerza total externa ejercida sobre el elemento i en el centro de masas.

N_i = Momento de fuerza total externo ejercido sobre el elemento i en el centro de masas.

I_i = Matriz de inercia del elemento i respecto de su centro de masas con respecto al sistema de coordenadas de la base.

f_i = Fuerza ejercida sobre el elemento i por el elemento i-1 en el sistema de coordenadas $(x_{i-1}, y_{i-1}, z_{i-1})$ para soportar al elemento i y los elementos por encima del elemento i.

n_i = Momento de fuerza ejercido sobre el elemento i por el elemento i-1 en el sistema de coordenadas $(x_{i-1}, y_{i-1}, z_{i-1})$.

q_i = variable generalizada de la articulación i correspondiente al ángulo girado en articulaciones de revolución y al desplazamiento en las articulaciones prismáticas.

$\dot{q}_i$ = velocidad generalizada de la articulación i correspondiente a la velocidad angular en una articulación de revolución y velocidad lineal en una articulación prismática.

$\ddot{q}_i$ = aceleración generalizada de la articulación i correspondiente a la aceleración angular en una articulación de revolución y a la aceleración lineal en el caso de una articulación prismática.

Γ_i = Par o fuerza aplicada en la articulación i.

Una vez definidas todas las variables vamos a establecer las ecuaciones que relacionan las mismas, a la vez que estableceremos su procedimiento de cálculo. Para el caso general de n articulaciones, debemos tener en cuenta las siguientes relaciones.

$$
\begin{aligned}
{}^{i}R_0 w_i &= \begin{cases} {}^{i}R_{i-1}({}^{i-1}R_0 w_{i-1} + z_0 \dot{q}_i) & \text{si el elemento i es rotacional} \\ {}^{i}R_{i-1}({}^{i-1}R_0 w_{i-1}) & \text{si el elemento i es traslacional} \end{cases} \\
{}^{i}R_0 \dot{w}_i &= \begin{cases} {}^{i}R_{i-1}\left[{}^{i-1}R_0 \dot{w}_{i-1} + z_0 \ddot{q}_i + ({}^{i-1}R_0 w_{i-1}) \times z_0 \dot{q}_i\right] & \text{si el elemento i es rotacional} \\ {}^{i}R_{i-1}({}^{i-1}R_0 \dot{w}_{i-1}) & \text{si el elemento i es traslacional} \end{cases} \\
{}^{i}R_0 \dot{v}_i &= \begin{cases} ({}^{i}R_0 \dot{w}_i) \times ({}^{i}R_0 p_i^*) + ({}^{i}R_0 w_i) \times \left[({}^{i}R_0 w_i) \times ({}^{i}R_0 p_i^*)\right] + \\ \quad + {}^{i}R_{i-1}({}^{i-1}R_0 \dot{v}_{i-1}) & \text{si el elemento i es rotacional} \\ {}^{i}R_{i-1}(z_0 \ddot{q}_i + {}^{i-1}R_0 \dot{v}_{i-1}) + ({}^{i}R_0 \dot{w}_i) \times ({}^{i}R_0 p_i^*) + \\ + 2({}^{i}R_0 w_i) \times ({}^{i}R_{i-1} z_0 \dot{q}_i) + \\ + ({}^{i}R_0 w_i) \times \left[({}^{i}R_0 w_i) \times ({}^{i}R_0 p_i^*)\right] & \text{si el elemento i es traslacional} \end{cases} \\
{}^{i}R_0 \vec{a}_i &= ({}^{i}R_0 \dot{w}_i) \times ({}^{i}R_0 \vec{s}_i) + ({}^{i}R_0 w_i) \times \left[({}^{i}R_0 w_i) \times ({}^{i}R_0 \vec{s}_i)\right] + {}^{i}R_0 \dot{v}_i
\end{aligned}
\qquad (2.41)
$$

Con x expresando el producto vectorial.

Obsérvese que con estas primeras ecuaciones se pueden conocer variables de la articulación i si conocemos estas mismas variables para la articulación i-1. Esto implica que se puede realizar un cálculo de un cierto conjunto de variables hacia delante, es decir, partiendo de la articulación 1 podemos obtener a través de las ecuaciones mostradas los valores de las variables para la articulación 2 y así sucesivamente. Por otra parte existe un segundo conjunto de ecuaciones a partir de las cuales podemos conocer ciertas variables de la articulación i-1 si conocemos los valores de dichas variables para la articulación i. Estas son:

$$
\begin{aligned}
{}^{i}R_0 f_i &= {}^{i}R_{i+1}({}^{i+1}R_0 f_{i+1}) + m_i \, {}^{i}R_0 \vec{a}_i \\
{}^{i}R_0 n_i &= {}^{i}R_{i+1}\left[{}^{i+1}R_0 n_{i+1} + ({}^{i+1}R_0 \, p_i^*) \times ({}^{i+1}R_0 f_{i+1})\right] + \\
&\quad + ({}^{i}R_0 \, p_i^* + {}^{i}R_0 \, \vec{s}_i) \times ({}^{i}R_0 F_i) + \\
&\quad + ({}^{i}R_0 \, I_i \, {}^{0}R_i)({}^{i}R_0 \, \dot{w}_i) + ({}^{i}R_0 \, w_i) \times \left[({}^{i}R_0 \, I_i \, {}^{0}R_i)({}^{i}R_0 \, w_i)\right] \\
\Gamma_i &= \begin{cases} ({}^{i}R_0 n_i)^T ({}^{i}R_{i-1} z_0) + b_i \dot{q}_i & \text{si el elemento i es rotacional} \\ ({}^{i}R_0 f_i)^T ({}^{i}R_{i-1} z_0) + b_i \dot{q}_i & \text{si el elemento i es tralacional} \end{cases}
\end{aligned}
\qquad (2.42)
$$

Debemos precisar que $z_0 = (0,0,1)^T$ y b_i hace referencia al coeficiente de amortiguamiento viscoso para la articulación i. Dicho coeficiente como primera aproximación suele tomarse igual a cero con objeto de simplificar las ecuaciones. La aplicación del modelo de Newton-Euler para resolver el problema de la dinámica inversa por tanto se puede dividir en dos pasos.

Como primer paso hemos de partir de unas condiciones iniciales para las velocidades angulares y lineales del sistema de referencia colocado en la base. Dichas velocidades

generalmente son nulas dado que suponemos que el sistema de coordenadas ligado a la base está en reposo, sin embargo la aceleración viene determinada por la aceleración de la gravedad como $\dot{v}_0 = (g_x, g_y, g_z)^T$ (dependiendo como se tome el sistema de referencia) donde $|g| = 9.8062 \frac{m}{s^2}$. Esto nos permite a través de las primeras ecuaciones del modelo de Newton-Euler obtener de forma iterativa los valores de las velocidades y aceleraciones angulares de los orígenes de los sistemas de referencia ligados a cada una de las articulaciones, además de las aceleraciones lineales de los mismos. Permitiéndonos también el obtener la aceleración lineal del centro de masas de cada una de las articulaciones.

Después en un segundo paso haciendo uso de los valores calculados en el primer paso, así como suponiendo que $n_{n+1} = f_{n+1} = 0$, lo cual equivale a despreciar el efecto de la carga no siendo siempre posible, podemos aplicando el segundo conjunto de ecuaciones calcular los pares y fuerzas de cada una de las articulaciones

A la vista del procedimiento de cálculo mostrado se puede observar que la complejidad desde el punto de vista computacional es menor en el caso del modelo de Newton-Euler que en el modelo de Euler-Lagrange dado que en el primer caso el cálculo se reduce a la multiplicación de vectores. El número de multiplicaciones en el modelo de Euler-Lagrange es de $\frac{128}{3}n^4 + \frac{512}{3}n^3 + \frac{739}{3}n^2 + \frac{160}{3}n$ donde n es el número de articulaciones mientras que en el modelo de Newton-Euler el número de multiplicaciones es igual a 132n, en cuanto a las sumas el modelo de Euler-Lagrange realiza $\frac{98}{3}n^4 + \frac{781}{6}n^3 + \frac{559}{3}n^2 + \frac{245}{6}n$ sumas frente a las 111n-4 realizadas en el modelo de Newton-Euler. Como se puede ver la complejidad del método de Newton-Euler es del orden del número de articulaciones del robot. Tanto en el modelo de Newton-Euler como en el de Euler-Lagrange las ecuaciones resultantes son complejas, por lo que sería de interés el aportar técnicas alternativas para caracterizar y controlar a los robots.

Capítulo 3
DISEÑO DE UNA RED NEURONAL DINÁMICA

3

DISEÑO DE UNA RED NEURONAL DINÁMICA

En este capítulo se muestra el modelo de red neuronal propuesto y los diferentes esquemas de aprendizaje aplicados, así como los detalles de la implementación. El orden de presentación en este capítulo se corresponde con el siguiente: Comenzamos dando el modelo, para después exponer diferentes algoritmos de aprendizaje aplicados al mismo, comentando algunos detalles referentes a las velocidades de aprendizaje e inicializaciones. Posteriormente se aplica la red y el algoritmo de aprendizaje a diferentes problemas, surgiendo de tal análisis diferentes modificaciones sobre el algoritmo de aprendizaje, para finalmente destacar las ventajas e inconvenientes del mismo, haciendo especial hincapié en estos últimos de cara a señalar algunas claves para subsanarlos.

3.1 ARQUITECTURA DE RED NEURONAL PROPUESTA.

En este apartado presentamos la arquitectura de red neuronal que ha sido utilizada en este trabajo. Las neuronas se disponen en forma de capas unas detrás de otras, con lo cual esta red adoptará la forma de una red feedforward. Cada neurona viene descrita por las siguientes ecuaciones:

$$n_i^k = \sum_{j=1}^{m_{k-1}} w_{ij}^k y_j^{k-1} + w_{i0}^k \qquad (3.1a)$$

$$\dot{x}_i^k = -a_i^k(x_i^k)\left[b_i^k(x_i^k) + n_i^k\right] \qquad (3.1b)$$

$$y_i^k = g(x_i^k) \qquad (3.1c)$$

Donde:

n_i^k = Entrada a la neurona i de la capa k.

y_i^k = Salida correspondiente a la neurona de la capa k.

w_{ij}^k = Peso que conecta la neurona j de la capa k-1 con la neurona i de la capa k.

w_{i0}^k = Peso de referencia del bias correspondiente a la neurona i de la capa k.

x_i^k = Estado correspondiente a la neurona i de la capa k.

m_k = Número de neuronas en la capa k.

$g(...)$ = Función no lineal (en nuestro caso utilizaremos la función tangente hiperbólica).

Obsérvese que en la ecuación (3.1b) $a_i^k(x_i^k)$ y $b_i^k(x_i^k)$ son dos coeficientes dependientes del estado, definidos para cada neurona. Estos pueden adoptar diferentes formas funcionales, dando este hecho una cierta generalidad al modelo. En nuestro caso dichas funciones las tomaremos como $a_i^k(x_i^k, a)$ y $b_i^k(x_i^k, b)$, es decir, dependientes de un parámetro a y otro b respectivamente. En esta sección tomaremos estos parámetros como constantes.

La diferencia fundamental entre el modelo propuesto y las redes neuronales estáticas está en la aparición de un término dinámico representado por la ecuación (3.1b). En la figura (3.1) se muestra un diagrama de la red propuesta, en donde se han obviado los pesos de bias por motivos de simplicidad. Obsérvese que las cajas representan las expresiones (3.1b).

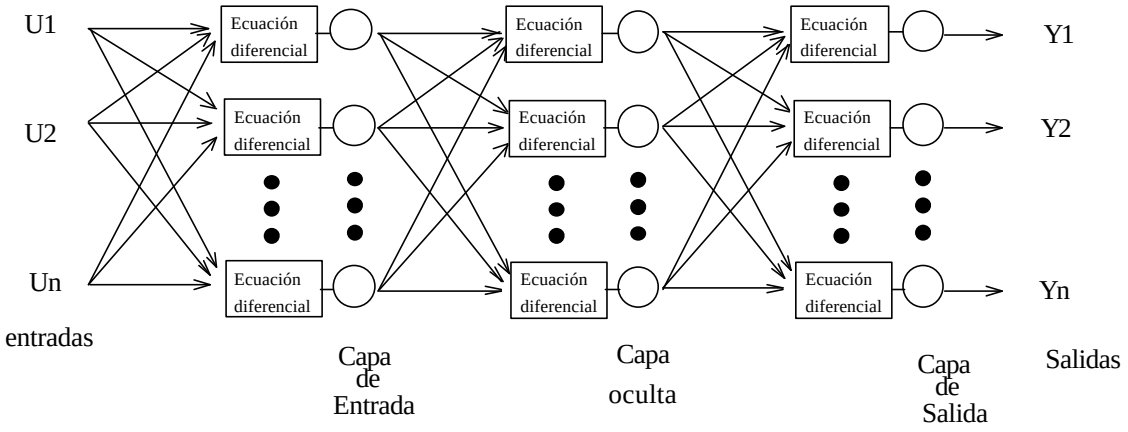


Figura 3.1 – Red neuronal de 3 capas

3.2 ALGORITMO DE ENTRENAMIENTO.

Dado el modelo establecido en la sección anterior, ahora propondremos un método que permita el aprendizaje de cualquier relación entre las entradas y salidas. Tal método se basará en el ajuste conveniente de los pesos w_{ij}^k correspondientes a las conexiones de las neuronas que configuran la red. Concretamente el método empleado está basado en el backpropagation pero teniendo en cuenta la estructura dinámica de la red. El primer paso consiste en definir una función de error que debe minimizarse. Dicha función de error viene definida de la siguiente forma:

$$E = \frac{1}{2}\sum_{i=1}^{M}\left(d_i - y_i^L\right)^2 \qquad (3.2)$$

Donde :

M = Número de neuronas en la última capa
L = Número de capas en la red.
d_i = Salidas deseadas.
y_i^L = Salidas de las neuronas de la última capa.

Para conseguir los pesos que minimicen la función de error, haremos uso del método del gradiente descendente, basado en que de forma iterativa se modifican los valores de los pesos de acuerdo a la ecuación (3.2) con el objetivo de que mediante esas modificaciones sucesivas nos aproximemos a los valores deseados:

$$\dot{w}_{ij}^k(t) = -\eta \frac{\partial E}{\partial w_{ij}^k} \tag{3.3}$$

Donde $\dot{w}_{ij}^k(t)$ constituye la derivada respecto al tiempo del peso que conecta la neurona j de la capa k-1 con la neurona i de la capa k para el instante de tiempo t. Obsérvese que j toma valores enteros en orden creciente comenzando en cero, de tal forma que reservamos el valor 0 de j para el peso correspondiente a la entrada de bias perteneciente a la neurona i de la capa k. Obsérvese también que el coeficiente η es una constante a la que denominamos velocidad de aprendizaje.

Ahora aplicando la regla de la cadena a $\frac{\partial E}{\partial w_{ij}^k}$ tenemos que:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial n_i^k}\frac{\partial n_i^k}{\partial w_{ij}^k} \tag{3.4}$$

Definiendo δ_i^k como:

$$\delta_i^k = \frac{\partial E}{\partial n_i^k} \tag{3.5}$$

y teniendo en cuenta que $\frac{\partial n_i^k}{\partial w_{ij}^k}$ es igual a la salida correspondiente a la neurona j de la capa k-1 o la unidad en el caso de un peso de bias sin más que tener presente la ecuación (3.1a) llegamos a la expresión más explícita:

$$\dot{w}_{ij}^k(t) = \begin{cases} -\eta \delta_i^k y_j^{k-1} \;\forall\; j \neq 0 \\ -\eta \delta_i^k \qquad j = 0 \end{cases} \tag{3.6a}$$

donde la ecuación (3.2) se ha expresado de forma diferente según se trate de pesos de bias o no. Esta ecuación a su vez se puede presentar en su forma discreta resultando:

$$w_{ij}^k(m+1) = w_{ij}^k(m) + \begin{cases} -\eta \delta_i^k y_j^{k-1} \;\forall\; j \neq 0 \\ -\eta \delta_i^k \qquad j = 0 \end{cases} \tag{3.6b}$$

En donde la variable m hace referencia a los diferentes instantes de tiempo en los que evaluamos la ecuación. Obsérvese por otro lado que la ecuación (3.6b) va a ser la ecuación de actualización de los pesos, por lo que sólo resta calcular δ_i^k para determinar los nuevos pesos a partir de los anteriores. Para calcular δ_i^k dividamos la red por capas:

Capa de salida: Aplicando la regla de la cadena sobre la expresión (3.5) tendremos:

$$\delta_i^k = \frac{\partial E}{\partial y_i^k}\frac{\partial y_i^k}{\partial x_i^k}\frac{\partial x_i^k}{\partial n_i^k} \tag{3.7}$$

Ahora siendo k=L, la última capa de la red neuronal, se observa fácilmente que:

$$\frac{\partial E}{\partial y_i^k} = -\left(d_i - y_i^L\right) = -e_i \tag{3.8}$$

y :

$$\frac{\partial y_i^k}{\partial x_i^k} = g'\left(x_i^k\right) \tag{3.9}$$

Para obtener el último término en (3.7) consideremos la siguiente expresión:

$$\frac{\partial x_i^k}{\partial n_i^k}\frac{\partial n_i^k}{\partial t} = \dot{x}_i^k = -a_i^k\left(x_i^k\right)\left[b_i^k\left(x_i^k\right) + n_i^k\right] \tag{3.10}$$

Con objeto de simplificar el algoritmo, en lo sucesivo supondremos que la variación del estado debida a la sinapsis es la única significativa, despreciando la variación explícita con respecto al tiempo. Tomando en consideración esta suposición la ecuación (3.10) se transformaría en la siguiente ecuación:

$$\frac{\partial x_i^k}{\partial n_i^k} = \frac{-a_i^k\left(x_i^k\right)\left[b_i^k\left(x_i^k\right) + n_i^k\right]}{\frac{\partial n_i^k}{\partial t}} \tag{3.11}$$

Ahora con el propósito de determinar una expresión para $\frac{\partial n_i^k}{\partial t}$, observemos que los pesos pueden considerarse que varían más lentamente que los estados y salidas de las neuronas, quedando tal derivada en la siguiente forma:

$$\frac{\partial n_i^k}{\partial t} = \frac{\partial\left(\sum_{j=1}^{m_{k-1}} w_{ij}^k y_j^{k-1}\right)}{\partial t} = \sum_{j=1}^{m_{k-1}}\left(w_{ij}^k \frac{dy_j^{k-1}}{dx_j^{k-1}}\dot{x}_j^{k-1}\right) = \tag{3.12}$$
$$= \sum_{j=1}^{m_{k-1}}\left(w_{ij}^k g'\left(x_j^{k-1}\right)\left(-a_j^{k-1}\left(x_j^{k-1}\right)\left[b_j^{k-1}\left(x_j^{k-1}\right)+n_j^{k-1}\right]\right)\right)$$

donde m_{k-1} se refiere al número de neuronas en la capa k-1. Teniendo presente esta ecuación obtenemos para δ_i^k :

$$\delta_i^k = -\left(d_i - y_i^L\right) g'\left(x_i^k\right) \frac{-a_i^k(x_i^k)\left[b_i^k(x_i^k)+n_i^k\right]}{\sum_{l=1}^{m_{k-1}}\left(w_{lj}^k g'\left(x_l^{k-1}\right)\left(-a_l^{k-1}\left(x_l^{k-1}\right)\left[b_l^{k-1}\left(x_l^{k-1}\right)+n_l^{k-1}\right]\right)\right)}$$

(3.13)

Capas ocultas: En neuronas de estas capas aplicaremos la siguiente fórmula que relaciona los δ_i^k de éstas con los de neuronas existentes en las capas que se encuentran delante de las mismas. Así tendremos:

$$\delta_i^k = \sum_{j=1}^{m_{k+1}}\left\{\frac{\partial E}{\partial n_j^{k+1}}\frac{\partial n_j^{k+1}}{\partial y_i^k}\right\}\frac{\partial y_i^k}{\partial x_i^k}\frac{\partial x_i^k}{\partial n_i^k} =$$
$$= \sum_{j=1}^{m_{k+1}}\left\{\delta_j^{k+1} w_{ji}^{k+1}\right\} g'\left(x_i^k\right) \frac{-a_i^k(x_i^k)\left[b_i^k(x_i^k)+n_i^k\right]}{\sum_{l=1}^{m_{k-1}}\left(w_{lj}^k g'\left(x_l^{k-1}\right)\left(-a_l^{k-1}\left(x_l^{k-1}\right)\left[b_l^{k-1}\left(x_l^{k-1}\right)+n_l^{k-1}\right]\right)\right)}$$

(3.14)

Primera capa: Para la primera capa o capa de entrada no existen salidas de neuronas en capas previas. Supondremos entonces que las entradas a la red constituyen las salidas y estados de una capa ficticia detrás de la primera:

$$y_i^0 = u_i;\ x_i^0 = u_i \tag{3.15}$$

Aplicando la regla de la cadena a δ_i^k tendremos:

$$\delta_i^1 = \sum_{j=1}^{m_2}\left\{\frac{\partial E}{\partial n_j^2}\frac{\partial n_j^2}{\partial y_i^1}\right\}\frac{\partial y_i^1}{\partial x_i^1}\frac{\partial x_i^1}{\partial n_i^1} \tag{3.16}$$

Desarrollando esta expresión quedará:

$$d_i^1 = \sum_{j=1}^{m_2} \{d_j^2 w_{ji}^2\} g'(x_i^2) \frac{-a_i^1(x_i^1)[b_i^1(x_i^1)+n_i^1]}{\sum_{l=1}^{m_0} (w_{il}^1 \dot{u}_l)} \qquad (3.17)$$

Aplicando las ecuaciones vistas hasta ahora se pueden obtener los nuevos pesos que variarán para minimizar la función de error definida en (3.2) siendo en general este ajuste insuficiente. Sin embargo recordemos que en la sección anterior comentamos que las funciones $a_i^k(x_i^k)$ y $b_i^k(x_i^k)$ correspondientes a la parte dinámica de las neuronas presentes en la ecuación (3.1b), las haríamos dependientes explícitamente de dos parámetros a y b respectivamente, denotando las funciones resultantes como $a_i^k(x_i^k, a)$ y $b_i^k(x_i^k, b)$. Esto abre una nueva línea en pro de minimizar la función de error, ya que variando estos se actúa directamente sobre la parte dinámica, de tal forma que se establece otro mecanismo de ajuste que refuerza el proceso de aprendizaje.

Antes de proseguir en la exposición sería conveniente modificar la ecuación de estado, de tal forma que se presente más adecuada de cara a su programación en computadoras . Tomando la ecuación de estado definida en (3.1b):

$$\dot{x}_i^k = -a_i^k(x_i^k)[b_i^k(x_i^k)+n_i^k] \qquad (3.18)$$

y aproximando la derivada sobre el estado como cociente de incrementos la ecuación queda:

$$x_i^k(\mathrm{m}+1) = x_i^k(\mathrm{m}) - H\, a_i^k(x_i^k(m))[b_i^k(x_i^k(m))+n_i^k(m)] \qquad (3.19)$$

siendo esta la ecuación de estado utilizada a efectos de implementación. Por otra parte H es una constante que corresponde al tiempo de discretización.

3.3 ALGORITMOS DE AJUSTE SOBRE LOS PARÁMETROS DE LAS ECUACIONES DE ESTADO.

En esta sección analizaremos el ajuste de los parámetros a y b de las ecuaciones de estado. Estudiaremos primeramente el ajuste de estos parámetros únicamente para las neuronas de la última capa, para posteriormente considerar el ajuste de los parámetros de todas las neuronas de la red.

3.3.1 ALGORITMO DE AJUSTE DE LOS A Y B CORRESPONDIENTES A LAS NEURONAS DE LA ÚLTIMA CAPA.

Como primer intento de cara a encontrar un algoritmo de optimización de los parámetros de la ecuación de estado nos hemos centrado en los parámetros de las ecuaciones de estado de las neuronas de las últimas capas. La última capa nos permite de una forma fácil obtener los estados deseados a partir de las salidas deseadas mediante la función inversa de la tangente hiperbólica.

El método de ajuste dependerá de la forma de la ecuación de estado, por lo que se hace necesario escoger alguna. En nuestro caso utilizaremos las siguientes expresiones para $a_i^k(x_i^k, a)$ y $b_i^k(x_i^k, b)$, donde la dependencia explícita de a y b en la notación será eliminada por motivos de simplicidad:

$$a_i^k(x_i^k) = a \tag{3.20a}$$
$$b_i^k(x_i^k) = b\ x_i^k \tag{3.20b}$$

De tal forma que la ecuación de estado quedará como:

$$x_i^k(\mathrm{m}+1) = x_i^k(\mathrm{m}) - Ha\left[b x_i^k(m) + n_i^k(m)\right] \tag{3.21}$$

Agrupemos la ecuación de estado de la siguiente forma:

$$x_i^k(\mathrm{m}+1) = x_i^k(\mathrm{m})(1 - Hab) - Ha\, n_i^k(m) \tag{3.22}$$

Ahora definiendo los siguientes pesos:

$$WR(1,1) = 1 - Hab \tag{3.23a}$$
$$WR(1,2) = -Ha \tag{3.23b}$$

Observamos que la ecuación de estado responde a una neurona de una red neuronal recurrente en el tiempo, donde las ecuaciones (3.23) definen los pesos y x_i^k corresponde a la salida de esa neurona hipotética. Seguidamente se muestra un diagrama de tal neurona:

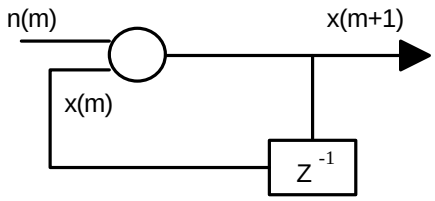


Figura (3.2) - Neurona de una red neuronal recurrente en el tiempo.

En la figura (3.2) n(m) corresponde a la entrada, que en nuestro caso particular será $n_i^k(m)$ mientras que x(m+1) corresponde a la salida de la neurona siendo en nuestro caso $x_i^k(\mathrm{m})$.

En vista de lo expuesto anteriormente únicamente habremos de aplicar un algoritmo de entrenamiento de los varios que existen para una red neuronal recurrente de tiempo discreto. Nosotros escogeremos el algoritmo de aprendizaje recurrente en tiempo real (RTRL) [WIL89]. Las salidas deseadas, se corresponderán con los estados deseados. El mecanismo de ajuste de a y b se conseguirá aplicando el RTRL, con una salida, una entrada y dos pesos, expresados estos últimos en función de a y b por las ecuaciones (3.23). El RTRL nos permitirá obtener los pesos adecuados y una vez obtenidos podremos hallar los nuevos parámetros a y b sin más que invertir las ecuaciones (3.23). Este mecanismo se integraría con el ajuste sobre los pesos, reforzando de esta forma el aprendizaje. Hemos de notar que los a y b pertenecientes a neuronas presentes en las capas ocultas y en la capa de entrada en este esquema se mantienen constantes. A la vista de este hecho parece razonable buscar algún algoritmo de ajuste que no solamente

tenga presente las neuronas de la última capa con el fin de mejorar de esta forma el aprendizaje. Un algoritmo de tales características se presenta en la próxima sección.

3.3.2 ALGORITMO DE AJUSTE DE LOS A Y B CORRESPONDIENTES A TODAS LAS NEURONAS DE LA RED NEURONAL.

El problema principal para generar un algoritmo de estas características, es buscar una forma de propagar el error hacia atrás. Comencemos planteando el método del gradiente descendente para los a y b correspondientes a cada neurona. Por comodidad de ahora en adelante denotaremos estos parámetros con un índice k correspondiente a la capa y un índice i correspondiente a la neurona dentro de la capa, quedando la fórmula del gradiente descendente para ambos como:

$$\dot{a}_i^k = -\eta_a \frac{\partial E}{\partial a_i^k} \tag{3.24a}$$

$$\dot{b}_i^k = -\eta_b \frac{\partial E}{\partial b_i^k} \tag{3.24b}$$

Las ecuaciones (3.24) serán las ecuaciones utilizadas en la actualización de los parámetros a_i^k y b_i^k. Dada la analogía de las mismas utilizaremos la ecuación (3.24a) para los desarrollos posteriores, siendo aplicables en igual forma a la ecuación (3.24b). Desarrollemos δ_i^k de la siguiente forma:

$$\delta_i^k = \frac{\partial E}{\partial n_i^k} = \frac{\partial E}{\partial a_i^k}\frac{\partial a_i^k}{\partial n_i^k} = \frac{\partial E}{\partial a_i^k}\frac{\partial x_i^k / \partial n_i^k}{\partial x_i^k / \partial a_i^k} \tag{3.25}$$

Ahora despejando $\frac{\partial E}{\partial a_i^k}$ tenemos:

$$\frac{\partial E}{\partial a_i^k} = \delta_i^k \frac{\partial x_i^k / \partial a_i^k}{\partial x_i^k / \partial n_i^k} \tag{3.26}$$

Esto nos permite poner la ecuación (3.24a) en la forma:

$$\dot{a}_i^k = -\eta_a \delta_i^k \frac{\partial x_i^k / \partial a_i^k}{\partial x_i^k / \partial n_i^k} \tag{3.27}$$

y análogamente nos permite poner la ecuación (3.24b) en la forma:

$$\dot{b}_i^k = -\eta_b \delta_i^k \frac{\partial x_i^k / \partial b_i^k}{\partial x_i^k / \partial n_i^k} \tag{3.28}$$

Estas dos ecuaciones marcarán las fórmulas de actualización para los parámetros a y b correspondientes a cada neurona. De cara a la programación estas expresiones se tomarán en su forma discreta:

$$a_i^k(m+1) = a_i^k(m) - \eta_a \delta_i^k \frac{\partial x_i^k / \partial a_i^k}{\partial x_i^k / \partial n_i^k} \tag{3.29a}$$

$$b_i^k(m+1) = b_i^k(m) - \eta_b \delta_i^k \frac{\partial x_i^k / \partial b_i^k}{\partial x_i^k / \partial n_i^k} \tag{3.29b}$$

Es importante destacar que el algoritmo propuesto es fácil de integrar con el algoritmo de ajuste de los pesos dado que las cantidades δ_i^k se utilizan tanto para un algoritmo como para el otro y por otro lado el cociente de derivadas parciales se obtiene sin más que disponer de la forma funcional de la ecuación de estado.

3.4 LAS VELOCIDADES DE APRENDIZAJE.

En el algoritmo de entrenamiento presentado en la sección 3.3 aparece un parámetro en las ecuaciones de actualización de los pesos que se denomina velocidad de aprendizaje. Esta se presentó como un valor constante que aparecía por igual en la actualización de todos los pesos. Con objeto de centrar el tema presentemos la ecuación de actualización de los pesos (3.6b):

$$w_{ij}^k(m+1) = w_{ij}^k(m) + \begin{cases} -\eta \delta_i^k y_j^{k-1} \ \forall\ j \neq 0 \\ -\eta \delta_i^k \qquad j = 0 \end{cases} \tag{3.30}$$

Sin embargo el modelo que nosotros proponemos plantea diferentes velocidades de aprendizaje según de que capa se trate y según se trate o no de pesos de referencia. Esta ligera generalización del modelo previamente expuesto en la sección 3.2, surge del hecho de que el comportamiento mejora considerablemente si a medida que nos acercamos a la capa de entrada a partir de la capa de salida aumentamos la velocidad de aprendizaje. Por tanto dentro de esta concepción más general la ecuación (3.30) queda en la forma:

$$w_{ij}^k(m+1) = w_{ij}^k(m) - \eta_k \delta_i^k y_j^{k-1} \tag{3.31a}$$

$$w_{i0}^k(m+1) = w_{i0}^k(m) - \eta_k^0 \delta_i^k \tag{3.31b}$$

Donde η_k^0 y η_k corresponden a las velocidades de aprendizaje para los pesos de referencia y no referencia de la capa k respectivamente. De igual forma ha de ampliar el esquema de aprendizaje para los a y b presentados en la sección 3.4.2, es decir, elegiremos las velocidades de aprendizaje tanto para a como para b diferentes según la capa. Estas formas de elegir las velocidades han mostrado ser más efectivas de cara al aprendizaje. En el siguiente apartado se ilustrará esta afirmación con algún ejemplo.

3.5 LA INICIALIZACIÓN DE PARÁMETROS.

En las secciones anteriores se han presentado diversos mecanismos de ajuste pero en ningún caso se ha hablado de que forma hemos de inicializar los diversos parámetros. Parece éste el momento de comentar estos detalles. Comencemos nuestro análisis con los pesos. Estos en general se inicializan al azar dentro de un rango de magnitud pequeño. La razón de escogerlos inicialmente pequeños se encuentra en intentar evitar que las neuronas se saturen, es decir, que los estados se sitúen en valores tales que al aplicarles la tangente hiperbólica el resultado no se encuentre en la zona de derivada nula. Es importante señalar que se han probado diversas formas de inicializar los pesos, siendo una de ellas la inicialización de Nguyen-Widrow [DEM93]. Sin embargo en los experimentos realizados no se han apreciado diferencias significativas respecto a los resultados obtenidos. Otros parámetros objeto de inicialización son los estados que han de tomar un valor pequeño, sin necesidad de ser al azar para las diferentes neuronas. La razón de escoger los estados pequeños es al igual que en el caso de los pesos, evitar una saturación prematura. De igual forma la inicialización de los parámetros a y b correspondientes a las diferentes neuronas se hará al azar y con valores de magnitud no muy altos.

3.6 APLICACIÓN DE LA RED Y DE LOS ALGORITMOS DE APRENDIZAJE.

Con objeto de determinar la bondad de la arquitectura y los algoritmos utilizados, así como averiguar cual de los algoritmos sobre el ajuste de los parámetros es más optimo, a la red inicialmente le hemos presentado un problema sencillo, consistente en aprender la operación identidad, es decir, las salidas son iguales a las entradas. Para aprender esta relación hemos utilizado una red compuesta por sólo dos neuronas, una en la capa de entrada y otra en la capa de salida. Por otra parte las formas de las funciones utilizadas en las ecuaciones de estado se han escogido como se describió en la sección anterior.

Como primera estrategia de aprendizaje hemos utilizado el algoritmo de aprendizaje expuesto en las secciones anteriores para los pesos, manteniendo constantes los parámetros correspondientes a las ecuaciones de estado. En la Figura 3.1 se presenta la evolución de la suma de errores cuadráticos para este caso. Se ha elegido como velocidad de aprendizaje para los pesos de la primera capa 0.001 y para los pesos de la segunda 0.0001. El error mínimo se consigue para la época 202 con un valor de 0.05028.

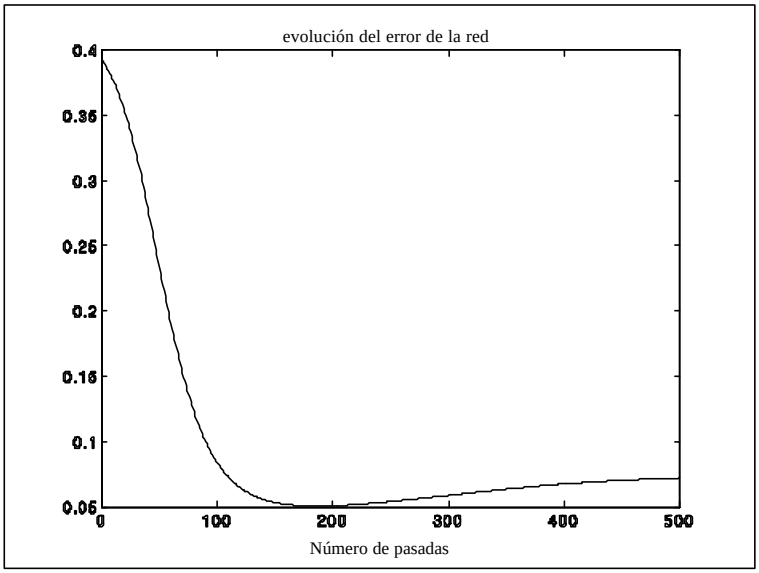


Figura 3.1 – Evolución de la suma de errores cuadráticos.

En la Figura 3.2 se presenta la salida de la red neuronal frente a la salida a aprender. Para el proceso de aprendizaje se han elegido 8 comandos en forma parabólica, de tal forma que la entrada a la red es una parábola constituida por 8 puntos mientras la salida es esta misma parábola. En la figura 3.2 se puede apreciar un acercamiento entre la salida de la red y la salida deseada, denominada esta última salida real.

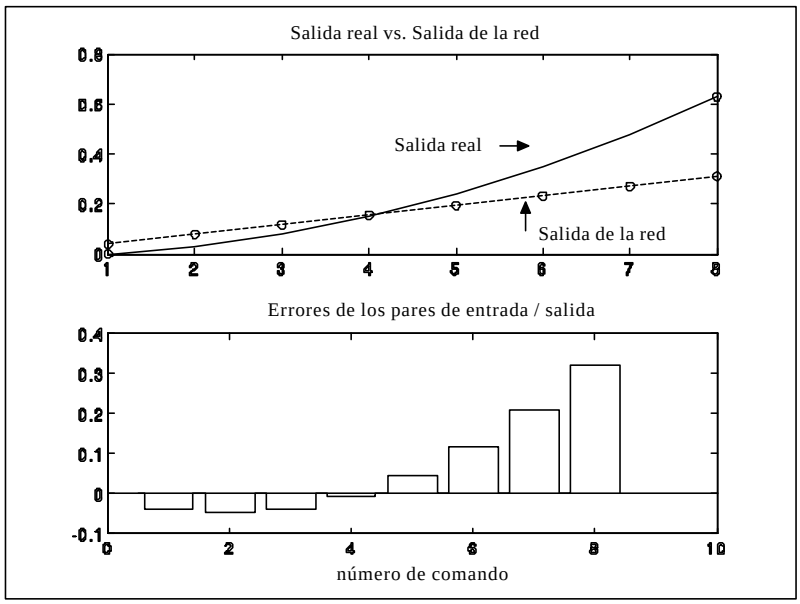


Figura 3.2 – Salida real vs. salida de la red.

A la vista de los resultados parece conveniente introducir nuevos mecanismos de ajuste de parámetros que contribuyan a reforzar el aprendizaje. El primer intento en este sentido sería el introducir el mecanismo de ajuste de los parámetros de la última capa

desarrollado en las secciones anteriores, es decir ajustar mediante el RTRL los parámetros de las neuronas de la última capa. En la figura 3.3 se muestra una curva de error cuadrático, donde se han utilizado las mismas velocidades de aprendizaje para los pesos que en el caso anterior y una velocidad de aprendizaje para los parámetros a y b de 0.01.

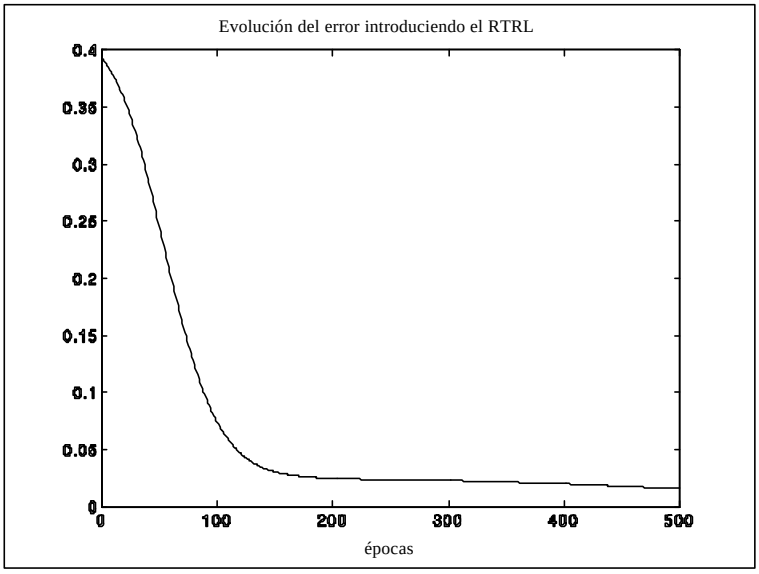


Figura 3.3 – Evolución de la suma de errores cuadráticos introduciendo RTRL.

Como se puede apreciar en la curva de error de la Figura 3.3, el comportamiento mejora considerablemente. El mínimo error se consigue en la época 500 con un valor de 0.01525. En la figura 3.4 presentamos la salida real respecto a la salida de la red, comprobándose la mejora significativa respecto a los resultados previos.

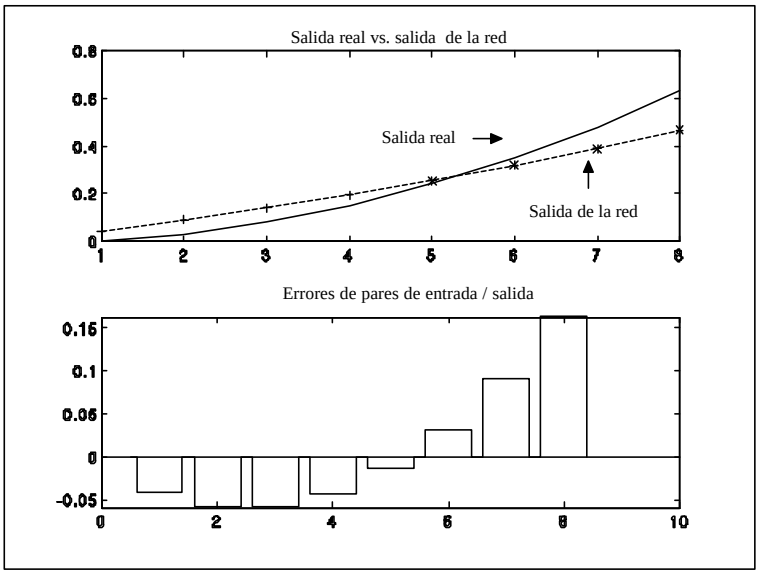


Figura 3.4 – Salida real vs. salida de la red.

Obsérvese que los * representan un acercamiento de la salida de la red hacia la curva real respecto a épocas previas y los + representan un alejamiento de la salida de la red de la curva real respecto a épocas previas.
Con el propósito de mejorar el proceso de aprendizaje se ha aumentado la velocidad de aprendizaje de los parámetros a 0.1. Sin embargo como se puede observar en la Figura 3.5 el proceso de aprendizaje se inestabiliza.

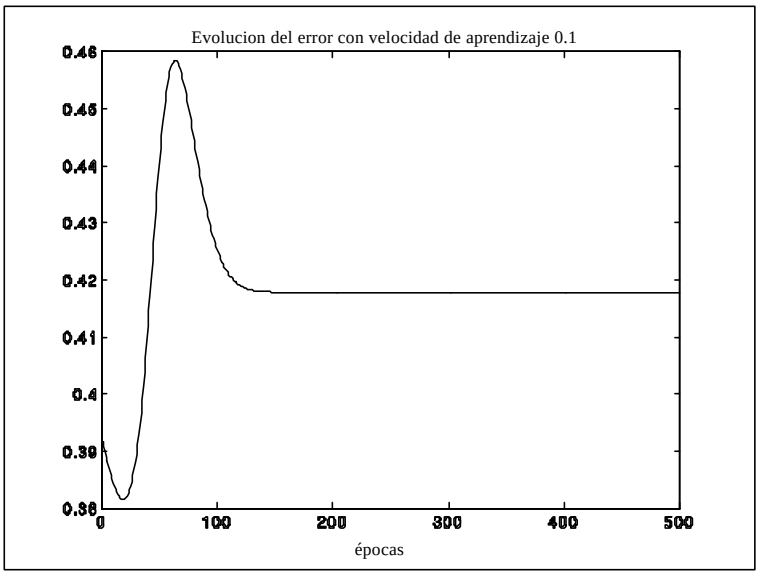


Figura 3.5 – Evolución de la suma de errores cuadráticos.

En un intento de mejorar los resultados parece conveniente extender el ajuste de los parámetros a y b a todas las neuronas de la red tal como se señaló en las secciones previas. En este caso hemos escogido las velocidades de aprendizaje de los pesos como se ha hecho en los casos anteriores y las velocidades de aprendizaje sobre los parámetros a y b como 0.1 para la primera capa y 0.01 para la capa de salida. Los resultados de este experimento se muestran en la figura 3.6, donde se puede apreciar un error cuadrático mínimo de 0.001429 después de 202 épocas.

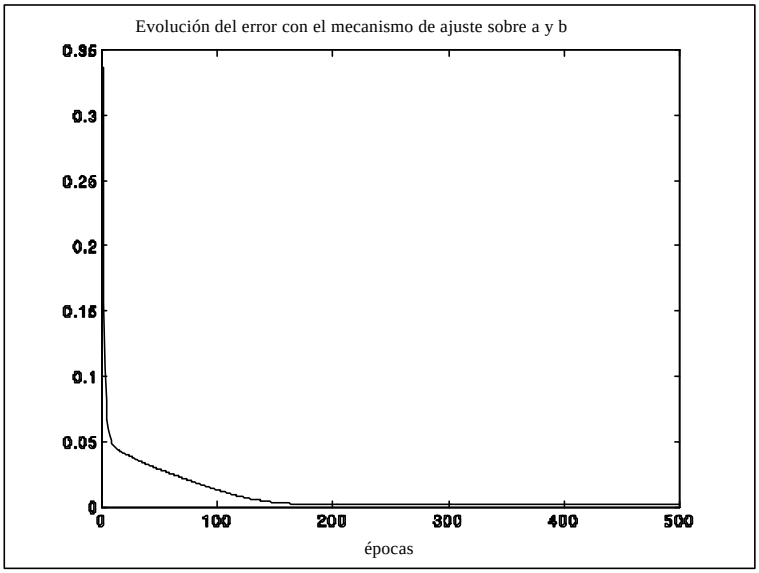


Figura 3.6 – Evolución de la suma de errores cuadráticos.

En la figura 3.7 se puede apreciar que forma adquiere la salida de la red en este caso :

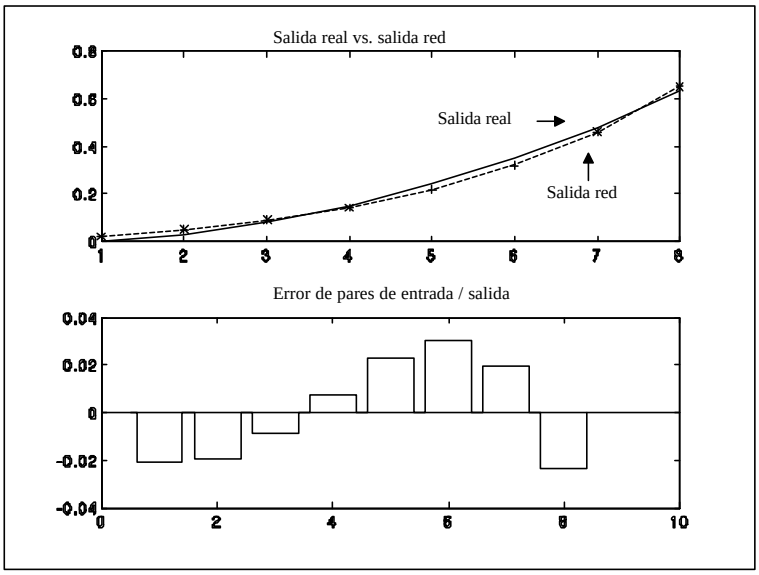


Figura 3.7 – Salida real vs. salida de la red.

Los resultados obtenidos nos llevan a considerar como mejor estrategia de aprendizaje la última utilizada, es decir, la actualización de los pesos conjuntamente con la actualización de los parámetros a y b para todas las neuronas. Es importante en este punto hacer hincapié sobre como se han tomado las velocidades de aprendizaje. En los experimentos realizados éstas han sido tomadas diferentes según las capas tanto en el caso de los parámetros a y b como en el caso de los pesos. Esta elección particular de las velocidades ha contribuido a obtener mejores resultados, de hecho han sido elegidas tal que aumenten a medida que nos acercamos desde la capa de salida a la capa de entrada. Sin embargo cuando las velocidades se eligen en orden creciente el proceso de aprendizaje se inestabiliza. En la figura 3.8 presentamos los resultados correspondientes

a un entrenamiento en donde únicamente se han intercambiado las velocidades de aprendizaje de los pesos respecto a los resultados presentados en la Figura 3.6 y 3.7.

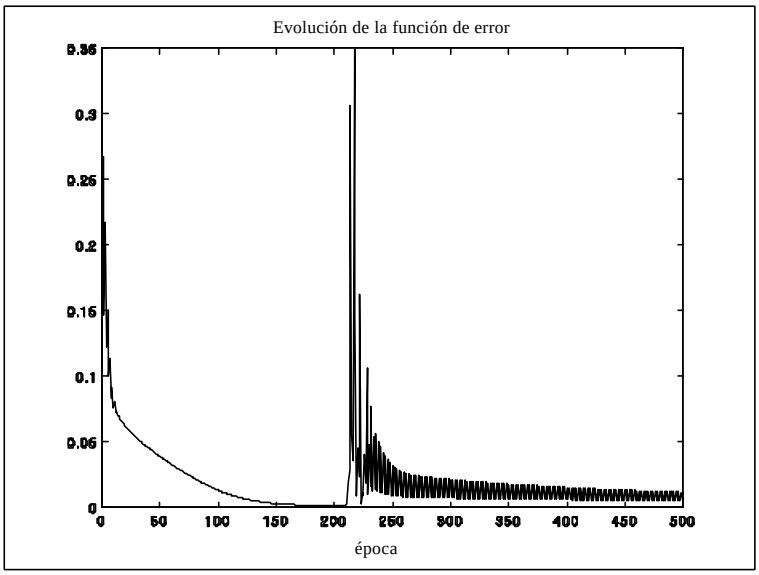


Figura 3.8 – Evolución de la suma de errores cuadráticos.

A continuación en la Figura 3.9 presentamos las salidas de la red frente a la salida objeto de aprendizaje para la época 250. En ésta se puede apreciar como la elección de las velocidades contribuye a un aprendizaje defectuoso.

También realizamos un ensayo con las velocidades de los parámetros a y b intercambiando las velocidades de las dos capas, es decir, tomando la velocidad mayor para la capa de salida, manteniendo las velocidades de aprendizaje para los pesos y las inicializaciones iguales a las utilizadas en el experimento presentado en la figura 3.6. Como puede observarse la curva de error presentada en la figura 3.10 refleja como esta elección de las velocidades paraliza la capacidad de aprendizaje de la red.

Los resultados de los diferentes experimentos presentados en esta sección muestran como mejor estrategia de aprendizaje la actualización de todos los pesos y parámetros de la red, eligiendo convenientemente las velocidades de aprendizaje tal que las mismas sean escogidas como crecientes desde la capa de salida a la capa de entrada. En lo sucesivo esta última aproximación será la utilizada.

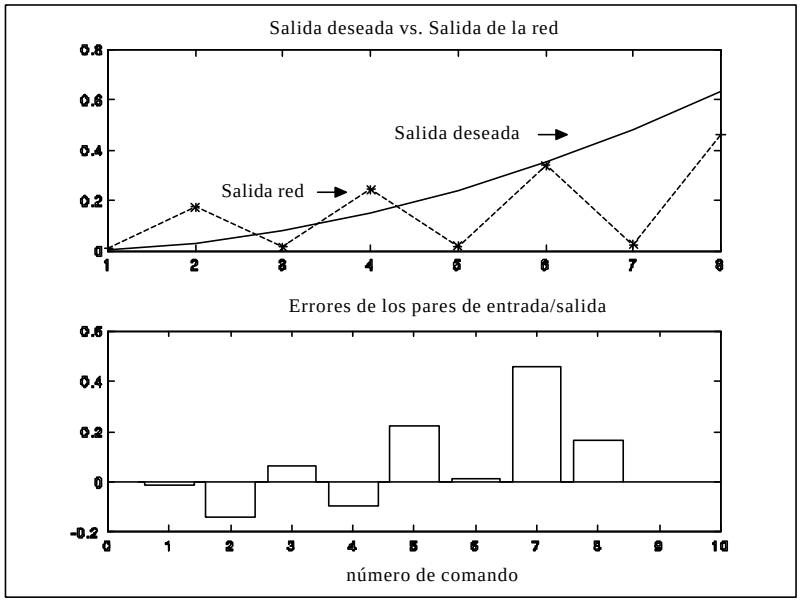


Figura 3.9 – Salida real vs. salida de la red.

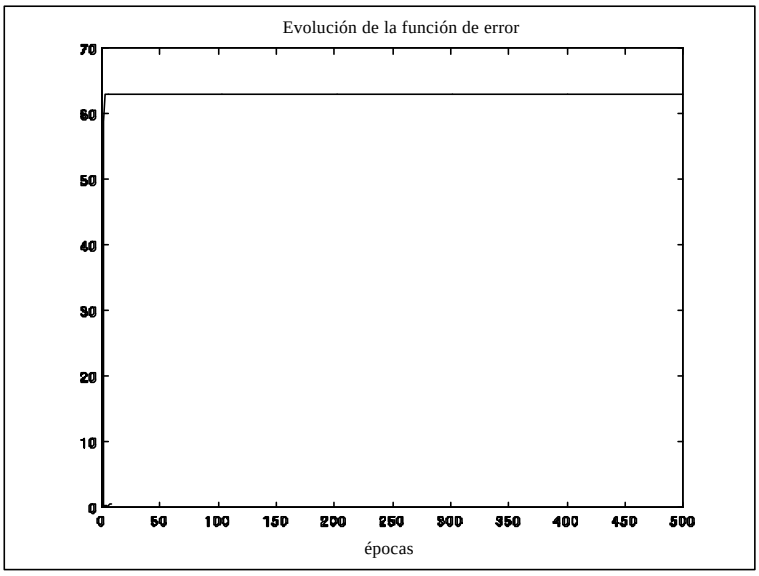


Figura 3.10 – Evolución de la suma de errores cuadráticos.

3.7 APLICACIÓN DE LA RED NEURONAL EN LA IDENTIFICACIÓN DE LA DINÁMICA DE UN ROBOT MANIPULADOR.

Una vez hemos visto las posibilidades de la red propuesta en la resolución de un problema sencillo, parece oportuno examinar sus prestaciones en la identificación de un sistema no lineal. En nuestro caso hemos elegido un brazo robot distribuido por la

empresa ALECOP. Este es un manipulador antropomórfico con 5 grados de libertad. Junto con el robot se ha utilizado una tarjeta IBM Turbo Board, que incorpora conversores analógico/digitales y digitales/analógicos. Esta tarjeta ha sido instalada en un PC y mediante la misma se permite un control sobre los motores situados en las articulaciones del robot. Cada una de las articulaciones del robot viene provista de un control proporcional integral, por lo que no se realiza un control directo sobre los motores, los comandos se especifican a través de una zona de la memoria RAM del PC, de tal forma que en diferentes posiciones de memoria sea posible situar las consignas para cada una de las articulaciones, es decir, en cada posición de memoria se hallará el ángulo que se desea para cada articulación. Posteriormente la tarjeta IBM Turbo Board se encarga de proporcionar esas consignas a los diferentes controladores PID. En la figura 3.11 se muestra un esquema de todo el sistema.

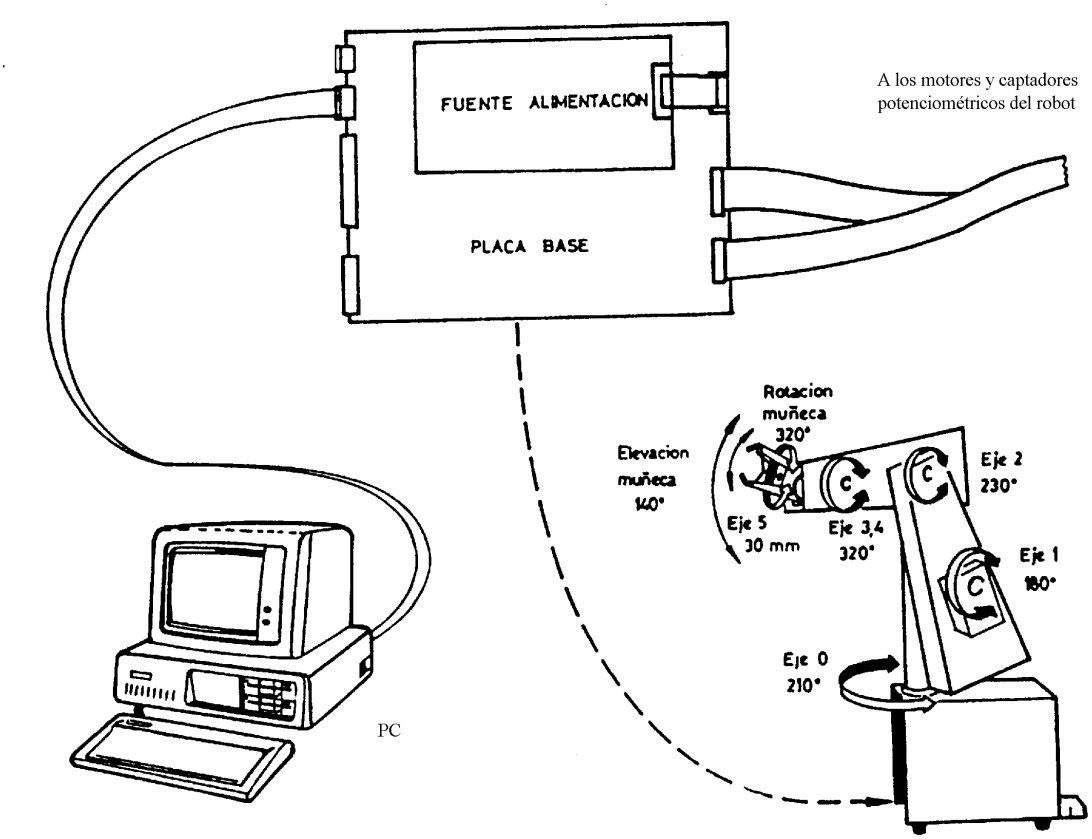


Figura 3.11 – Esquema del sistema de control.

Con objeto de identificar la dinámica se obtuvieron varias curvas de ángulos correspondientes a las diferentes articulaciones. Para ello se suministraron trayectorias en forma de curvas de ángulos a cada una de las articulaciones, es decir, nuestro objetivo fue que las articulaciones siguieran unas ciertas trayectorias en el dominio temporal. Con ese propósito cada 70 milisegundos se proporcionaron los nuevos ángulos a los diferentes controladores PID, recogiendo posteriormente los ángulos reales. Para tal propósito se desarrollaron diferentes subrutinas de temporización, de tal manera que por medio de las mismas se determinase cuando se debe proporcionar un nuevo comando y recoger la subsiguiente respuesta. Por otro lado las unidades de las consignas, así como las respuestas de las articulaciones han sido tomadas como números enteros entre 0 y 255, tal que el ángulo en radianes vendría dado como ese número multiplicado por el cociente entre el máximo ángulo para la articulación a la

que nos referimos (tal como se señalan en la figura 3.11) y 255. Teniendo presente este hecho, hemos introducido señales sinusoidales de diferentes frecuencias como consigna a seguir por el robot, escalando linealmente sus valores entre 0 y 255. En la Figura 3.12 se puede apreciar una de las curvas de ángulos de consigna en donde se ha utilizado una frecuencia de 3/5 rad/seg.

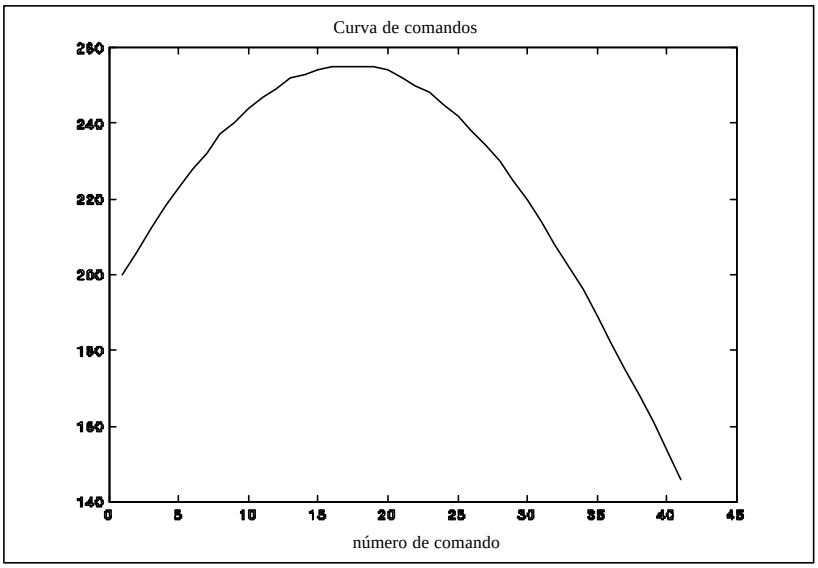


Figura 3.12 – Curva de comandos.

En la figura (3.13) mostramos la respuesta a esta curva de comandos para la primera articulación. Mientras que en las figuras (3.14) y (3.15) presentamos sendas respuestas pero en esta ocasión corresponden a curvas de comandos de una frecuencia de 2 y 4 rad/seg.

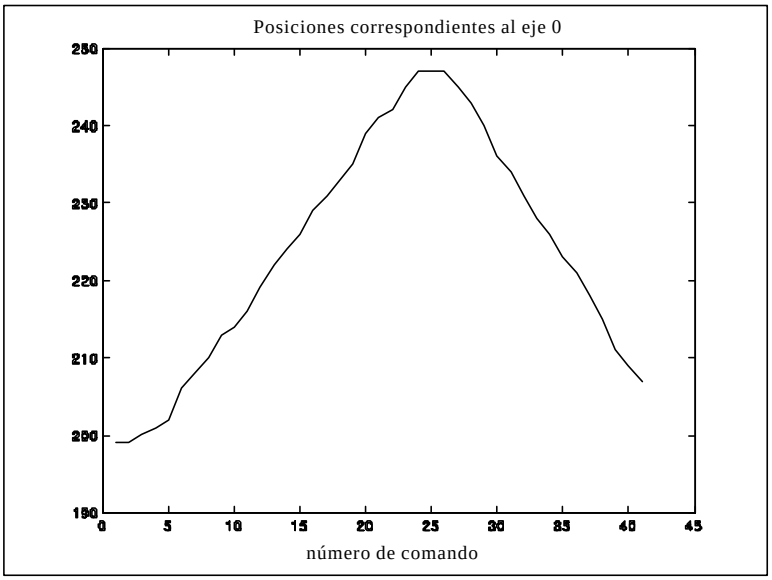


Figura 3.13 – Posiciones correspondiente al eje 0.

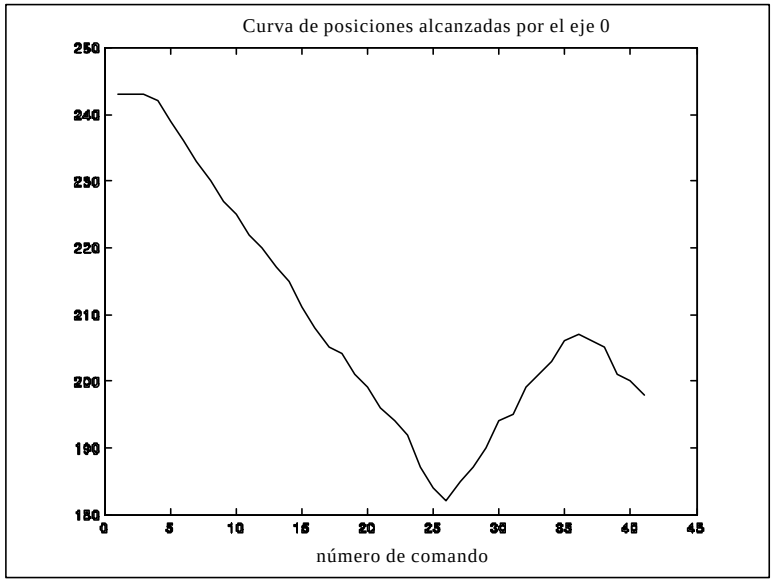


Figura 3.14 – Posiciones alcanzadas por la articulación 0.

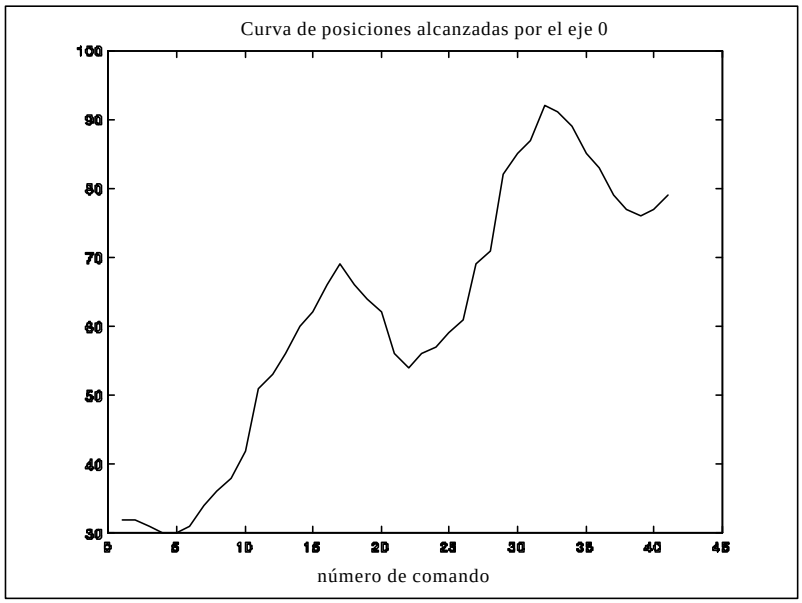


Figura 3.15 – Curva de posiciones alcanzadas por la articulación 0.

Estas curvas de comandos y respuestas han sido utilizadas como datos para el aprendizaje de la dinámica por parte de la red neuronal propuesta. Antes de aplicar estos datos a la red hemos de considerar previamente dos hechos que afectarán al aprendizaje. Por un lado hemos de apreciar que la derivada de la curva de comandos ha de ser calculada con objeto de aplicar el algoritmo de aprendizaje propuesto. Nosotros hemos calculado ésta por medio de un cociente de diferencias finitas a partir de las curvas de comandos. En la Figura (3.16) mostramos el resultado correspondiente a la curva de comandos de más baja frecuencia.

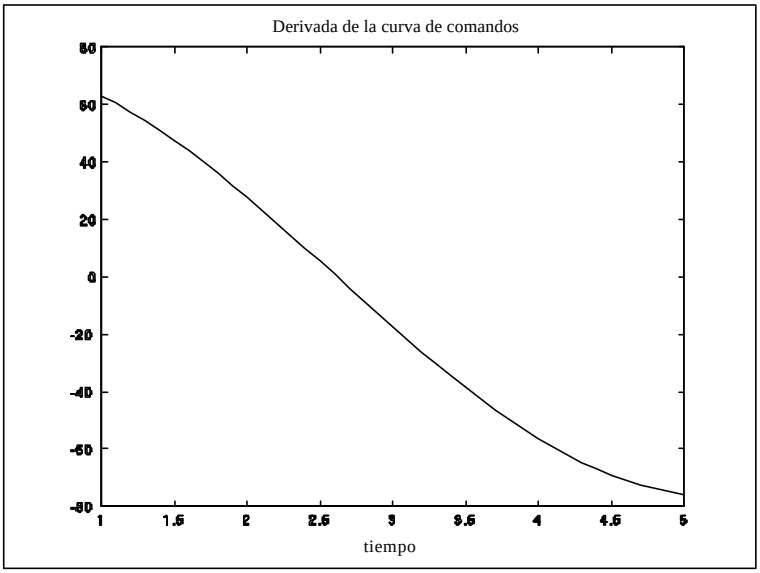


Figura 3.16 – Derivada de la curva de comandos.

Sin embargo, como se puede apreciar en la Figura (3.16) tal curva de derivadas en un momento dado pasa por un valor cero lo cual implica una dificultad añadida de cara a aplicar el algoritmo de aprendizaje. En dicho algoritmo la derivada de los comandos aparece en el denominador de algunas expresiones. Con objeto de subsanar este hecho hemos tomado en consideración las derivadas de los comandos si éstos se encuentran por encima de una cierta cota, tal que se asegure la convergencia del aprendizaje. Cuando esto ocurra, hemos optado por no actualizar los pesos, es decir éstos se mantienen constantes hasta que en una nueva iteración los nuevos valores de la derivada del comando se sitúen por encima de esa cota.

Aparte de este problema surge un segundo, debido a que las unidades elegidas tienen un rango numérico desde 0 a 255. Esto provocaría un fenómeno de saturación en la red dado que al ser las activaciones de las neuronas grandes las salidas de las mismas se situarían en el rango de saturación de la función de activación. Con objeto de evitar este problema, se han multiplicado las entradas a la red por un factor que disminuya los valores de entrada a un rango donde se evite la saturación, mientras las salidas también han sido multiplicadas por otro factor de cara a disminuir sus valores. Además la curva de derivadas de los comandos ha sido multiplicada por el mismo factor que las entradas. Obsérvese que todas estas multiplicaciones en absoluto plantean un inconveniente respecto a la identificación de la dinámica del sistema, dado que únicamente representan una multiplicación por un factor constante en las capas de entrada y salida de la red.

Teniendo en consideración las observaciones realizadas en los párrafos previos, se han realizado varios experimentos. En este caso la ecuación de estado para las neuronas ha sido elegida como:

$$X(k+1) = (1 - Hab)X(k) - Ha\,n(k) \qquad (3.32)$$

En la Figura (3.17) se presenta el resultado para el caso de la sinusoide de más baja frecuencia. Para esta hemos tomado como factores multiplicativos 1/100 para las entradas y derivadas de la entrada y 1/600 para las salidas, eligiendo en este caso una red de 3 capas y 3 neuronas cada una. En lo que respecta a los pesos y parámetros de las ecuaciones de estado (a y b) han sido inicializados de forma aleatoria. El resultado de la

Figura (3.17) se ha tomado después de haber transcurrido 200 épocas en el entrenamiento, alcanzando en esta época un error cuadrático de 0.007381.

Obsérvese que en la figura (3.17) se han indicado con el signo + los puntos que se alejan respecto a la curva real con relación a épocas anteriores y con el símbolo * los puntos que suponen un acercamiento. Esta simbología será adoptada en los sucesivos resultados que se presentan en esta sección, con objeto de determinar la evolución de las salidas de la red respecto a las mismas en épocas previas.

Con objeto de ver si existe dependencia del número de capas en la resolución del problema se ha realizado el mismo entrenamiento con una red de dos capas dando el resultado en la época 200 que se muestra en la figura (3.18). Como se puede apreciar la reducción en el número de capas trae consigo un peor resultado siendo incapaz la red neuronal de aprender el comportamiento decreciente.

En la figura (3.19) presentamos el resultado correspondiente a la curva de comandos sinusoidal de frecuencia 2 rad/seg. Este resultado ha sido obtenido utilizando una red de 3 capas y 3 neuronas. Además hemos tomado un factor multiplicativo de 1/600 tanto para las entradas y sus derivadas, como para las salidas. En la figura (3.19) se presenta la época 5657 para la cual se ha obtenido un error de 0.007112. Debemos destacar que la salida de la red intenta seguir la curva objeto de aprendizaje, sin embargo la última parte no llega a ser aprendida. Tal fenómeno puede ser explicado por el hecho de que la red actúa primero disminuyendo los errores mayores.

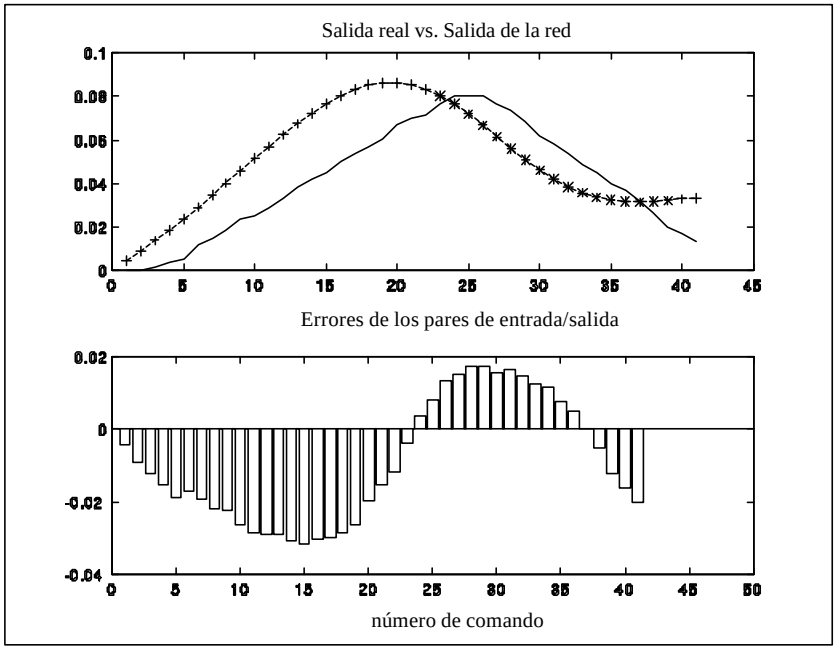


Figura 3.17 – Salidas reales vs. salidas de la red.

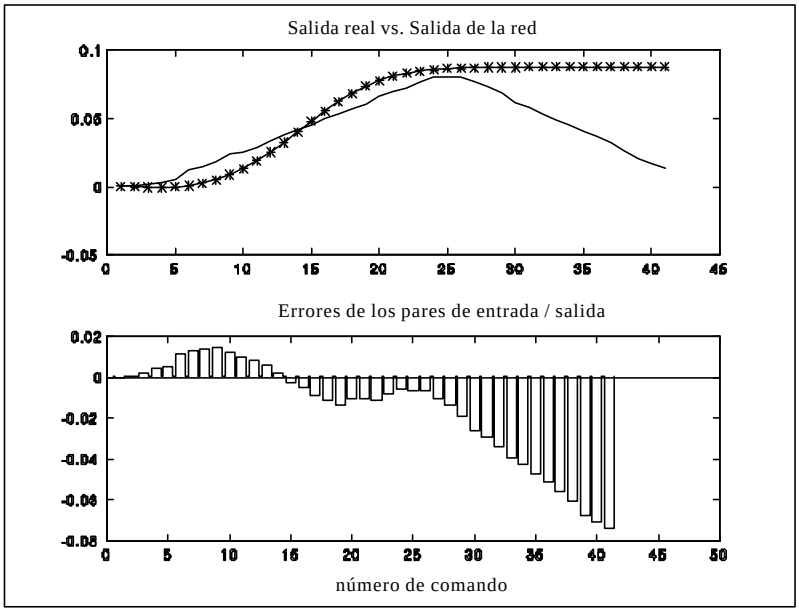


Figura 3.18 – Salidas reales vs. salidas de la red.

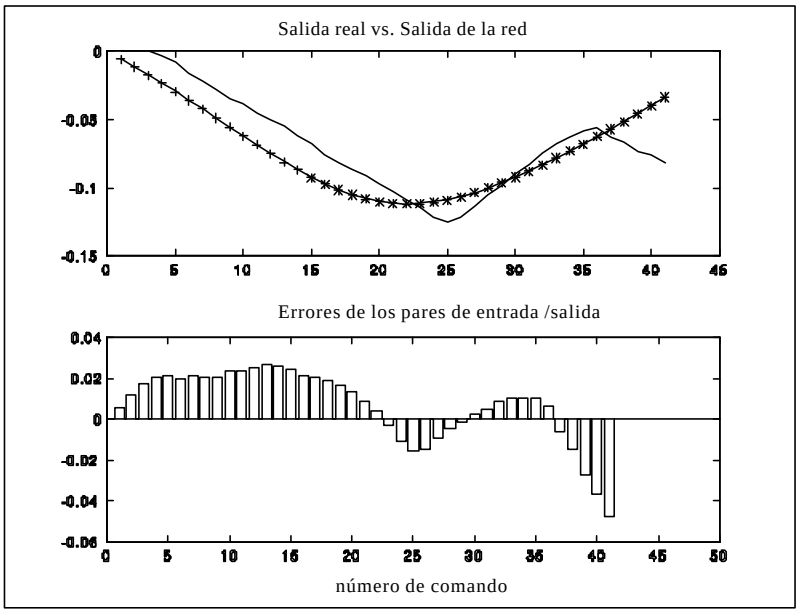


Figura 3.19 – Salidas real vs. salida de la red.

Para la respuesta de la sinusoide de mayor frecuencia se han elegido factores multiplicativos de 1/300 para las entradas y derivadas y 1/600 para las salidas. En este caso como en el anterior se ha tomado una red de 3 capas con 3 neuronas por capa. En la figura (3.20) presentamos los resultados para la época 5000 donde se ha alcanzado un error de 0.02509.

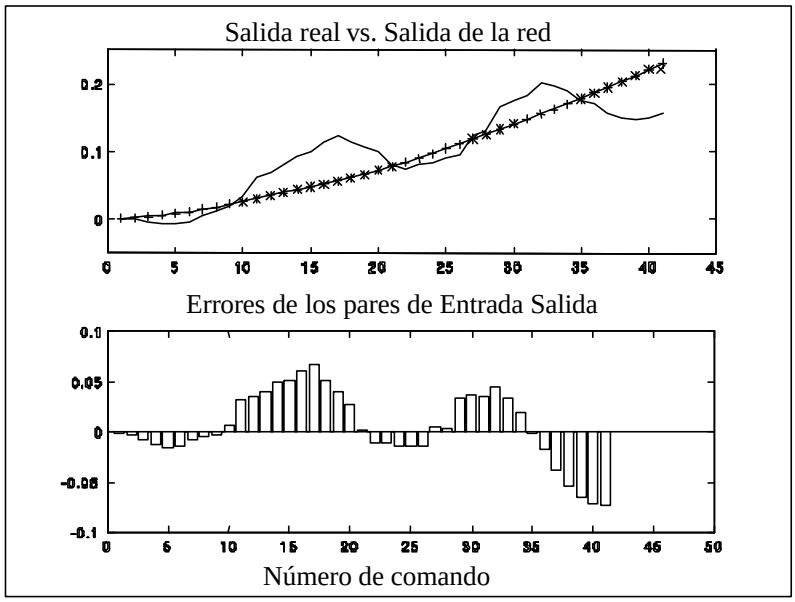


Figura 3.20 – Salida real vs. salida de la red.

Los resultados de estos primeros experimentos muestran como a medida que las curvas son más complejas, es decir, su comportamiento con el tiempo es más variable el aprendizaje se hace más difícil. Esto se refleja en el hecho de que a medida que se aumenta la frecuencia hemos de tomar un mayor número de épocas para conseguir un resultado óptimo. También es importante destacar que en los resultados mostrados la red neuronal únicamente es capaz de seguir el comportamiento medio sin llegar a apreciar los pequeños cambios en el comportamiento.

Por otra parte, aunque finalmente hemos obtenido los resultados mostrados anteriormente, en general el proceso de aprendizaje tiene gran tendencia a inestabilizarse de tal manera que es necesario un número considerablemente alto de experimentos de cara a obtener un buen comportamiento de la red.

Estos resultados nos llevan a plantearnos nuevas estrategias de aprendizaje con el objeto de mejorar los resultados obtenidos. En las siguientes secciones describiremos todas estas modificaciones, así como el resultado de las mismas.

3.8 VARIACIONES EN EL ALGORITMO DE APRENDIZAJE.

Con objeto de mejorar los resultados hemos introducido en el algoritmo de aprendizaje varias modificaciones encaminadas a reforzar el aprendizaje.

La primera de las estrategias introducidas ha sido el incorporar en el algoritmo un método para prevenir que grandes errores influyan en la actualización de los pesos y parámetros de forma drástica. De esta manera evitamos el efecto desestabilizador que este hecho tiene sobre el proceso de aprendizaje. El método consiste en introducir un factor de escala de la siguiente forma:

1) Comenzamos cada época con el factor de escala igual a 1. Si denotamos el factor de escala por la variable "*escalado*" entonces tomamos *escalado*=1.

2) Para cada nuevo par de entrada/salida, modificar la escala como sigue:

2.1) *error* = salida deseada – salida de la red.
2.2) *r*(t) = *escalado*error*
2.3) si *r(t)* > 1 hacemos *r(t)=1* y *escalado= escalado*/*error*.
2.4) si *r(t)* < -1 hacemos *r(t)=-1* y *escalado* = - *escalado*/*error*.

Donde *r(t)* se toma como el error en la última capa de la red neuronal, de tal manera que el gradiente se calculará en este caso utilizando *r(t)* en vez de la diferencia entre la salida deseada y la salida de la red. Aplicando este método podremos conseguir que los errores se hallen en un rango pequeño dado que *r(t)* se hallará aproximadamente entre –1 y 1.

Este método ha sido aplicado al caso de una curva de comandos de frecuencia 4 rad/seg, dando el resultado que se muestra en la figura (3.21). Sin embargo, aunque dicho método ha contribuido a estabilizar el proceso de aprendizaje, los resultados son similares a los mostrados en la sección 3.7 con únicamente una ligera mejora. Los resultados mostrados en la figura (3.21) corresponden a la época 5000 donde se ha obtenido un error cuadrático de 0.01925, utilizando una red de 3 capas y 3 neuronas por capa.

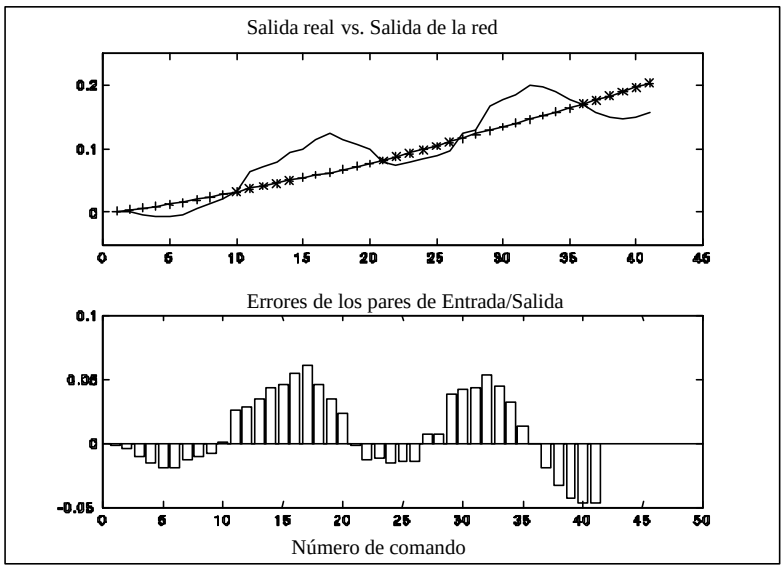


Figura 3.21 – Salida real vs. salida de la red.

Como segunda modificación se realizaron pruebas en donde las velocidades de aprendizaje para los parámetros a y b fueron escogidas según la siguiente proporción según las capas: $1, s^2, s^3, s^4$ .. etc. Desde la capa de salida a la de entrada, con *s* un valor comprendido entre 1 y 2. Esta forma de escoger las velocidades es debida a que como el error se propaga desde las capas de salida a las de entrada, es deseable que pueda llegar a las primeras capas sin que lo impida una saturación en las últimas. Por ello las velocidades se toman en orden creciente desde la capa de salida a la de entrada. Sin embargo, se obtuvieron resultados similares a los ya mostrados.

Siguiendo esta línea de razonamiento aplicamos el algoritmo de adaptación de las velocidades de aprendizaje que se muestra a continuación, aunque en este caso

únicamente sobre las velocidades de los pesos. Por otra parte con el objeto de mejorar la habilidad de la red para moverse por la superficie de error añadimos un valor de momentum [HUS93] a la actualización de los pesos y parámetros, siendo éste también actualizado en el proceso de aprendizaje. En las líneas siguientes resumimos el algoritmo de actualización de velocidades y momentums:

Si *error* (t) < *error* (t-1) Entonces:
Velocidad (t+1)=*velocidad* (t) * 1.15;
Momentum (t+1)= *Momentum* (0);
En caso contrario:
Velocidad (t+1)=*velocidad* (t) * 0.75;
Momentum (t+1)= 0;

Obsérvese que t se refiere al instante actual de tal forma que según el error, considerado éste como la diferencia entre la salida de la red y la salida real, sea mayor o menor que el del instante previo multiplicaremos la velocidad por 1.15 ó 0.75. Mientras el momentum permanecerá igual al momentum inicial dado por el usuario (*Momentum*(0)) si el error actual es inferior al anterior o, se igualará a cero si el error actual es mayor o igual que el del instante anterior. También con el objeto de reforzar el aprendizaje le hemos presentado a la red los mismos pares de entradas y salidas objeto de aprendizaje varias veces. Es decir, se han realizado varias actualizaciones de pesos y parámetros consecutivas con el mismo patrón de aprendizaje.

Utilizando esta estrategia se han realizado varios experimentos comprobando en general que el proceso de aprendizaje se hace más robusto, produciéndose con menos frecuencia la inestabilización. En lo que respecta a la bondad de los resultados estos no han mejorado considerablemente. En la Figura (3.22) mostramos el resultado de aplicar un momentum inicial de 0.9 y repetir los pares entrada/salida 25 veces. El resultado mostrado corresponde a la época 5000 siendo el error cuadrático igual a 0.01564.

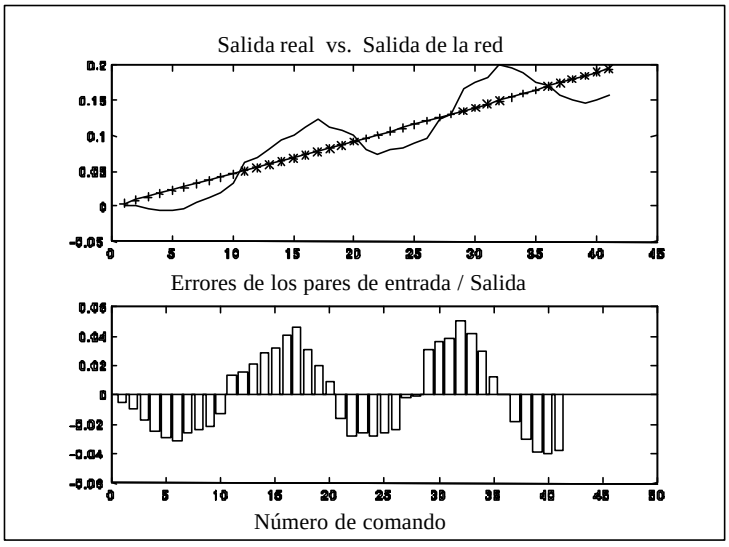


Figura 3.22 – Salida real vs. salida de la red.

En un intento por mejorar los resultados, finalmente hemos cambiado la ecuación de estado de las neuronas en dos formas diferentes. Obsérvese que esto implica modificar

la dinámica de la red neuronal, de tal manera que nuestra idea inicial era el modificar la ecuación de estado con el objeto de conseguir un comportamiento de la red neuronal con mayor variabilidad en la salida. Por un lado, sustituimos la ecuación de estado dada por la expresión (3.1b) por una ecuación de estado de segundo orden, es decir, tal que la evolución del estado de cada una de las neuronas viniese dada por la solución de una ecuación diferencial de segundo orden. Es importante destacar que este aumento en el orden de la ecuación de estado lleva aparejado un aumento en el número de parámetros a actualizar, puesto que los parámetros que multiplican a los términos dependientes temporalmente de dos instantes previos deben también de ser modificados en el proceso de aprendizaje. En los experimentos realizados, los resultados fueron similares a los ya obtenidos, siendo en algunos casos peores. Estos resultados nos llevaron a no considerar el aumentar el orden de las ecuaciones de estado, dado que el aumento en la complejidad de la red en absoluto compensa los resultados obtenidos.

La segunda opción probada fue el considerar una ecuación de estado con un comportamiento no lineal. De hecho elegimos la siguiente dinámica:

$$X(k+1) = X(k) - Hab\,X(k)^2 - H\,a\,n(k) \qquad (3.33)$$

En la figura (3.23) presentamos el resultado después de 500000 épocas en donde el error cuadrático alcanzado es de 0.01418. En la figura se puede apreciar un ligero cambio de comportamiento, sin embargo no parece realmente significativo.

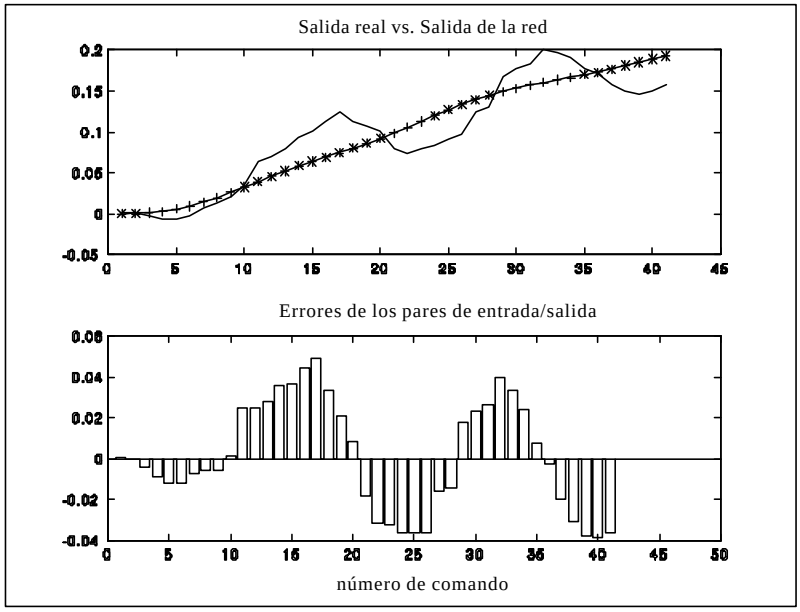


Figura 3.23 – Salida real vs. salida de la red.

Con el propósito de mejorar el resultado llevamos el entrenamiento hasta 1500000 épocas consiguiendo un error cuadrático de 0.01276. Este resultado nos muestra que el aumento en el número de épocas no viene acompañado de una disminución proporcional en el error cuadrático, con lo cual se puede concluir que en ningún caso los resultados obtenidos son debidos a tomar un número insuficiente de épocas en el entrenamiento.

En general podemos argumentar que todas las modificaciones sobre el algoritmo de aprendizaje descritas en esta sección no han mostrado mejoras significativas respecto al

aprendizaje. Se deduce de los resultados obtenidos que la red es únicamente capaz de aprender el comportamiento medio, siendo bastante costoso el aprendizaje de pequeños cambios frente a este comportamiento medio. De hecho hemos realizado experimentos en donde se han tomado varias redes neuronales dinámicas como la propuesta en este capítulo, en cascada, de tal forma que la primera aprenda a seguir la salida real, es decir, las curvas de ángulos del robot, mientras que la siguiente aprenda la diferencia entre la salida real y la salida de la red anterior , y así sucesivamente. Los resultados obtenidos mostraron la incapacidad de las redes para seguir las irregularidades de la curva a aprender, sobre todo en las redes colocadas en cascada a partir de la primera . En la figura (3.24) presentamos el resultado de la segunda red, en donde se puede apreciar que la red únicamente tiende a seguir el comportamiento medio.

Los resultados obtenidos nos obligan a replantearnos el algoritmo de aprendizaje, dado que a pesar del gran número de modificaciones introducidas los resultados muestran un comportamiento similar: el aprendizaje del comportamiento medio. Como primer paso de cara a ver las dificultades del algoritmo de aprendizaje propuesto hemos de analizar cuales son las suposiciones sobre las cuales se sustenta. Es importante resaltar que de cara a determinar los valores δ_i^k fue necesario, en la expresión (3.9) hacer una suposición sobre el estado. Repetimos aquí dicha ecuación para facilitar la comprensión de la naturaleza de tal suposición:

$$\frac{\partial x_i^k}{\partial n_i^k}\frac{\partial n_i^k}{\partial t}+\frac{\partial x_i^k}{\partial t}=\dot{x}_i^k=-a_i^k\left(x_i^k\right)\left[b_i^k\left(x_i^k\right)+n_i^k\right] \qquad (3.34)$$

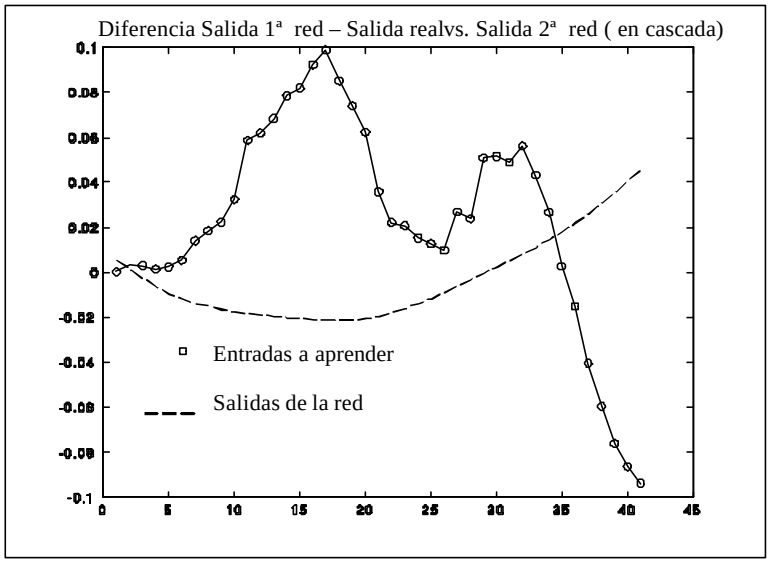


Figura 3.24 – salida real vs. salida de la red.

Se supuso que la variación del estado con respecto al tiempo era dominada por la dependencia respecto a la sinapsis, considerando que la variación explícita del estado con respecto al tiempo no era significativa. Esta suposición nos llevó a considerar $\frac{\partial x_i^k}{\partial t}=0$ y de esta manera calcular los valores δ_i^k para cada una de las neuronas. Sin

embargo, tal suposición implica restringir la variabilidad del estado con respecto al tiempo, y esto en definitiva se traduce en restringir el comportamiento dinámico de la red. También hemos de considerar que esta suposición nos llevó a obtener la expresión (3.17) para el calculo de los δ_i^1 de las neuronas de la primera capa, en donde nos aparece en el denominador la derivada de las entradas a la red. Esto representa un inconveniente añadido dado que cuando trabajamos con señales que tomen el valor cero o valores próximos al mismo en algún instante de tiempo el algoritmo se inestabiliza. Para evitar este efecto en los experimentos realizados hemos definido una cota por debajo de la cual no se realizaban actualizaciones de pesos y parámetros. Sin embargo, esto vuelve a incidir en una pérdida de dinamismo de la red, dado que existen intervalos de tiempo en el proceso de aprendizaje en donde forzosamente hemos de frenar el aprendizaje.

Todas estas consideraciones nos llevan a reformular el algoritmo de aprendizaje propuesto en la sección 3.3 intentando evitar suposiciones respecto a la evolución temporal del estado. En el próximo capítulo realizaremos este análisis.

Capítulo 4
PROPONIENDO UN NUEVO ALGORITMO DE APRENDIZAJE

4
PROPONIENDO UN NUEVO ALGORITMO DE APRENDIZAJE

En este capítulo volveremos a analizar los pasos seguidos en el diseño del algoritmo de aprendizaje para la red propuesta en el capítulo anterior, introduciendo modificaciones que resultarán en la obtención de un algoritmo de aprendizaje más adecuado desde el punto de vista matemático, eliminando algunos de los problemas que nos presentaba el algoritmo desarrollado inicialmente. Consideraremos el problema de la estabilidad de las ecuaciones de estado de las neuronas, introduciendo procedimientos para asegurar en todo momento su estabilidad. Y finalmente, aplicaremos la red neuronal en la resolución de un problema de identificación de un sistema no lineal simple, para posteriormente aplicarla al caso de la identificación de la dinámica de un brazo robot.

4.1 REPLANTEAMIENTO DEL ALGORITMO DE APRENDIZAJE PARA LA ARQUITECTURA DE RED PROPUESTA.

4.1.1 MECANISMO DE AJUSTE DE LOS PESOS.

Tal como se ha señalado, antes de aplicar nuevas técnicas sería conveniente replantearnos los pasos seguidos en el diseño del algoritmo de aprendizaje, en la línea indicada al final del capítulo anterior. Con objeto de simplificar la comprensión de esta nueva versión del algoritmo de aprendizaje repetiremos los pasos seguidos en el capítulo anterior.

El objetivo del algoritmo de aprendizaje es minimizar la función de error siguiente, a través de la modificación de los diferentes parámetros de la red, en nuestro caso, los pesos y parámetros determinantes de la dinámica interna de la misma:

$$E = \frac{1}{2}\sum_{i=1}^{M}\left(d_i - y_i\right)^2 \qquad (4.1)$$

Donde:

M = Número de salidas.

d_i = Salida deseada i-ésima.

y_i = Salida de la red i-ésima.

Con tal objetivo hemos aplicado el algoritmo del gradiente descendente sobre la función de error, obteniendo las siguientes expresiones de actualización para los pesos:

$$w_{ij}^k(m+1) = w_{ij}^k(m) - \eta \frac{\partial E}{\partial w_{ij}^k} \tag{4.2}$$

$$w_{ij}^k(m+1) = w_{ij}^k(m) + \begin{cases} -\eta \delta_i^k \, y_j^{k-1} \; \forall \; j \neq 0 \\ -\eta \delta_i^k \qquad j = 0 \end{cases} \tag{4.3}$$

con $\delta_i^k = \frac{\partial E}{\partial n_i^k}$

Obsérvese que η es la velocidad de aprendizaje y m es el índice temporal sobre la actualización de los pesos. A la vista de la ecuación (4.3), podemos concluir que calculando el factor δ_i^k para cada una de las neuronas podríamos calcular los nuevos pesos. Es importante señalar antes de proseguir que hasta el momento las ecuaciones obtenidas no se hayan ligadas a ninguna suposición expresada de forma explícita. Por lo que el algoritmo permanece exactamente igual al presentado en el capítulo anterior. En este punto parece apropiado dividir el problema por capas, con objeto de simplificar el análisis:

Capa de Salida: Aplicando la regla de la cadena a la expresión (4.1) y tomando k=L (capa de salida), obtenemos:

$$\delta_i^L = \frac{\partial E}{\partial y_i^L} \frac{\partial y_i^L}{\partial x_i^L} \frac{\partial x_i^L}{\partial n_i^L} = -(d_i - y_i) g'(x_i^L) \frac{\partial x_i^L}{\partial n_i^L} \tag{4.4}$$

Todos los términos en la expresión (4.4) son conocidos menos la derivada del estado con respecto a la activación $\frac{\partial x_i^L}{\partial n_i^L}$. El cálculo de este término esta condicionado a la forma de las ecuaciones de estado de las neuronas, pues son éstas quienes determinan la relación del estado x_i^k y la activación n_i^k. Debido a ello de cara al desarrollo de las ecuaciones asociadas al algoritmo hemos modificado la forma de la ecuación de estado tomada en el capítulo anterior con el objeto de simplificar esta relación y facilitar el cálculo de las derivadas. La ecuación de estado elegida se presenta discretizada de la siguiente forma, en donde se han evitado los productos de las funciones $A_i^k(x_i^L, a_i^L)$ y $B_i^k(x_i^L, b_i^L)$:

$$x_i^L(r+1) = x_i^L(r) + H B_i^L(x_i^L, b_i^L) + H \; A_i^L(x_i^L, b_i^L) \, n_i^L \tag{4.5}$$

Donde H es el tamaño del paso temporal, r es el índice sobre el tiempo y $A_i^k(x_i^L, a_i^L)$ y $B_i^k(x_i^L, b_i^L)$ son dos funciones arbitrarias dependientes del estado y sendos parámetros a_i^L y b_i^L respectivamente. Aprovechando la ecuación (4.5) podríamos calcular la derivada del estado con respecto a la activación $\frac{\partial x_i^L}{\partial n_i^L}$ dado que una buena aproximación a su valor sería la solución de la ecuación lineal:

$$z(r+1) = z(r)\left[1+H\frac{\P B_i^L\left(x_i^L,\, b_i^L\right)}{\P x_i^L}+H\frac{\P A_i^L\left(x_i^L,\, a_i^L\right)}{\P x_i^L}n_i^L(r)\right] \quad + H \quad A_i^L\left(x_i^L,\, a_i^L\right) \qquad (4.6)$$

Donde $z(r)=\dfrac{\P x_i^L(r)}{\P n_i^L}$ y $z(0)=0$. Por tanto resolviendo esta ecuación de forma iterativa podremos obtener la derivada del estado respecto a la activación y a partir de la misma terminar obteniendo $\boldsymbol{d}_i^L$. Debemos destacar que esta forma de calcular esta derivada representa un cambio sustancial respecto a la forma que se utilizó en el capítulo anterior, dado que en ningún caso se impone ninguna suposición, tendiendo este hecho a beneficiar notablemente al algoritmo y eliminando restricciones en el proceso de aprendizaje.

Capas Ocultas: Para el caso de las neuronas de las capas ocultas hemos de tener presente que los δ_i^k de éstas se relacionan con los de las capas que se encuentran delante por la ecuación:

$$\boldsymbol{d}_i^k = \sum_{j=1}^{m_{k+1}}\left\{\frac{\P E}{\P n_j^{k+1}}\frac{\P n_j^{k+1}}{\P y_i^k}\right\}\frac{\P y_i^k}{\P x_i^k}\frac{\P x_i^k}{\P n_i^k} = \sum_{j=1}^{m_{k+1}}\left\{\boldsymbol{d}_j^{\,k+1} w_{ji}^{k+1}\right\} g'\left(x_i^k\right)\frac{\P x_i^k}{\P n_i^k} \qquad (4.7)$$

Donde el único término desconocido es la derivada del estado con respecto a la activación $\dfrac{\P x_i^k}{\P n_i^k}$. Si como en el caso de las neuronas de la capa de salida tomamos en consideración la ecuación de estado de las neuronas:

$$x_i^k(r+1) = x_i^k(r)+HB_i^k\left(x_i^k, b_i^k\right)+H \quad A_i^k\left(x_i^k, a_i^k\right) n_i^k \qquad (4.8)$$

Podríamos obtener el valor de esta derivada resolviendo la ecuación lineal:

$$z(r+1) = z(r)\left[1+H\frac{\P \quad B_i^k\left(x_i^k,\, b_i^k\right)}{\P x_i^k}+H\frac{\P \quad A_i^k\left(x_i^k,\, a_i^k\right)}{\P x_i^k}n_i^k(r)\right] \quad + H \quad A_i^k\left(x_i^k,\, a_i^k\right) \qquad (4.9)$$

Siendo $z(r)=\dfrac{\partial x_i^k(r)}{\partial n_i^k}$ y $z(0)=0$ en este caso. Con ello se consiguen los valores de δ_i^k para las neuronas de las capas ocultas.

Capa de Entrada: En cuanto a la capa de entrada el principal problema es la no existencia de ninguna capa previa. Sin embargo es posible suponer que la capa de entrada se haya precedida de una capa virtual, cuyas salidas son las entradas a la red. Considerando esta suposición los $\boldsymbol{d}_i^1$ se calculan a través de la ecuación (4.7) tomando *k=1*. Obsérvese que para calcular $\boldsymbol{d}_i^1$ no intervienen las salidas de las capas previas por lo que el cálculo es exactamente igual al realizado para las capas ocultas. Las salidas de

la capa previa sólo intervienen a través de la ecuación de actualización de los pesos (4.3), siendo en este caso las entradas a la red. Señalemos que aun cuando hemos introducido la suposición de una capa virtual detrás de la capa de entrada esto en absoluto representa una limitación de cara al algoritmo, dado que ésta únicamente se ha introducido con el objeto de facilitar el entendimiento aprovechando las ecuaciones desarrolladas para las otras capas.

Podemos concluir que con el planteamiento elegido hemos evitado todas aquellas suposiciones a las que nos vimos obligados en un principio. Sin embargo hasta ahora únicamente hemos analizado el algoritmo de actualización de los pesos, en la próxima sección nos centraremos en el efecto que los cambios expuestos tienen en el ajuste de los parámetros de las ecuaciones de estados.

Antes de acabar esta sección parece oportuno destacar la relación de este algoritmo con el algoritmo backpropagation para redes multi-capa estáticas. Obsérvese que la arquitectura de red presentada podría verse como una extensión directa de una red multi-capa éstatica en donde únicamente hemos introducido un sistema dinámico entre la activación y la función de activación de cada neurona. Por tanto, dado que el algoritmo expuesto se basa en las mismas ideas que el backpropagation, modificado en este caso por el carácter dinámico de la red, parecería lógico que si se eliminase ese comportamiento dinámico el algoritmo de aprendizaje se convirtiese en el backpropagation para una red estática. Este es el caso cuando por alguna elección particular de los parámetros a_i^k y b_i^k la ecuación (4.8) se convierte en:

$$x_i^k(r+1) = n_i^k(r) \qquad (4.10)$$

Entonces tenemos que $\frac{\partial x_i^k(r)}{\partial n_i^k} = 1$ con lo que se confirma que todo el algoritmo de aprendizaje se convierte en el backpropagation para una red multi-capa estática, sin más que eliminar el comportamiento dinámico en las ecuaciones de estado (4.8). Esta característica es deseable desde el punto de vista de la implementación del algoritmo dado que nos permite detectar fallos en la implementación al eliminar a propósito la dinámica con una elección adecuada de los parámetros dinámicos a_i^k y b_i^k, de tal forma que los valores de las diferentes variables puedan ser comparadas con las del algoritmo backpropagation standard.

4.1.2 AJUSTE DE LOS PARÁMETROS DE LAS ECUACIONES DE ESTADO.

Antes de iniciar esta discusión es necesario elegir expresiones concretas para las funciones que aparecen en las ecuaciones de estado $A_i^k\left(x_i^k, a_i^k\right)$ y $B_i^k\left(x_i^k, b_i^k\right)$. En nuestro caso las hemos tomado de esta forma:

$$A_i^k(x_i^k, a_i^k) = a_i^k \;,\; B_i^k(x_i^k, b_i^k) = -b_i^k\, x_i^k \qquad (4.11)$$

Teniendo estas expresiones ahora es necesario aplicar el algoritmo del gradiente descendente sobre cada uno de los parámetros. Por tanto, las ecuaciones de actualización de los mismos vendrán dadas como:

$$a_i^k(r+1) = a_i^k(r) - \eta_a \frac{\partial E}{\partial a_i^k} \tag{4.12}$$

$$b_i^k(r+1) = b_i^k(r) - \eta_b \frac{\partial E}{\partial b_i^k}\ ; \tag{4.13}$$

Donde η_a y η_b son las velocidades de aprendizaje. El principal problema es determinar las derivadas parciales de la función de error respecto a los parámetros a_i^k y b_i^k obteniendo de esta forma todos los términos presentes en las ecuaciones de actualización de los parámetros (4.12) y (4.13). Vamos por tanto a centrarnos en el cálculo de $\frac{\partial E}{\partial a_i^k}$. Para ello observemos que δ_i^k se puede expresar como:

$$\delta_i^k = \frac{\partial E}{\partial n_i^k} = \frac{\partial E}{\partial a_i^k}\frac{\partial a_i^k}{\partial n_i^k} = \frac{\partial E}{\partial a_i^k}\frac{\partial x_i^k / \partial n_i^k}{\partial x_i^k / \partial a_i^k} \tag{4.14}$$

Y despejando $\frac{\partial E}{\partial a_i^k}$ obtenemos:

$$\frac{\partial E}{\partial a_i^k} = \delta_i^k \frac{\partial x_i^k / \partial a_i^k}{\partial x_i^k / \partial n_i^k} \tag{4.15}$$

De igual forma obtenemos la misma ecuación para b_i^k:

$$\frac{\partial E}{\partial b_i^k} = \delta_i^k \frac{\partial x_i^k / \partial b_i^k}{\partial x_i^k / \partial n_i^k} \tag{4.16}$$

Hasta el momento no hemos introducido ninguna modificación respecto al algoritmo de ajuste sobre los parámetros presentado en el capítulo anterior, permaneciendo válido dado que no se introduce ninguna suposición que limite el aprendizaje. Sin embargo, en el capítulo anterior advertíamos que la aparición de la derivada del estado con respecto a la activación $\partial x_i^k / \partial n_i^k$ en el denominador de las expresiones (4.15) y (4.16) podría traer problemas ya que valores pequeños de la misma nos llevarán a la inestabilización del proceso de aprendizaje. De hecho se demuestra que la aparición de tal derivada en el denominador no es necesaria para la actualización de los parámetros. Para ello definamos nuevos δ_i^k como:

$$\delta_i^k = \delta_{dyn_i}^k \, \partial x_i^k / \partial n_i^k \tag{4.17}$$

Donde $\boldsymbol{d}_{dyn_i}^{k}$ hace referencia a los nuevos δ_i^k. Introduciendo estos nuevos δ_i^k en las expresiones (4.15) y (4.16) eliminamos la derivada parcial del estado respecto de la activación $\partial x_i^k / \partial n_i^k$ del denominador:

$$\frac{\partial E}{\partial a_i^k} = \boldsymbol{d}_{dyn_i}^{k} \, \partial x_i^k / \partial a_i^k \tag{4.18}$$

$$\frac{\partial E}{\partial b_i^k} = \boldsymbol{d}_{dyn_i}^{k} \, \partial x_i^k / \partial b_i^k \tag{4.19}$$

Destaquemos que los $\boldsymbol{d}_{dyn_i}^{k}$ aunque han sido definidos a partir de los δ_i^k no necesariamente han de calcularse a partir de éstos. De hecho teniendo en cuenta que δ_i^k puede ser expresarse como:

$$\boldsymbol{d}_i^k = \frac{\partial E}{\partial y_i^k}\frac{\partial y_i^k}{\partial x_i^k}\frac{\partial x_i^k}{\partial n_i^k} \tag{4.20}$$

Los $\boldsymbol{d}_{dyn_i}^{k}$ se calculan a través de la siguiente ecuación:

$$\boldsymbol{d}_{dyn_i}^{k} = \frac{\partial E}{\partial y_i^k}\frac{\partial y_i^k}{\partial x_i^k} \tag{4.21}$$

En el caso de tratarse de la capa de salida tomando $k=L$, siendo L el número de capas, estas derivadas son:

$$\boldsymbol{d}_{dyn_i}^{L} = -(d_i - y_i) g'(x_i^L) \tag{4.22}$$

Para el caso de las capas ocultas teniendo en cuenta la ecuación (4.7) y sustituyendo $\boldsymbol{d}_i^k$ por $\boldsymbol{d}_{dyn_i}^{k}$ mediante la ecuación (4.17) queda:

$$\boldsymbol{d}_{dyn_i}^{k} \frac{\partial x_i^k}{\partial n_i^k} = \sum_{j=1}^{m_{k+1}} \left\{ \boldsymbol{d}_{dyn_j}^{k+1} \frac{\partial x_j^{k+1}}{\partial n_j^{k+1}} w_{ji}^{k+1} \right\} g'(x_i^k) \frac{\partial x_i^k}{\partial n_i^k} \tag{4.23}$$

Obteniendo finalmente tras simplificar en ambos miembros $\frac{\partial x_i^k}{\partial n_i^k}$:

$$\boldsymbol{d}_{dyn_i}^{k} = \sum_{j=1}^{m_{k+1}} \left\{ \boldsymbol{d}_{dyn_j}^{k+1} \frac{\partial x_j^{k+1}}{\partial n_j^{k+1}} w_{ji}^{k+1} \right\} g'(x_i^k) \tag{4.24}$$

Donde $\frac{\partial x_j^{k+1}}{\partial n_j^{k+1}}$ se obtiene a partir de la ecuación (4.9), considerando la capa k+1 y la neurona j.

Finalmente los $\delta_{dyn_i^1}^1$ de la capa de entrada se calculan aprovechando la ecuación (4.24) como:

$$\delta_{dyn_i^1}^1 = \sum_{j=1}^{m_2} \left\{ \delta_{dyn_j^2}^2 \frac{\partial x_j^2}{\partial n_j^{k+1}} w_{ji}^2 \right\} g'(x_i^1) \qquad (4.25)$$

Desde el punto de vista de la implementación de este algoritmo cabrían dos posibilidades. Por un lado implementar las ecuaciones para el cálculo de los δ_i^k y los $\delta_{dyn_i^k}^k$ por separado, tal y como se ha indicado. Sin embargo, otra posibilidad sería el seguir utilizando los δ_i^k como se hace en las expresiones (4.15) y (4.16), teniendo en cuenta que la derivada del estado con respecto a la activación $\partial x_i^k / \partial n_i^k$ que aparece en el denominador también aparece como producto implícito en el numerador, es decir, δ_i^k es el producto de $\delta_{dyn_i^k}^k$ por $\partial x_i^k / \partial n_i^k$, con lo que sí $\partial x_i^k / \partial n_i^k$ tomase valores pequeños esto no nos llevaría a obtener valores muy grandes de cara a la actualización de los parámetros dado que en el numerador aparece $\partial x_i^k / \partial n_i^k$ como factor. Esta segunda opción tiene la ventaja de que utilizamos una única definición de δ_i^k para la actualización de los pesos y los parámetros de las ecuaciones de estados, con lo que esto representa desde el punto de vista de la simplicidad del algoritmo, reutilizando los δ_i^k en la actualización de los parámetros a_i^k y b_i^k. En nuestro caso hemos optado por esta segunda opción.

4.1.3 ESTABILIDAD DE LAS ECUACIONES DE ESTADO.

En la sección 4.2 hemos presentado un mecanismo de actualización de los parámetros de las ecuaciones de estado basado en el método del gradiente descendente. Sin embargo, existen otros aspectos respecto a la actualización de los parámetros que debieran estudiarse, tal como la estabilidad de las ecuaciones de estado. Iniciaremos esta exposición presentando la forma general de estas ecuaciones de estado:

$$x_i^k(r+1) = x_i^k(r) + H B_i^k(x_i^k, b_i^k) + H \; A_i^k(x_i^k, a_i^k)\, n_i^k \qquad (4.26)$$

En particular tomaremos las funciones $A_i^k(x_i^k, a_i^k)$ y $B_i^k(x_i^k, b_i^k)$ como:

$$A_i^k(x_i^k, a_i^k) = a_i^k \;\; , \;\; B_i^k(x_i^k, b_i^k) = -b_i^k x_i^k \qquad (4.27)$$

Esta elección de las funciones $A_i^k(x_i^k, a_i^k)$ y $B_i^k(x_i^k, b_i^k)$ da la siguiente ecuación de estado:

$$x_i^k(r+1) = (1 - H b_i^k)\, x_i^k(r) + H a_i^k n_i^k \qquad (4.28)$$

Es importante señalar que con esta elección se consigue una ecuación de estado en donde las actualizaciones del parámetro a_i^k influyen únicamente en la dependencia sobre la activación, mientras las actualizaciones del parámetro b_i^k afectan únicamente a la dependencia con el estado en un paso de tiempo anterior. En definitiva se puede decir que b_i^k controla la dinámica de la ecuación de estado, mientras que a_i^k controla la ganancia que se aplica a las entradas. Esta forma de elegir la ecuación de estado parece más conveniente que la utilizada en el capítulo anterior dado que se separa la influencia sobre la dinámica y las entradas permitiendo un mayor control en la modificación de los parámetros.

La ecuación (4.26) representa un sistema lineal dinámico de primer orden. Por tanto, como tal sistema dinámico el comportamiento del mismo se puede hacer inestable para alguna elección particular de b_i^k. Esto nos lleva a no permitir cualquier valor del parámetro b_i^k, sino únicamente aquellos que hagan a la ecuación de estado estable. Considerando que esta ecuación de estado es la de un sistema lineal de primer orden, éste tendría un polo de valor $\left(1 - H\, b_i^k\right)$ y atendiendo a la teoría clásica de la estabilidad de los sistemas discretos, la ecuación de estado se convertirá en inestable cuando no se cumpla que $\left\| 1 - H\, b_i^k \right\| < 1$ donde $\| \ \|$ es el operador norma. Este hecho nos obliga a evaluar esta condición sobre los parámetros b_i^k de tal forma que si los nuevos b_i^k al realizar una actualización del parámetro no la cumplen entonces deberemos no actualizar los parámetros dinámicos, esperando a que en nuevas actualizaciones tal condición se vuelva a cumplir. Destaquemos también que en este caso hemos elegido ecuaciones de estado lineales y de primer orden.

4.2 APLICACIÓN DEL ALGORITMO.

4.2.1 EN LA IDENTIFICACIÓN DE UN SISTEMA DINÁMICO NO LINEAL.

Como primer paso en la comprobación del algoritmo hemos elegido un sistema dinámico de la forma:

$$y(r+1) = f(y(r), u(r)) \qquad (4.29)$$

Donde $u(r)$ es la entrada del sistema dinámico, e $y(r)$ es la salida y hemos elegido $f(y(r),u(r))$ como:

$$f(y(r), u(r)) = 1.1\,\mathrm{sen}(\cos(y(r))) + 1.5u(r) \qquad (4.30)$$

El problema se reduce a la identificación de un sistema dinámico no lineal. Con objeto de identificar este sistema mediante la red neuronal propuesta hemos tomado una red neuronal de dos entradas, 3 capas y 3 neuronas por capa. Hemos elegido para las entradas a la red las entradas al sistema dinámico *u(r)* y las salidas en el instante actual del sistema dinámico *y(r)*. En nuestro caso de cara al entrenamiento hemos tomado los valores de la entrada *u(r)* al azar con una distribución uniforme entre –1 y 1, mientras que las salidas vienen dadas por la ecuación (4.29). En la Figura (4.1) mostramos una comparación de la salida de la red con respecto a la salida del sistema dinámico. Como se puede observar la red neuronal se acerca considerablemente a la salida objeto de aprendizaje.

El resultado mostrado en la Figura (4.1) ha sido obtenido después de transcurridas 989 épocas y con una velocidad de aprendizaje para los pesos de 0.1. Un aspecto de particular interés es la inicialización de pesos y parámetros. Los primeros se han inicializado en el intervalo (-0.1, 0.1) evitando de esta forma la saturación de las neuronas. En el caso de los parámetros a_i^k han sido tomados como valores al azar en el intervalo (1,25), de tal forma que no supongan un factor muy grande respecto a las partes dinámicas (el primer término de la ecuación 4.28), y por tanto las hagan despreciables. En lo que respecta a la inicialización de los parámetros b_i^k estos han sido escogidos al azar en el intervalo (51,85). Tal elección ha sido motivada por el hecho de que al tomar el paso de discretización H igual a 0.01, este rango de valores garantiza el cumplimiento de la condición de estabilidad de las ecuaciones de estado $\left\|1 - H\, b_i^k\right\| < 1$. Obsérvese que los valores de los parámetros b_i^k en la aplicación del algoritmo son controlados a lo largo del proceso de aprendizaje de tal manera que se siga cumpliendo la condición $\left\|1 - H\, b_i^k\right\| < 1$, con lo que las ecuaciones de estado son estables en todo momento.

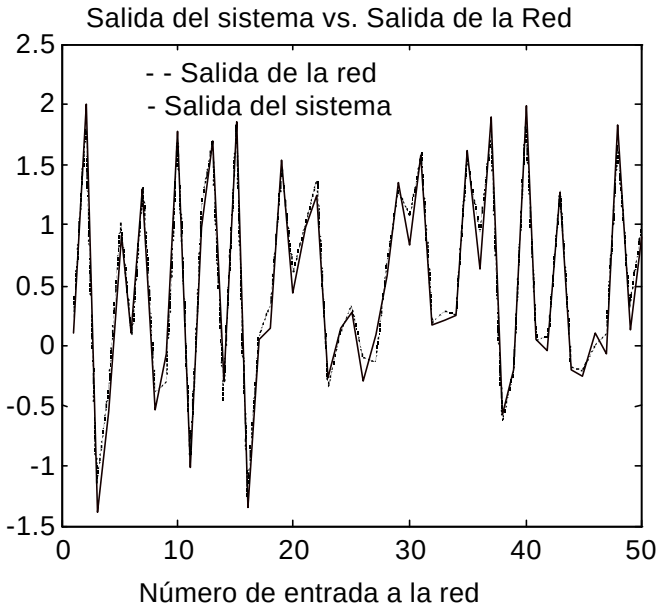


Figura 4.1 – Comparación de la salida de la red vs. Salida del sistema dinámico.

Con el propósito de comparar los resultados obtenidos con los de otras técnicas basadas en redes neuronales, hemos aplicado a este mismo problema una red neuronal Muti-capa estática. El primer problema con el que nos hemos encontrado es el que al elegir la misma velocidad de aprendizaje para los pesos la red neuronal ha mostrado un comportamiento inestable. En la Figura (4.2) se puede ver la curva de error para el caso de una velocidad de aprendizaje de 0.05.

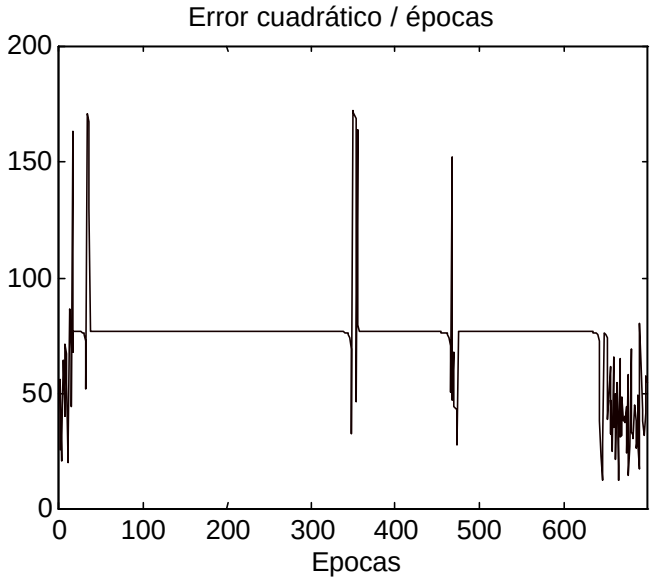


Figura 4.2 – Curva de error para una velocidad de aprendizaje de 0.05

Hemos tenido que reducir la velocidad de aprendizaje en un factor de 10, es decir, hemos elegido una velocidad de aprendizaje de 0.01. Con esta velocidad se ha obtenido la curva de error mostrada en la Figura (4.3), en donde se puede observar que comparativamente con la red neuronal propuesta se obtiene un comportamiento ligeramente mejor.

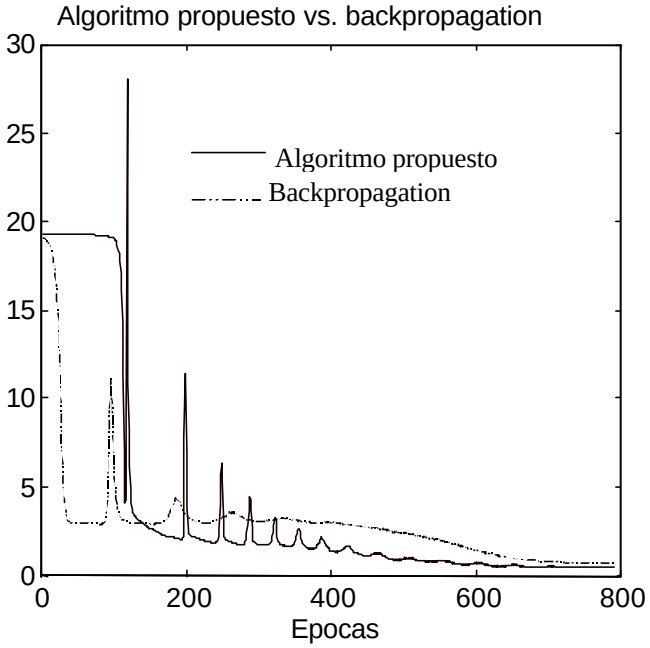


Figura 4.3 – Comparación de la red propuesta y la MLP.

Destaquemos con relación a los resultados obtenidos que en los experimentos realizados hemos introducido como entrada a la red la salida del sistema dinámico en el instante actual. Introduciendo esta información la red neuronal realmente aprende una relación estática, de ahí el que la red Multi-capa estática tenga todavía para este problema un buen comportamiento. En la próxima sección se volverá a aplicar este tipo de red pero en un caso en que realmente este presente una relación dinámica.

4.2.2 APLICACIÓN AL EJEMPLO TRATADO EN EL CAPÍTULO ANTERIOR.

Una vez que hemos visto los resultados obtenidos en el caso del sistema no lineal descrito por la ecuación (4.30), en donde se observa un buen comportamiento, en este apartado presentamos los resultados obtenidos de aplicar el algoritmo de aprendizaje tal y como se plantea en este capítulo al caso tratado en el capítulo anterior. En la figura (4.4) presentamos la salida de la red frente a la salida real después de transcurridas 106991 épocas. En esta época se consigue un valor del error cuadrático de 0.004558 notablemente inferior a los presentados cuando se utiliza el algoritmo de aprendizaje tal y como se mostró en el capítulo anterior. En el resultado mostrado en la figura (4.4) las velocidades de aprendizaje de los pesos se han tomado iguales a 0.001, mientras las velocidades de aprendizaje de los parámetros a y b de las ecuaciones de estado se han tomado iguales a 8.9741. Es importante destacar que en este caso no hemos empleado momentum en la ecuación de actualización de los pesos y las velocidades de aprendizaje se han mantenido constante a lo largo del proceso de aprendizaje. Como se puede observar la salida de la red es capaz de seguir las variaciones de la salida real, a diferencia de lo que ocurría en los resultados mostrados en el capítulo anterior, donde se ponía de manifiesto que la red únicamente seguía el comportamiento medio.

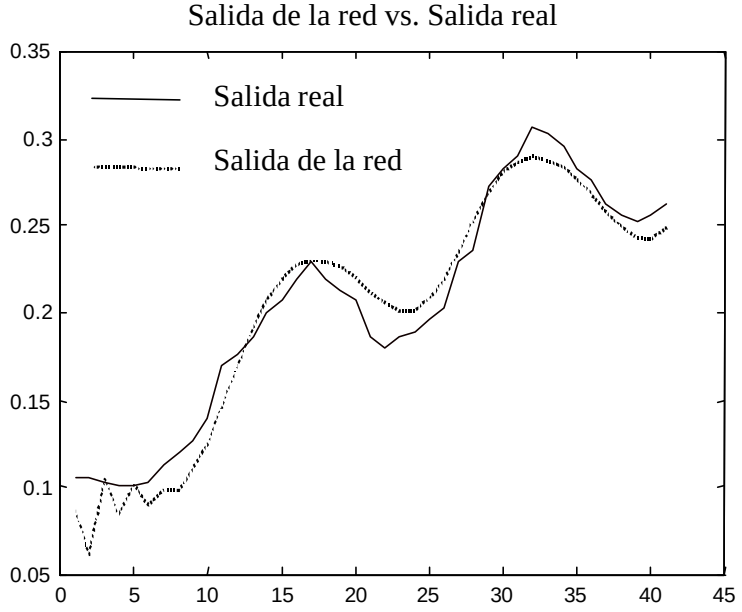


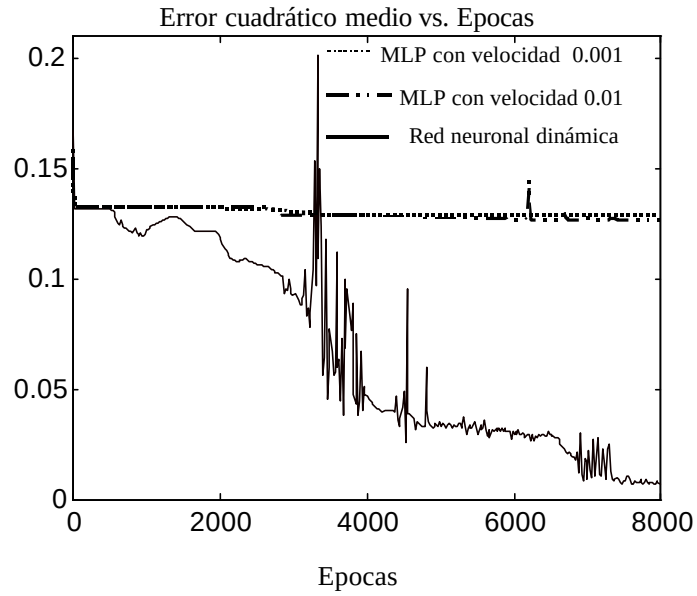
Figura 4.4 – Salida real vs. Salida de la red

4.2.3 EN LA IDENTIFICACIÓN DE UN BRAZO ROBOT PUMA 560.

Como se ha presentado en capítulos previos la dinámica de un robot manipulador es altamente no lineal. En nuestro caso, hemos aplicado la red propuesta a la identificación de un modelo simulado del Puma 560. Con objeto de simplificar el análisis nos hemos centrado en la identificación de la segunda articulación, tomando en consideración los torques de todas las articulaciones como entradas. De cara al entrenamiento hemos utilizado como entradas torques sinusoidales y como salidas objeto de aprendizaje los ángulos de la segunda articulación expresados en radianes. En cuanto a la constitución de la red esta consta de 3 capas y 3 neuronas por capa, mientras que las inicializaciones de los pesos y parámetros dinámicos se han realizado igual que en la sección 4.2.1. Las entradas a la red han sido los torques de las 5 articulaciones del robot. En la Figura (4.4) presentamos la evolución de la curva de error para una velocidad de aprendizaje de los pesos de 0.1. Como se puede observar, después de 8000 épocas se obtiene un error cuadrático medio de 0.0055. En un intento por comparar los resultados hemos aplicado una red multi-capa estática tomando como en el caso de la seccion 4.2.1 una velocidad de aprendizaje de 0.01, dado que para el caso de una velocidad de 0.1 se produce la inestabilización del proceso de aprendizaje. Con esta velocidad se obtiene después de 8000 épocas un error cuadrático medio de 0.1268, y reduciendo ésta a 0.001 se alcanza un error cuadrático medio de 0.1291. En la Figura (4.4) presentamos las curvas de error correspondientes a estos casos en donde se puede apreciar que la red propuesta alcanza un error cuadrático medio inferior en dos ordenes de magnitud a los obtenidos por medio de la red multi-capa estática.

Intentando mejorar el comportamiento de la red multi-capa estática se ha aplicado esta junto con el algoritmo de aprendizaje backpropagation con velocidades de aprendizaje adaptativas y momentum standard [DEM93]. El resultado de tal experimento se muestra en la figura (4.6) en donde se puede apreciar que se alcanza en este caso un error cuadrático medio de 0.0655 en la época 8000, lo cual supone una considerable mejora respecto a los resultados obtenidos hasta ahora. En la Figura (4.7) presentamos la evolución de las velocidades de aprendizaje.

A la vista de los buenos resultados al introducir un algoritmo adaptativo para las velocidades de aprendizaje y el momentum en la red multi-capa estática en donde el error cuadrático medio ha disminuido un orden de magnitud respecto al algoritmo de aprendizaje con velocidades fijas y sin momentum parece conveniente introducir una estrategia similar en el caso de la red neuronal dinámica.



Nota: se ha utilizado MLP como abreviatura de red Multi-capa éstatica

Figura 4.5 – Curva de error de la Red dinámica y la MLP para diferentes velocidades de aprendizaje

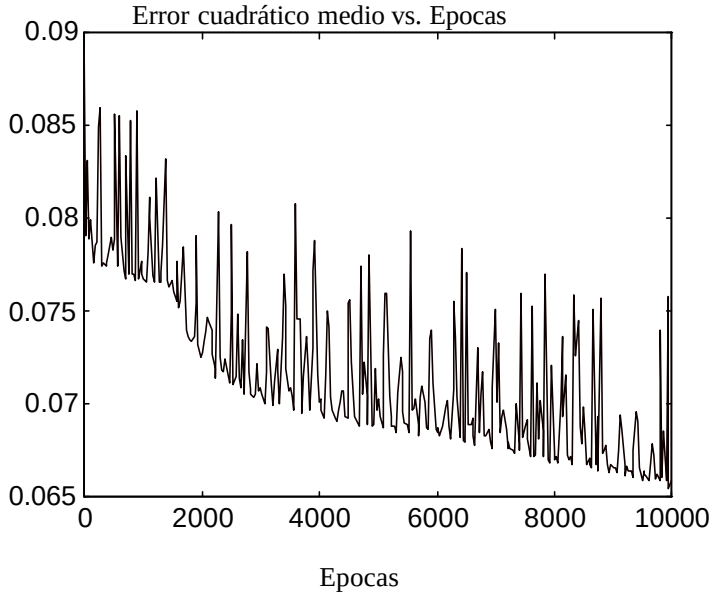


Figura 4.6 – Evolución del error cuadrático medio de la red multi-capa estática cuando se introduce el algoritmo adaptativo de las velocidades de aprendizaje y el momentum.

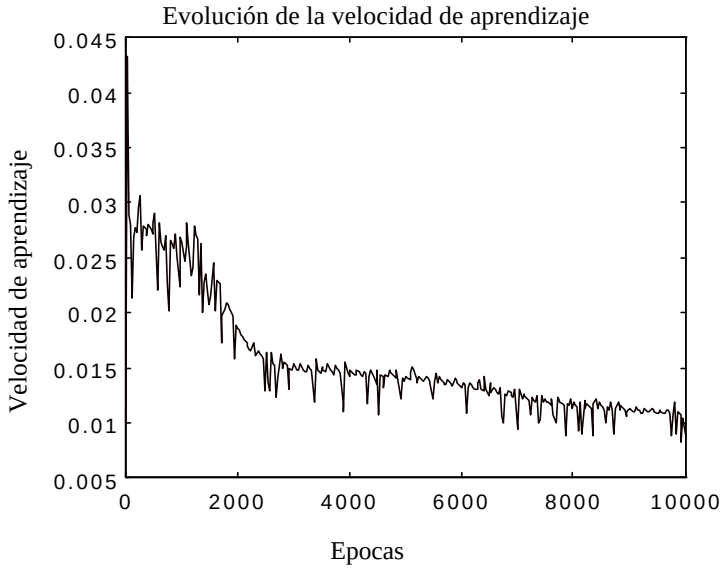


Figura 4.7 – Evolución de las velocidades de aprendizaje.

Teniendo presente que en el capítulo anterior habíamos introducido como modificación del algoritmo inicialmente propuesto un mecanismo para adaptar las velocidades de aprendizaje, además de introducir un momentum variable según los valores que fuese tomando el error podríamos volverlo a utilizar aquí. En resumen este consistía en:

Si *error (t)* < *error (t-1)* Entonces:
Velocidad (t+1)=*velocidad* (t) * 1.15;
Momentum (t+1)= *Momentum* (0);
En caso contrario:
Velocidad (t+1)=*velocidad* (t) * 0.75;
Momentum (t+1)= 0;

El momentum se ha introducido de la siguiente forma para los pesos:

$$w_{ij}^{k}(r+1) = w_{ij}^{k}(r) - m_c h(r-1)\frac{\partial E}{\partial w_{ij}^{k}}(r-1) - (1-m_c)h(r)\frac{\partial E}{\partial w_{ij}^{k}}(r) \qquad (4.31)$$

Donde:

m_c = mometum.

$h(r-1)$, $h(r)$ = velocidades de aprendizaje en los instantes r y (r-1).

$\frac{\partial E}{\partial w_{ij}^{k}}(r-1)$, $\frac{\partial E}{\partial w_{ij}^{k}}(r)$ = Derivada parcial del error con respecto al peso w_{ij}^{k} en los instantes (r-1) y r.

De igual forma para los parámetros también se ha introducido un momentum, considerando el mismo valor que para los pesos. En este caso interviene en las expresiones de actualización de los parámetros de la siguiente forma:

$$a_{ij}^{k}(r+1) = a_{ij}^{k}(r) - m_c \eta_a \frac{\partial E}{\partial a_{ij}^{k}}(r-1) - (1-m_c)\eta_a \frac{\partial E}{\partial a_{ij}^{k}}(r) \qquad (4.32)$$

$$b_{ij}^{k}(r+1) = b_{ij}^{k}(r) - m_c \boldsymbol{h}_b \frac{\P E}{\P b_{ij}^{k}}(r-1) - (1-m_c)\boldsymbol{h}_b \frac{\P E}{\P b_{ij}^{k}}(\mathrm{r}) \qquad (4.33)$$

Donde:

$\boldsymbol{h}_a$, $\boldsymbol{h}_b$ = velocidades de aprendizaje de los parámetros a_i^k y b_i^k.

$\frac{\P E}{\P a_{ij}^{k}}(r-1)$, $\frac{\P E}{\P a_{ij}^{k}}(r)$ = Derivada parcial del error con respecto al peso a_i^k en los instantes (r-1) y r.

$\frac{\P E}{\P b_{ij}^{k}}(r-1)$, $\frac{\P E}{\P b_{ij}^{k}}(r)$ = Derivada parcial del error con respecto al peso b_i^k en los instantes (r-1) y r.

Obsérvese que en este caso las velocidades de aprendizaje $\boldsymbol{h}_a$ y $\boldsymbol{h}_b$ se mantienen constantes a lo largo del entrenamiento. Introduciendo esta estrategia en el algoritmo y realizando la misma inicialización que en el caso anterior obtuvimos un error cuadrático medio después de 8000 épocas de 0.0011, el cual es considerablemente menor que el conseguido sin tener en cuenta esta modificación en el algoritmo.

Con el objeto de comparar los resultados obtenidos con los que proporcionan otras técnicas hemos elegido aplicar un algoritmo de aprendizaje sobre una red Multi-capa estática no basado en las ideas del gradiente descendente como es el algoritmo de aprendizaje de Levenberg-Marquard [DEM93]. Este nos llevó a un error cuadrático medio inferior al resultante de aplicar el backpropagation con velocidades adaptativas y momentum, 0.0498 transcurridas 8000 épocas, el cual todavía se haya lejos de los errores obtenidos con la arquitectura y algoritmo de aprendizaje propuestos.

También se han comparado los resultados con los de una red neuronal de retardos en el tiempo, es decir, una arquitectura de red consistente en una red multi-capa estática en donde como entradas se presentan las entradas actuales y las retrasadas en el tiempo hasta un instante determinado. Según el paso de tiempo al que lleguemos hacia atrás se define el orden de la red de retardo en el tiempo, conocida en la terminología inglesa como TDNN (Time delay Neural Network) [LAN90][LAP87]. En nuestro caso hemos probado una TDNN de orden 3, es decir, hemos considerado como entradas a la red Multi-capa estática la entrada actual, la entrada previa y la entrada retrasada dos instantes de tiempo. Aplicando esta red hemos obtenido un error cuadrático medio de 0.0373 después de 8000 épocas, el cual es unas treinta veces mayor al obtenido con la red neuronal propuesta.

A modo de resumen las comparaciones realizadas y los resultados obtenidos se muestran en la tabla 4.1 en donde se especifican las redes y algoritmos utilizados, junto con el error cuadrático medio al que se ha llegado en cada caso.

También se ha realizado un ensayo para comprobar si la red ha aprendido la dinámica del robot. En la Figura 4.8 presentamos el resultado de haberle presentado como entradas a la red torques sinusoidales de una frecuencia no utilizada en la fase de entrenamiento, concretamente 95 rad/seg. Los ángulos mostrados corresponden a la segunda articulación de un PUMA 560. Como se puede observar la red neuronal es capaz de seguir la trayectoria de esta articulación. Como comparación en la Figura 4.9 se muestra el resultado de un experimento similar, pero en este caso utilizando una red multi-capa estática con el algoritmo de entrenamiento de velocidades de aprendizaje adaptativas y momentum. Como se puede comprobar a la vista de la gráfica, la red no es

capaz de aprender la relación dinámica entre los torques y los ángulos de la segunda articulación.

Algoritmos de aprendizaje	**Error cuadrático medio**
Red Neuronal Dinámica	0.0055
MLP con velocidad de aprendizaje = 0.01	0.1268
MLP con velocidad de aprendizaje = 0.001	0.1291
MLP con momentum y velocidad de aprendizaje adaptativa	0.0655
Red Neuronal Dinámica con momentum y velocidad de aprendizaje adaptativa	0.0011
Algoritmo de Levenberg-Marquardt	0.0498
TDNN (Red neuronal de retardo en el tiempo)	0.0373

Tabla 4.1 – Error cuadrático medio de diferentes algoritmos de aprendizaje

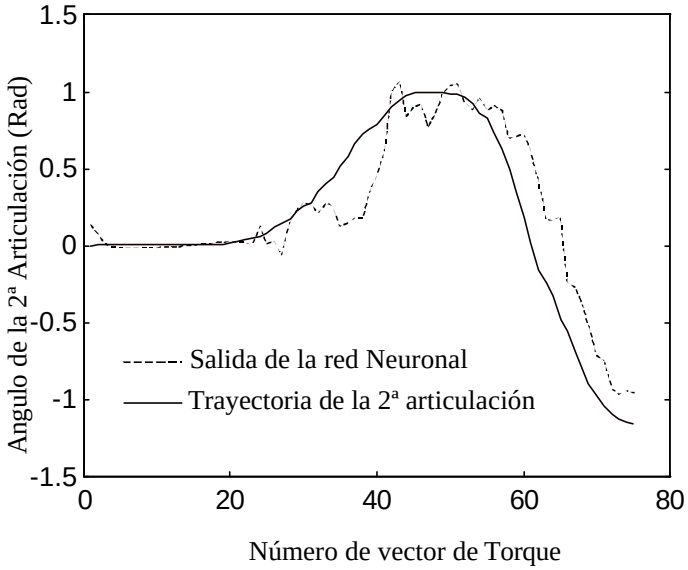


Figura 4.8 – Salida de la Red propuesta vs. Trayectoria en ángulos de la 2ª articulación

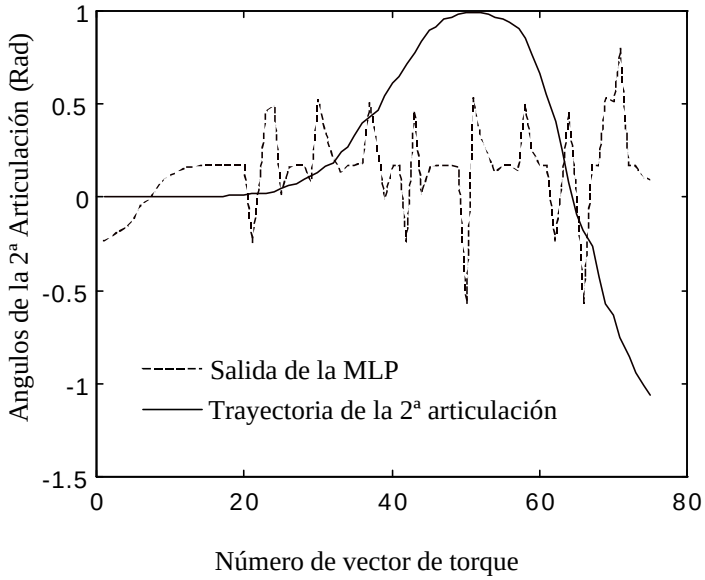


Figura 4.9 – Salida de la MLP vs. Trayectoria en ángulos de la 2ª articulación del PUMA 560.

Capítulo 5
RED NEURONAL DINÁMICA RECURRENTE

5

RED NEURONAL DINÁMICA RECURRENTE

En este capítulo se analiza un caso particular de la red neuronal propuesta, caracterizado principalmente por la incorporación a través de las ecuaciones de estado de dependencia respecto de otras neuronas de la misma capa. Adaptaremos el algoritmo de aprendizaje a este caso, introduciendo algunas modificaciones respecto al algoritmo propuesto en los capítulos previos. Este caso particular nos lleva a una arquitectura de red más simple en cuanto al número de capas, así como a simplificar las expresiones del algoritmo de aprendizaje. Por último utilizaremos la red neuronal en la resolución del problema de la dinámica inversa, tomando como conjunto de entrenamiento datos obtenidos a partir de la aplicación de la formulación de Newton-Euler.

5.1 PARTICULARIZACIÓN DE LA RED NEURONAL PROPUESTA.

5.1.1 MODELO DE RED NEURONAL.

En este capítulo vamos a introducir una particularización de la red propuesta. Elegiremos un modelo de neurona lineal, es decir, las funciones de activación serán escogidas como funciones lineales del tipo f(x)=x, de tal manera que el comportamiento no lineal se introduce a través de la ecuación de estado. Tomando en consideración como expresión para la ecuación de estado:

$$\dot{x}_i^k = B_i^k\left(x_i^k, b_i^k\right) + A_i^k\left(x_i^k, a_i^k\right) n_i^k \qquad (5.1)$$

Donde $A_i^k\left(x_i^k, a_i^k\right)$ y $B_i^k\left(x_i^k, b_i^k\right)$ son funciones de forma arbitraria. En nuestro caso estas funciones se eligieron como:

$$B_i^k\left(x_i^k, b_i^k\right) = \sum_{j=1}^{N_k} b_{ij}^k\, g_j\left(x_j^k\right) - x_i^k \qquad (5.2)$$

$$A_i^k\left(x_i^k, a_i^k\right) = 1 \qquad (5.3)$$

Donde:

$g_i\left(x_i^k\right)$ = función no lineal monótonamente creciente.

N_k = Número de neuronas en la capa k.

Tomando en cuenta las expresiones para $A_i^k\left(x_i^k, a_i^k\right)$ y $B_i^k\left(x_i^k, b_i^k\right)$ el modelo de red neuronal puede expresarse como:

$$n_i^k = \sum_{j=1}^{m_{k-1}} w_{ij}^k y_j^{k-1} + w_{i0}^k \tag{5.4}$$

$$\dot{x}_i^k = -x_i^k + \sum_{j=1}^{N_k} b_{ij}^k \; g_j^k\left(x_j^k\right) + n_i^k \tag{5.5}$$

$$y_i^k = x_i^k \tag{5.6}$$

Como se puede observar en este caso se ha prescindido de la función $A_i^k\left(x_i^k, a_i^k\right)$, por lo que el parámetro a_i^k no interviene en el proceso de aprendizaje. Por otro lado hemos de remarcar que el comportamiento no lineal de la red se ha introducido por medio de las funciones $g_j\left(x_j^k\right)$. Todas estas consideraciones nos llevan a una red de neuronas lineales dado que las funciones de activación son lineales, con la incorporación a las mismas de un comportamiento no lineal a través de las ecuaciones de estado, siendo susceptibles de ajuste los parámetros b_{ij}^k y los pesos w_{ij}^k. También hemos introducido en las ecuaciones de estado una dependencia de los estados de una neurona con respecto a los estados de las otras neuronas en la misma capa, en el convencimiento de que esta dependencia entre neuronas contiguas podría aumentar la capacidad de aprendizaje de la red.

Destaquemos por otro lado que hemos introducido una modificación de cara al cálculo de la salida de las neuronas en aras de simplificar el algoritmo de aprendizaje. En vez de tomar el estado en el instante actual, se deja evolucionar hasta que se alcance el estacionario siendo éste el que se toma como salida. Este estado estacionario vendrá dado por la ecuación (5.5) tomando la derivada del estado igual a cero:

$$0 = -\tilde{x}_i^k + \sum_{j=1}^{N_k} b_{ij}^k \; g_j^k\left(\tilde{x}_j^k\right) + n_i^k \tag{5.7}$$

Donde:

$\tilde{x}_i^k$ = estado estacionario de la neurona i-ésima en la capa k-ésima.

Por lo tanto el modelo de neurona, teniendo en cuenta la ecuación (5.7), se puede expresar como:

$$n_i^k = \sum_{j=1}^{m_{k-1}} w_{ij}^k y_j^{k-1} + w_{i0}^k \tag{5.8}$$

$$0 = -\tilde{x}_i^k + \sum_{j=1}^{N_k} b_{ij}^k \; g_j^k\left(\tilde{x}_j^k\right) + n_i^k \tag{5.9}$$

$$y_i^k = \tilde{x}_i^k \tag{5.10}$$

En las aplicaciones presentadas en esta memoria, se ha utilizado una red de dos capas donde la primera corresponde a neuronas cuyo comportamiento viene dado por las ecuaciones (5.8), (5.9) y (5.10), mientras que la segunda corresponde a la capa de salida constituida por neuronas puramente lineales. Esta capa de neuronas lineales ha sido introducida con el objeto de mejorar la capacidad de la red respecto a adaptarse más convenientemente a la salida objeto de aprendizaje. Hemos de hacer hincapié en que esta capa de neuronas lineales realmente no contribuye en el aprendizaje del comportamiento dinámico, únicamente interviene en determinar una relación lineal entre las salidas de la red y la capa de entrada. En la figura (5.1) se presenta un diagrama de esta red, en donde se puede apreciar que aun cuando se han introducido modificaciones ésta es similar a la ya expuesta considerando neuronas lineales e introduciendo el comportamiento no lineal a través de las ecuaciones de estado. Por otra parte también se han hecho depender los estados de cada neurona de los estados de las otras neuronas de la misma capa.

Aunque la arquitectura es similar a la de la red propuesta en capítulos anteriores, la forma de operar en el dominio temporal es diferente. También se debe destacar que hemos reducido la red a una única capa de neuronas con comportamiento dinámico, siendo ésta una característica favorable en cuanto a simplicidad en la implementación.

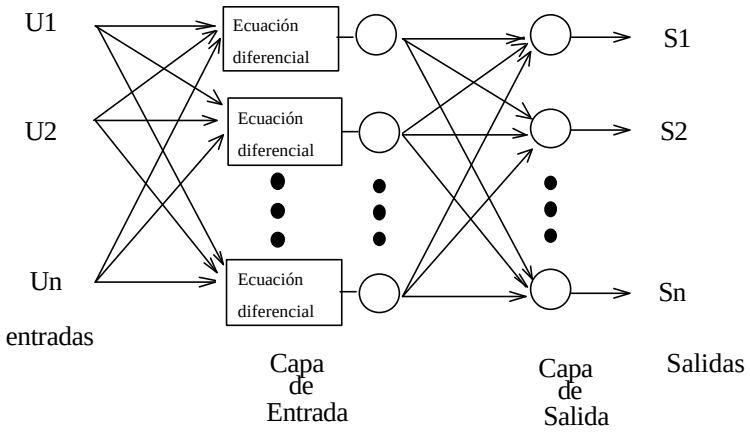


Figura 5.1 – Esquema de la red neuronal.

Definida la arquitectura de red, sólo resta hallar un algoritmo de aprendizaje que permita a la red reproducir las relaciones dinámicas que se desean modificando para ello los diferentes parámetros y pesos. Tengamos en cuenta que serían tres clases de parámetros y pesos los que habría que modificar. La primera serían los valores de $\{w_{ij}^1\}$ que multiplican las entradas a la red siendo w_{ij}^1 el peso entre la entrada de la red i-ésima con la neurona j-ésima de la capa de entrada. La segunda se refiere a los valores $\{b_{ij}^1\}$ que influyen de forma decisiva en el comportamiento dinámico y en el acoplamiento entre los diferentes estados de las neuronas de la capa de entrada, siendo b_{ij}^1 el parámetro correspondiente al estado de la neurona i-ésima que multiplica al estado de la neurona j-ésima de la capa de entrada. Por último la tercera clase de parámetros hace mención a los pesos correspondientes a las neuronas lineales de la última capa, estos

pesos se denotarán por $\{d_{ij}\}$ donde j es el índice del número de salida e i es un índice sobre las neuronas de la capa de entrada.

Antes de concluir esta sección reescribamos las ecuaciones para la red mostrada en la figura (5.1), la cuales pueden simplificarse como se indica a continuación prescindiendo de la ecuación (5.10):

$$n_i^1 = \sum_{j=1}^{N_e} w_{ij}^1 u_j + w_{i0}^1 \tag{5.11}$$

$$0 = -\tilde{x}_i^1 + \sum_{j=1}^{N_1} b_{ij}^1 \, g^1{}_j\left(\tilde{x}_j^1\right) + n_i^1 \tag{5.12}$$

$$s_l = \sum_{j=1}^{N_1} d_{jl} \tilde{x}_j^1 \tag{5.13}$$

Donde:

u_j es la j-ésima entrada a la red.

N_1 es el número de neuronas en la capa de entrada.

N_e es el número de entradas a la red

s_l es la salida l-ésima de la red

5.1.2 ALGORITMO DE APRENDIZAJE.

Teniendo en cuenta la arquitectura de red mostrada en la sección anterior dado que presenta cambios respecto a la arquitectura propuesta en los capítulos anteriores, presentamos aquí como se modificaría el algoritmo de aprendizaje en este caso. Como siempre se debe, a través de la modificación de los diferentes parámetros y pesos de la red, minimizar la función de error:

$$E = \frac{1}{2}\sum_{i=1}^{M}\left(r_i - s_i\right)^2 \tag{5.14}$$

Donde :

M = Número de neuronas en la capa de salida.

r_i = Salidas deseadas.

s_i = Salidas de la red.

En nuestro caso estos parámetros y pesos son w_{ij}^1 , b_{ij}^1 y d_{jl}. En lo sucesivo omitiremos el superíndice del parámetro b_{ij}^1 y el peso w_{ij}^1 dado que únicamente tenemos una capa de neuronas con comportamiento dinámico. Las actualizaciones de parámetros y pesos vienen dadas por las siguientes ecuaciones:

$$w_{ij}(m+1) = w_{ij}(m) - \eta_w \frac{\partial E}{\partial w_{ij}} \tag{5.15}$$

$$b_{ij}(m+1) = b_{ij}(m) - \eta_b \frac{\partial E}{\partial b_{ij}} \tag{5.16}$$

$$d_{jl}(m+1) = d_{jl}(m) - \eta_d \frac{\partial E}{\partial d_{jl}} \tag{5.17}$$

Donde:

η_w, η_b y η_d = Velocidades de aprendizaje.
m es un índice sobre las actualizaciones.

Para actualizar parámetros y pesos por tanto sólo es necesario calcular las derivadas parciales del error con respecto a pesos y parámetros. Empecemos calculando la derivada parcial del error con respecto a los pesos w_{ij} :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j^1} \frac{\partial y_j^1}{\partial w_{ij}} \tag{5.18}$$

Siendo:

$$\frac{\partial E}{\partial y_j^1} = \sum_{l=1}^{N_s} \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial y_j^1} \tag{5.19}$$

Si tomamos en consideración las ecuaciones (5.10) y (5.13) tendremos:

$$\frac{\partial s_l}{\partial y_j^1} = d_{jl} \tag{5.20}$$

Mientras que utilizando la ecuación (5.14) $\frac{\partial E}{\partial s_l}$ vendría dada como:

$$\frac{\partial E}{\partial s_l} = -(r_l - s_l) \tag{5.21}$$

Con lo que la ecuación (5.19) quedaría como:

$$\frac{\partial E}{\partial y_j^1} = -\sum_{l=1}^{N_s} (r_l - s_l) d_{jl} \tag{5.22}$$

Por último teniendo presente que:

$$\frac{\partial y_j^1}{\partial w_{ij}} = u_i \tag{5.23}$$

La ecuación de actualización para los pesos w_{ij} se puede expresar como:

$$w_{ij}(m+1) = w_{ij}(m) + \eta_w \sum_{l=1}^{N_s} (r_l - s_l) \mathrm{d}_{jl}\, \mathrm{u}_i \tag{5.24}$$

De igual forma para los parámetros b_{ij} :

$$\frac{\partial E}{\partial b_{ij}} = \frac{\partial E}{\partial y_i^1} \frac{\partial y_i^1}{\partial b_{ij}} \tag{5.25}$$

Aplicando la regla de la cadena a $\frac{\partial E}{\partial y_i^1}$:

$$\frac{\partial E}{\partial y_i^1} = \sum_{l=1}^{N_s} \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial y_i^1} \tag{5.26}$$

Aprovechando la ecuación (5.20) y (5.21) la ecuación (5.26) se transforma en:

$$\frac{\partial E}{\partial y_i^1} = -\sum_{l=1}^{N_s} (r_l - s_l) d_{il} \tag{5.27}$$

Mientras que utilizando la ecuación (5.12):

$$\frac{\partial y_i^1}{\partial b_{ij}} = g_j(\tilde{x}_j) \tag{5.28}$$

Siendo finalmente la ecuación para la actualización de los parámetros:

$$b_{ij}(m+1) = b_{ij}(m) + \eta_b g_j(\tilde{x}_j) \sum_{l=1}^{N_s} (r_l - s_l) d_{il} \tag{5.29}$$

Por último la ecuación de actualización de los pesos d_{jl} corresponde a la ecuación de actualización de los pesos de una capa de neuronas lineales, obteniéndose las mismas ecuaciones que en el caso de una red de neuronas de este tipo. Considerando que:

$$\frac{\partial E}{\partial d_{jl}} = \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial d_{jl}} \tag{5.30}$$

y calculando las derivadas parciales a través de las ecuaciones (5.14) y (5.13) tenemos:

$$\frac{\partial E}{\partial s_l} = -(r_l - s_l) \tag{5.31}$$

$$\frac{\partial s_l}{\partial d_{jl}} = \tilde{x}_j \tag{5.32}$$

Obteniendo la siguiente ecuación de actualización para los pesos d_{jl} :

$$d_{jl}(m+1) = d_{jl}(m) + \eta_d (r_l - s_l) \tilde{x}_j \tag{5.33}$$

Obsérvese como el hecho de dejar evolucionar el estado hasta su valor estacionario lleva consigo una simplificación considerable de las ecuaciones de actualización de parámetros y pesos. En la sección siguiente aplicaremos este algoritmo de aprendizaje junto con la arquitectura de red mostrada al problema de la determinación de un modelo dinámico para un robot.

5.2 APLICACIÓN DE LA RED NEURONAL

5.2.ALGORITMO PARA EL ROBOT PUMA.

Antes de aplicar la red neuronal, vamos a exponer cual es el problema donde será aplicada. Con este objeto hemos desarrollado un programa para controlar el robot PUMA mediante la aplicación de un algoritmo de control de par calculado. En definitiva consiste en un programa que solicita como entrada la trayectoria en coordenadas cartesianas que deseamos que el robot recorra. Las velocidades y aceleraciones iniciales y finales se han escogido como nulas. Se pretende que el robot recorra una trayectoria recta en el espacio cartesiano donde se han fijado las velocidades y aceleraciones iniciales y finales como nulas y la posición final e inicial de la recta. Sin embargo es más conveniente expresar la trayectoria en función de variables generalizadas del robot, así como velocidades y aceleraciones generalizadas del robot. En nuestro caso al tratarse de un Robot PUMA nos estamos refiriendo a los ángulos de las articulaciones, las velocidades angulares de las mismas y las aceleraciones angulares. Estas variables resultan más útiles desde el punto de vista del control de las articulaciones, por lo que es necesario transformar la trayectoria especificada en coordenadas cartesianas a coordenadas generalizadas. Esta tarea se ha realizado implementando un planificador de trayectorias, el cual mediante la aplicación de la cinemática inversa del robot, así como la cinemática inversa para las velocidades y aceleraciones, nos da finalmente la trayectoria expresada en coordenadas generalizadas. Una vez se ha expresado la trayectoria en coordenadas generalizadas, hemos de utilizar algún método para determinar los pares a aplicar sobre las articulaciones. En nuestro caso hemos empleado la formulación de Newton-Euler, que nos permite a partir de la especificación de la trayectoria en forma de variables generalizadas, sus derivadas primeras y segundas determinar los pares que se deben aplicar a cada articulación. Es importante observar que hemos elegido la formulación de Newton-Euler frente a la formulación de Lagrange-Euler dada la reducción de carga computacional de la primera.

Destaquemos también que en nuestro caso particular las trayectorias se han elegido como líneas rectas en el espacio cartesiano, donde velocidad y aceleración son nulas al inicio y al final del movimiento. Se realiza por tanto en una primera etapa un movimiento acelerado hasta adquirir una velocidad máxima para después sufrir una desaceleración llegando al punto final de la trayectoria con velocidad nula. El tomar las trayectorias de esta forma nos lleva a poder especificar la trayectoria tan solo mediante la aceleración a la que se somete inicialmente el robot y el tiempo que se mantiene esta aceleración, ya que el programa evalúa si tales parámetros son consistentes con la distancia recorrida.

Por otra parte, hemos de precisar que no hemos considerado en el programa variación de la orientación del efector, suponiendo ésta constante a lo largo de la trayectoria, y atendiendo únicamente al movimiento del centro de masa del efector.

5.2.1 APLICACIÓN DE LA RED NEURONAL PROPUESTA A UN ROBOT PUMA.

Uno de los principales problemas del algoritmo de control desarrollado en la sección anterior es su carga computacional. Aunque dijimos en la sección anterior que el algoritmo de Newton-Euler resulta más conveniente desde el punto de vista computacional que el algoritmo de Lagrange-Euler, todavía tiene un alto costo computacional.

Por este motivo hemos aplicado la red propuesta en la sección 5.1 al aprendizaje de la relación entre pares, ángulos, velocidades y aceleraciones de las articulaciones. Por simplicidad en lo sucesivo nos centraremos en una articulación, concretamente la primera, de tal manera que consideramos como entradas los ángulos, velocidades y aceleraciones de todas las articulaciones, mientras que como salida sólo consideramos los pares de la primera articulación.

De hecho hemos entrenado la red para que el efector del robot siga una trayectoria recta en el espacio cartesiano, tal que el robot es acelerado con una aceleración de 0.15 m/seg^2 durante 0.8 seg. Mientras los puntos inicial y final han sido elegidos como (-0.15 m,0.5 m,0 m) y (0.2 m, 0.3 m,0 m) en el espacio cartesiano.

En nuestro caso hemos utilizado un PUMA 560 cuyos parámetros de Denavit-Hartenberg vienen dados como:

articulación	alfa (ºC)	teta(ºC)	d (en cm)	a (en cm)
1	-90	90+giro	0	0
2	0	0+giro	0	43.2
3	-90	-90+giro	15	1.9
4	-90	180+giro	43.2	0
5	90	0+giro	0	0
6	0	0+giro	0	0

Tabla 5.1 – Parámetros de Denavit-Hartenberg.

Mientras que los centros de masas de las articulaciones y las masas de las mismas son:

articulación	$\bar{X}$	$\bar{Y}$	$\bar{Z}$	m (en Kg)
1	0	0.309	0.039	12.95
2	-0.329	0.005	0.2038	22.36
3	0.020	0.014	0.0037	5.0
4	0	0.086	-0.0029	1.117
5	0	-0.010	0.0013	0.618
6	0	0	0.0029	0.157

Tabla 5.2 – Centros de masa y masas de las articulaciones

Y los momentos de inercia de las articulaciones y motores vienen dados como:

articulación	Ixx (kg-m2)	Iyy(kg-m2)	Izz(kg-m2)	Ia(kg-m2)
1	2.35100	0.1968	2.3457	0.7766
2	1.33130	4.313	3.4116	2.3616
3	0.07582	0.07766	0.01038	0.5827
4	0.01410	0.00338	0.01395	1.060
5	0.00055	0.00057	0.000546	0.0949
6	0.00012	0.00012	0.00006	0.107

Tabla 5.3 – Momentos de Inercia de las articulaciones y momentos de inercia de los motores.

Antes de aplicar la red neuronal hemos de reconsiderar las ecuaciones de la misma dado que éstas se han dado como ecuaciones diferenciales, las cuales no pueden llevarse a un ordenador de forma directa. Debido a esto hemos discretizado dichas ecuaciones con lo que las ecuaciones implementadas quedan como:

$$n_i^k = \sum_{j=1}^{m^{k-1}} w_{ij}^k y_j^{k-1} + w_{i0}^k \tag{5.34}$$

$$x_i^k(r+1) = \sum_{j=1}^{N_k} H\, b_{ij}^k\; g_j^k\left(x_j^k(r)\right) + H\, n_i^k \tag{5.35}$$

$$y_i^k = x_i^k \tag{5.36}$$

Donde dado que a priori no conocemos el estado estacionario hemos iterado la ecuación (5.35) un número de veces suficientemente grande como para aproximarnos al valor del estado estacionario. Por otra parte hemos tomado un paso de discretización H igual a 0.1.

De cara al entrenamiento hemos desplazado la curva de pares tal que el par inicial fuese cero, es decir, hemos restado a la curva de pares el valor del par tomado inicialmente. Esta operación en absoluto representa una limitación en el aprendizaje del comportamiento dinámico. Sin embargo en el proceso de aprendizaje esta característica es deseable dado que las actualizaciones de los parámetros y pesos se realizan de una forma más suave, al tener un error inicial pequeño.

Aplicando la red para el tramo indicado hemos obtenido el resultado presentado en la figura (5.2). En esta ocasión, sólo hemos hecho uso de los ángulos de las articulaciones como entradas a la red, siendo las salidas los pares calculados para la primera articulación. El resultado mostrado corresponde a la época 37000, donde se consigue un error cuadrático de 10.75. Es importante destacar que la red aplicada posee 4 neuronas en la capa de entrada, todas las velocidades de aprendizaje para parámetros y pesos han sido escogidas iguales a 0.001 y los diferentes parámetros y pesos iniciales han sido escogidos al azar entre –0.1 y 0.1. Por otra parte, hemos tomado un número de iteraciones igual a 50 para llegar al estado estacionario.

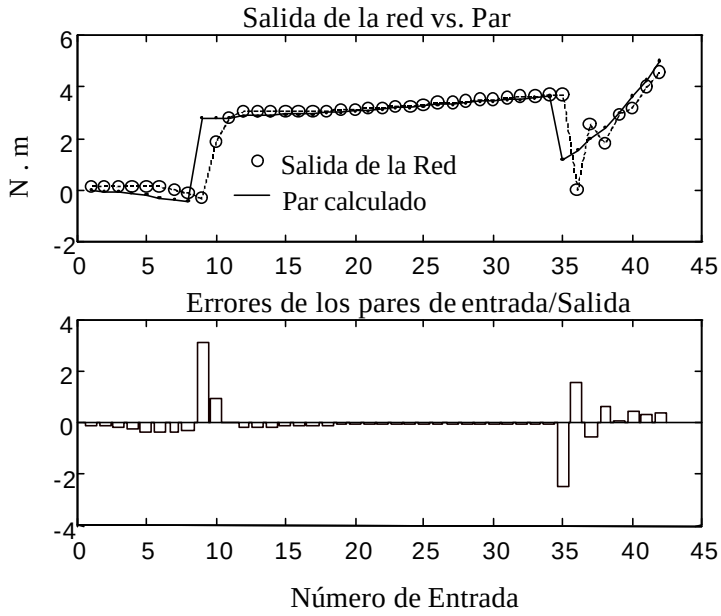


Figura 5.2 – Salida de la red vs. El par calculado para la primera articulación

Destaquemos que el resultado mostrado en la figura (5.2) corresponde a la fase de entrenamiento, es decir, las actualizaciones de los parámetros se han realizado de forma "on-line", cada vez que se presenta un nuevo comando se realiza la actualización de los parámetros sin esperar a que se presenten todos los comandos correspondientes a una época para realizar las actualizaciones. Esto nos lleva a cuestionarnos la bondad del resultado presentado en la figura (5.2), dado que una vez entrenada la red si realmente ha habido un aprendizaje de la relación entre entradas y salidas debería ser parecida a la presentada en la figura (5.2). En la figura (5.3) se muestra el resultado de este ensayo para este caso.

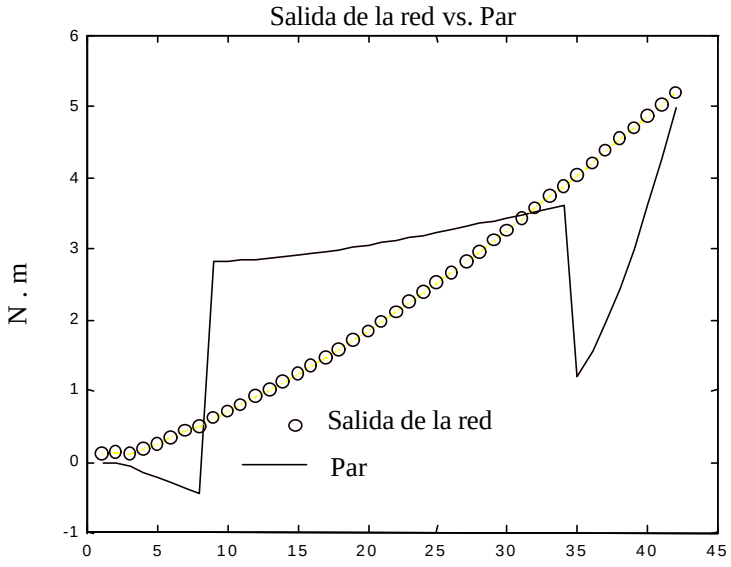


Figura 5.3 - Salida de la red vs. El par calculado para la primera articulación

Habida cuenta de los resultados obtenidos, nuestro interés se centró en aplicar la red con las mismas características expuestas pero esta vez tomando como entradas no tan solo los ángulos de las diferentes articulaciones sino también las velocidades y aceleraciones angulares de las mismas. El resultado obtenido se muestra en la Figura (5.4), después de transcurrir 40000 épocas con un error cuadrático de 9.742, el cual es ligeramente inferior al obtenido anteriormente.

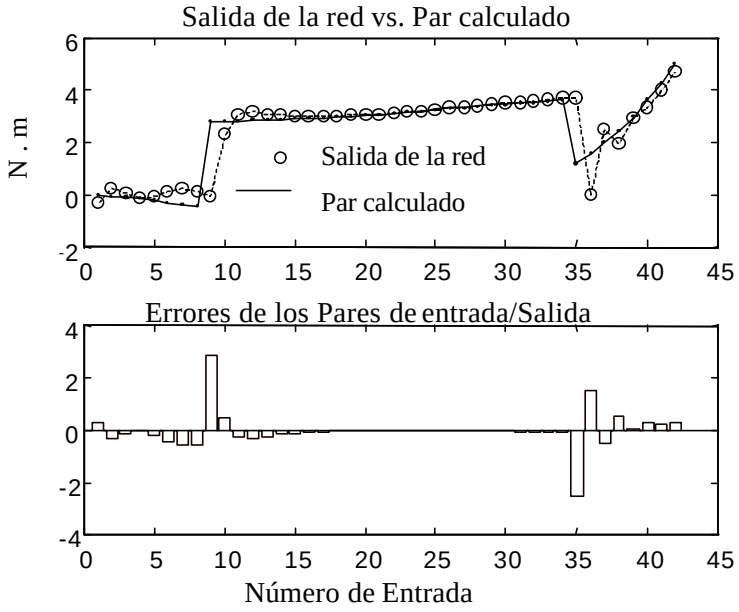


Figura 5.4 – Salida de la red vs. Par calculado para la primera articulación

También en este caso una vez concluida la fase de entrenamiento le hemos presentado a la red todos los ángulos, velocidades y aceleraciones de las articulaciones. Los resultados se muestran en la figura (5.5) donde se aprecia como en este caso la red aprende el comportamiento a diferencia del caso donde sólo utilizamos como entradas los ángulos de las articulaciones.

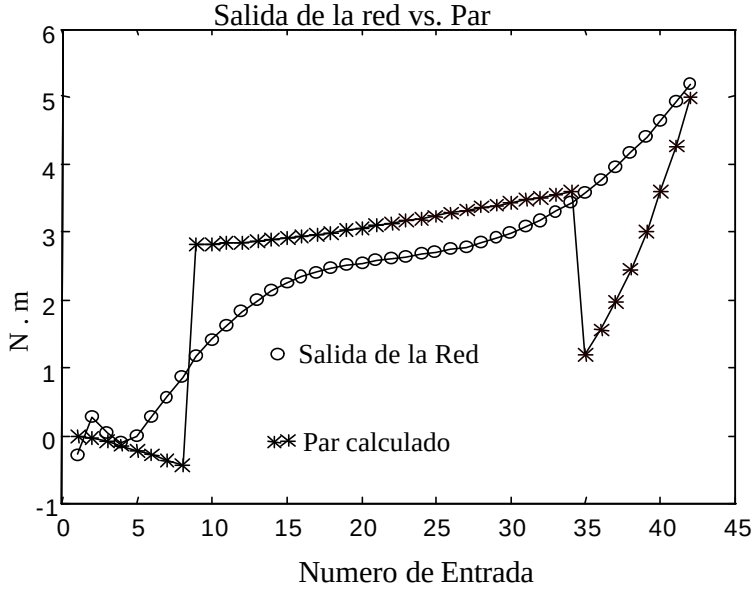


Figura 5.5 – Respuesta de la red una vez concluida la fase de entrenamiento.

Para determinar la influencia que sobre el aprendizaje tiene el número de neuronas hemos aumentado en la red el número de éstas a cinco en la capa de entrada y le hemos presentado los mismos valores de entrada y salida. Los resultados se presentan en la figura (5.6) donde se aprecia un deterioro en el comportamiento, en consecuencia el aumento del número de neuronas empeora el aprendizaje. Una vez la red ha sido entrenada el resultado se muestra en la figura (5.7) donde se aprecia un comportamiento peor que el mostrado en la figura (5.5), sobre todo en la parte final de la curva de pares donde la red no es capaz de seguir el comportamiento creciente.

Por último se han realizado ensayos disminuyendo el número de neuronas de la capa de entrada a 3, obteniendo similares resultados a los presentados en las figuras (5.6) y (5.7). Lo que indica que no hay una relación clara entre el número de neuronas y la capacidad para resolver el problema en los casos estudiados. Hemos de precisar que los datos presentados corresponden todos a un número similar de épocas, así como las velocidades de aprendizaje y los rangos de los parámetros y pesos iniciales, de tal forma que las condiciones de entrenamiento de las redes neuronales escogidas han sido similares.

Hemos de destacar que en los casos mostrados se ha aplicado el conjunto de comandos utilizados en el entrenamiento una vez la red ha sido entrenada, sin embargo cuando cambiamos los comandos las salidas de la red difieren en un mayor grado de los valores de las salidas reales.

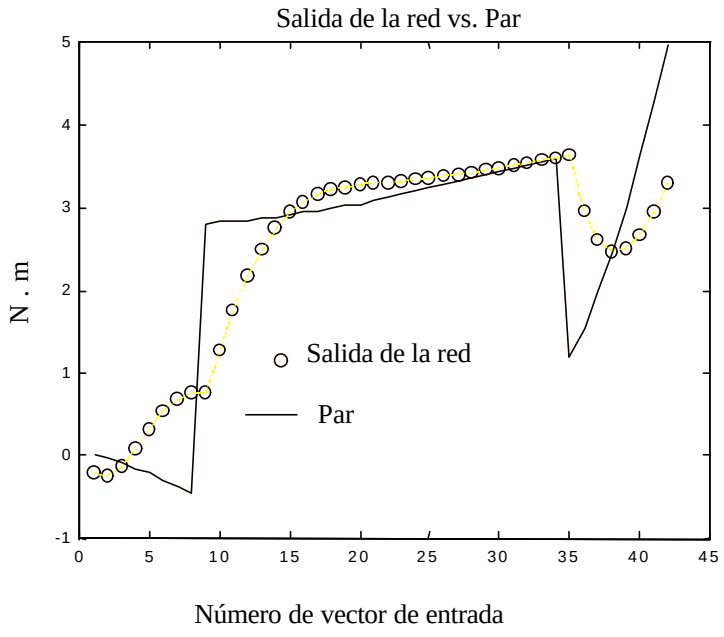


Figura 5.6 – Red neuronal con 5 neuronas en la capa de entrada

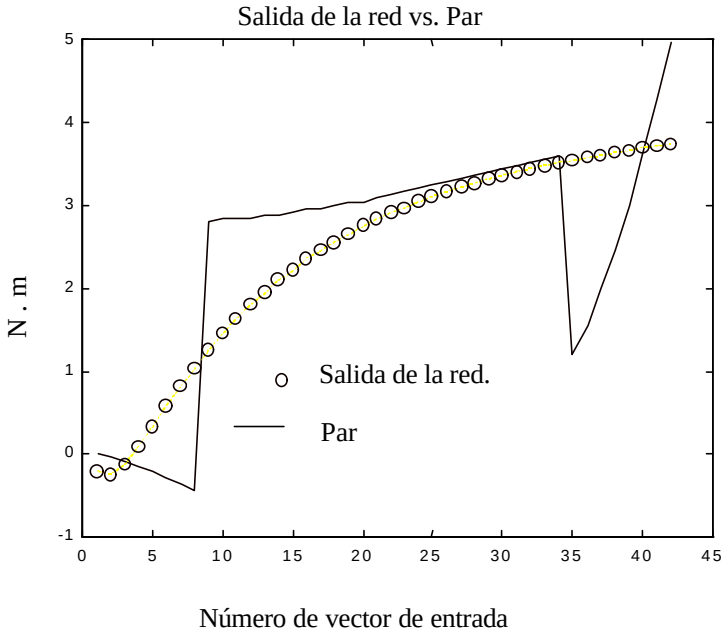


Figura 5.7 – Respuesta ante los ángulos, velocidades y aceleraciones de las articulaciones una vez la red ha sido entrenada.

Capítulo 6
APLICACIÓN DE LAS REDES NEURONALES EN EL PROBLEMA DE LA CINEMÁTICA INVERSA

6

APLICACIÓN DE LAS REDES NEURONALES EN EL PROBLEMA DE LA CINEMÁTICA INVERSA

En este capítulo trataremos el problema de la cinemática inversa, aplicando las redes neuronales como técnica alternativa de solución al problema. Comenzaremos viendo los resultados de una red neuronal estática multi-capa con el algoritmo de aprendizaje backpropagation, para posteriormente aplicar un algoritmo de aprendizaje basado en la utilización de algoritmos genéticos.

6.1 INTRODUCCIÓN

En los capítulos anteriores hemos descrito el problema de la determinación de la cinemática inversa de los robots manipuladores. Como es bien conocido la resolución de este problema tiene una considerable importancia dentro del mundo de la robótica dado que aunque es más fácil especificar una tarea por medio de coordenadas cartesianas del efector, sin embargo desde el punto de vista del control cualquier tarea se describe de forma más fácil en términos de las coordenadas generalizadas asociadas a las diferentes articulaciones. Por otro lado, se debe destacar que la relación entre las coordenadas cartesianas y las coordenadas generalizadas de las articulaciones es una relación altamente no lineal. Una de las dificultades más significativas que encontramos en la resolución de la cinemática inversa es la existencia de varias formas de escoger las variables generalizadas para obtener una misma posición del efector. Debido a la no linealidad del problema en este capítulo emplearemos las redes neuronales como una forma de obtener esta relación. En nuestro caso el estudio se ha aplicado sobre un robot planar de tres articulaciones como el mostrado en la figura (6.1). Tomando como longitudes para las articulaciones un valor de 10 cm para cada una de ellas. Es importante indicar que el robot planar de tres articulaciones se caracteriza por un comportamiento altamente redundante, es decir, para una misma posición del efector expresada en coordenadas cartesianas existen varios conjuntos de coordenadas generalizadas (en este caso los ángulos de las diferentes articulaciones) que dan como resultado la misma posición del efector.

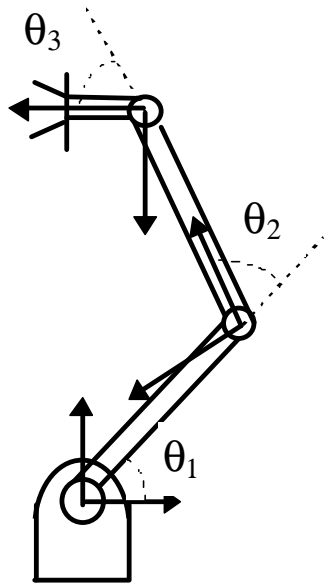


Figura 6.1 – Robot planar de tres articulaciones.

6.2 APLICACIÓN DE REDES NEURONALES ESTÁTICAS EN LA RESOLUCIÓN DE LA CINEMÁTICA INVERSA DE UN ROBOT.

Como primera aproximación a la resolución del problema hemos aplicado una red multicapa estática tomando como función de activación la tangente hiperbólica. Nuestro objetivo en este caso se centra en a partir de algunas posiciones tomadas por el efector del robot expresadas en coordenadas cartesianas(las coordenadas x e y en el plano) y los ángulos de las articulaciones correspondientes a esos puntos entrenar la red neuronal, de tal forma que finalmente la red pueda obtener como salidas los ángulos de las articulaciones correspondientes a diferentes entradas expresadas en coordenadas cartesianas. En nuestro caso hemos empleado el algoritmo de entrenamiento Backpropagation standard.

El primer problema que se nos presentó fue el adaptar el problema de la cinemática inversa a las redes neuronales dado que las coordenadas cartesianas pueden tomar valores arbitrarios y los ángulos toman valores entre 0 y 2π radianes, sin embargo en las redes neuronales utilizadas hemos usado como función de activación la tangente hiperbólica cuyo rango de salida se sitúa entre –1 y 1. Por otra parte, el introducir grandes valores como entradas a la red nos lleva inevitablemente a la saturación de las neuronas. Con el objeto de subsanar esta dificultad hemos aplicado una función lineal a los valores de entrada (las coordenadas x e y). Hemos elegido esta función lineal de la forma $f(x)= A\ x + B$ donde los valores A y B se han tomado tal que el rango de las entradas se sitúe entre –0.8 y 0.8. También los ángulos que se expresan en radianes, se han escalado a través de una función lineal de las mismas características que la utilizada para las entradas dentro del rango de –0.8 a 0.8, acondicionándose a los valores de salida de la red. Destaquemos que esto en absoluto supone el distorsionar la relación entre las coordenadas cartesianas y los ángulos de las articulaciones dado que una vez entrenada la red, únicamente bastará con aplicar a las coordenadas cartesianas la función lineal utilizada para escalar éstas entre (–0.8, 0.8) antes de introducirlas como entradas a la red y aplicarle a las salidas de la red la función lineal inversa de la utilizada

para escalar los ángulos de las articulaciones entre –0.8 y 0.8 obteniendo finalmente los ángulos de las articulaciones para esa posición de efector.

Teniendo en cuenta estas funciones lineales hemos realizado varios ensayos tomando diferentes conjuntos de posiciones del efector al objeto de determinar las prestaciones de las redes neuronales en la resolución de la cinemática inversa. En la figura (6.2) presentamos la curva de error correspondiente a uno de los ensayos, como se puede observar el error cuadrático medio toma un valor de 0.0959 después de 100 épocas. Hemos tomado una red neuronal constituida por 3 capas con 3 neuronas cada una de ellas, la velocidad de aprendizaje para los pesos se ha tomado igual a 0.1 y la inicialización de los pesos se ha realizado en el rango de –0.3 a 0.3. En la tabla 6.1 mostramos el conjunto de posiciones correspondientes al conjunto de entrenamiento.

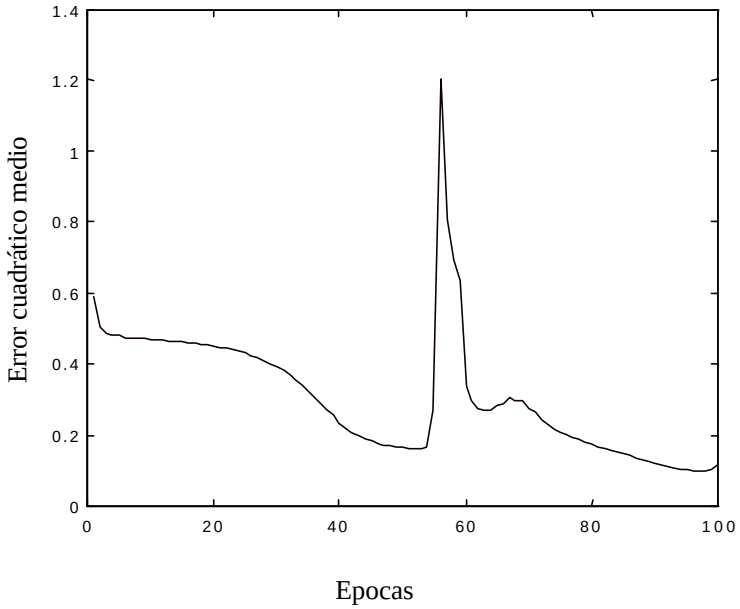


Figura 6.2 – Evolución del error cuadrático medio.

P	X(cm)	Y(cm)	q_1(Rad)	q_2(Rad)	q_3(Rad)
1	1.6837	19.2379	0.1745	1.5708	0.5236
2	-2.9230	13.1072	0.5236	1.2217	1.5708
3	-5.1558	19.2378	0.8727	0.8727	1.2217
4	-1.7346	28.6317	1.2217	0.5236	0.1745
5	-10.3927	24.8391	1.5708	0.1745	0.8727

Tabla 6.1 – conjunto de posiciones utilizadas en el entrenamiento.

En las figuras (6.3), (6.4) y (6.5) presentamos las salidas de la red correspondientes a las posiciones en coordenadas cartesianas (x,y) mostradas en la tabla 6.1 una vez la red ha sido entrenada.

Sin embargo desde el punto de vista de la especificación de tareas al robot estamos más interesados en las coordenadas cartesianas del efector, no en los ángulos de las

articulaciones. Debido a ello de las Figuras (6.3), (6.4) y (6.5) no podemos deducir si las posiciones del efector son cercanas a las especificadas. Con el objetivo de determinar las coordenadas cartesianas que corresponden a los ángulos obtenidos por las salidas de la red se ha desarrollado un programa en donde se ha implementado la cinemática directa, tal que proporcionándole los valores de los ángulos de las articulaciones es capaz de calcular las coordenadas del efector. Las coordenadas del efector para los ángulos mostrados en las figuras anteriores se presentan en las figuras (6.6) y (6.7).

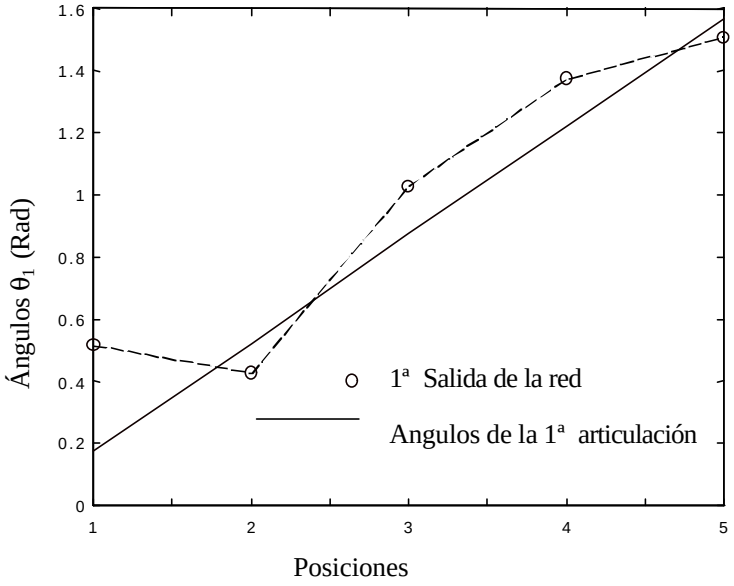


Figura 6.3 – 1ª salida de la red vs. Ángulos de la 1ª articulación

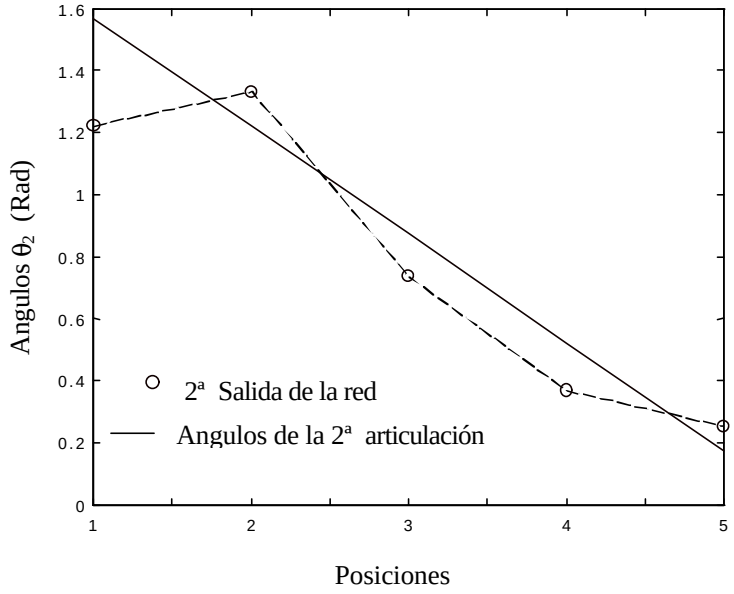


Figura 6.4 – 2ª salida de la red vs. Ángulos de la 2ª articulación

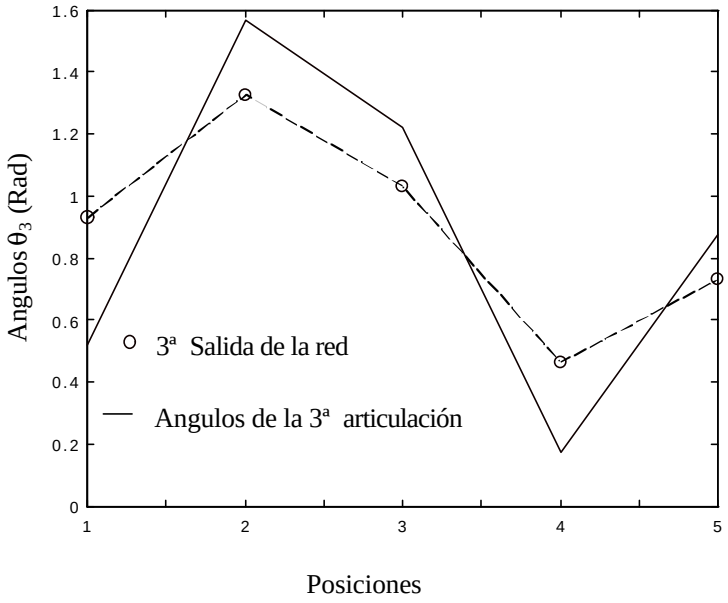


Figura 6.5 – 3ª salida de la red vs. Ángulos de la 3ª articulación

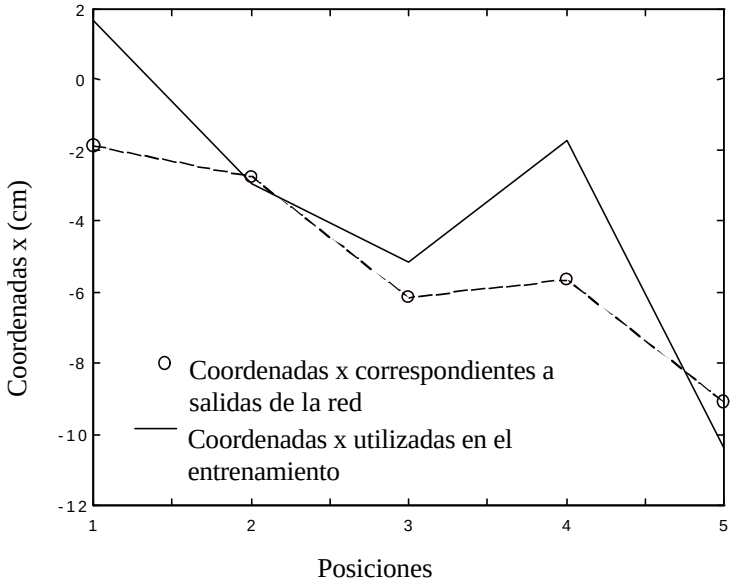


Figura 6.6 – Coordenadas x correspondientes a las salidas de la red neuronal

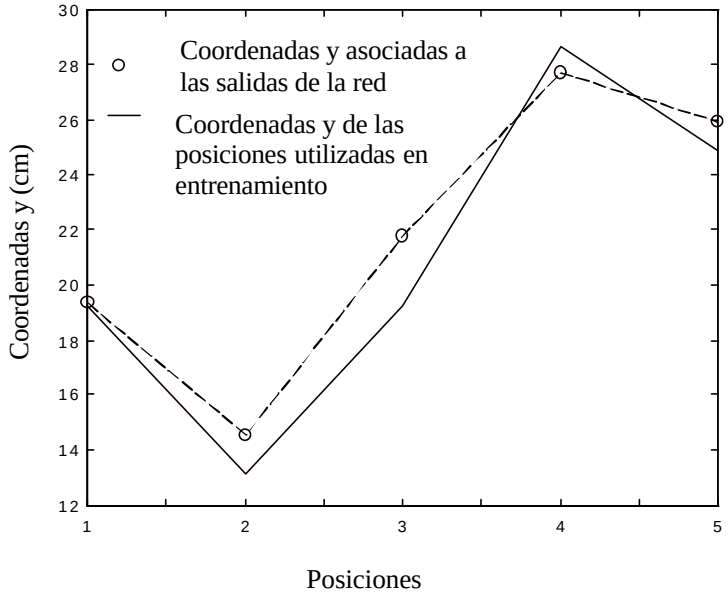


Figura 6.7 – Coordenadas y correspondientes a las salidas de la red neuronal

Hemos de observar que los resultados mostrados en las figuras previas corresponden a las entradas utilizadas en la fase de entrenamiento, parece de interés por tanto presentarle entradas a la red que no hayan sido utilizadas previamente en el entrenamiento, viendo de esta forma la capacidad de generalización de la red. En la tabla 6.2 se muestran los valores presentados a la red una vez esta ha sido entrenada. Obsérvese que las posiciones escogidas no pertenecen al conjunto de posiciones utilizadas en la fase de entrenamiento. En las figuras (6.8), (6.9) y (6.10) se presentan las salidas de la red comparadas con los valores de los ángulos de las articulaciones calculados mediante la aplicación de relaciones geométricas. Como se puede observar, en general existe un seguimiento de dichas curvas.

P	X(cm)	Y(cm)	q_1(rad)	q_2(rad)	q_3(rad)
1	1.0405	18.6556	0.1745	1.5708	0.6109
2	-3.5872	12.0711	0.5236	1.3090	1.5708
3	-6.8144	18.7223	0.9599	0.8727	1.2217
4	-4.2262	28.3816	1.3090	0.5236	0.1745
5	-10.3967	24.8481	1.5708	0.1745	0.8727

Tabla 6.2 – conjunto de posiciones utilizadas de cara a estudiar la generalización.

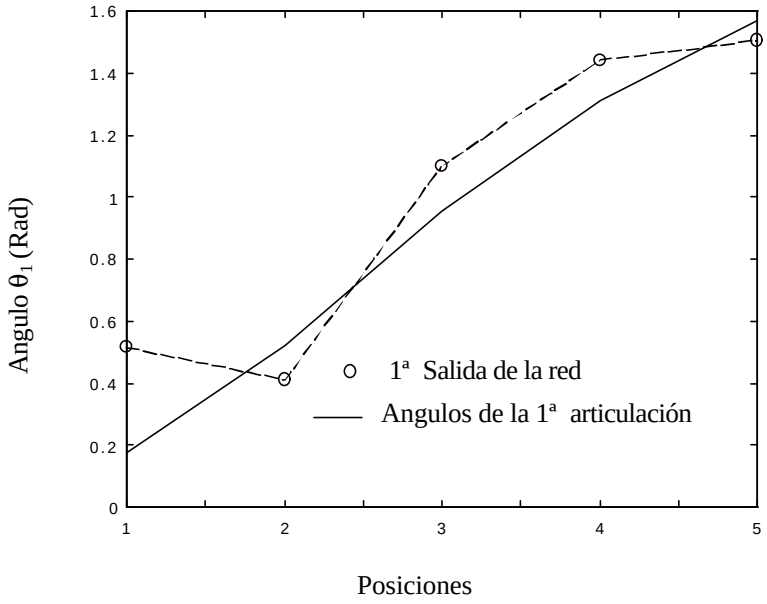


Figura 6.8 – 1ª salida de la red vs. Ángulos de la 1ª articulación

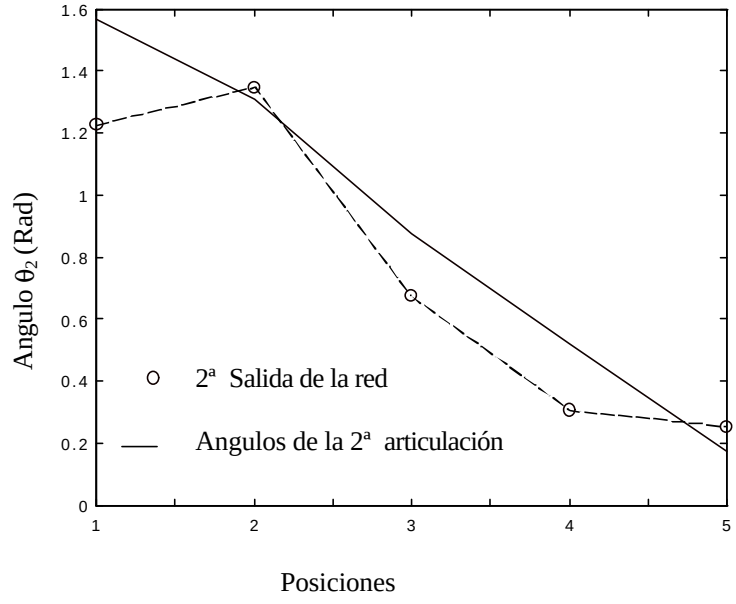


Figura 6.9 – 2ª salida de la red vs. Ángulos de la 2ª articulación

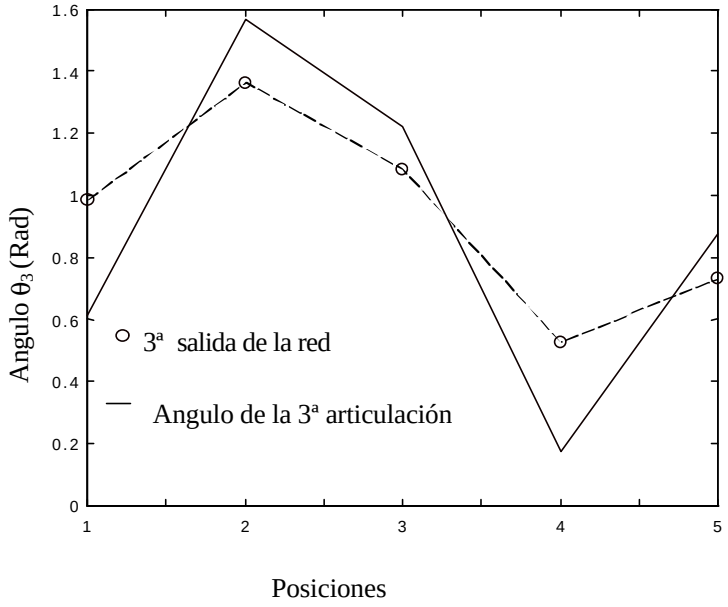


Figura 6.10 – 3ª salida de la red vs. Ángulos de la 3ª articulación

En este caso de la misma forma que se hizo con el conjunto de ángulos utilizados en la fase de entrenamiento, tiene gran interés el comprobar cuales son las desviaciones que se producen por elegir los ángulos dados por la red, frente a tomar los ángulos calculados mediante relaciones geométricas a partir de las coordenadas cartesianas del efector. En las figuras (6.11) y (6.12) presentamos las coordenadas asociadas a los ángulos mostrados en las figuras (6.8), (6.9) y (6.10).

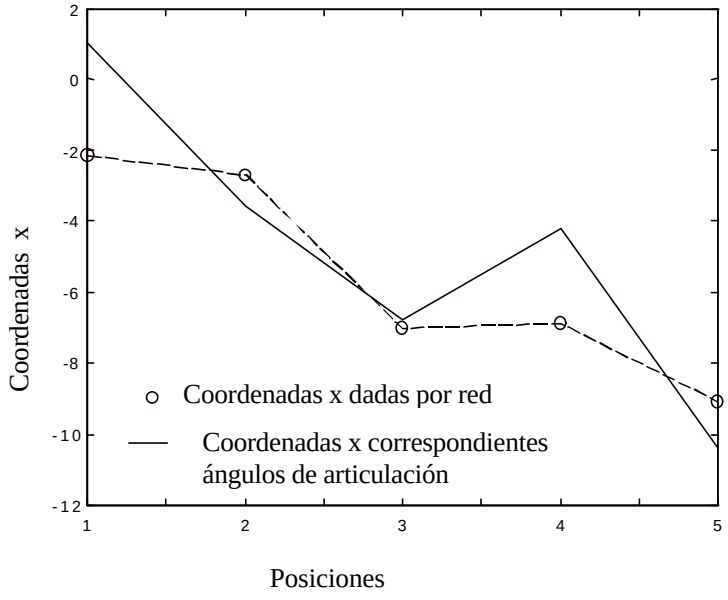


Figura 6.11 – Coordenadas x obtenidas a partir de las salidas de la red

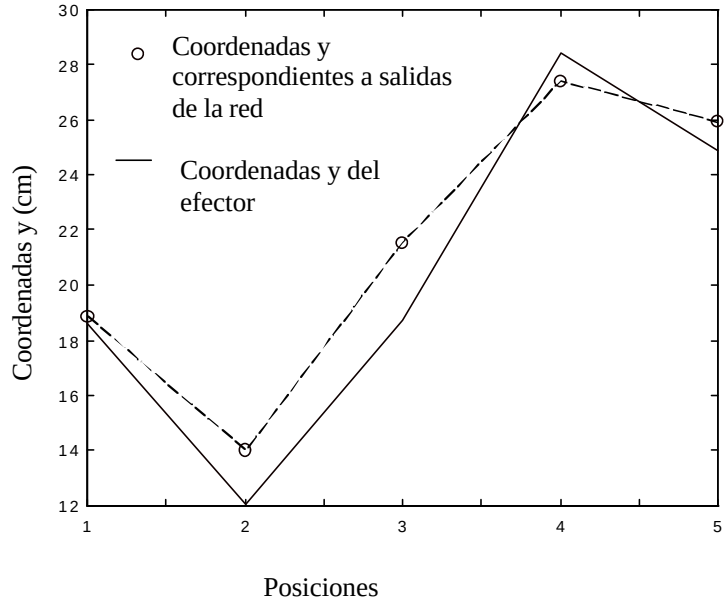


Figura 6.12 – Coordenadas y obtenidas a partir de salidas de la red

También hemos realizado ensayos tomando un mayor número de puntos en la fase de entrenamiento. Concretamente hemos tomado 25 posiciones y hemos aplicado el algoritmo de entrenamiento, tomando una red de 3 capas y 3 neuronas, con una velocidad de aprendizaje de 0.1. Es importante reseñar que en este caso hemos llevado el entrenamiento hasta 62290 épocas, obteniendo un error cuadrático medio de 0.0036. En la figura (6.13) presentamos las salidas de la red correspondientes a los ángulos de la 1ª articulación después de transcurridas 62290 épocas. Hemos de comentar que en este caso hemos escogido la misma curva de ángulos para cada articulación siendo el resultado el mismo para todas las articulaciones, por lo que en la figura (6.13) sólo presentamos los resultados de una articulación.

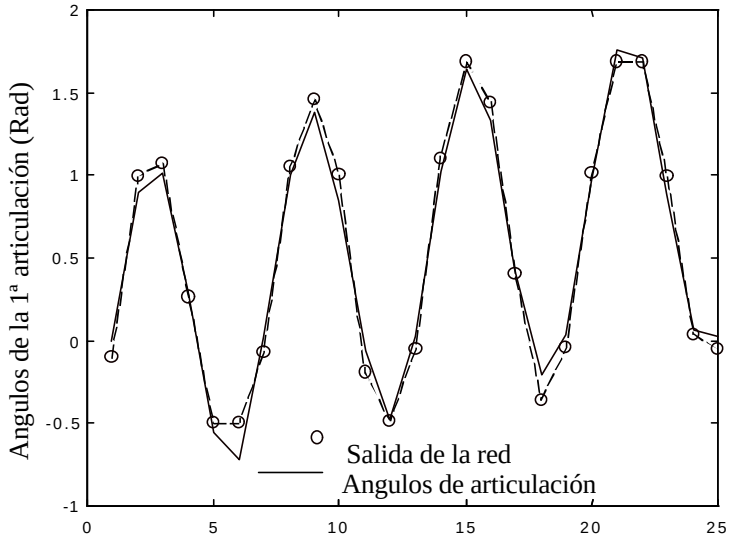


Figura 6.13 – 1ª salida de la red vs. Ángulos de la 1ª articulación

Destaquemos que los puntos tomados en la fase de entrenamiento corresponden a elegir los ángulos de las articulaciones como:

$$\theta_1 = \text{sen}(10\,t) + \text{sen}(0.5\,t) \quad (6.1)$$

$$\theta_2 = \text{sen}(10\,t) + \text{sen}(0.5\,t) \quad (6.2)$$

$$\theta_3 = \text{sen}(10\,t) + \text{sen}(0.5\,t) \quad (6.3)$$

Donde los valores para la variable t se han tomado en el intervalo (0,2.5), obteniendo de esta forma los 25 puntos utilizados. A la vista de cómo se han escogido los puntos de cara al entrenamiento parece lógico ver como responde la red a puntos correspondientes a valores de t fuera del intervalo (0, 2.5). Podríamos por tanto tomar valores de ángulos según las relaciones (6.1) (6.2) y (6.3) para un intervalo de t entre 0 y 5 estudiando de esta forma la generalización de la red. En la figura (6.14) presentamos los valores de la 1ª salida de la red correspondiente a los ángulos de la primera articulación del robot planar. Véase como los ángulos de la primera articulación correspondientes al intervalo para t entre 2.5 y 5 no utilizados en la fase de entrenamiento se aproximan a los ángulos calculados a partir de la aplicación de relaciones geométricas.

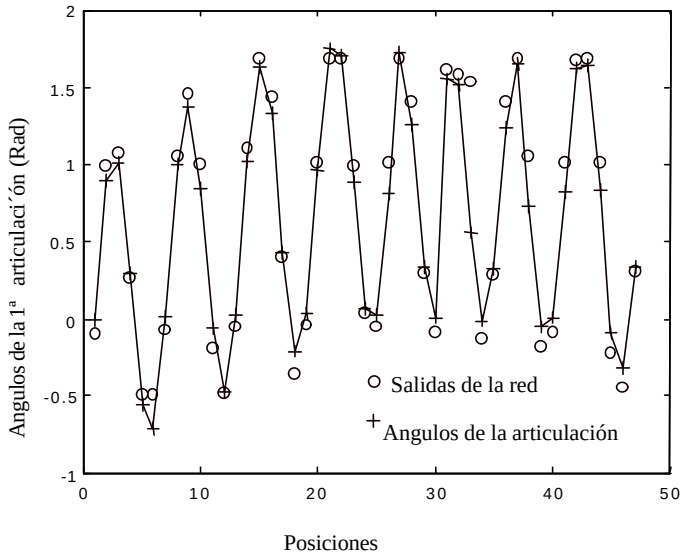


Figura 6.14 – Salida de la red para posiciones correspondientes a t entre 0 y 5.

Obsérvese que en la simulación mostrada anteriormente hemos elegido la misma forma funcional para los ángulos de todas las articulaciones del robot. Si los ángulos fuesen elegidos de diferente forma el aprendizaje por parte de la red neuronal se dificulta. En las Figuras (6.15), (6.16) y (6.17) se pone de manifiesto este hecho dado que los resultados mostrados cuando se eligen ángulos distintos para las diferentes articulaciones corresponden a una vez han transcurrido 1331057 épocas, tomando igual que en la simulación anterior una red de 3 capas con 3 neuronas por capa y una velocidad de aprendizaje de 0.1. El error cuadrático medio conseguido en este caso es de 0.0066.

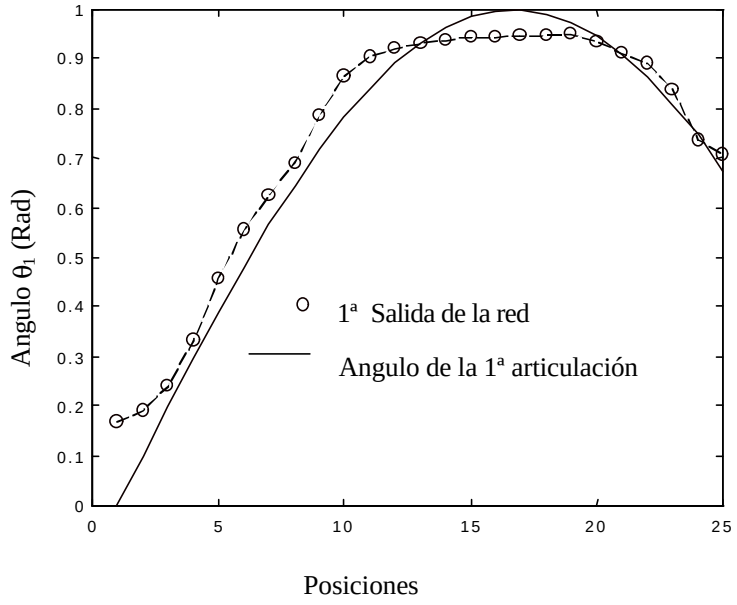


Figura 6.15 – 1ª salida de la red vs. Ángulos de la 1ª articulación

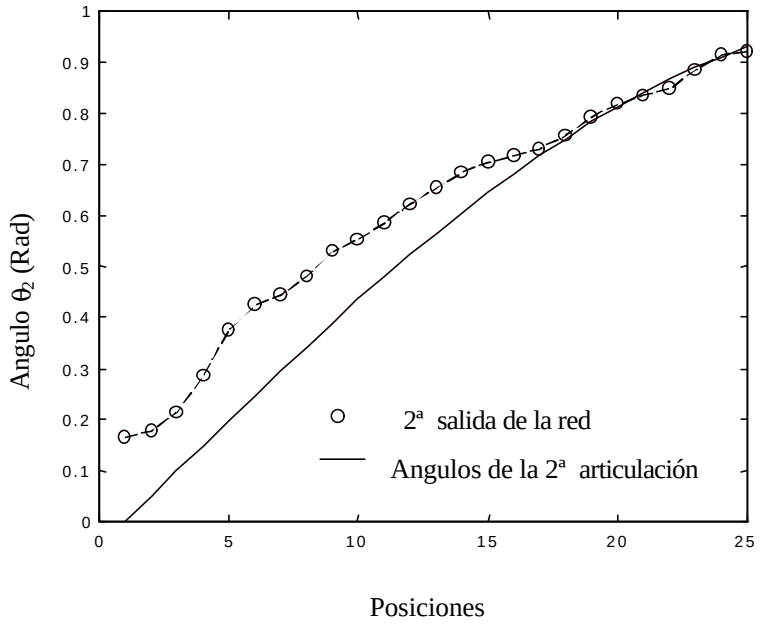


Figura 6.16 – 2ª salida de la red vs. Ángulos de la 2ª articulación

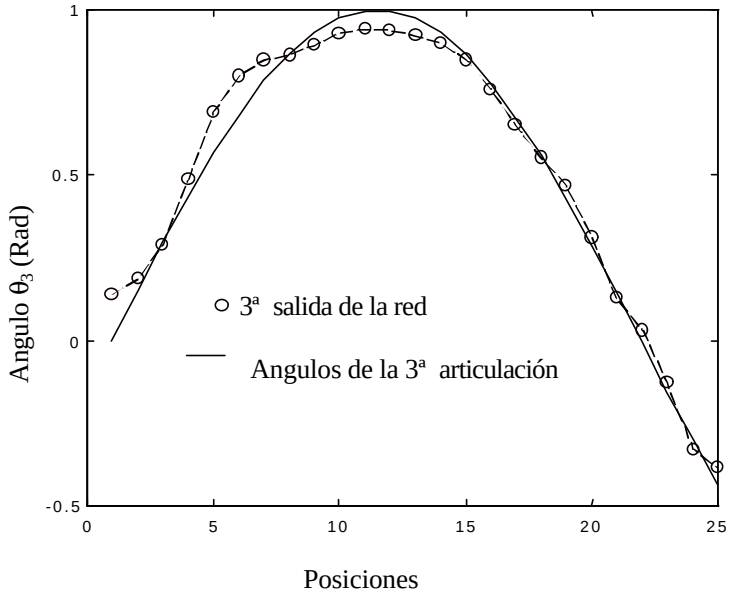


Figura 6.17 – 3ª salida de la red vs. Ángulos de la 3ª articulación

Estos resultados nos llevan a considerar como más conveniente el utilizar una red de una salida para determinar las salidas de cada articulación, dado que el proceso de aprendizaje se acelera considerablemente

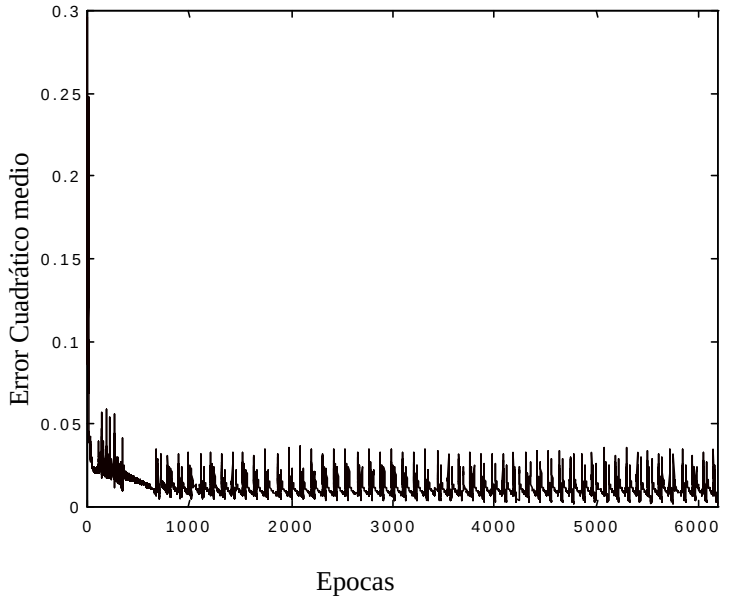


Figura 6.18 – Evolución de la suma de error cuadrático.

En la figura (6.18) mostramos la evolución del error cuadrático medio para el caso de considerar como entradas las posiciones del efector correspondientes a la elección de los ángulos de las articulaciones como se indica en las expresiones (6.1), (6.2) y (6.3), y como salida los ángulos de una única articulación. Después de 6189 épocas, se obtiene un error cuadrático medio de 0.0018. Debemos comentar que en este caso se ha elegido

una red de 3 capas con 3 neuronas por capa y una velocidad de aprendizaje de 0.1. Es importante señalar que aunque el error cuadrático medio más pequeño en la figura (6.18) se obtiene para la época 6189, se puede ver que a partir de la época 1000 los resultados no difieren demasiado de ese valor mínimo.

6.3 APLICACIÓN DE ALGORITMOS GENÉTICOS EN EL ENTRENAMIENTO DE LAS REDES.

En un intento por buscar nuevas formas de resolver el problema de la cinemática inversa hemos introducido en el algoritmo de entrenamiento de las redes vistas en la sección anterior algoritmos genéticos. Una de la razones que nos ha animado a introducir estas técnicas en el aprendizaje es que los algoritmos genéticos presentan una buena respuesta a los requerimientos de búsqueda de soluciones en un espacio multi-dimensional. Una de las características más importantes de los algoritmos genéticos es que mantienen la variedad de potenciales soluciones, permitiendo la búsqueda multi-direccional. De este modo se logra eficientemente la optimización de una función multivariable, escapando de los óptimos locales. Esta cualidad, llamada robustez, radica en que, siendo algoritmos probabilísticos, combinan elementos de la búsqueda directa y estocástica, convergiendo hacia una solución óptima global.
En el caso del entrenamiento de una red neuronal los algoritmos de aprendizaje pueden verse como búsquedas en un espacio multi-dimensional. Teniendo en cuenta este hecho parece natural introducir otras técnicas de búsqueda tales como los algoritmos genéticos.
Los algoritmos genéticos fueron introducidos por John Holland a finales de los 60 y principios de los 70 [HOL75], realizando el primer tratamiento sistemático en su libro "Adaptation in Natural and Artificial Systems" publicado en 1975. Los algoritmos genéticos fueron inspirados por los mecanismos de evolución introducidos por Darwin, de tal forma que las especies están bajo un continuo desarrollo mediante procesos combinados de selección natural, recombinación sexual y mutación. En su investigación John Holland puso de manifiesto la habilidad que representaciones simples tales como grupos de genes, cadenas de bits, etc tienen a la hora de codificar estructuras más complicadas y la potencia de simples transformaciones sobre estas representaciones para mejorar tales estructuras.

Los algoritmos genéticos se basan en las siguientes ideas:

1) En la evolución, cada forma de vida (o especie) se enfrenta con el problema de adaptarse a entornos cambiantes y a menudo complejos.

2) La adaptación se realiza creando nuevas estructuras o individuos, algunos de los cuales se adaptan mejor teniendo más probabilidad de sobrevivir y procrear.

3) Cada individuo dentro de las especies se caracteriza por su estructura cromosómica, es decir, por una cadena de símbolos.

4) Esta estructura cromosómica es la información básica que se reproduce y recombina con otras estructuras cromosómicas para crear nuevos individuos.

5) No es el individuo el que en realidad se adapta para acondicionarse al medio sino las especies en sí mismas.

Teniendo en cuenta estas ideas los algoritmos genéticos responden al siguiente pseudocódigo:

Procedimiento ALGORITMO_GENÉTICO
{ Objetivo minimizar *evalua* (p) tal que p ε U }
P,Q,R: conjuntos de soluciones ⊂ U;
Inicializamos (P);
Mientras no *finalizamos* (P) **haz**:
Q:= *Selecciona* (P);
R:= *Crea* (Q);
P:= *recombina* (P,Q,R)
Final
Final {ALGORITMOS_GENÉTICO}

El algoritmo genético comienza con una población (generalmente aleatoria), P, de cromosomas, cada uno representando una solución al problema y cada uno evaluado de acuerdo a alguna función de coste. En el pseudocódigo es la función *evalua* la que realiza esta última tarea, mientras que la inicialización corresponde a la función *Inicializamos*. Después algunos de los cromosomas se seleccionan a través de la función *selecciona* dándonos un conjunto de cromosomas Q. Esta selección se realiza aplicando principios darwinianos de selección de los mejores. A partir de este conjunto Q se puede crear un nuevo conjunto de cromosomas R, aplicando la función *Crea*. Esta función únicamente aplicará una serie de operadores denominados operadores genéticos para crear este nuevo conjunto, denominado en la terminología inglesa “Offspring pool” o “Child pool”. Los operadores genéticos más comúnmente utilizados son los operadores de crossover y mutación. Los operadores de crossover combinan características de dos cromosomas padres para formar dos cromosomas hijos mediante el intercambio de partes de los cromosomas padres. Los operadores de mutación por su parte introducen modificaciones aleatorias en los cromosomas. Una vez obtenidos estos nuevos cromosomas deben evaluarse mediante la función de coste.
El siguiente paso es la reconstrucción de la población P, a partir de la población original, P, la población de cromosomas seleccionados Q, y la población de hijos, R. En el pseudocódigo esta operación se realiza mediante la función *recombina*. El proceso de reconstrucción de la nueva población P se realiza de diferentes formas, de tal manera que podemos reemplazar enteramente la población original P por la nueva población R o únicamente reemplazar unos pocos cromosomas. Este proceso continua tal y como se indica en el pseudocódigo hasta que se cumple una condición de finalización, indicada por la función *finalizamos*. A estas iteraciones se las conoce habitualmente con el nombre de generaciones, atendiendo a las ideas que inspiraron los algoritmos genéticos.

Una de las ventajas de los algoritmos genéticos es la adaptación de los mismos a muchos problemas. La clave de su versatilidad se encuentra en el hecho de que únicamente hay que proponer una representación de las soluciones del problema objeto de estudio en forma de cromosomas y una función de coste que contemple la bondad de las soluciones.

Varios investigadores han aplicado los algoritmos genéticos al campo de las redes neuronales. Así Reeves y Steele [REE91] emplearon los algoritmos genéticos en el diseño de la arquitectura y establecimiento de los parámetros de una red neuronal

artificial, mientras Motana [MON91] los ha empleado como algoritmo de entrenamiento de redes neuronales, en sustitución del algoritmo Back-propagation ya descrito. En la siguiente sección hemos aplicado los algoritmos genéticos al aprendizaje de la cinématica inversa mediante una red neuronal.

6.4 APLICACIÓN EN LA DETERMINACIÓN DE LA CINÉMATICA INVERSA.

El primer paso para introducir los algoritmos genéticos es dar una representación, es decir, determinar cual va a ser la estructura que juega el papel de cromosoma o individuo. Nosotros identificaremos cada peso de la red neuronal con un gen, de tal forma que cada individuo o cromosoma vendrá dado por un vector de pesos cuyas componentes son los pesos de la red neuronal, siendo cada individuo una solución al problema de la cinemática inversa. Es decir cada individuo vendrá dado como:

$$\nu_i = (w_1, w_2, w_3, ..., w_n) \qquad (6.4)$$

Donde:

ν_i = Individuo i-ésimo.

$(w_1, w_2, w_3, ..., w_n)$ = Vector cuyas componentes son todos los pesos de la red neuronal.

En nuestro caso hemos utilizado una red constituida por 4 capas, de tal forma que la primera capa está compuesta por dos neuronas, ya que las entradas a la red son las coordenadas cartesianas (x,y), mientras que para las capas ocultas hemos elegido respectivamente 10 y 5 neuronas. En cuanto a la capa de salida sólo hemos elegido una neurona de tal forma que únicamente nos concentraremos en analizar uno de los ángulos de las tres articulaciones, sin pérdida de generalidad dado que podríamos utilizar redes neuronales similares para aprender las relaciones de las coordenadas del efector con los demás ángulos. Concretamente hemos elegido los ángulos correspondientes a la primera articulación como salidas de la red. Esta elección de la red nos lleva por tanto a tener vectores de pesos de 91 componentes, es decir, cada cromosoma estará compuesto por 91 genes, o desde otro punto de vista, cada solución del problema de la cinemática inversa estará determinada por los 91 pesos de la red. Obsérvese que en este caso los genes se han elegido como números reales.

De cara a la aplicación del algoritmo genético hemos de definir también una población, es decir, un conjunto de cromosomas. En el caso tratado en este estudio hemos tomado una población compuesta por 50 cromosomas. Por tanto la población queda definida como:

$$P(t) = (\nu_1, \nu_2, \nu_3, ..., \nu_{50}) \qquad (6.5)$$

Donde t hace referencia a la iteración en la que nos encontramos en la aplicación del algoritmo genético. Como punto de partida de la aplicación del algoritmo habremos de definir la población para la primera iteración, es decir hemos de definir P(1). En las simulaciones realizadas ésta ha sido elegida aleatoriamente tomando valores para los pesos entre –1.0 y 1.0.

Además de elegir la representación es necesario especificar alguna forma de relacionar la búsqueda en el espacio mutidimensional de los pesos con el problema de la

cinemática inversa. Esto se realiza mediante algún criterio de calidad de tal forma que podamos asignar a cada cromosoma un valor acorde con el grado de bondad de la solución. Este valor lo hemos especificado mediante una función, a la cual se le suele dar el nombre de función de coste o "fitness". Nosotros hemos elegido esta función de coste de la siguiente forma:

$$f(v_i) = \frac{\sum_{n=1}^{N} \left((\theta_0^n - \theta_d^n)^2 + \alpha (\theta_0^n - \theta_d^{n-1})^2 \right)}{N} \qquad (6.6)$$

N es el número de puntos tomados en el entrenamiento. De esta manera podemos obtener una media sobre ese conjunto de puntos. Teniendo en consideración este hecho la función de coste queda reducida a dos términos. El primero de ellos es la diferencia $(\theta_0^n - \theta_d^n)$ donde θ_0^n es la salida de la red para la posición n-ésima, mientras que q_d^n es el ángulo de la primera articulación calculado mediante relaciones geométricas. Este término da cuenta de la desviación del comportamiento de la red frente al comportamiento deseado. Sin embargo existe un segundo término en la función de coste, concretamente la diferencia $(\theta_0^n - \theta_d^{n-1})$ entre el valor de la salida de la red θ_0^n y el valor del ángulo de la articulación correspondiente al punto que se le ha presentado previamente a la red θ_d^{n-1}. Con este término hemos querido pesar el hecho de que los nuevos ángulos de la articulación se hallen cercanos a los anteriores, es decir, a los ángulos correspondientes a las posiciones previamente adoptadas por el efector. Recordemos en este punto que cuando introducimos el problema de la cinemática inversa en la sección (6.1) comentamos que una de las características de este problema era la existencia de varias soluciones para una misma posición del efector, con lo que este segundo término en la función de coste viene a elegir de las posibles soluciones aquellas que se encuentren más cercanas a las anteriores, influyendo de forma positiva en la consecución de movimientos suaves. A este segundo término lo denominaremos de penalización. Destaquemos que la influencia de este término puede controlarse a través del parámetro α mostrado en la expresión (6.6). En los casos estudiados este parámetro ha sido elegido como 0.2 en algunos y 0.3 en otros. Por otro lado, hemos de comentar que la función de coste se evalúa en cada iteración del algoritmo genético. La evaluación de la función de coste para cada cromosoma lleva únicamente consigo el utilizar la red para obtener los valores de salida de la misma con los pesos correspondientes a ese cromosoma.

Una vez establecida una representación conveniente de cara a la aplicación de los algoritmos genéticos, así como definida la función de coste que liga los algoritmos genéticos con el problema que nos ocupa, hemos de plantear ahora la forma de los operadores genéticos. En nuestro caso hemos empleado un operador de cruce y un operador de mutación con el objeto de ir generando nuevos individuos que se ajusten mejor a la solución óptima del problema.

El operador de cruce creará un vector de pesos hijo o individuo hijo a partir de dos vectores de pesos padres o individuos padres. Para determinar los pesos del individuo hijo, los pesos correspondientes a cada neurona del hijo se copian de los correspondientes a uno de los individuos padre seleccionado al azar. El proceso se repite hasta completar el individuo hijo. En lo que respecta al operador de mutación se ha elegido de la siguiente forma: tomando un individuo hemos seleccionado n neuronas al azar y añadido a sus pesos un valor al azar entre –1.0 y +1.0. De esta manera creamos un nuevo individuo mutante en la población. En nuestro caso n=5, sin que este valor

tenga un significado relevante dado que se podría tomar un número diferente de neuronas. En los casos estudiados se ha aplicado el operador de cruce con una probabilidad de 0.2, mientras que el operador de mutación se ha aplicado con una probabilidad de 0.8, de tal forma que creamos un individuo mutante 8 veces de cada 10 generaciones o iteraciones.

Por otra parte a la hora de construir la nueva población en cada iteración sustituimos el individuo con menor valor de coste por el nuevo individuo hijo, sea este fruto de la aplicación del operador de cruce o el de mutación.

En teoría aplicando los operadores de cruce y de mutación descritos en los párrafos previos, el algoritmo genético debería converger hacia individuos con un valor de la función de coste óptima. Sin embargo, en la implementación hemos visto que el algoritmo converge prematuramente a mínimos locales. El problema surge del hecho de que el algoritmo en pocas iteraciones crea copias múltiples del mismo cromosoma en las nuevas poblaciones, de tal forma que al ya limitado tamaño de la población se le suma el hecho de que las nuevas poblaciones representan cada vez más a un único cromosoma, conduciéndonos inevitablemente hacia un valor óptimo, posiblemente local. La causa más común de convergencia prematura es la aparición de superindividuos que tienen valores de la función de coste mucho mejores que el promedio de la función de coste de la población. Con el objeto de eliminar este fenómeno hemos realizado la siguiente operación: una vez transcurrida una iteración evaluamos la diferencia entre el valor de la función de coste menor en la población y el valor medio de la función de coste para la población, si esta diferencia estuviese por debajo de una cierta cota, se volvería a inicializar la segunda parte de la población tal como se hizo para la población P(1), mientras que la primera parte de la misma se conservaría tal y como está. Obsérvese que la primera parte conservaría los cromosomas obtenidos hasta esta iteración con lo cual no se pierden los resultados parciales del algoritmo, evolucionando finalmente hacia un óptimo global.

Teniendo en cuenta las ideas mostradas hemos aplicado los algoritmos genéticos al problema de la cinemática inversa realizando varias simulaciones. En la figura (6.19) presentamos la curva de evolución de la función de coste para una de las simulaciones, donde se puede apreciar que después de 20000 iteraciones se llega a un valor de la función de coste de 0.0007. En la tabla (6.3) mostramos las posiciones utilizadas en el entrenamiento así como el ángulo de la primera articulación, además de las salidas de la red una vez ésta ha sido entrenada. En la figura (6.20) hemos mostrado las coordenadas (x,y) deseadas y las coordenadas obtenidas utilizando el modelo de la cinemática directa a partir de los ángulos proporcionados por varias redes cuyas salidas son los ángulos de cada articulación.

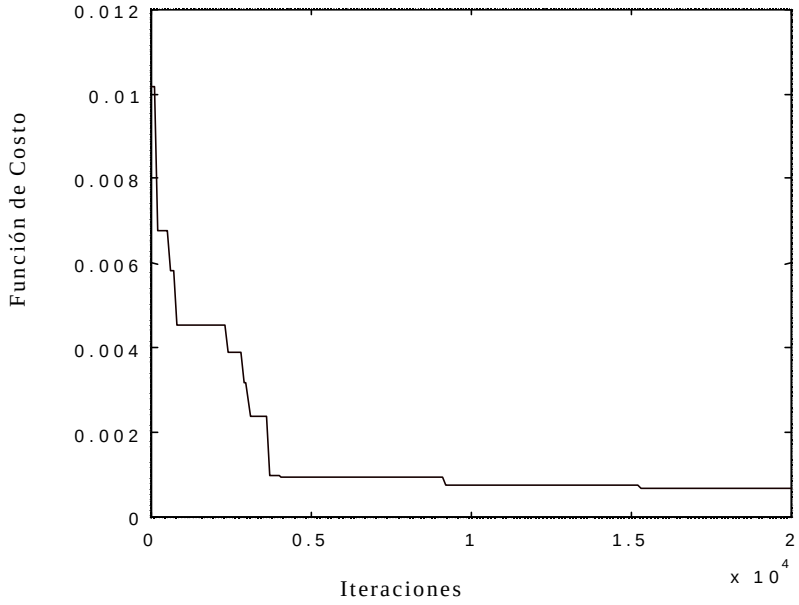


Figura 6.19 – Valores de la Función de coste para 20000 generaciones

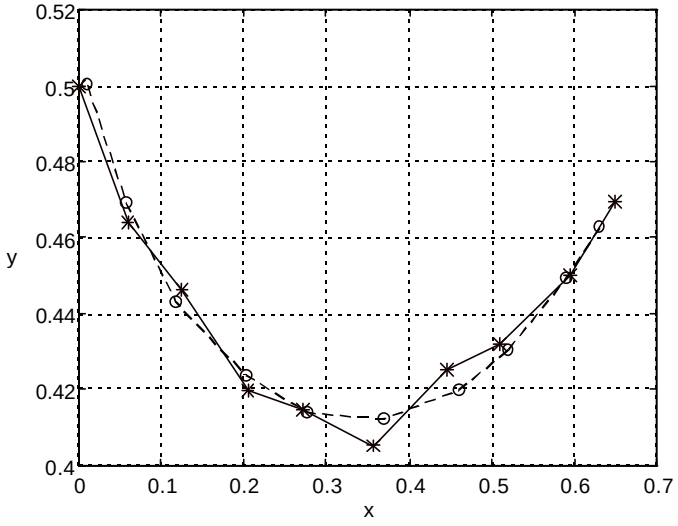


Figura 6.20- Trayectoria objetivo (línea discontinua) frente a trayectoria aprendida (línea continua)

En la tabla 6.3 presentamos los valores de los ángulos y coordenadas representados gráficamente en la figura 6.20.

P	*X(cm)*	*Y(cm)*	q_d^p (Rad)	q_o^p (Rad)
1	0.0499	0.5025	1.5708	1.54366
2	0.0777	0.4729	1.5008	1.50983
3	0.1364	0.4476	1.4708	1.46721
4	0.1956	0.4065	1.4208	1.41116
5	0.8848	0.5885	1.3908	1.36195
6	0.3573	0.4052	1.3708	1.34550
7	0.4462	0.4252	1.3508	1.32888
8	0.5095	0.4318	1.3208	1.31410
9	0.5945	0.4500	1.2908	1.30484
10	0.6486	0.4592	1.2708	1.29935

Tabla 6.3 – Posiciones, ángulos y salidas de la red

En la tabla 6.4 presentamos el resultado de otro entrenamiento en donde hemos introducido como entradas a la red 2 posiciones que no pertenecen al conjunto de entrenamiento, observando la capacidad de generalización una vez la red ha sido entrenada.

P	q_d^p **(Rad)**	q_o^p **(Rad)**	q_o^p **(Rad) (Posiciones no utilizadas en la fase de entrenamiento)**
1	1.5708	1.56647	
2	1.3963	1.40957	
3	1.2217	1.23835	
4	1.0472	1.06343	
5	0.9996		0.98882
6	0.8727	0.88471	
7	0.6981	0.70575	
8	0.5712		0.549372
9	0.5236	0.515073	
10	0.3491	0.309609	
11	0.1745	0.147081	
12	0	0.002763	

Tabla (6.4) – Posiciones y ángulos presentados a la red

Es importante destacar que en el caso de las dos posiciones no utilizadas en la fase de entrenamiento se observan valores muy cercanos a los calculados mediante relaciones geométricas. También debemos resaltar que en los experimentos realizados se ha observado un mejor resultado en aquellos casos en los cuales los puntos objeto de entrenamiento se encontraban cercanos entre ellos, degradándose el comportamiento cuando esta distancia va aumentando. De hecho los casos en donde la secuencia de posiciones está relativamente más lejos entre sí nos llevan a valores de la función de coste mayores, de tal forma que las distancias entre los ángulos de las articulaciones y las salidas de la red se hacen mayores. A la vista de los resultados este método se muestra como una alternativa al entrenamiento mediante el algoritmo de backpropagation.

Capítulo 7
UN CONTROL BASADO EN REDES NEURONALES PARA UN ROBOT PUMA.

7

UN CONTROL BASADO EN REDES NEURONALES PARA UN ROBOT PUMA

Hasta ahora hemos trabajado con las redes neuronales en los problemas de identificación de robots manipuladores y cinemática inversa, en este capítulo comenzamos revisando algunos de los algoritmos de control sobre robots manipuladores en general y mostramos sobretodo aquellos basados en redes neuronales. Posteriormente proponemos un nuevo algoritmo de control basado en redes neuronales estáticas, para finalmente implementarlo en simulación y mostrar los resultados obtenidos.

7.1 INTRODUCCIÓN.

En este capítulo introduciremos un algoritmo de control para un robot manipulador. Una primera tentativa de control podría consistir en el esquema representado en la figura 7.1.

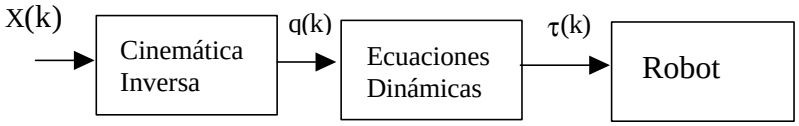


Figura 7.1 – Control Directo de un Robot

Siendo X(k) las posiciones cartesianas, q(k) las variables generalizadas (ángulos y desplazamientos) y $\tau(k)$ los pares de fuerza de los motores que se encuentran en las articulaciones.

Aun disponiendo del modelo inverso se puede ver que esta configuración en lazo abierto no es robusta y las perturbaciones o incertidumbres en el modelo real alterarán el comportamiento. Es necesario por tanto trabajar en lazo cerrado.

El control de robots se ha basado fundamentalmente en métodos tradicionales de control. El controlador más común es un PID con parámetros fijos. La acción derivativa es básica para suavizar las fuertes oscilaciones que aparecen. Sin embargo, teniendo en cuenta que los robots son sistemas fuertemente no lineales, ningún controlador con parámetros fijos puede proporcionar un buen control en todo el espacio de trabajo.
Los robots controlados por PID serán excesivamente lentos en algunas regiones y pueden oscilar en otras. Conseguir movimientos suaves y precisos sobre una trayectoria especificada requiere un conocimiento exacto de los parámetros cinemáticos y dinámicos, y en estas condiciones los algoritmos de control alcanzan un alto grado de complejidad.
Sin embargo el ser humano mueve sus brazos con respuestas suaves y precisas para realizar una gran cantidad de tareas sin necesidad de resolver ecuaciones cinemáticas o dinámicas. En un intento de emular el comportamiento humano se han desarrollado sistemas de control adaptivo, en los cuales los parámetros del controlador se ajustan automáticamente dependiendo de la región del espacio de estados (posiciones y velocidades). Estos sistemas se comportan mejor que los sistemas con parámetros fijos, pero con el costo adicional de una gran complejidad. Es más, el diseño de un buen sistema de control adaptivo requiere incluso modelos más precisos de los robots.
Como alternativa a estas estrategias de control se han desarrollado estrategias de control basadas en redes neuronales. En este capítulo nos centraremos en aquellas que se dirigen al aprendizaje del modelo dinámico inverso supuesto que ya tenemos la tarea a realizar por el manipulador especificada en variables generalizadas. En el caso de un robot Puma, tal como se muestra en la figura 7.2, presuponemos que conocemos todos los ángulos de las articulaciones que corresponden a las posiciones tomadas por el efector para realizar la tarea especificada.
Las estrategias mostradas suponen que los problemas de determinación de la trayectoria para llegar a un punto objetivo, así como el problema de la cinemática inversa se resuelven por medio de los algoritmos standard. Partimos por tanto de tener los ángulos deseados para cada articulación como objetivo de control. El problema del control se reduce por tanto a determinar los torques que son necesarios aplicar a las diferentes articulaciones con el objetivo de obtener los ángulos deseados. Varias aproximaciones basadas en redes neuronales han sido aplicadas a este problema por parte de varios investigadores. Como primera aproximación aparece la modelización de la dinámica inversa de forma directa, propuesta por Jordan [MIL90] en 1988. En la terminología inglesa conocida con el nombre de “direct inverse modeling”. En la figura (7.3) presentamos un diagrama de esta aproximación. En definitiva se trata de entrenar una red neuronal de tal forma que las entradas son los ángulos de las articulaciones, mientras que las salidas de la red corresponden a los torques a aplicar, siendo la señal de error utilizada para entrenar la red. Este error es la diferencia entre el torque realmente aplicado al robot y el torque que nos proporciona la red. De esta forma se podría aprender la dinámica inversa. En la figura (7.3) hemos representado la red neuronal mediante el bloque etiquetado como “modelo dinámico inverso”. Es importante destacar que este bloque en absoluto está limitado a una red neuronal, siendo posibles otras técnicas tales como la utilización de una memoria direccionada por contenido (memoria CAM) con una búsqueda del vecino más próximo, una simple tabla de búsqueda u otros métodos que sean capaces de hacer corresponder una serie de valores de entradas a una serie de valores de salidas. Atkerson y Reinkensmeyer [MIL90] utilizaron esta aproximación en 1988 utilizando una memoria CAM. Sin embargo, aunque el problema de la dinámica inversa se puede tratar mediante una red neuronal, el esquema de control

tiene la estructura del mostrado en la figura (7.1), con lo que presenta los mismos defectos.

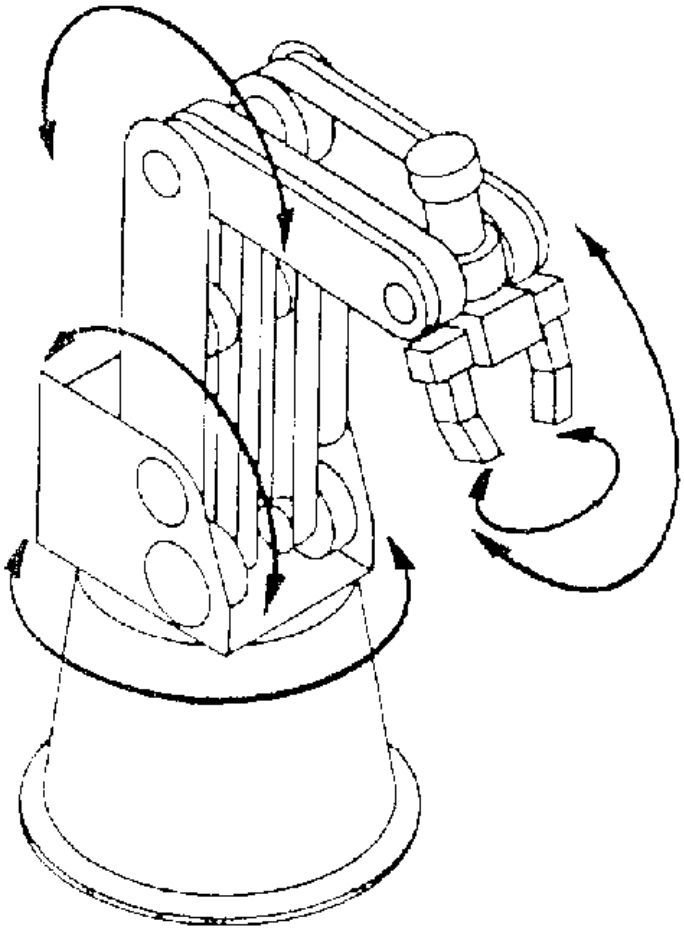


Figura 7.2 – Robot PUMA

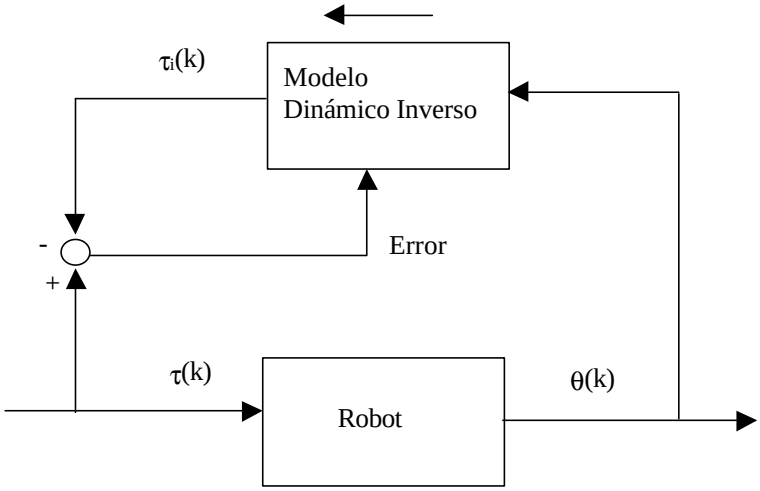


Figura 7.3- “Direct Inverse modeling”

Otra de las aproximaciones utilizadas es la que se muestra esquemáticamente en la figura (7.4). Este algoritmo es conocido con el nombre de modelización inversa y directa, o en el caso de la terminología inglesa se le conoce con el nombre de “forward and inverse modeling”. Dicha estrategia de control fue propuesta por Jordan y Rumelhart [MIL90] en 1988. Esta aproximación presenta dos etapas en su aplicación, por un lado hemos de aprender la dinámica directa. Es decir, a partir de los torques suministrados a las articulaciones del robot y los ángulos resultantes de aplicar estos torques la red neuronal debe aprender la relación entre los torques y los ángulos. Una vez la dinámica directa ha sido aprendida por la red, la cual se indica en la figura (7.4) con el bloque etiquetado como “Modelo Dinámico Directo”, estaremos en disposición de realizar el control sobre las articulaciones en una segunda etapa. Si especificamos en esta segunda etapa los ángulos deseados para las articulaciones, estos podrían introducirse en otra red cuyo objetivo es aprender la dinámica inversa, dándonos esta segunda red un valor de torque a aplicar a las articulaciones. Esta red se indica en la figura (7.4) a través del bloque etiquetado “modelo de dinámica inversa”. Una vez aplicados los torques sobre las articulaciones, éstas adquirirán nuevos ángulos. Los errores constituidos por la diferencia de los valores de estos ángulos y los ángulos deseados se introducen como entradas de la red utilizada en la primera etapa. Sin embargo en este caso esta red se toma con entradas y salidas intercambiadas, es decir, ahora las entradas de la red juegan el papel de salidas mientras que las salidas de la red juegan el papel de entradas. Considerando la red de esta forma las salidas estarían relacionadas con el error en los torques. Es decir, nos darían idea de la diferencia entre los torques aplicados y los torques que debían de haber sido aplicados para conseguir nuestro objetivo de posicionar las articulaciones en los ángulos deseados. Estas salidas de la primera red se utilizan como señal de error en el proceso de entrenamiento de la segunda red, la cual modeliza la dinámica inversa del robot.

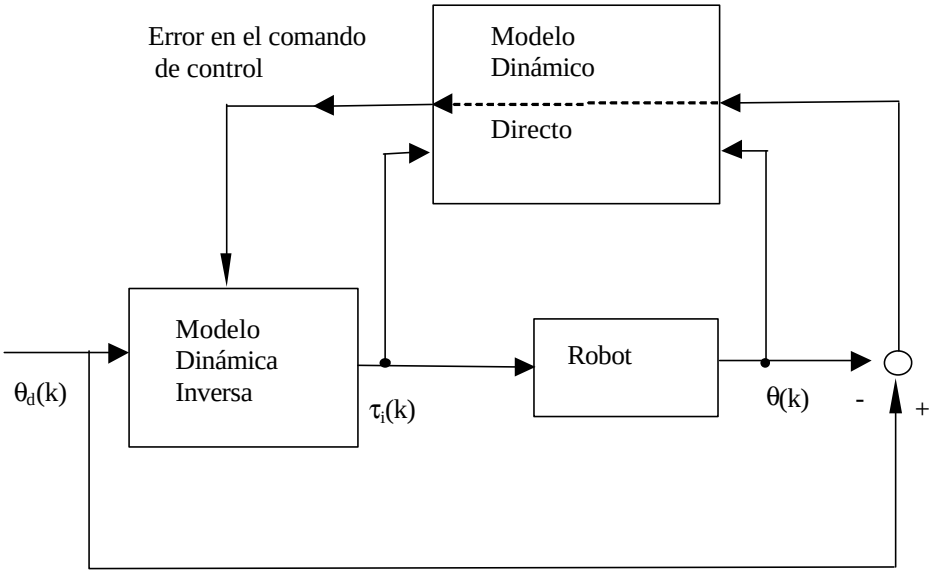


Figura 7.4 – “Forward and inverse modeling”.

Esta estrategia aunque plantea un esquema en lazo cerrado, sin embargo presenta como mayor inconveniente que separa la fase de aprendizaje de la fase de control, con lo cual la respuesta del algoritmo de control se deteriora considerablemente cuando se producen cambios dinámicos en el robot, tal es el caso cuando el robot toma una pieza por medio de la pinza o efector.

Antes de introducir la estrategia utilizada en nuestro caso es interesante presentar una última aproximación al problema de controlar un manipulador robótico. Esta estrategia se conoce con el nombre de aprendizaje de error realimentado, siendo su nombre en la terminología inglesa “feedback error learning”. Esta fue presentada por primera vez por Kawato, Furukawa y Suzuki [KAW87] [MIY88] en 1987. En la figura (7.5) mostramos un diagrama esquemático de dicha aproximación, en donde el único bloque donde se produce aprendizaje viene etiquetado como “modelo dinámico inverso”. Es importante destacar que en este caso se ha incluido un algoritmo de control realimentado standard como parte del algoritmo global de control. Concretamente en la figura (7.5) se ha introducido un control proporcional, sin embargo sería factible el introducir otros tipos de controladores. A este sistema de control realimentado se le ha introducido una red neuronal con el objeto de corregir los torques dados por el sistema de control realimentado. Es decir, los torques aplicados a las articulaciones son la suma de los torques calculados según el sistema de control realimentado y la salida de la red neuronal, representada en la figura (7.5) a través del bloque etiquetado como “modelo dinámico inverso”. Obsérvese que las entradas a dicha red son los ángulos deseados para las diferentes articulaciones. En cuanto a la señal de error utilizada en este caso para el entrenamiento de la red neuronal ha sido escogida igual al torque suministrado por el sistema de control realimentado.

En el caso mostrado en la figura (7.5) la salida del controlador proporcional. Este torque debe evolucionar hacia un valor nulo, dado que si la red neuronal aprendiese correctamente la dinámica inversa del robot los errores entre los ángulos de las articulaciones y los ángulos deseados serían nulos dando consecuentemente un valor nulo para el torque obtenido por el controlador proporcional.

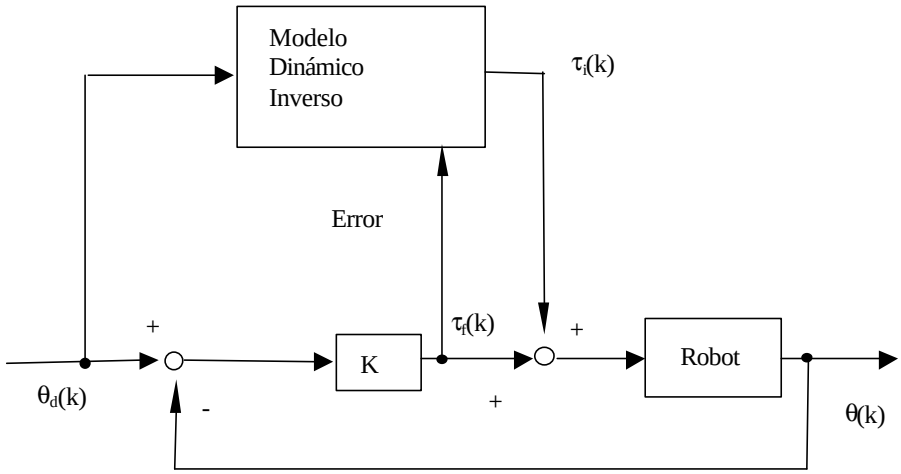


Figura 7.5 – "Feedback error learning".

Como característica más destacable se puede comentar que esta estrategia no separa la fase de aprendizaje de las redes, del control propiamente del manipulador. Es importante resaltar que esta aproximación lleva consigo la utilización de las técnicas de control tradicionales, mejorando sus prestaciones a través de la introducción de una red neuronal con el objeto de aprender el modelo dinámico inverso del robot mientras controlamos las articulaciones del robot. En este capítulo nosotros proponemos introducir las redes neuronales no para aprender la dinámica inversa del robot sino utilizarlas para sintonizar los parámetros de los controladores. Esta estrategia permitiría superar los problemas de las técnicas de control clásico en tanto en cuanto a través de las redes neuronales podríamos seleccionar estrategias de control apropiadas según la región del espacio de estados.

7.2 CONTROL DE UN ROBOT PUMA.

Con el objetivo de controlar las articulaciones de un Robot PUMA tal como el mostrado en la figura (7.2) hemos utilizado controladores basados en redes neuronales. En el algoritmo de control presentado no se realiza ninguna suposición acerca del modelo paramétrico del robot. Las redes neuronales nos proporcionan los parámetros de los controladores. Concretamente, hemos empleado un controlador proporcional derivativo (PD) y un controlador proporcional integral derivativo (PID), de tal forma que las salidas de las redes se corresponden con los parámetros del PD o PID según el caso. En la figura (7.6) se presenta un esquema de esta estrategia de control. Obsérvese que $(q_{d1}, q_{d2}, q_{d3}, q_{d4}, q_{d5})$ se refiere al vector de ángulos deseados para las respectivas articulaciones, mientras que $(q_1, q_2, q_3, q_4, q_5)$ se refiere al vector de ángulos de las articulaciones que el robot posee en el instante de tiempo considerado.

Es importante remarcar que en el esquema de control propuesto se ha supuesto un modelo descentralizado del robot. Es decir, cada torque de cada articulación se determina por un controlador diferente. Señalemos también que en nuestro caso hemos considerado el mismo tipo de controlador para todas las articulaciones. En lo que respecta a la elección de los parámetros para cada controlador ha sido realizada

considerando los ángulos y las velocidades angulares de todas las articulaciones para cada uno de ellos.

En el esquema presentado en la figura (7.6) las entradas a la red neuronal son los ángulos y las velocidades angulares de las articulaciones, mientras que los parámetros son las salidas. En el caso del controlador PD estas serían la ganancia de la acción proporcional y la ganancia de la acción derivativa, mientras que en el PID serían además de la ganancia de la acción proporcional y la ganancia de la acción derivativa, la ganancia de la acción integral.

En cuanto al algoritmo de entrenamiento utilizado para las redes ha sido el backpropagation standard. Con el objetivo de hacer más facil la exposición del algoritmo de control vamos a presentar la notación que será utilizada, así como revisar los pasos seguidos en la aplicación del algoritmo de entrenamiento backpropagation. Comencemos introduciendo la siguiente notación:

$v_{l,j}$, Salida de la neurona j-ésima en la capa l.

$w_{l,j,i}$, Peso que conecta la salida de la neurona i-ésima de la capa l-1 con la entrada j-ésima de la capa l.

x, Conjunto de entradas utilizado en la fase de entrenamiento.

$d_j(\mathbf{x})$, Valor deseado para la j-ésima salida correspondiente al conjunto de entradas **x**.

N_l, Número de neuronas en la capa *l*.

L, Número de capas.

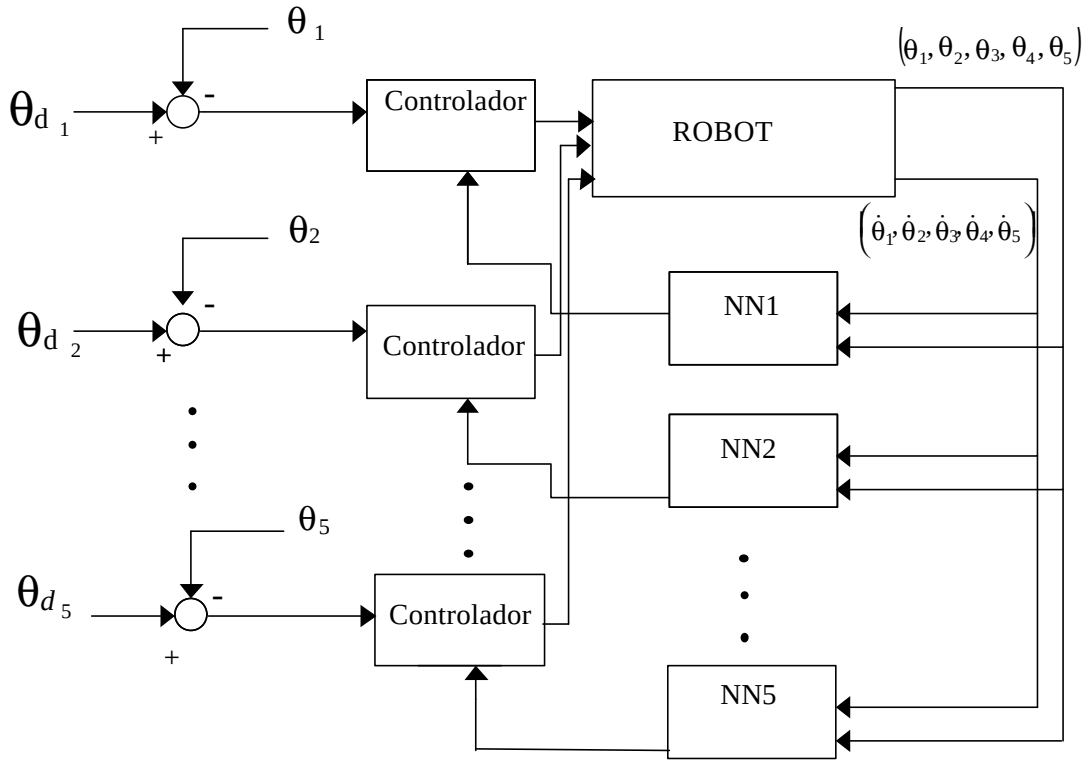


Figura 7.6 – Esquema del sistema de control.

La salida de la neurona vendrá dada por:

$$v_{l,j} = f\left(\sum_{i=0}^{N_{l-1}} w_{l,j,i} v_{l-1,i}\right) \tag{7.1}$$

Donde *f*(.) es la función sigmoide.

El objetivo del algoritmo de entrenamiento es modificar los pesos de tal forma que consigamos pesos que minimicen la siguiente función de costo:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{q=1}^{N_L} (d_q(\mathbf{x}) - v_{L,q}(\mathbf{x}))^2 \tag{7.2}$$

El backpropagation estándar establece que los pesos deben ser modificados de acuerdo a la siguiente ecuación:

$$w_{l,j,i}(k+1) = w_{l,j,i}(k) - m\frac{\partial J(\mathbf{w})}{\partial w_{l,j,i}} \tag{7.3}$$

Donde μ es la velocidad de aprendizaje. La función de sensibilidad puede ser expresada utilizando la regla de la cadena como:

$$\frac{\partial J(\mathbf{w})}{\partial w_{l,j,i}} = \frac{\partial J(\mathbf{w})}{\partial v_{l,j}} \frac{\partial v_{l,j}}{\partial w_{l,j,i}}; \tag{7.4}$$

$$\frac{\partial J(\mathbf{w})}{\partial v_{l,j}} = \sum_{m=1}^{N_{l+1}} \frac{\partial J(\mathbf{w})}{\partial v_{l+1,m}} (v_{l,j}(1 - v_{l+1,m}) w_{l+1,m,j}); \tag{7.5}$$

$$\frac{\partial v_{l,j}}{\partial w_{l,j,i}} = (v_{l,j}(1 - v_{l,j}) v_{l-1,i}); \tag{7.6}$$

Con:

$$\frac{\partial J}{\partial v_{L,j}} = v_{L,j}(\mathbf{x}) - d_j(\mathbf{x}) \tag{7.7}$$

Teniendo en cuenta las expresiones anteriores el backpropagation se reduce a realizar las siguientes operaciones:

1.- *Inicializar los pesos con valores pequeños al azar.*
2.- *Propagar la señal de error a través de la red neuronal*.
3.- *Calcular la función de sensibilidad para cada peso de la red.*
4.- *Modificar los pesos.*
5.- *Saltar al paso 2 y repetir los pasos hasta que se alcance una cierta condición de finalización.*

En el caso de un robot manipulador J(w) puede escogerse de la forma siguiente:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{q=1}^{N}(\theta_{d_q} - \theta_q)^2 \qquad (7.8)$$

Donde:

θ_{d_q} = Angulo deseado para la articulación q-ésima.

θ_q = Angulo de la articulación q-ésima.

N = Número de articulaciones del robot.

En la exposición del algoritmo de control nos centraremos únicamente en una articulación. Obsérvese que aunque se considere un único controlador la red neuronal que determina los parámetros del controlador recibe información de todas las articulaciones. Dado que asumimos para el algoritmo de control un modelo descentralizado, las expresiones para todos los controladores serán análogas.

La modificación de los pesos se realiza de acuerdo con la ecuación (7.3). Por lo que el único término que es necesario calcular para hacer efectiva la actualización de los pesos es $\partial J(k)/\partial w_{l,j,i}$. Si la salida de la red neuronal fuera directamente θ_j, esta derivada parcial podría calcularse de la siguiente forma sin más que aplicar la regla de la cadena:

$$\frac{\partial J(k)}{\partial w_{L,j,i}} = \frac{\partial J(k)}{\partial \theta_j(k)}\frac{\partial \theta_j(k)}{\partial w_{L,j,i}} = -e_j(k)\frac{\partial \theta_j(k)}{\partial w_{L,j,i}} \qquad (7.9)$$

Donde $e_j(k)$ es el error correspondiente al ángulo de la articulación j-ésima. Sin embargo en nuestro caso las salidas de la red neuronal son los m_j (los diferentes parámetros del controlador), que no aparecen de forma explícita en la función de costo. De tal forma que la función de sensibilidad viene expresada como:

$$\frac{\partial J(k)}{\partial w_{Lji}} = \frac{\partial J(k)}{\partial m_j}\frac{\partial m_j}{\partial w_{l,j,i}} \qquad (7.10)$$

Donde:

$$\frac{\partial J(k)}{\partial m_j} = -\sum_{i}^{N} e_i(k)\frac{\partial \theta_i(k)}{\partial \tau_h(k)}\frac{\partial \tau_h(k)}{\partial m_j} \qquad (7.11)$$

Obsérvese que τ_h es el torque aplicado a la articulación h-ésima. El factor $\partial\theta_i(k)/\partial\tau_h(k)$ se obtiene inmediatamente si conocemos el modelo inverso del robot. Sin embargo si queremos evitar cualquier suposición acerca del modelo podríamos obtener este término aproximando la derivada parcial por un cociente de incrementos.

$$\frac{\partial \theta_i(k)}{\partial \tau_h(k)} \approx \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} \qquad (7.12)$$

Es importante destacar que cuando las derivadas parciales de los ángulos de las articulaciones con respecto al torque aplicado sobre las mismas no se puedan calcular a través de la ecuación (7.12) debido a la aparición de problemas numéricos es posible utilizar las redes neuronales dinámicas propuestas en los capítulos anteriores. Una vez estas redes han sido entrenadas, éstas en si mismas constituyen un modelo de la dinámica directa, donde tenemos como entradas los torques aplicados y como salidas los ángulos de las articulaciones. De tal forma que a través de la propagación de la derivada parcial de las salidas hacia las entradas de modo similar a la propagación de error realizada en el algoritmo backpropagation podemos obtener las derivadas parciales de los ángulos de las articulaciones con respecto a los torques aplicados. Veamos entonces como se obtiene esta derivada para la red neuronal dinámica sin realimentación entre neuronas propuesta en los capítulos anteriores.
Obsérvese que:

$$\frac{\partial \boldsymbol{q}_i}{\partial \boldsymbol{t}_h} = \sum_{j=1}^{N_1} \frac{\partial \boldsymbol{q}_i}{\partial x_j^1} \frac{\partial x_j^1}{\partial \boldsymbol{t}_h} \tag{7.13}$$

Donde:

N_1= Número de neuronas en la primera capa.

x_j^1 = Estado de la neurona j-ésima de la primera capa.

Definiendo:

$$\boldsymbol{d}_j^k = \frac{\partial \boldsymbol{q}_i}{\partial x_j^k} \tag{7.14}$$

Y teniendo en cuenta que $\frac{\partial x_j^1}{\partial \boldsymbol{t}_h}$ es igual al peso $w_{j,h}^1$ entre la entrada h-ésima y la neurona j-ésima de la primera capa, la expresión (7.13) puede expresarse como:

$$\frac{\partial \boldsymbol{q}_i}{\partial \boldsymbol{t}_h} = \sum_{j=1}^{N_1} \partial_j^1 \, w_{j,h}^1 \tag{7.15}$$

Donde únicamente hemos de calcular ∂_j^1. Ahora si se tiene en cuenta que:

$$\begin{aligned} \boldsymbol{d}_j^k &= \sum_{p=1}^{N_{k+1}} \frac{\partial \boldsymbol{q}_i}{\partial x_p^{k+1}} \frac{\partial x_p^{k+1}}{\partial y_j^k} \frac{\partial y_j^k}{\partial x_j^k} = \sum_{p=1}^{N_{k+1}} \boldsymbol{d}_p^{k+1} \frac{\partial x_p^{k+1}}{\partial y_j^k} g'(x_j^k) = \\ &= \sum_{p=1}^{N_{k+1}} \boldsymbol{d}_p^{k+1} \frac{\partial x_p^{k+1}}{\partial n_p^{k+1}} \frac{\partial n_p^{k+1}}{\partial y_j^k} g'(x_j^k) = \sum_{p=1}^{N_{k+1}} \boldsymbol{d}_p^{k+1} \frac{\partial x_p^{k+1}}{\partial n_p^{k+1}} w_{p,j}^{k+1} g'(x_j^k) \end{aligned} \tag{7.16}$$

Donde n_p^{k+1} es la activación correspondiente a la neurona p-ésima de la capa k+1, $g'(x_j^k)$ la derivada de la función de activación, N_{k+1} el número de neuronas de la capa

k+1 y $\frac{\partial x_p^{k+1}}{\partial n_p^{k+1}}$ se puede calcular a partir de la ecuación de estado de las neuronas. Si tomamos la ecuación de estado como:

$$x_p^{k+1}(r+1) = x_p^{k+1}(r) + H\, B_p^{k+1}\left(x_p^{k+1}, b_p^{k+1}\right) + H \quad A_p^{k+1}\left(x_p^{k+1}, a_p^{k+1}\right) n_p^{k+1} \qquad (7.17)$$

Podemos obtener el valor de esta derivada resolviendo la ecuación lineal:

$$c(r) = z(r)\left[1 + H\frac{\partial \quad B_p^{k+1}\left(x_p^{k+1}, b_p^{k+1}\right)}{\partial x_p^{k+1}} + H\frac{\partial \quad A_p^{k+1}\left(x_p^{k+1}, a_p^{k+1}\right)}{\partial x_p^{k+1}} n_p^{k+1}(r)\right] \qquad (7.18a)$$

$$z(r+1) = c(r) + H \quad A_p^{k+1}\left(x_p^{k+1},\, a_p^{k+1}\right) \qquad (7.18b)$$

Siendo z(r) = $\frac{\partial x_p^{k+1}(r)}{\partial n_p^{k+1}}$ y $z(0) = 0$ en este caso. De tal forma que se consiguen los valores de δ_j^k para la capa k a partir de los de la capa k+1. Ahora considerando que:

$$\delta_p^L = \begin{cases} Si\ p = i & g\,'(x_i^L) \\ Si\ p \neq i & 0 \end{cases} \qquad (7.19)$$

Donde:

L = número de capas de la red.

Se calcula ∂_j^1 y a partir de la ecuación (7.15) $\frac{\partial q_i}{\partial t_h}$.

En cuanto al factor $\partial \tau_h(k)/\partial m_j$ puede calcularse a través de la expresión para el controlador. Si cambiásemos el controlador sólo sería necesario volver a calcular esta derivada parcial utilizando la expresión de ese nuevo controlador, mientras que el resto del algoritmo se mantiene igual.

En el caso de un PD la relación del torque con los ángulos de las articulaciones viene dada como:

$$\tau_h = m_1(\theta_{d_h} - \theta_h) - m_2\dot{\theta}_h \qquad (7.20)$$

m_1 = Ganancia de la acción proporcional.

m_2 = Ganancia de la acción derivativa.

θ_{d_h} = Angulo deseado correspondiente a la articulación h-ésima

θ_h = Angulo correspondiente a la articulación h-ésima.

Nótese que m_1 y m_2 deberían haberse denotado como m_{1h} y m_{2h} dado que los parámetros son diferentes para cada articulación, sin embargo con el objetivo de simplificar las expresiones este subíndice será suprimido en lo sucesivo.

Teniendo en cuenta la expresión para el torque en este caso, las derivadas parciales de los torques con respecto a los parámetros son:

$$\frac{\partial \tau_h}{\partial m_1} = (\theta_{dh} - \theta_h) \tag{7.21}$$

$$\frac{\partial \tau_h}{\partial m_2} = -\dot{\theta}_h \tag{7.22}$$

Por otra parte, necesitamos calcular las derivadas de la función de costo con respecto a los parámetros para obtener la función de sensibilidad correspondiente a los pesos de la última capa. Estas pueden ser calculadas usando la expresión (7.11) como:

$$\frac{\partial J(k)}{\partial m_1} = -\sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} (\theta_{dh} - \theta_h) \tag{7.23}$$

$$\frac{\partial J(k)}{\partial m_2} = \sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} \dot{\theta}_h \tag{7.24}$$

Y finalmente la función de sensibilidad $\frac{\partial J(k)}{\partial w_{L,j,i}}$ correspondiente a los pesos de la última capa es:

$$\frac{\partial J(k)}{\partial w_{L1s}} = -\sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau_h(k) - \tau_h(k-1)} (\theta_{dh} - \theta_h) \frac{\partial m_1}{\partial w_{L,1,s}} \tag{7.25}$$

$$\frac{\partial J(k)}{\partial w_{L2s}} = \sum_i^N e_i(k) \frac{\theta_i(k) - \theta_i(k-1)}{\tau(k) - \tau(k-1)} \dot{\theta}_1 \frac{\partial m_2}{\partial w_{L,2,s}} \tag{7.26}$$

Donde $\frac{\partial m_1}{\partial w_{L,1,s}}$ y $\frac{\partial m_2}{\partial w_{L,2,s}}$ se pueden expresar de la siguiente forma utilizando la expresión (7.6):

$$\frac{\partial m_1}{\partial w_{L,1,s}} = (m_1(1-m_1)v_{L-1,s}); \tag{7.27}$$

$$\frac{\partial m_2}{\partial w_{L,2,s}} = (m_2(1-m_2)v_{L-1,s}); \tag{7.28}$$

Los pesos de las neuronas de la capa de salida se pueden hallar a través de las funciones de sensibilidad dadas por las expresiones (7.25) y (7.26), donde todos los términos son conocidos. Por el contrario, la actualización de los pesos de las demás neuronas se realizaría como es habitual en las redes estáticas con el algoritmo de entrenamiento backpropagation usando las ecuaciones (7.4), (7.5) y (7.7).

Si en vez de un PD utilizamos como controlador un PID, la relación entre el torque y los ángulos vendría dada como:

$$\tau_h = m_1(\theta_{d_h} - \theta_h) - m_2\dot{\theta}_h + m_3\int_0^t(\theta_{d_h} - \theta_h)dt \qquad (7.29)$$

Donde:

m_1 = Ganancia de la acción proporcional.

m_2 = Ganancia de la acción derivativa.

m_3 = Ganancia de la acción integral.

El mismo análisis que con el controlador PD se puede hacer en este caso, considerando una salida adicional en la red neuronal, correspondiente al parámetro m_3. En este caso la red neuronal tendría 3 salidas correspondientes a los tres parámetros del controlador PID. La derivada parcial del torque con respecto a este tercer parámetro se calcula fácilmente sin más que tener en cuenta la ecuación (7.29).

Es importante remarcar que, las ecuaciones para todos los controladores que actúan sobre las articulaciones se obtienen de la misma manera.

7.3 RESULTADOS

Una vez hemos presentado el algoritmo de control, en esta sección daremos algunos de los resultados de la aplicación de este algoritmo sobre un robot Puma. Para ello hemos hecho uso de un PC Pentium en el cual se ha simulado esta política de control. Para ver la efectividad del algoritmo de control hemos realizado varios ensayos en donde como ángulos deseados para cada articulación hemos elegido una curva sinusoidal. En la figura (7.7) presentamos los resultados de uno de los entrenamientos. En esta figura podemos ver los ángulos correspondientes a la primera articulación junto con los ángulos deseados para esta articulación. Estos resultados corresponden a haber elegido controladores PD. Hemos de comentar también que las salidas de las redes neuronales han sido multiplicadas por un factor de 10 para conseguir valores mayores de los parámetros, y en cuanto a la inicialización de los pesos ésta se ha hecho como es habitual al azar dentro del rango entre –0.1 y 0.1, intentado evitar fenómenos de saturación en las neuronas. Los resultados se pueden observar que no son satisfactorios, dado que aparece un gran offset entre los ángulos deseados para la primera articulación y los ángulos reales de esta articulación.

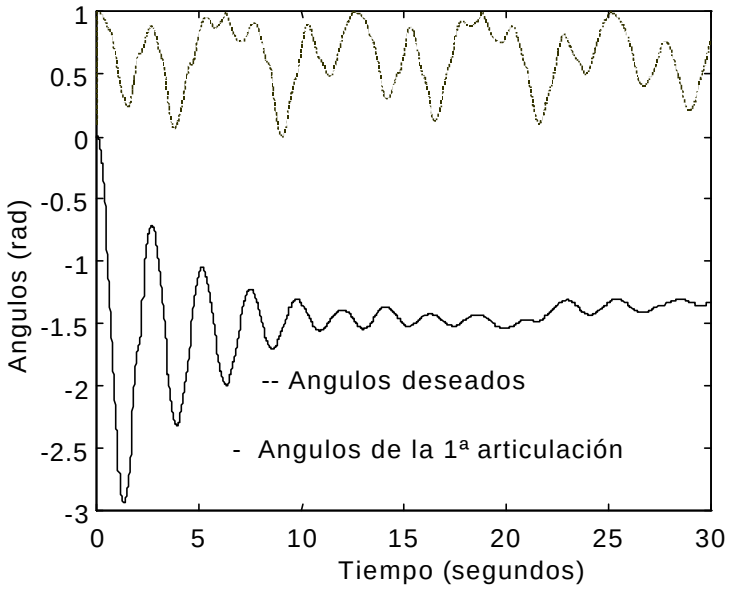


Figura 7.7 – Angulos de la 1ª articulación frente a la curva de ángulos deseados en el caso de un controlador PD.

Vista la respuesta del controlador PD, parecía oportuno considerar nuevos controladores. Aumentando en grado de complejidad los controladores consideramos el controlador PID. En la figura (7.8) presentamos uno de los resultados utilizando este tipo de controlador. De igual forma que para el PD, hemos multiplicado las salidas de la red por un factor de 10 y hemos considerado como inicialización para los pesos valores aleatorios entre –0.1 y 0.1. Sin embargo, al introducir un nuevo parámetro(la ganancia de la acción integral) hemos tomado 3 salidas para las redes neuronales. Para el caso de la salida correspondiente a la ganancia de la acción integral hemos utilizado otro factor de escala, es decir, en este caso la salida ha sido multiplicada por 0.1, para ajustar las salidas al rango de valores conveniente para este parámetro. Como se puede observar en la figura (7.8) los resultados son mejores que para el PD eliminando el offset que aparecía en aquel caso. Se puede apreciar que los ángulos de la 1ª articulación siguen a los ángulos deseados después de transcurridos algunos segundos, tiempo éste utilizado por la red para hacer converger los valores de los parámetros a valores convenientes. La evolución de los parámetros se muestra en la figura (7.9) en donde se puede apreciar claramente este hecho. Es también importante resaltar que una de las dificultades más significativas desde el punto de vista de la implementación es la respuesta del robot en el primer intervalo de tiempo, en donde los ángulos toman valores arbitrarios, llevándonos a movimientos erráticos de las articulaciones. Este problema proviene de la inicialización al azar de los pesos, que nos lleva inevitablemente a valores arbitrarios de los parámetros situándose éstos lejos de los valores deseados. Una solución a este problema es realizar una elección más conveniente de los pesos iniciales. Una posibilidad sería el tomar los valores de los pesos iniciales igual a los correspondientes a un ensayo anterior.

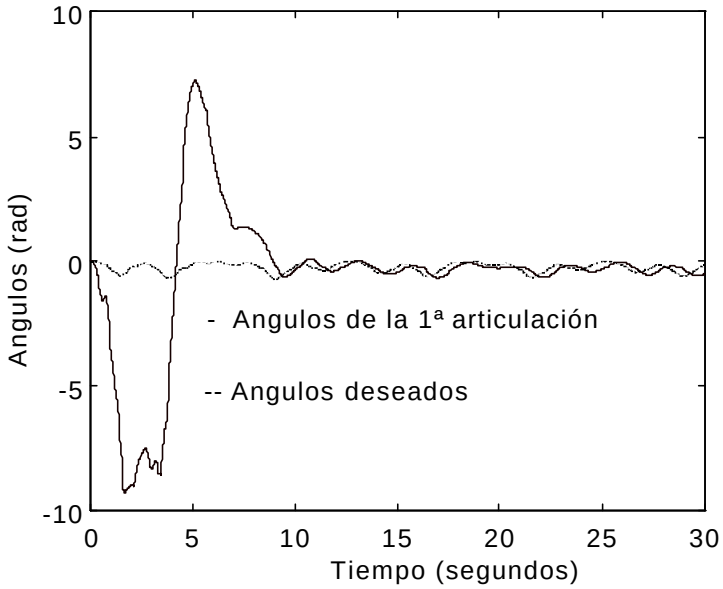


Figura 7.8 - Angulos de la 1ª articulación frente a la curva de ángulos deseados para un PID.

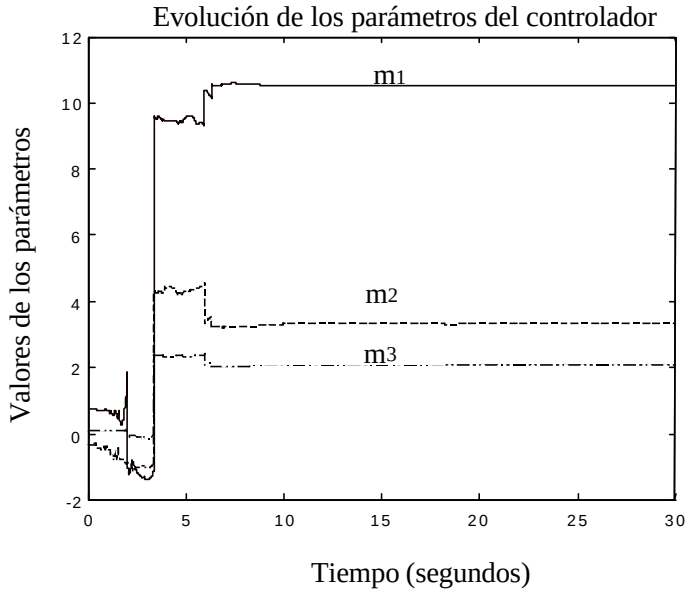


Figura 7.9 – Evolución de los parámetros para un controlador PID.

Capítulo 8
APLICACIÓN DE UN SISTEMA DE CONTROL NEURO-FUZZY A UN ROBOT MÓVIL

8

APLICACIÓN DE UN SISTEMA DE CONTROL NEURO-FUZZY A UN ROBOT MÓVIL.

En este capítulo introduciremos el problema del control de un robot móvil. Presentaremos primeramente dos robots móviles desarrollados en el grupo de Computadoras y Control de la Universidad de La Laguna los cuales se les ha aplicado un control basado en lógica difusa. Describiremos con cierto detalle esta estrategia de control para posteriormente presentar otra estrategia de control basada en un sistema Neuro-Fuzzy. Describiremos la estructura del sistema Neuro-Fuzzy, así como el método de entrenamiento utilizado, para finalmente aplicarlo en simulación para dirigir un robot móvil en un espacio de trabajo con obstáculos.

8.1 INTRODUCCION

En este capítulo nos centraremos en los robots móviles atendiendo a las estrategias diseñadas para controlar su movimiento. En general un robot móvil se puede definir como cualquier estructura mecánica con capacidad de movimiento absoluta respecto al medio y con capacidad para interactuar con ese medio. En la figura 8.1 presentamos un esquema general en donde se ilustra la estructura de control de un robot móvil típico.
El esquema presentado en la figura (8.1) está constituido por cinco módulos, cada uno de los cuales tiene asociado ciertas tareas específicas.

El módulo gestor del sistema sensorial se encarga de la gestión y control de los sensores, siendo el módulo de navegación quien le indica los sensores que deben usarse.

El módulo actualizador del modelo del entorno y posición está constituido por un modelo del entorno susceptible de ser variado si el robot se mueve en un entorno variable. Este módulo tiene como misión el averiguar la posición y orientación del robot en todo momento.

El módulo Planificador de trayectorias tiene como objetivo fundamental determinar trayectorias libres de colisión, resolviendo este problema a partir de la aplicación de algoritmos de búsqueda en grafos ó mediante una función de costo que pese factores tales como el gasto energético para ir de un punto a otro de la zona de trabajo.

El módulo de Navegación por su parte genera las primitivas de guiado que después realizará el módulo piloto.

Y por último el módulo piloto dará los valores de referencia a los controladores de los motores, teniendo también en cuenta las características mecánicas del movimiento del robot, tales como el número de motores, tipo, distribución, etc.

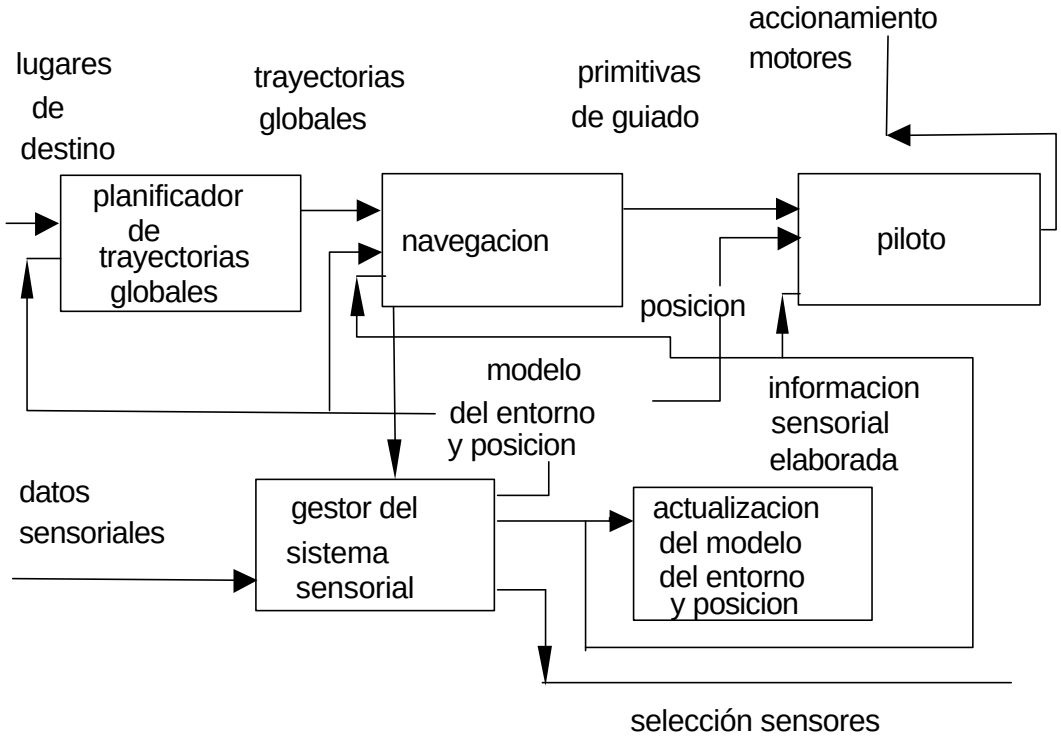


Figura 8.1 – Esquema general de la estructura de control de un robot móvil.

En este capítulo nos centraremos en el módulo de navegación puesto que es en éste en donde se generan las políticas de control para ser aplicadas directamente a los motores a través del módulo piloto, sin menoscabo de que otros módulos puedan influir de forma indirecta en la política de control a seguir.

8.2 ROBOT MÓVIL TÍPICO.

Una de las políticas de control más comunes es la utilización de lógica difusa para que un robot móvil pueda moverse dentro de la zona de trabajo, en la cual encuentra obstáculos a su paso. Con el objeto de ilustrar esta técnica, vamos a exponer estos conceptos en relación a un proyecto de robot autónomo desarrollado en la Universidad de La Laguna por el grupo de Computadoras y Control.

En nuestro grupo se han desarrollado dos prototipos denominados RAP y Sultán I. En la figura (8.2) mostramos una fotografía de éstos, donde el mayor es RAP, constituido por una placa base de PC basada en una unidad central de proceso Intel 286, mientras que el más pequeño corresponde a Sultán I, que utiliza una tarjeta Handy board basada en un procesador 68HC11. Es importante destacar que mientras para RAP se mueve mediante ruedas, el sistema de tracción utilizado por Sultán I consiste en la utilización de correas que unen las ruedas del mismo. Este último sistema le confiere un mayor grado de movilidad, permitiéndole escapar de ciertas situaciones en las cuales con únicamente el sistema de tracción de ruedas el robot quedaría atrapado.

En las figuras (8.3) y (8.4) mostramos un esquema general del prototipo Sultán I en donde se pueden apreciar los diferentes módulos del mismo. En el apéndice A mostramos las especificaciones técnicas de cada uno de los robots móviles. Sin embargo desde el punto de vista de la aplicación de la estrategia de control basada en lógica difusa, hemos de decir que los robots están dotados de sensores ultrasónicos con poca resolución direccional que únicamente permiten el detectar obstáculos que se le presenten en un entorno local al robot, mientras que para determinar la posición y orientación del robot desde un punto de vista global, dado que el objetivo final es llevar el robot a una cierta posición, se dispone de un sistema de posicionamiento global basado en una brújula electrónica.

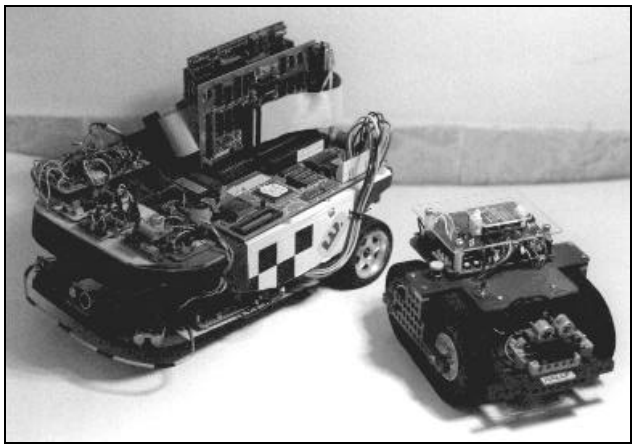


Figura 8.2. – Fotografía de los dos prototipos.

En cuanto a la aplicación de comandos con el objeto de conseguir mover el robot con la dirección y velocidades adecuadas se dispone en ambos casos de dos servomotores laterales, tal que proporcionando comandos a cada uno de ellos se puede controlar el movimiento del robot.

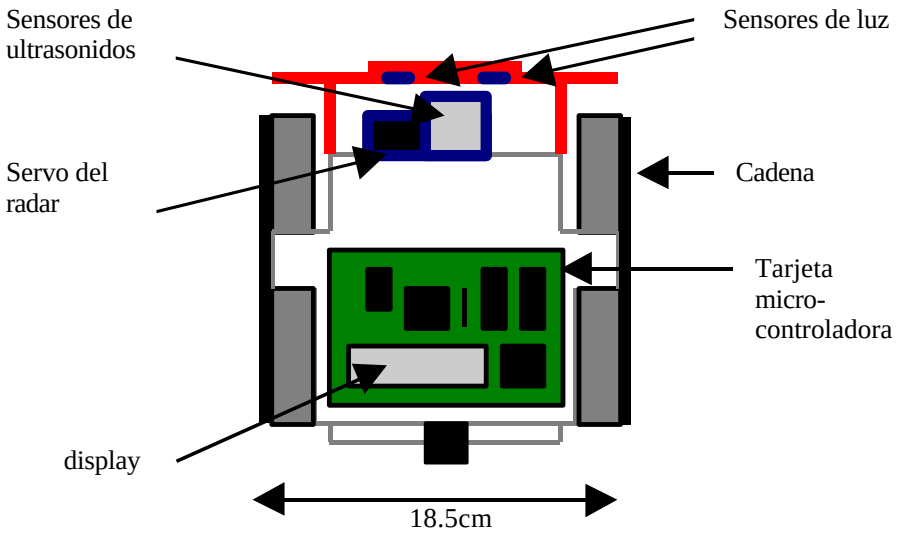


Figura 8.3 : Vista Superior.

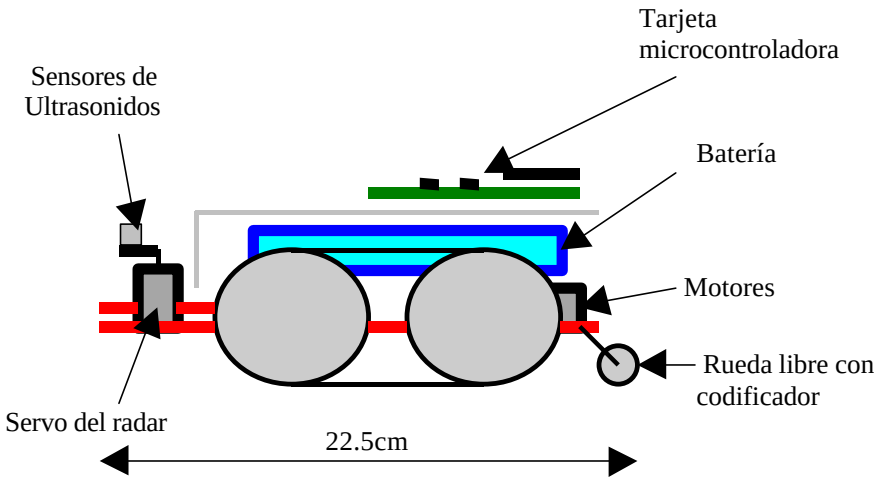


Figura 8.4 : Vista lateral

Por lo tanto desde el punto de vista del control hemos de considerar las variables mostradas en la figura (8.5). Es decir, se considerarán las velocidades de avance del motor izquierdo y derecho indicadas en la figura (8.5) como *v_izquierda* y *v_derecha* , así como el ángulo entre el eje vertical de simetría del robot y la línea recta que une el punto medio del robot y la posición destino del mismo tal como se indica en la figura (8.5). Esta última variable nos da cuenta de la orientación global del robot, entendiendo por orientación global la orientación del robot respecto a la posición destino. Finalmente, a través de los sensores ultrasónicos podemos analizar la distancia del robot con respecto a los posibles obstáculos. En la figura (8.5) hemos destacado que tomaremos medidas de las distancias a través de tres sensores, localizados respectivamente en el centro, parte izquierda y parte derecha de la parte frontal, indicadas en la figura (8.5) como *d_izquierda*, *d_frente* y *d_derecha*.

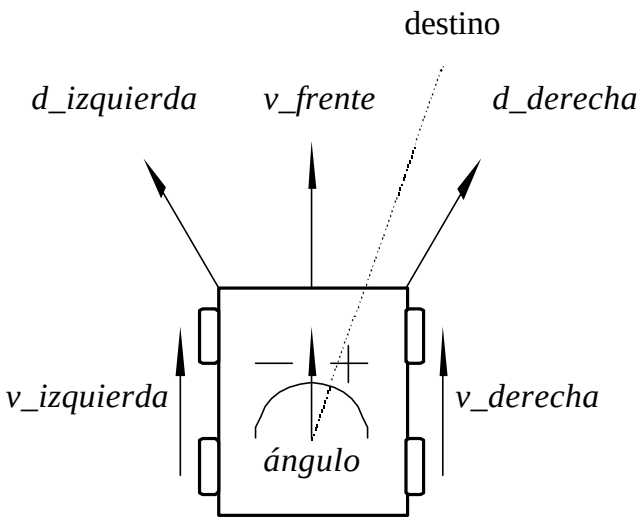


Figura 8.5 – Esquema del robot móvil.

8.3 ESTRATEGIA DE CONTROL DIFUSO CLÁSICA

Parece oportuno reseñar al comienzo de esta sección que existen aproximaciones clásicas aplicadas al movimiento de un robot móvil autónomo basadas en la teoría de campos de potencial, cuya forma de operar consiste en asociar a cada obstáculo que se le presente al robot un valor de energía potencial elevado, mientras que a la posición destino se le asocia un valor de energía potencial muy bajo, de tal forma que el destino actúa como una fuerza de atracción mientras que los objetos actúan como una fuerza de repulsión. Aunque desde el punto de vista teórico parece que esta teoría es coherente, sin embargo en la práctica se encuentra con algunos inconvenientes, dirigiéndose el robot muchas veces a puntos donde la energía potencial constituye únicamente un mínimo local. Estos inconvenientes en la aplicación de la teoría de campos nos llevan pues a plantearnos otras soluciones. Una de las soluciones que más ha sido empleada durante los últimos años es la incorporación de políticas de control difusas [PED93] [PAL96]. En esta sección intentaremos dar las claves de éstas utilizando para ello el esquema de robot móvil presentado en la figura (8.5) correspondiente a los prototipos desarrollados en la universidad de La Laguna.

Como primer paso de cara a la aplicación de un control difuso hemos de tener en cuenta las variables de interés desde el punto de vista del control y fuzzificarlas, es decir, definir una serie de conjuntos difusos a los cuales cada una de las variables puede pertenecer con un cierto grado de pertenencia. Las variables se muestran en la figura (8.5). Estas pueden pertenecer a los siguiente conjuntos difusos:

Para las velocidades de avance: Lenta, media y rápida
Para las distancias: Lejos, media y cerca.
Para el ángulo: positivo, cero y negativo (según se encuentre a la izquierda, en frente o a la derecha del destino)

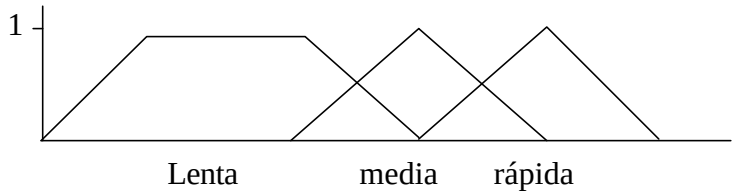


Figura 8.6 – Conjunto difusos

En la figura (8.6) mostramos como se han tomado las funciones de pertenencia. Aunque en la figura (8.6) indicamos las funciones de pertenencia para el caso de las velocidades de avance, para los conjuntos difusos de las demás variables también se han escogido las funciones de pertenencia en forma triangular o trapezoidal.

Obsérvese que el nombre de los conjuntos hace referencia a un rango de valores para cada una de las variables. Es esta una de las características fundamentales de la lógica difusa, en tanto en cuanto en lógica difusa no se hace mención a valores exactos de las variables si no a que estos valores se encuentran dentro de un rango con una cierta probabilidad.

Aparte de la definición de los conjuntos difusos es necesario definir una serie de reglas difusas que permitan al robot autónomo esquivar cualquier obstáculo.

En el caso de que en el camino no se le presenten obstáculos las reglas tienden a hacer que el robot avance directamente hacia el destino, corrigiendo la dirección si ésta no fuese correcta. En la tabla (8.1) presentamos las reglas que habrían que tenerse en cuenta, mientras que en la figura (8.7) ilustramos el movimiento resultante en este caso.

Cuando el robot detecta algún obstáculo cercano debe reducir la velocidad para evitarlo. Las reglas que tienen en cuenta esta situación y la ilustración de la misma se muestran en la tabla (8.2) y la figura (8.7) respectivamente.

1) Si *d_izq* es *lejos* y *d_frente* es *lejos* y *d_dcha* es *lejos* y *ángulo* es *cero* entonces *v_izq* es *rápida* y *v_dcha* es *rápida.*
2) Si *d_izq* es *lejos* y *d_frente* es *lejos* y *d_dcha* es *lejos* y *ángulo* es *negativo* entonces *v_izq* es *lenta* y *v_dcha* es *rápida.*
3) Si *d_izq* es *lejos* y *d_frente* es *lejos* y *d_dcha* es *lejos* y *ángulo* es *positivo* entonces *v_izq* es *rápida* y *v_dcha* es *lenta.*

Tabla 8.1 – Reglas en el caso de no encontrarse ningún obstáculo

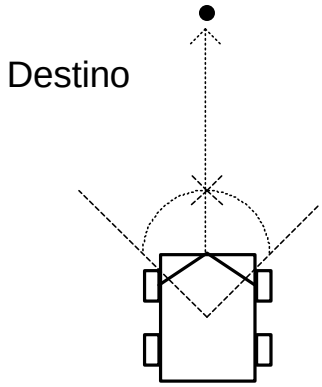


Figura 8.7 – Robot avanzando en línea recta hacia el destino

4) Si *d_izq* es *media* y *d_frente* es *cerca* y *d_dcha* es *cerca* entonces *v_izq* es *lenta* y *v_dcha* es *rápida.*
5) Si *d_izq* es *cerca* y *d_frente* es *cerca* y *d_dcha* es *media* entonces *v_izq* es *rápida* y *v_dcha* es *lenta.*

Tabla 8.2 – Evitando obstáculos cercanos.

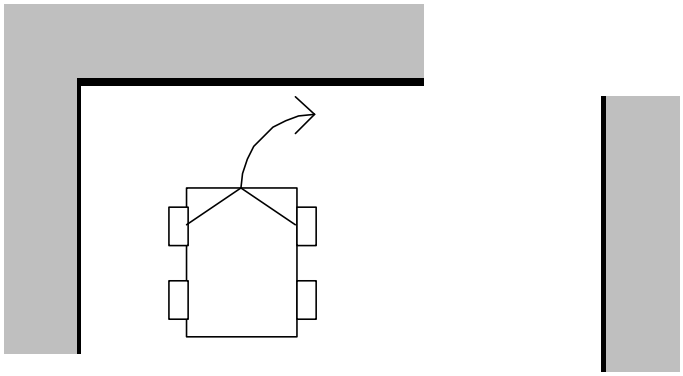


Figura 8.8 – Robot evitando obstáculo cercano.

Otra situación a considerar es cuando al robot se le presentan obstáculos cercanos a los dos lados del mismo. Este caso se resuelve mediante la regla que se presenta en la tabla 8.3, mientras que la ilustración de tal situación se hace en la figura (8.9).

6) Si d_izq es cerca y d_frente es media y d_dcha es cerca entonces *v_izq* es *media* y *v_dcha* es *media.*

Tabla 8.3 – Regla difusa correspondiente a obstáculos laterales cercanos.

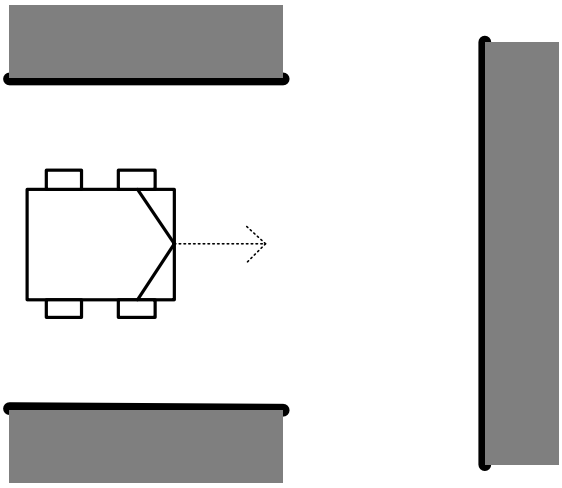


Figura 8.9 – obstáculos laterales.

Cuando al robot se le presentan obstáculos en todas direcciones, la acción que el robot realiza es escapar de esa zona, por ejemplo mediante un giro hacia la derecha. Las reglas mostradas en la tabla 8.4 tratan este caso, siendo ilustrado en la figura (8.10).

7) Si d_izq es cerca y d_frente es cerca y d_dcha es cerca entonces *v_izq* es *rápida* y *v_dcha* es *lenta.*
8) Si *d_izq* es *media* y *d_frente* es *cerca* y *d_dcha* es *media* entonces *v_izq* es *rápida* y *v_dcha* es *lenta.*

Tabla 8.4 – Reglas para el caso de obstáculos en todas direcciones.

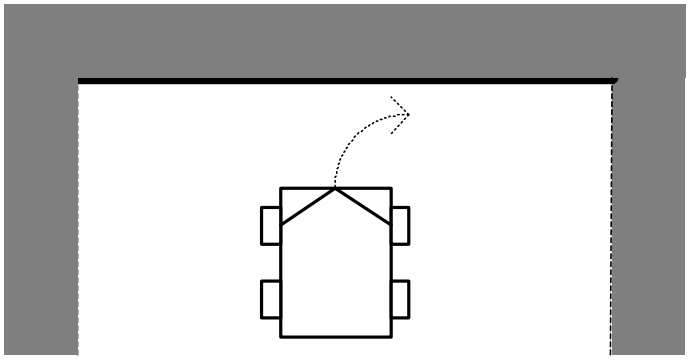


Figura 8.10 – Robot ante un obstáculo sin salida.

En los últimos casos que hemos mostrado únicamente nos hemos concentrado en esquivar los obstáculos suponiendo que el destino se encuentra a una cierta distancia delante del robot, sin embargo hemos de añadir nuevas reglas que tengan en cuenta otras formas de esquivar los obstáculos dirigiéndose finalmente a la posición destino.

Una situación de este estilo ocurre cuando el robot se halla en una habitación cerca de una pared que posee una puerta y desea salir de la misma para dirigirse a un destino que se encuentra fuera de la habitación. El robot en este caso seguiría la dirección paralela a la pared para finalmente al llegar a la puerta girar hacia la posición de destino. Esta situación se representa en la figura (8.11) y las reglas difusas correspondientes se muestran en la tabla (8.5).

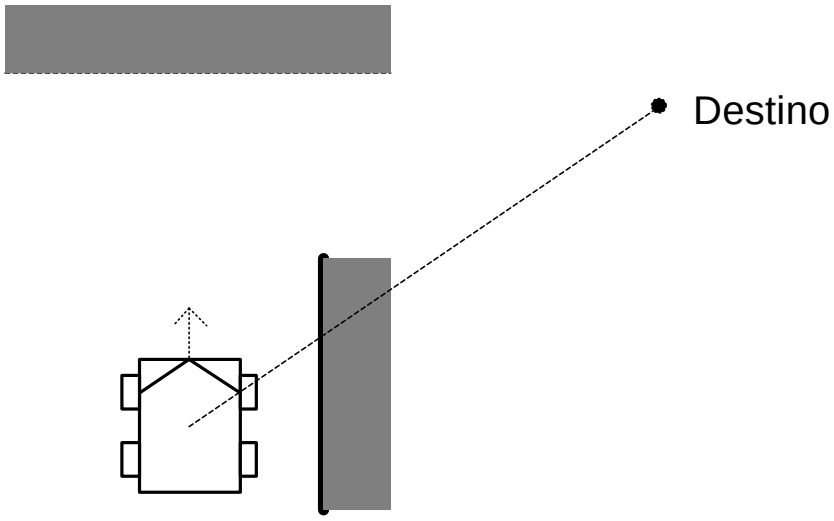


Figura 8.11 – Esquivando obstáculo en forma de pared con puerta teniendo en cuenta el punto de destino.

9) Si *d_izq* es *lejos* y *d_frente* es *lejos* y *d_dcha* es *cerca* y *ángulo* es *positivo* entonces *v_izq* es *media* y *v_dcha* es *media.*
10) Si *d_izq* es *cerca* y *d_frente* es *lejos* y *d_dcha* es *lejos* y *ángulo* es *negativo* entonces *v_izq* es *media* y *v_dcha* es *media.*
11) Si *d_izq* es *lejos* y *d_frente* es *medio* y *d_dcha* es *cerca* y *ángulo* es *positivo* entonces *v_izq* es *media* y *v_dcha* es *media.*
12) Si *d_izq* es *cerca* y *d_frente* es *media* y *d_dcha* es *lejos* y *ángulo* es *negativo* entonces *v_izq* es *media* y *v_dcha* es *media.*

Tabla (8.5) – Reglas saliendo a través del desplazamiento paralelo al obstáculo.

Debemos comentar que esta última situación no sería resuelta por una aproximación clásica puesto que el robot quedaría atrapado en un mínimo local de potencial. En la figura (8.12) presentamos una situación más difícil que se resolvería a través de la aplicación de las reglas difusas ya mostradas. Con objeto de simplificar el análisis únicamente hemos permitido que en cada momento se aplique sólo una de las reglas difusas. En la figura (8.12) éstas se indican por el número correspondiente. Sin embargo, hemos de tener presente que se aplican simultáneamente varias reglas. La situación presentada en la figura (8.12) tiene especial interés dado que mediante la aplicación de la teoría del campo de fuerzas el robot no sería capaz de alcanzar el destino dado que como en el caso mostrado en la figura (8.11) el robot quedaría atrapado en un mínimo de potencial, concretamente dentro del obstáculo en forma de U.

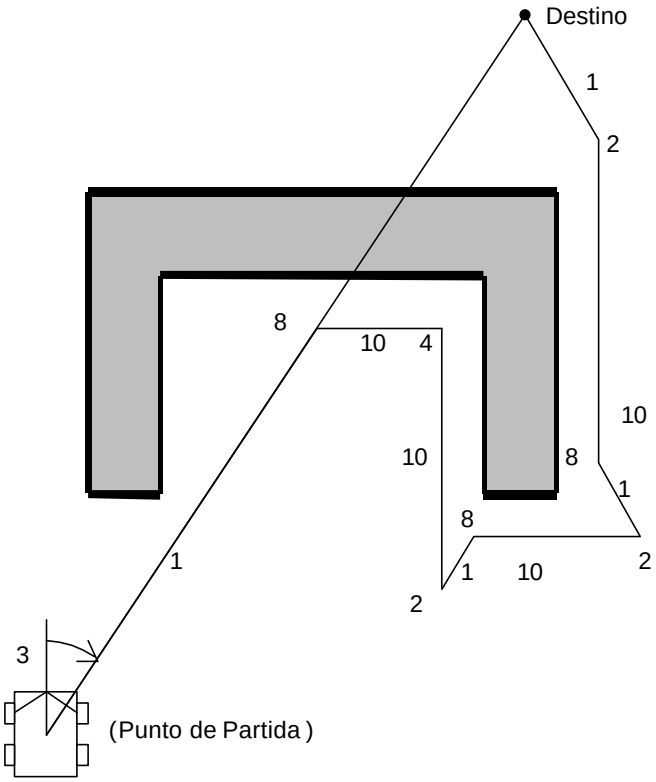


Figura 8.12 – Robot móvil esquivando un obstáculo en forma de U

8.4 ESTRATEGIA NEURO-FUZZY

Una de las dificultades de cara a aplicar la estrategia difusa es la obtención de las reglas difusas o Fuzzy y las funciones de pertenencia. En esta sección presentamos una estrategia Neuro-Fuzzy que permite la extracción automática de esas reglas y las funciones de pertenencia. Utilizaremos las redes neuronales como herramienta para generar de forma automática un sistema difuso o Fuzzy de prestaciones similares a las del sistema Fuzzy presentado en la sección anterior. En este caso el operador nos proporcionaría trayectorias para el robot móvil que esquiven con éxito los obstáculos encontrados a su paso y finalmente permitan al robot llegar a su destino. Estas trayectorias serían suministradas en base a la experiencia del operador, sin embargo el suministrar esa información se presenta como una tarea mucho más fácil que el definir las reglas difusas y las funciones de pertenencia. En las siguientes secciones mostraremos la estructura de este sistema Neuro-Fuzzy, así como el algoritmo de entrenamiento necesario para la extracción automática de las reglas y funciones de pertenencia.

8.4.1 ESTRUCTURA DEL SISTEMA NEURO-FUZZY.

El sistema Neuro-Fuzzy [LIN94] se podría ver como una red neuronal de 3 capas. Los nodos de dicha red se encuentran agrupados por capas, y la estructura global de dicha

red corresponde a la de una red feedforward. En la figura (8.13) presentamos un diagrama de la estructura del sistema Neuro-Fuzzy.

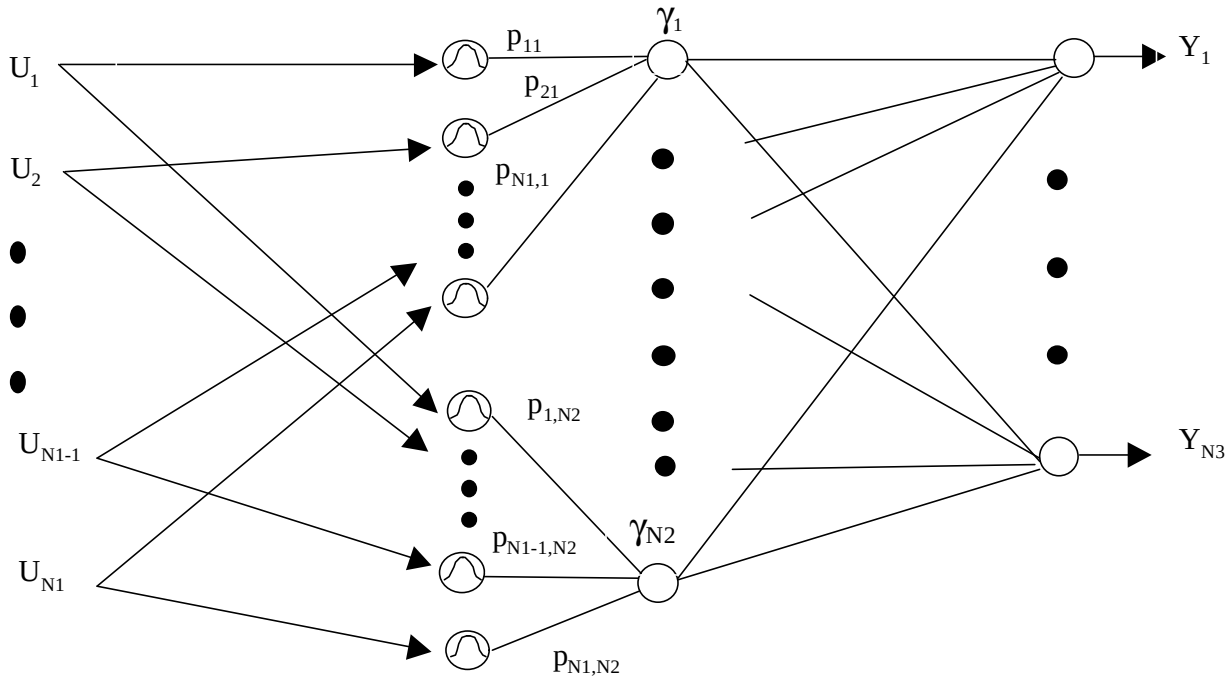


Figura 8.13 – Estructura del Sistema Neuro-Fuzzy

La primera capa o capa de entrada está compuesta por varias neuronas, cada una de las cuales corresponde a una neurona de base radial. Las entradas a las neuronas de base radial son las entradas al sistema Neuro-Fuzzy, mientras que las salidas vienen dadas como:

$$p_{ij} = \exp\left(-\frac{\left(U_i - m_{ij}\right)^2}{s_{ij}^{\,2}} \right) \qquad \begin{matrix} \mathrm{j} = 1,2,...,\mathrm{N2} \\ \mathrm{i} = 1,2,...,\mathrm{N1} \end{matrix} \qquad (8.1)$$

Donde:

m_{ij} = Centro de función de pertenencia correspondiente a la entrada i-ésima y la neurona j-ésima de la capa oculta.

U_i = Entrada i-ésima del Sistema Neuro-Fuzzy.

s_{ij} = Anchura de la función de pertenencia correspondiente a la entrada i-ésima y la neurona j-ésima de la capa oculta.

p_{ij} = Salida de la neurona de Base Radial (o grado de pertenencia para la entrada i-ésima correspondiente a la neurona j-ésima).

N2 = Número de Neuronas en la capa oculta.

N1 = Número de entradas del sistema Neuro-Fuzzy.

Nótese que las funciones de pertenencia se han tomado como gaussianas. Por otra parte, las salidas de los nodos de la capa oculta se calculan como:

$$g_j = min\left\lfloor p_{1j}, p_{2j}, ..., p_{ij}, ..., p_{N1j} \right\rfloor \quad j = 1, 2, ..., N2 \qquad (2)$$

Donde:

g_j = Salida del nodo j-ésimo de la capa oculta.

Finalmente, la capa de salida podría considerarse como una capa de neuronas lineales, donde los pesos que conectan la capa oculta y la capa de salida son los valores estimados de las salidas. Las salidas de estos nodos se calculan mediante la expresión:

$$Y_k = \frac{\sum_j sv_{jk}\, g_j}{\sum_j g_j} \qquad \begin{matrix} \mathrm{j} = 1,2,..., \mathrm{N2} \\ \mathrm{k} = 1,2,..., \mathrm{N3} \end{matrix} \qquad (8.3)$$

Donde:

Y_k = k-ésima salida del sistema Neuro-Fuzzy.

sv_{jk} = Valor estimado de la k-ésima salida proporcionado por el nodo j-ésimo de la capa oculta.

N3= Número de salidas del sistema Neuro-Fuzzy.

Por lo tanto, se puede decir que la capa de salida es la encargada de realizar el proceso de defuzzificación, proporcionando valores exactos para las salidas.

En resumen, la estructura del sistema Neuro-Fuzzy se podría ver como una red neuronal de Base Radial típica, donde se ha insertado una capa adicional entre la capa de neuronas de Base Radial (La capa de entrada) y la capa lineal (La capa de salida). Las neuronas de esta capa adicional se encargan de calcular los grados de pertenencia correspondientes a las diferentes reglas, es decir, aplican el operador difuso min, siendo N2 el número total de reglas difusas. Una vez estos cálculos se han realizado, la capa de salida aplica un proceso de defuzzificación para obtener valores numéricos exactos para las variables de salida del sistema. En el caso tratado en la sección anterior las velocidades de avance de los motores.

En el sistema Neuro-Fuzzy hay algunos parámetros que determinan la relación entre las entradas y las salidas. En este caso, el comportamiento del sistema Neuro-Fuzzy depende de los siguientes parámetros: Los centros de las funciones de pertenencia, las anchuras de las funciones de pertenencia y los valores estimados de las salidas. Para determinar estos parámetros hemos de emplear un algoritmo de aprendizaje sobre la red.

8.4.2 ALGORITMO DE APRENDIZAJE.

El algoritmo de aprendizaje se divide en tres fases. La primera y la segunda se concentran en la obtención de valores para algunos de los parámetros y la optimización del número de neuronas en la capa oculta o desde otro punto de vista el número de reglas difusas. La última fase por su parte se centra en el refinamiento de los diferentes parámetros del sistema Neuro-Fuzzy tomando como valores iniciales aquellos dados por las fases de entrenamiento previamente realizadas.

8.4.2.1 PRIMERA FASE DE ENTRENAMIENTO.

Esta primera fase consiste en la obtención de valores adecuados para los centros de las funciones de pertenencia y los valores estimados de las salidas. Estos valores se refinan en la tercera fase del entrenamiento, en esta fase únicamente daremos valores iniciales. Para obtener estos valores iniciales hemos empleado el algoritmo de mapa de características auto-organizativo de Kohonen [KOH87]. El vector de pesos inicial del mapa auto-organizativo se selecciona con valores al azar, siendo la dimensión del vector de pesos igual al número de entradas (N1) más el número de salidas (N3), de tal forma que:

$$W_j = (w_{1j}, w_{2j}, \ldots, w_{N1j}, w_{N1+1j}, \ldots, w_{N1+N3j}) \qquad j = 1, 2, \ldots, N2 \quad (8.4)$$

Las entradas al mapa auto-organizativo se toman como:

$$V = (U_1, U_2, \ldots, U_{N1}, Y_{D1}, \ldots, Y_{DN3}) \quad (8.5)$$

Donde:

$(U_1, U_2, \ldots, U_{N1})$ = Vector de entrada al Sistema Neuro-Fuzzy

$(Y_{D1}, Y_{D2}, \ldots, Y_{DN1})$ = Vector de salidas deseadas.

Ahora para modificar los pesos de la red según el algoritmo de mapa de características auto-organizativo de Kohonen, hemos de calcular el nodo "ganador". Dicho nodo obedece a la siguiente ecuación:

$$\| W_x - V \|^2 = \min_j \| W_j - V \|^2 \qquad \begin{matrix} j = 1, 2, \ldots, N2 \\ x = Nodo\ "ganador" \end{matrix} \quad (8.6)$$

En nuestro caso hemos utilizado un mapa auto-organizativo unidimensional, es decir, sólo se considera una capa de nodos. Nótese que el número de vectores de pesos se toma igual al número de neuronas en la capa oculta del sistema Neuro-Fuzzy. Una vez que se ha determinado el nodo ganador los nuevos pesos pueden calcularse mediante la siguiente ecuación:

$$W_j(t+1) = W_j(t) + \left(\frac{mc - t}{mc} \right) lr0 \exp\left(-\frac{(j-s)^2}{\sigma^2} \right) (V - W_j(t)) \qquad \begin{matrix} j = 1, 2, \ldots, N2 \\ t = 1, 2, \ldots, mc \end{matrix} \quad (7)$$

Donde:

σ = Varianza.

$W_j(t)$ = Vector de pesos j-ésimo en el instante t.

x = Nodo "ganador"

lr0 = Velocidad de aprendizaje inicial.

mc = Número de etapas de entrenamiento.

Obsérvese que cada vez que modificamos los pesos consideramos un nuevo par de entrada-salida deseada. Después que hemos tomado mc pares de entrada-salida deseada obtendremos los pesos finales. Parecería lógico suponer que el siguiente paso correspondería a asignar estos pesos a los valores de los centros de funciones de pertenencia y a los valores estimados de las salidas. Cada vector de pesos podría asociarse con cada neurona de la capa oculta del sistema Neuro-Fuzzy, tal que las funciones de pertenencia y los valores de las salidas estimadas conectados con una neurona particular de la capa oculta sean determinados por su correspondiente vector de pesos. Sin embargo, las neuronas en la capa oculta del sistema Neuro-Fuzzy están ligadas directamente al número de reglas difusas, por lo que parece más apropiado previamente a la asignación de valores a las funciones de pertenencia y a los valores estimados de las salidas optimizar el número de neuronas en la capa oculta, dado que esto significa quedarnos con un número mínimo de reglas difusas para resolver el problema evitando en lo posible cualquier clase de redundancia. Por eso, antes de realizar este proceso de asignación trataremos en una segunda fase del entrenamiento de optimizar el número de nodos en la capa oculta.

8.4.2.2 SEGUNDA FASE DE ENTRENAMIENTO.

De la primera fase de entrenamiento podemos deducir qué reglas difusas similares se representan por neuronas de la capa oculta que se encuentran cercanas, apoyados en el hecho de que se aplicó el algoritmo de mapa de características auto-organizativo de Kohonen. Como decíamos al final de la sección anterior el propósito de esta fase es obtener un número mínimo de reglas difusas de tal forma que podamos conseguir una mejor capacidad de generalización.

Las distancias Euclídeas entre dos vectores de pesos correspondientes a dos neuronas de la capa oculta están directamente relacionadas con el grado de similitud entre las respectivas reglas implementadas por esas neuronas. Partiendo de este principio esta segunda fase de entrenamiento se basa en la determinación de estas distancias.

Teniendo en cuenta estas ideas el algoritmo presentado en esta sección se puede resumir como sigue:

1- Evaluar las distancias Euclídeas entre los vectores de pesos de las neuronas adyacentes de la capa oculta.

$$d_{j,j+1} = \left\| W_j - W_{j+1} \right\| \qquad j = 1, 2, \ldots, N2 \tag{8.8}$$

2- Si la distancia $d_{j,j+1}$ es menor que un cierto valor ε, las dos neuronas se unen en una, o desde el punto de vista del sistema difuso al que tratamos de llegar, las reglas difusas correspondientes a esas neuronas se combinan en una. Esta combinación de reglas se traduce desde el punto de vista de la estructura del sistema Neuro-Fuzzy en la realización de las siguientes operaciones:

2.1 Hemos de asignar un nuevo valor al vector de pesos correspondiente al nodo j-ésimo de la capa oculta:

$$W_j = \frac{W_j + W_{j+1}}{2} \qquad (8.9)$$

2.2 El número de nodos en la capa oculta se disminuye en una unidad como resultado de la combinación:

$$N2 = N2 - 1 \qquad (8.10)$$

2.3 Los vectores de pesos de los nodos de la capa oculta se modifican como se indica a continuación siempre y cuando j+1 sea menor que el número de neuronas N2:

$$W_n = W_{n+1} \quad n = j+1,\ldots,N2 \qquad (8.11)$$

3- Repetimos el paso 1 hasta todas las distancias sean mayores que ε.

Después de aplicar este algoritmo nos quedamos con un número mínimo de nodos en la capa oculta. En este punto ya si nos encontramos en disposición de asignar valores a los centros de las funciones de pertenencia y a las salidas estimados mediante los vectores de pesos correspondientes a los nodos de la capa oculta. Obsérvese que en la sección anterior esta asignación se pospuso hasta tener un número óptimo de nodos en la capa oculta, o desde otra perspectiva , un número de reglas difusas mínimo.
Los valores iniciales de los centros de las funciones de pertenencia se obtienen como:

$$m_{ij} = w_{ij} \qquad \begin{matrix} j = 1,2,\ldots,N2 \\ i = 1,2,\ldots,N1 \end{matrix} \qquad (8.12)$$

Mientras que los valores estimados de las salidas se toman como:

$$sv_{jk} = w_{N1+k,j} \qquad \begin{matrix} k = 1,2,\ldots,N3 \\ j = 1,2,\ldots,N2 \end{matrix} \qquad (8.13)$$

Una vez que se han aplicado estas dos fases de entrenamiento, tenemos por un lado un número mínimo de nodos en la capa oculta, así como valores, aunque iniciales, para los centros de las funciones de pertenencia y las salidas estimadas. Sin embargo, es necesaria una fase de aprendizaje adicional que determine valores para otros parámetros de la estructura del sistema Neuro-Fuzzy, así como, que mejore los valores obtenidos para algunos de los parámetros. Este es el propósito de la tercera fase de entrenamiento que se muestra en la siguiente sección.

8.4.2.3 TERCERA FASE DE ENTRENAMIENTO.

Teniendo en cuenta la analogía entre la estructura del Sistema Neuro-Fuzzy y una red neuronal de base radial (RBFN) típica, podríamos utilizar el algoritmo de aprendizaje de mínimos cuadrados utilizado habitualmente en este tipo de red.
El objetivo de este algoritmo se centra en minimizar la siguiente función de costo:

$$E = \frac{1}{2}\sum_{k=1}^{N3}\left(Y_k - \hat{y}_k\right)^2 \quad k = 1,\ldots,N3 \tag{8.14}$$

Donde:

$\hat{y}_k$ = k-ésima salida deseada.

Y_k = k-ésima salida del sistema Neuro-Fuzzy.

Si *m* es cualquier parámetro del sistema Neuro-Fuzzy se actualizaría como:

$$m(t+1) = m(t) - lr\frac{\partial E}{\partial m} \quad t = 1, 2, \ldots, mf \tag{8.15}$$

Donde :

lr= Velocidad de aprendizaje.
m(t) = Parámetro m en el instante t.
mf = Número de épocas.

Teniendo presente la ecuación (8.15) sólo es necesario calcular la derivada parcial de la función de costo con respecto al parámetro particular que estemos considerando. Las expresiones para las derivadas parciales de los centros y las anchuras de las funciones de pertenencia, así como, la derivada parcial con respecto a los valores estimados de las salidas se muestran en las ecuaciones (8.16 a), (8.16 b) y (8.16 c) respectivamente:

$$\frac{\partial E}{\partial m_{ij}} = -2\left[\sum_{k=1}^{N3}\left(y_k - \hat{y}_k\right)\frac{sv_{jk}\sum_{j=1}^{N2} p_{ij} - \sum_{j=1}^{N2} sv_{jk}\, p_{ij}}{\left(\sum_{j=1}^{N2} p_{ij}\right)^2}\right] p_{ij}\left[\frac{\left(u_i - m_{ij}\right)}{s_{ij}^2}\right] \quad \begin{matrix} i = 1,\ldots,N1 \\ k = 1,\ldots,N3 \end{matrix} \tag{8.16a}$$

$$\frac{\partial E}{\partial s_{ij}} = -2\left[\sum_{k=1}^{N3}\left(y_k - \hat{y}_k\right)\frac{sv_{jk}\sum_{j=1}^{N2} p_{ij} - \sum_{j=1}^{N2} sv_{jk}\, p_{ij}}{\left(\sum_{j=1}^{N2} r_j\right)^2}\right] p_{ij}\left[\frac{\left(u_i - m_{ij}\right)^2}{s_{ij}^3}\right] \quad \begin{matrix} i = 1,\ldots,N1 \\ k = 1,\ldots,N3 \end{matrix} \tag{8.16b}$$

$$\frac{\partial E}{\partial sv_{jk}} = -\left(y_k - \hat{y}_k\right)\frac{g_j}{\sum_j g_j} \quad \begin{matrix} j = 1,\ldots,N2 \\ k = 1,\ldots,N3 \end{matrix} \tag{8.16c}$$

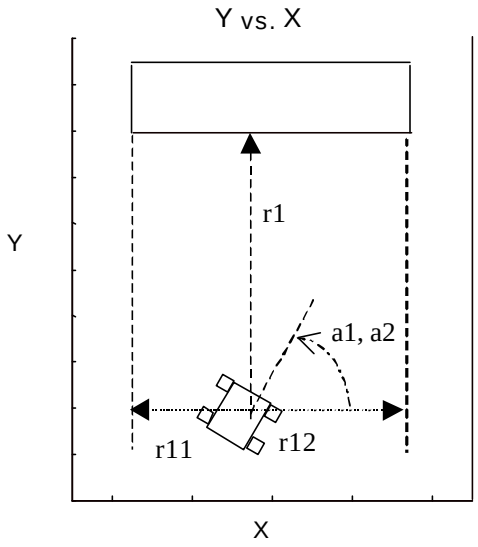
Donde:

p_{ij} = Salida correspondiente a la entrada i-ésima del sistema Neuro-Fuzzy, la cual es una de las entradas del nodo j-ésimo de la capa oculta.

γ_j = Salida del j-ésimo nodo de la capa oculta.

8.5 RESULTADOS DE LA APLICACIÓN DEL SISTEMA NEURO-FUZZY

Después de haber presentado el sistema Neuro-Fuzzy, en esta sección daremos algunos resultados de su aplicación. La aplicación del sistema Neuro-Fuzzy presupone como primer punto el que un humano proporcionará un número de trayectorias que superen con éxito los obstáculos que se le presenten al robot móvil a su paso. El número de trayectorias proporcionadas debe de ser tal que permita la extracción de las reglas necesarias para dirigir los movimientos del robot, es decir, en ellas debe de estar reflejado el comportamiento que se desea para el robot. Como primer caso hemos estudiado la situación en la cual el robot móvil debe superar un obstáculo rectangular como el que se muestra en la figura (8.14). Las variables r1 y r2 indican distancias entre el robot y el obstáculo tal como se indica en la figura, mientras que a1 es el ángulo en el instante presente y a2 sería el nuevo valor del ángulo que tomaría el robot. Obsérvese que r2 se ha definido como -r11/(r11-r12). Mientras que los rangos en los cuales se encuentran estas variables se han indicado en la figura (8.14). En definitiva la tarea del sistema Neuro-Fuzzy se centra en determinar el valor del nuevo ángulo a2, a través del valor del ángulo actual a1 y las distancias al obstáculo r1 y r2. De cara a la aplicación del sistema Neuro-Fuzzy hemos multiplicado las entradas del mismo por un factor menor de uno. De esta forma disminuimos los valores de las entradas y los adaptamos mejor a la aplicación de la estrategia Neuro-Fuzzy. En las simulaciones mostradas se ha hecho uso de un factor de 0.75. En la figura (8.15) mostramos los resultados obtenidos en este caso, en donde también se ha señalado mediante un rectángulo en línea discontinua la zona en la cual se han tomado las trayectorias para el proceso de entrenamiento del sistema Neuro-Fuzzy. Como se puede observar en la figura mostramos dos trayectorias correspondientes a dos puntos de partida diferentes en donde se puede apreciar el buen comportamiento del sistema Neuro-Fuzzy.



r1 >0 ; r11 < 0 ; r12 >0 ; a1 >0 ; a2>0

Figura 8.14 -Variables r1, r2=-r11/(r11 r12), ángulo actual a1 y nuevo ángulo a2

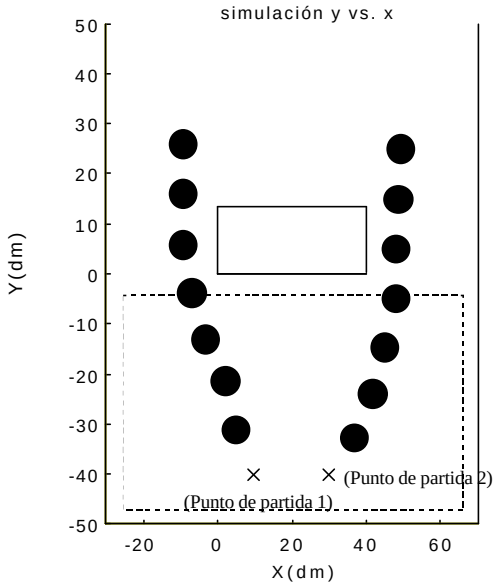
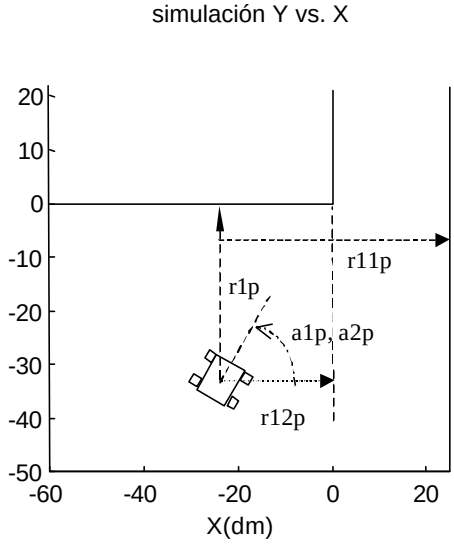


Figura 8.15 – Robot esquivando un obstáculo rectangular.

Antes de proseguir parece oportuno hacer una observación respecto a la elección de las variables, éstas han sido elegidas de tal forma que evitemos en lo posible dependencia respecto a la dinámica directa del robot móvil, es decir la relación entre el comando que se aplica y el movimiento resultante. El sistema Neuro-Fuzzy únicamente precisa el valor de ángulo que debería de tomar el robot en el siguiente instante de tiempo no haciendo mención en ningún caso a características dinámicas del movimiento, siendo función del módulo piloto indicado en la figura (8.1) el conseguir el ángulo deseado.



r1p >0 ; r11p > 0 ; r12p >0 ; a1p >0 ; a2p>0

Figura 8.16 - Variables r1p, r2p=-r11p/(r11p-r12p), ángulo actual a1p y nuevo ángulo a2p.

Otra situación tratada mediante el sistema Neuro-Fuzzy es la de superar un obstáculo en forma de esquina tal como se muestra en la figura (8.16). Para este caso hemos elegido

como variables que nos indiquen a que distancia se encuentran los obstáculos r1p y r2p, donde r2p es igual al cociente -r11p/(r11p-r12p). En este caso al igual que en el caso del obstáculo rectangular también hemos tomado como variables el ángulo en el instante actual señalado en la figura (8.16) como a1p y el valor del nuevo ángulo a2p. En la figura (8.16) también hemos señalado los rangos de valores en los cuales se encuentran cada una de las variables.

Al igual que en el caso anterior el sistema Neuro-Fuzzy debe de suministrar valores para los ángulos en el instante de tiempo t+1 a través de los valores de las distancias al obstáculo (r1p y r2p) y el valor del ángulo para el instante t tal como se indica en la figura (8.16). En la figura (8.17) presentamos una trayectoria resultante una vez el sistema Neuro-Fuzzy ha sido entrenado. En este caso las trayectorias de entrenamiento se han tomado dentro del área indicada en la figura (8.17) en línea discontinua.

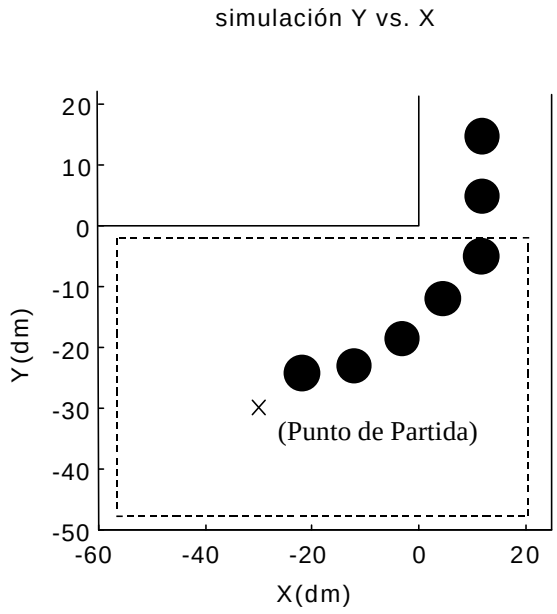


Figura 8.17 – Robot esquivando un obstáculo en forma de esquina.

Una vez que hemos entrenado el sistema Neuro-Fuzzy para superar los dos tipos de obstáculos vistos hasta ahora, hemos colocado al robot móvil en una situación en la cual se encontrase con obstáculos que se pudiesen clasificar como alguno de los casos estudiados. El resultado se muestra en la figura (8.18). Obsérvese que la zona de trabajo en este caso se asemeja a un laberinto, esquivando con éxito todos los obstáculos que se encuentran a su paso. También debemos destacar que aunque el sistema Neuro-Fuzzy ha sido entrenado para superar obstáculos rectangulares otras formas de obstáculos también podrían ser superadas aproximando tales obstáculos por un rectángulo. Es importante destacar que aunque el sistema Neuro-Fuzzy ha sido entrenado para superar las primeras esquinas que se le presentan en la trayectoria al robot móvil que tiene como posición de partida la número 3, la tercera esquina antes de llegar al destino también es superada por el robot móvil siendo ésta diferente a la que ha sido utilizada en el proceso de entrenamiento.

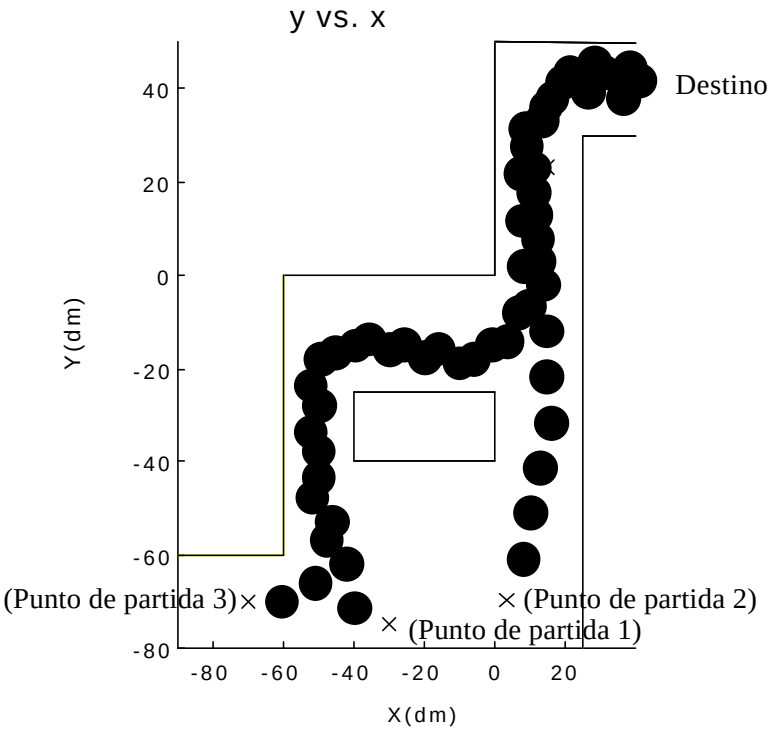


Figura 8.18 – Robot móvil dentro de un laberinto.

Una vez que se han comprobado las técnicas en simulación se ha procedido a implementarlas sobre el prototipo Sultan I. El proceso de entrenamiento del sistema Neuro-Fuzzy se realiza en un PC. Este proceso es el más costoso desde el punto de vista computacional, dado que el proceso de entrenamiento se basa en un proceso de cálculos iterativos tal y como se ha mostrado en las secciones anteriores, dependiente del número de ejemplos proporcionados por el operador. Este número, en general ha de ser grande con el objeto de que el sistema Neuro-Fuzzy pueda extraer las reglas de modo adecuado. Para establecer una forma sencilla de introducir estos ejemplos se ha desarrollado una interface gráfica que ayuda al operador a proporcionar su cononocimiento acerca de cómo esquivar los obstáculos evitando cualquier conocimiento específico del sistema. Una vez concluido el proceso de entrenamiento, el sistema Neuro-Fuzzy se reduce a una red neuronal de base Radial (RBF), donde los parámetros de la misma han sido fijados. De tal manera que la evalución de la reglas Fuzzy por el prototipo es sustituida por una red neuronal de base Radial, donde las entradas son las coordenadas (x,y) del robot y el ángulo tomado por el mismo en el instante previo, mientras que la salida es el nuevo ángulo de giro. Desde el punto de vista del tiempo de cómputo el operar con la red RBF representa un pequeño porcentaje comparado con el tiempo invertido en la fase de entrenamiento, destacando que la optimización del número de reglas influye muy positivamente en este tiempo. En la implementación hemos de tener en cuenta que el prototipo posee poca memoria para programas y datos, siendo esta de 32 Kbytes de RAM. Esto nos ha obligado a implementar el proceso de cálculo en un PC de tal forma que los datos necesarios para el control del robot se envien mediante un enlace de radio. Para realizar este enlace de radio se han utilizado dos módulos transmisores a 433 MHz, mientras que la recepción se ha realizado mediante dos módulos receptores sintonizados a 433 MHz. Hemos colocado un módulo transmisor en el PC y el otro en el robot, los dos asociados al puerto serie del PC y de la tarjeta microcontroladora del robot, respectivamente. Por

tanto a través de los puertos series y con la electrónica complementaria para transformar señales de niveles RS-232 a niveles TTL, utilizados por los módulos transmisores y receptores, se transmite la información necesaria en forma digital desde el PC al robot y del robot al PC a una velocidad de 1200 baudios. La comunicación se ha realizado de forma semi-duplex de tal forma que no se permita la emisión y recepción simultánea en el PC o en el robot, con objeto de evitar interferencias, sin embargo esto no representa ninguna limitación en la aplicación de la estrategia Neuro-Fuzzy. En el apéndice A mostramos las características técnicas de los modulos transmisores y receptores, así como los programas desarrollados para la comunicación entre la tarjeta Handy-Board situada en Sultan I y el PC. De cara a la aplicación de la estrategía Neuro-Fuzzy el robot debe enviar las coordenadas de posición (x,y) y el ángulo actual, mientras que el PC como resultado de la aplicación del sistema Neuro-Fuzzy envía al robot los comandos necesarios para que el robot se dirija según un ángulo en línea recta una distancia igual a la que ha sido utilizada para los movimientos del robot en el proceso de entrenamiento del sistema Neuro-Fuzzy. Un punto importante de la aplicación del sistema Neuro-Fuzzy es la obtención de las coordenadas de posición (x,y) y del ángulo del robot. Esta se realiza en el robot mediante una brújula electrónica que nos fija un ángulo considerando el Norte magnético como dirección absoluta, y un codificador basado en una rueda dentada atravesada por un haz de luz en el rango del infrarojo que nos permite obtener la distancia recorrida. Estos dos sensores nos permiten obtener valores de la posición (x,y) y del ángulo, sin embargo debemos remarcar que al tener la brújula un error de 2 grados, además de verse afectada por las estructuras metálicas cercarnas que distorsionan el campo magnético, las posiciones reales difieren de las medidas por el robot. Otro factor a tener en cuenta son los movimientos desarrollados por el robot en donde el codificador se deslice lateralmente, dado que en este caso también se introduce una diferencia entre la distancia real recorrida y la medida por el codificador. Estas imprecisiones en los sensores hacen diferir los resultados simulados de los resultados reales. Sin embargo si disminuimos las distancias recorridas por el robot en cada movimiento respecto a las utilizadas en simulación se observa como el sistema Neuro-Fuzzy es bastante robusto a los errores introducidos en los sensores. Otra de las características observadas es la robustez que presenta el sistema Neuro-Fuzzy frente al control del ángulo de partida de cada movimiento del robot, puesto que errores de 10 grados permiten seguir obteniendo resultados satisfactorios.

Además de los sensores ya vistos el robot posee un sonar situado sobre un servomotor que permite su movimiento angular, de tal forma que podemos detectar obstáculos presentes delante del robot. Mediante este sistema sonar podemos identificar el tipo de obstáculo con el que nos encontramos (una esquina, un rectángulo, etc.), de tal forma que apliquemos los parámetros del sistema Neuro-Fuzzy adecuados para ese tipo de obstáculo. El reconocimiento de obstáculos mediante el sistema de sonar permite al robot moverse en cualquier entorno donde únicamente se encuentren obstáculos para los que ha sido entrenado el sistema Neuro-Fuzzy u otros que puedan ser aproximados de alguna forma a estos últimos. En la figura 8.19 se presenta una fotografía en donde observamos el robot Sultan I dentro de un entorno en forma de laberinto similar al presentado en la figura 8.18. Obsérvese que aunque los obstáculos en forma de esquinas y en forma de rectángulo aparecen en diferentes posiciones y orientaciones a las que han sido tomadas en el proceso de aprendizaje, es fácil generalizar los resultados a estos casos dado que únicamente se realizan traslaciones y rotaciones respecto a los considerados en el entrenamiento. Por otra parte, hemos de destacar que el espacio de trabajo, así como las distancias recorridas por el robot en cada movimiento han sido escaladas linealmente un factor 1/5 respecto a las utilizadas en simulación.

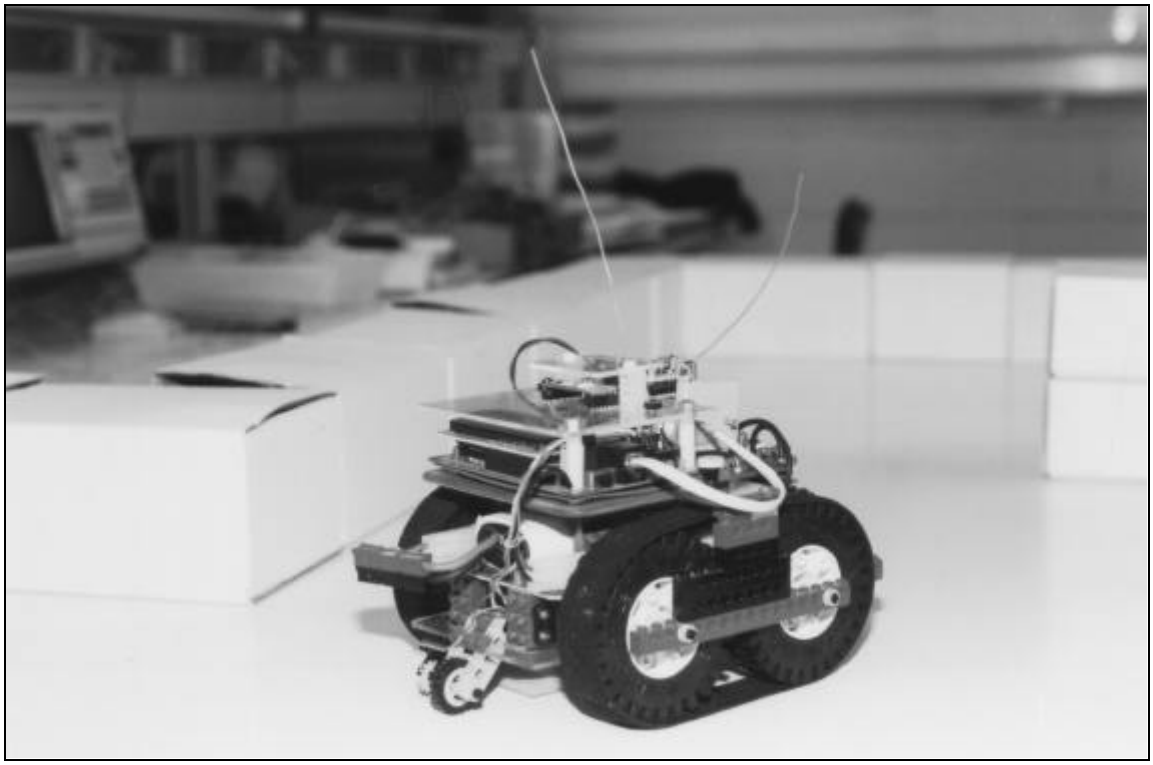


Figura 8.19 – Sultan I recorriendo un circuito en forma de laberinto.

Conclusiones

CONCLUSIONES

- En esta tesis se han abordado una serie de problemas del mundo de la robótica mediante técnicas conexionistas de inteligencia artificial.
- Las Redes Neuronales estáticas no pueden resolver el problema del modelado de un robot, al ser este un problema dinámico donde las salidas dependen de lo ocurrido en instantes anteriores. La dependencia además de ser dinámica es no lineal, y debido a las perturbaciones no con una estructura fija. Por lo tanto se justifica emplear las Redes Neuronales como modelo global o para caracterizar las desviaciones respecto a un modelo nominal.
- Se han propuesto varios modelos, en concreto una Red Neuronal LRGF (Locally Recurrent Globally Feedforward) y se han estudiado sus ventajas e inconvenientes proponiéndose mecanismos de aprendizaje efectivos. Los resultados se han comparado con los esquemas de TDNN (Time Delay Neural Network), más simples pero menos eficaces en este caso. Posteriormente se ha particularizado esta red para el estado estacionario comprobándose su buen comportamiento, aunque peor que la red anterior, sobre todo en generalización.
- Para implementar un esquema de control es necesario resolver la cinemática inversa. Este es un problema geométrico, y por tanto estático, aunque no lineal y multievaluado. Para el caso de los robots redundantes es interesante la aplicación de las Redes Neuronales Estáticas. Se ha realizado un estudio de esta aplicación, utilizándose también algoritmos genéticos en el aprendizaje, introduciendo un término de penalización en la función de coste, con el objeto de evitar las redundancias en las soluciones.
- Se ha propuesto un esquema de control basado en redes neuronales estáticas, en el cual las redes actúan sintonizando los parámetros de un controlador clásico, tal como un PID. Como modelo de la planta para calcular los gradientes se utiliza el modelo dinámico presentado.
- Cuando el dispositivo robótico es móvil surge la necesidad de obtener trayectorias libres de obtáculos. Esto se ha planteado clásicamente como un problema de optimización, o mediante un conjunto de reglas Fuzzy que tiene que dar el observador. Recientemente se han propuesto varios esquemas neurofuzzy para realizar este tipo de tareas, nosotros adaptamos un esquema donde a partir de un conjunto de trayectorias que sean soluciones al problema dadas por un operador humano experto se obtienen un conjunto de reglas que permiten al robot evitar los obstáculos que se le presentan. Hemos entrenado este sistema Neuro-Fuzzy para

diferentes obstáculos. Teniendo en cuenta diversas transformaciones de coordenadas hemos utilizado el sistema Neuro-Fuzzy ya entrenado para esquivar los obstáculos orientados de diferente forma. Esto nos ha permitido introducir el robot en un entorno en forma de laberinto superando los diferentes obstáculos que se encuentra en su camino hasta la posición objetivo.

Apéndice

Apéndice A

ESPECIFICACIONES TÉCNICAS

A.1 ROBOT PUMA.

longitud de las articulaciones : 1 m.

Peso de cada articulación : 1 Kg.

Número de articulaciones: 6.

A.2 ESPECIFICACIONES DE RAP.

Dimensiones (largo, ancho y alto) : 37cm x 24.5 cm x 24 cm

Peso (sin batería) : 2200gr.

Peso (con batería) : 2500 gr.

Motor : Mabuchi de 2.5cm de diámetro.

Caja reductora : Kyosho con reducción 20:1

Control del motor : PWM.

Resolución del PWM : 4 bits.

Velocidad máxima : 1.25 m/s

Velocidad normal de funcionamiento : 0.6 m/s

Tipo de batería : Batería estándar de Ni-Cad de 7.2 voltios con conector tipo “kyosho”

Consumo : 3.5 Amperios aproximadamente.

Autonomía : 20 minutos aprox. con batería de 1400mAH

Número de sensores (emisor-receptor) del sonar : 3 distribuidos en la parte frontal.

Rango de detección del sonar : 0 - 120 cm.

Resolución del sonar : 4 bits.

Servo de la dirección : Marca FUTABA modelo S-28.

Ángulo de giro de la rueda delantera : -70º a 70º.

Resolución del controlador del servo : 4 bits con 9 valores útiles.

Procesador principal : 80286 de AMD funcinando a 16Mhz.

Memoria : 512Kb en dos módulos S.I.M.M.

A.3 ESPECIFICACIONES DE SULTAN I.

Dimensiones (largo, ancho y alto) : 22.5 cm x 18.5 cm x 15 cm.

Peso (sin batería) : 990 gr.

Peso (con batería) : 1290 gr.

Motores : Dos servos FUTABA S3003.

Caja reductora : Incluida con los motores.

Control de los motores : PWM por Software.

Resolución del PWM : Cada motor tiene 7 valores distintos en cada sentido.

Velocidad máxima : 0.4 m/s

Tipo de batería : Batería estándar de Ni-Cad de 7.2 voltios con conector tipo "kyosho"

Consumo : 0.6 Amperios aproximadamente.

Autonomía : 2 horas aprox. con batería de 1400mAH.

Número de sensores (emisor-receptor) del sonar : 1 montado sobre un servo móvil.

Resolución del servo móvil del sonar : 1º aprox.

Rango de detección del sonar : 0 - 150 cm.

Resolución del sonar : 16 bits. Aproximadamente 1microseg. de resolución temporal.

Procesador principal : MC68HC11A1FN funcionando a 2Mhz.

Memoria : 32Kb de RAM estática.

Visualización : Display con dos lineas de 16 caracteres.

A.4 ESPECIFICACIONES DEL MÓDULO TRANSMISOR MOD TX 433SAW.

Tipo de Circuito: Circuito híbrido de alta fiabilidad.

Frecuencia de trabajo: 433.92 MHz.

Frecuencia de modulación máxima para lógica TTL: 4KHz.

Impedancia de antena: 50 ohm.

Potencia de salida de RF: 10 mW

Dimensiones: 12.2 x 38.1 x 6 mm.

Patillaje:

1 Tierra
2 Entrada de modulación con Vc<8V.
3 Entrada de modulación con Vc>8V.
4 Tierra
11 Salida de antena.
13 Tierra
15 Vc desde 4V hasta +12 V.

A.5 ESPECIFICACIONES DEL MÓDULO RECEPTOR MOD BC-NB.

Tipo de circuito: Circuito híbrido de alta fiabilidad.

Frecuencia de modulación máxima para lógica TTL: 2KHz.

Frecuencia de sintonía: de 220 a 433 MHz.

Ancho de Banda: entre +/- 1MHz y +/- 1.5 MHz.

Sensibilidad de RF medida con una señal de entrada ON/OFF: 3 μV.

Dimensiones: 12.2 x 38.1 x 6 mm.

Patillaje:

1 +5 V.
2 Tierra.
3 Antena
7 Tierra
11 Tierra.
13 Salida de testeo.
14 Salida.
15 +5V.

En ausencia de señal de Radio frecuencia salida a nivel lógico bajo.
Antena de longitud λ/4.
Fuente con circuito de filtrado RC para evitar problemas de ruido.

Apéndice B

LISTADOS DE LOS PROGRAMAS DE SIMULACIÓN Y CONTROL DE LOS ROBOTS

B.1 CONTROL NEURONAL.

Programas realizados en Matlab para el control neuronal de robots manipuladores. Estos programas utilizan funciones de la toolbox de redes neuronales de Matlab.

B.1.1 robot.m

```
% robot.m
%     sistema de control sobre un robot puma.
%
load rob2lnk1.mat; % parámetros del robot
global tau_new;
nact=2; % articulaciones a controlar.
n=nact; % n. de articulaciones consideradas por el modelo.
n_net=nact;% n. de redes neuronales.

 % inicializacion constantes en control realimentado

 K=[k1 k2]; % según número de articulaciones
 B=[b1 b2]; % según número de articulaciones

 % inicializacion variables

  Qd=[1 1];
  Q_old=[0 0];
  Q1_old=[0 0];
  tau_old=[0 0];
  Qevolucion(1,:)=Q_old;
  Q1evolucion(1,:)=Q1_old;

% tau inicial distinto de cero

tau_new=[0.00001 0.00001];
tau_evolucion(1,:)=tau_new;

 % parametros simulacion

 t=0:0.0001:5;
 n_sim=max(size(t));

% inicializacion de las redes
```

```
 ne=4; % n³ de entradas
 n1=3; % n³ de neuronas de primera capa
 n2=3; % n³ de neuronas de segunda capa
 ns=2; % n³ de neuronas de salida.
 for indice=1:nact
  eval(['w1',int2str(indice),'=randv(-0.1,0.1,n1,ne);']);
  eval(['b1',int2str(indice),'=randv(-0.1,0.1,n1,1);']);
  eval(['w2',int2str(indice),'=randv(-0.1,0.1,n2,n1);']);
  eval(['b2',int2str(indice),'=randv(-0.1,0.1,n2,1);']);
  eval(['w3',int2str(indice),'=randv(-0.1,0.1,ns,n2);']);
  eval(['b3',int2str(indice),'=randv(-0.1,0.1,ns,1);']);
 end;
for i=1:(n_sim-1)
   tau_evolucion(i,:)=tau_new;
   Kevolucion(i,:)=K;
   Bevolucion(i,:)=B;
   tol=1e-3;
   minstep=0.00001;
   maxstep=0.001;

 % rob2lnk modelo del robot implementado en Simulink

   [T X1 Y]=rk45('rob2lnk',[t(i) t(i+1)],...
            [Q1_old(1) Q_old(2) Q_old(1) Q1_old(2)],...
            [tol,minstep,maxstep],'torque');
  Qt=[Y(:,1) Y(:,4)];
  Qt1=[Y(:,2) Y(:,5)];
  n_sol=max(size(T));
  Q_new=Qt(n_sol,1:n);
  Q1_new=Qt1(n_sol,1:n);
  Qr_new=Q_new(:,1:nact);
  Q1r_new=Q1_new(:,1:nact);
  Qevolucion(i+1,:)=Q_new;
  Q1evolucion(i+1,:)=Q1_new;
  e=Qd-Q_new; % error de los titas
  Eevolucion1(i,:)=e;
  dQ=Q_new-Q_old; % diferencia de los angulos.
  dtau=tau_new-tau_old;
  Q_old=Q_new;
  Q1_old=Q1_new;
 for j=1:nact
   prod=(1/dtau(j))*sum(e.*dQ);
   err_new=[e(j);Q1_new(j)];
   err_new=err_new*prod;
  if j==1
%   [w11,b11,w21,b21,w31,b31]=net_new(w11,b11,'tansig',...
%        w21,b21,'tansig',w31,b31,'purelin',[Qr_new
Q1r_new]',0.01,err_new);
%   [a1,a2,K(j)]=simuff([Qr_new Q1r_new]',w11,b11,'tansig',w21,b21,...
%        'tansig',w31,b31,'purelin');
   tau_old(j)=tau_new(j);
   tau_new(j)=K(j)*e(j)+B(j)*Q1_new(j);
  elseif j==2
%   [w12,b12,w22,b22,w32,b32]=net_new(w12,b12,'tansig',...
%        w22,b22,'tansig',w32,b32,'purelin',[Qr_new
Q1r_new]',0.01,err_new);
%   [a1,a2,K(j)]=simuff([Qr_new Q1r_new]',w12,b12,'tansig',w22,b22,...
%        'tansig',w32,b32,'purelin');
   tau_old(j)=tau_new(j);
   tau_new(j)=K(j)*e(j)+B(j)*Q1_new(j);
  elseif j==3
```

```
   [w13,b13,w23,b23,w33,b33]=net_new(w13,b13,'tansig',...
         w23,b23,'tansig',w33,b33,'purelin',[Qr_new
Q1r_new]',0.01,err_new);
   [a1,a2,K(j)]=simuff([Qr_new Q1r_new]',w13,b13,'tansig',w23,b23,...
         'tansig',w33,b33,'purelin');
   tau_old(j)=tau_new(j);
   tau_new(j)=K(j)*e(j)+B(j)*Q1_new(j);
  elseif j==4
   [w14,b14,w24,b24,w34,b34]=net_new(w14,b14,'tansig',...
         w24,b24,'tansig',w34,b34,'purelin',[Qr_new
Q1r_new]',0.01,err_new);
   [a1,a2,K(j)]=simuff([Qr_new Q1r_new]',w14,b14,'tansig',w24,b24,...
         'tansig',w34,b34,'purelin');
   tau_old(j)=tau_new(j);
   tau_new(j)=K(j)*e(j)+B(j)*Q1_new(j);
  elseif j==5
   [w15,b15,w25,b25,w35,b35]=net_new(w15,b15,'tansig',...
         w25,b25,'tansig',w35,b35,'purelin',[Qr_new
Q1r_new]',0.01,err_new);
   [a1,a2,K(j)]=simuff([Qr_new Q1r_new]',w15,b15,'tansig',w25,b25,...
         'tansig',w35,b35,'purelin');
   tau_old(j)=tau_new(j);
   tau_new(j)=K(j)*e(j)+B(j)*Q1_new(j);
  end;
 end;
end;
clear Q_old Q1_old n_sim tau_old tau_new n_sol e dQ dtau Q_new Q1_old;
clear Qr_new Q1r_new i indice prod j B K Q1_new;
```

B.1.2 net_new.m

```
function [w1,b1,w2,b2,w3,b3] =
net_new(w1,b1,f1,w2,b2,f2,w3,b3,f3,p,tp,err_new)
% net_new.m
%           funcion basada en tbp3.m, para implementar redes
neuronales.
%
%TBP3 Train 3-layer feed-forward network w/backpropagation.
%
%     [W1,B1,W2,B2,W3,B3,TE,TR] =
TBP3(W1,B1,F1,W2,B2,F2,W3,B3,F3,P,T,TP,err_new)
%       Wi - SixR weight matrix of ith layer.
%       Bi - Six1 bias vector of ith layer.
%       F  - Transfer function (string) of ith layer.
%       P  - RxQ matrix of input vectors.
%       T  - S2xQ matrix of target vectors.
%       TP - Training parameters (optional).
%
%
%         err_new    error en la ultima capa
%
%     Returns:
%       Wi - new weights.
%       Bi - new biases.
%
%     Training parameters are:
%       TP(1) - Learning rate, 0.01.
%     Missing parameters and NaN's are replaced with defaults.
```

```
%     Mark Beale, 1-31-92
% Revised 12-15-93, MB
% Copyright (c) 1992-94 by the MathWorks, Inc.
% $Revision: 1.1 $  $Date: 1994/01/11 16:29:35 $



if nargin < 11,error('Not enough arguments.'),end

% TRAINING PARAMETERS

lr = tp(1);
df1 = feval(f1,'delta');
df2 = feval(f2,'delta');
df3 = feval(f3,'delta');

% PRESENTATION PHASE

a1=feval(f1,w1*p,b1);
a2=feval(f2,w2*a1,b2);
a3=feval(f3,w3*a2,b3);


% BACKPROPAGATION PHASE

  d3 = feval(df3,a3,err_new);
  d2 = feval(df2,a2,d3,w3);
  d1 = feval(df1,a1,d2,w2);

  % LEARNING PHASE

  [dw1,db1] = learnbp(p,d1,lr);
  [dw2,db2] = learnbp(a1,d2,lr);
  [dw3,db3] = learnbp(a2,d3,lr);
  w1 = w1 + dw1; b1 = b1 + db1;
  w2 = w2 + dw2; b2 = b2 + db2;
  w3 = w3 + dw3; b3 = b3 + db3;
```

B.2 SISTEMA DE CONTROL NEURO-FUZZY.

Scripts en Matlab que realizan entrenamiento del sistema Neuro-Fuzzy y permiten simular el movimiento del robot móvil desde cualquier posición inicial.

B.2.1 neurofuzzy.m

```
  function [m,sv,sigma]=neufuzzy(U,Y,cota)
%
% function [m,sv,sigma]=neufuzzy(U,Y,cota)
%
% Descripcion:
%
% sistema neuro-fuzzy que determina parametros de tal sistema
% mediante entrenamiento.
%
% entrada:
%           U entrada de dimension (RxQ)
%           Y salida de dimension (SxQ)
% salidas:
%           m     centros de las funciones de pertenencia.
%          sv     valores estimados para los nodos de reglas.
%          sigma  anchuras de las funciones de pertenencia.
%
 % eligiendo cota por defecto.

 if nargin~=3
  cota=0.001;
 end

 % entradas al sistema neuro-fuzzy


    [R,Q]=size(U); % dimension (RxQ)

    Umin=min(U');
    Umin=Umin';

    Umax=max(U');
    Umax=Umax';

 % salidas del sistema neuro-fuzzy

    [S,Q1]=size(Y);  % dimension (SxQ)
    if Q1~=Q
     error('entrada y salida tienen que tener igual Q');
    end

    Ymin=min(Y');
    Ymin=Ymin';
    Ymax=max(Y');
    Ymax=Ymax';

   % KOHONEN'S SELF-ORGANIZING FEATURE MAP ALGORITM

 U=U'; % dimension (QxR)
 Y=Y'; % dimension (QxS)
```

```
Umin=Umin'; % dimension (1xR)
Umax=Umax';
Ymin=Ymin'; % dimension (1xS)
Ymax=Ymax';

N1=R; % numero de entradas
N2=100; % numero de reglas fuzzy
N3=S; % numero de salidas.
alfa=0.75;
sigma0=1.75;
I=[alfa*U Y];

Imin=[Umin Ymin];
Imax=[Umax Ymax];
Ilimite=[Imin;Imax]; % dimension (2x(R+S))

    % dimension (QxR) R=N1+N3

w=initsmt(Ilimite',N2);
me1=500;
mf=50;
lr1=1;
w=trainfmt(w,I','h',sigma0,[mf me1 lr1 N2]);
 % RULE-EXTRACTING PHASE

  delta=0.05  % valor de distancia bajo del cual se mezclan
              % reglas.


 d=zeros(N2,N2); % vector de distancias.
 for j=1:(N2-1)
   d(j,j+1)=norm(w(j,:)-w(j+1,:));
 end
 j=1;
 while (j<=(N2-1))
    if (d(j,j+1)<delta)

     % asignacion de nuevos pesos (nuevos nodos).

      w(j,:)=(w(j,:)+w(j+1,:))/2;
      for l=j+1:(N2-1)
       w(l,:)=w(l+1,:);
      end
      N2=N2-1;
      w=w(1:N2,:);

     % definicion de las nuevas distancias entre los
     % nuevos nodos.

      for j=1:(N2-1)
        d(j,j+1)=norm(w(j,:)-w(j+1,:));
      end
      j=0;
    end
  j=j+1;
 end

   % asignacion de pesos
```

```
      % asignacion de las centros de las funciones de
      % pertenencia.

       for i=1:N1
        for j=1:N2
         m(i,j)=w(j,i);
        end
       end

      % asignacion de los valores estimados para los
      % nodos de reglas.

       for k=1:N3
        for j=1:N2
          sv(j,k)=w(j,N1+k);
        end
       end

  % LEARNING PHASE

    erold=inf;
    sigmaold=[];
    mold=[];
    svold=[];

    me=5000 % numero de entrenamientos
    lr=0.05 % velocidad de aprendizaje
    sigma=ones(N1,N2); % inicializacion de las sigmas

    U=U'; % dimension (RxQ)
    Y=Y'; % dimension (SxQ)

    [R,Q]=size(U);
    [S,Q1]=size(Y);
    if Q~=Q1
      error('dimension de U e Y incorrecta para LMS');
    end

for ent=1:me

 % presentation phase
for q1=1:Q

 for i=1:N1
  for j=1:N2
   P(i,j)=exp(-((alfa*U(i,q1)-m(i,j))^2)/(sigma(i,j)^2));
  end
 end

 for j=1:N2
  ro(j)=min(P(:,j));
 end

 for k=1:N3
  sy(k)=sum(sv(:,k).*ro')/sum(ro);
 end

  e=Y(:,q1)-sy;
  eseg(q1)=e;

 % learning phase
```

```
   for j=1:N2
    for k=1:N3
     sv(j,k)=sv(j,k)+lr*e(k)*ro(j)/sum(ro);
    end
   end

   for j=1:N2
    for i=1:N1
     if ro(j)==P(i,j)
       q(i,j)=1;
     else
       q(i,j)=0;
     end

     % actualizacion centro de funcion de pertenencia

     buf1=0;
     for k=1:N3
      buf2=sv(j,k)*sum(ro)-sum(sv(:,k).*((P(i,:))'));
      buf1=buf1+e(k)*buf2;
     end
     den=sum(P(i,:))^2;
     buf1=buf1*P(i,j)/den;
     buf2=buf1*((alfa*U(i,q1)-m(i,j))/(sigma(i,j)^2));
     m(i,j)=m(i,j)+2*lr*q(i,j)*buf2;

     % actualizacion de la varianza

     buf3=buf1*(alfa*(U(i,q1)-m(i,j))^2)/(sigma(i,j)^3);
     sigma(i,j)=sigma(i,j)+2*lr*q(i,j)*buf3;
    end
   end
  end

  % error cuadratico


   e1(ent)=sqrt(sum(eseg.^2));
   if rem(ent,10)==0
     if (e1(ent)<=cota);
      fprintf('valor minimo er= %g \n',e1(ent))
%     fprintf('valor siguiente er= %g \n',e1(ent))
%     sigma=sigmaold;
%     m=mold;
%     sv=svold;
      % error cuadratico al final del entrenamiento

      for q1=1:Q

      for i=1:N1
        for j=1:N2
         P(i,j)=exp(-((alfa*U(i,q1)-m(i,j))^2)/(sigma(i,j)^2));
        end
       end

       for j=1:N2
        ro(j)=min(P(:,j));
       end

       for k=1:N3
```

```
        sy(k)=sum(sv(:,k).*ro')/sum(ro);
       end

       e=Y(:,q1)-sy;
       eseg(q1)=e;
       ypr=simnf(U(:,q1),m,sv,sigma,alfa);
       eseg1(q1)=Y(:,q1)-ypr;
      end;
      disp(['sum squared error=',num2str(sumsqr(eseg))]);
      disp(['sum squared error1=',num2str(sumsqr(eseg1))]);

      plot(e1);
      xlabel('epocas');
      ylabel('error cuadratico');
      disp(' --- Kohonen self-organizing feature map ----');
      disp(' ');
      disp(['N1=',num2str(N1),' % numero de entradas']);
      disp(['N2=',num2str(N2),' % numero de reglas fuzzy']);
      disp(['N3=',num2str(N3),' % numero de salidas.']);
      disp(['numero de epocas=',num2str(me1)]);
      disp(['frecuencia de presentacion=',num2str(mf)]);
      disp([' velocidad de aprendizaje inicial=',num2str(lr1)]);
      disp(['alfa=',num2str(alfa),' % alfa que multiplica a U']);
      disp(['sigma0=',num2str(sigma0),' % sigma en funcion de
vecindad']);
      disp(' ');
      disp(' --- Rule-extracting phase ---');
      disp(' ');
      disp(['delta=',num2str(delta),' % cota de separacion de
reglas']);
      disp(' ');
      disp('----   LMS algorithm sobre ----');
      disp(' ');
      disp(['numero de epocas= ',num2str(me)]);
      disp(['lr=',num2str(lr)]);
      return;
     end
     fprintf('epo= %.0f, er= %g \n',ent,e1(ent))
     sigmaold=sigma;
     mold=m;
     svold=sv;
     erold=e1(ent);
   end
 end
   % error cuadratico al final del entrenamiento

 for q1=1:Q

  for i=1:N1
   for j=1:N2
    P(i,j)=exp(-((alfa*U(i,q1)-m(i,j))^2)/(sigma(i,j)^2));
   end
  end

  for j=1:N2
   ro(j)=min(P(:,j));
  end

  for k=1:N3
   sy(k)=sum(sv(:,k).*ro')/sum(ro);
  end
```

```
   e=Y(:,q1)-sy;
   eseg(q1)=e;
   ypr=simnf(U(:,q1),m,sv,sigma,alfa);
   eseg1(q1)=Y(:,q1)-ypr;
 end;
   disp(['sum squared error=',num2str(sumsqr(eseg))]);
   disp(['sum squared error1=',num2str(sumsqr(eseg1))]);

   plot(e1);
   xlabel('epocas');
   ylabel('error cuadratico');
   disp(' --- Kohonen self-organizing feature map ----');
   disp(' ');
   disp(['N1=',num2str(N1),' % numero de entradas']);
   disp(['N2=',num2str(N2),' % numero de reglas fuzzy']);
   disp(['N3=',num2str(N3),' % numero de salidas.']);
   disp(['numero de epocas=',num2str(me1)]);
   disp(['frecuencia de presentacion=',num2str(mf)]);
   disp([' velocidad de aprendizaje inicial=',num2str(lr1)]);
   disp(['alfa=',num2str(alfa),' % alfa que multiplica a U']);
   disp(['sigma0=',num2str(sigma0),' % sigma en funcion de
vecindad']);
   disp(' ');
   disp(' --- Rule-extracting phase ---');
   disp(' ');
   disp(['delta=',num2str(delta),' % cota de separacion de reglas']);
   disp(' ');
   disp('----   LMS algorithm sobre ----');
   disp(' ');
   disp(['numero de epocas= ',num2str(me)]);
   disp(['lr=',num2str(lr)]);
```

B.2.2 p2ob2.m

```
 Ytra=[];
 Utra=[];
trayec3; % dibuja la trayectoria a recorrer

x0=input('valor de x inicial:');
y0=input('valor de y inicial:');

pob=input('primer obstaculo esquina (s/n)','s');
if upper(pob)=='S'

% primer obstaculo esquina 1.
  load caso11.mat;
  if ~exist('alfa')
   alfa=1;
  end;
   alfa
  xorig=-60; % coordenadas nuevo sistema de referencia.
  yorig=-60;
  rot=0;
  a2=20;

   % cambio variables relativas.
```

```
    [x1 y1]=x2xp(0,[xorig yorig],[x0 y0]);
    [u0 u1]=x2vr([x1 y1],a2);
    u2=input('angulo inicial en grados:');
    u2=u2*pi/180;
    y=u2;
    mov=input('movimientos para superar esquina');
 for i=1:mov
  [u0new,u1new]=sigen2(u0,u1,y,xorig,yorig,a2)
  U1=[u0;u1;y];
  Utra=[Utra U1];
  y=simnf([u0 u1 y],m,sv,sigma,alfa)
  Ytra=[Ytra y];
  u0=u0new;
  u1=u1new;
 end
 [xp yp]=vr2x(u0new,u1new,a2);
 [x0 y0]=xp2x(rot,[xorig yorig],[xp yp]);
end;

% primer obstaculo (obstaculo1). Necesitamos (x0,y0) respecto a
% sistema normal.

 load ob1ca8.mat;
 if ~exist('alfa')
  alfa=1;
 end;
  alfa
 xorig=-40;
 yorig=-40;
 rot=0;
 aob1=40;
 mov=input('movimientos para superar rectangulo');

 [xp yp]=x2xp(0,[xorig yorig],[x0 y0]);
 xp
 yp
 [u0 u1]=xp2vrob1([xp yp],aob1);
 u2=input('angulo inicial en grados:');
 u2=u2*pi/180;
 y=u2;

 for i=1:mov
  [u0new,u1new]=sigxpob1(u0,u1,y,aob1,rot,[xorig,yorig]);
  U1=[u0;u1;y];
  Utra=[Utra U1];
  y=simnf([u0 u1 y],m,sv,sigma,alfa);
  y*180/pi
  Ytra=[Ytra y];
  u0=u0new;
  u1=u1new;

 end
 [xp yp]=vr2xpob1(u0,u1,aob1);
 [x0 y0]=xp2x(rot,[xorig yorig],[xp yp]);


% mirar los resultados de prueba con nuevas esquinas.
 load caso11.mat;
 if ~exist('alfa')
  alfa=1;
```

```
 end;
  alfa
% clg;
 a1=25; % anchura primera esquina.

 u0=-y0
 u12=x0
 u11=x0-a1.*ones(size(x0))
 u1=u11./(u11-u12);
 u2=y;
 u2=input('angulo inicial en grados:');
 u2=u2*pi/180
 mov=input('movimientos para superar esquina 2');
 y=u2;
 for i=1:mov
  [u0new,u1new]=sigen(u0,u1,y);
  U1=[u0;u1;y];
  Utra=[Utra U1];
  if u0new<-3
   break % condicion para cambiar de eje de coordenadas.
  end
  y=simnf([u0 u1 y],m,sv,sigma,alfa)
  Ytra=[Ytra y];
  u0=u0new;
  u1=u1new;
 end

  xorig=25; % coordenadas nuevo sistema de referencia.
  yorig=30;
  a2=50-30;

   % cambio variables relativas.

    [x0 y0]=vr2x(u0new,u1new,a1);
    [x1 y1]=x2xp(90,[xorig yorig],[x0 y0]);
    [u0 u1]=xp2vr([x1 y1],a2);
    y=pi/2-y; % nueva variable relativa para el angulo.
    mov=input('movimientos para superar esquina 3');
 for i=1:mov
  [u0new,u1new]=sigen1(u0,u1,y,xorig,yorig,a2);
  U1=[u0;u1;y];
  Utra=[Utra U1];
  y=simnf([u0 u1 y],m,sv,sigma,alfa)
  Ytra=[Ytra y];
  u0=u0new;
  u1=u1new;
 end
```

B.2.3 trayec3.m

```
  % trayecto.m

 minx=-90;
 maxx=40;
 miny=-80;
 maxy=50;
 axis([minx maxx miny maxy]);
 hold on;
 title('simulacion y vs. x');
```

```
 plot([-60 0],[0 0],'-y');
 plot([0 0],[0 50],'-y');
 plot([25 25],[miny 30],'-y');

 plot([25 40],[30 30],'-y');
 plot([0 40],[50 50],'-y');
% obtaculo ob1
% obseve que las lineas se colocan plot([x1 x2],[y1 y2],'-y');
 plot([-40 0],[-40 -40],'-y');
 plot([-40 0],[-25 -25],'-y');
 plot([-40 -40],[-40 -25],'-y');
 plot([0 0],[-40 -25],'-y');

% nueva esquina
 plot([-60 -60],[0,-60],'-y');
 plot([minx -60],[-60 -60],'-y');
 plot([minx -60],[-80 -80],'-y');

 xlabel(' X(m)');
 ylabel(' Y(m)');
```

B.2.4 sigen2.m

```
  function [u0new,u1new]=sigen2(u0,u1,u2,xorig,yorig,a2)
%
%  simula el paso de una posicion a otra segun dinamica del
%  robot con velocidad v=1. El sistema de refencia donde
%  se han dado las variables tiene origen (xorig,yorig).
%  desarrollado para obstaculo esquina sin rotar solo
%  trasladado.
%

 t=0:0.1:10;
 dimt=max(size(t));
 X=[];
 Y1=[];
 U2=[];

      % coordenadas respecto a sistema (xorig,yorig).

 [x1 y1]=vr2x(u0,u1,a2);

      % coordenadas respecto a sistema inercial.

 [x0 y0]=xp2x(0,[xorig yorig],[x1 y1]);


 X=[X x0];
 Y1=[Y1 y0];
 plot(x0,y0,'X');
 U2=[U2 u2];

 v=1;
 x=x0+v*cos(u2).*t;
 y=y0+v*sin(u2).*t;
 ye=[x0 y0]';
 L=10; % numero de pasos
 Y=ye*ones(1,L);
```

```
head=line('color','r','linestyle','.','markersize',50,'erase','xor',..
.
            'xdata',ye(1),'ydata',ye(2));
 body=line('color','y','linestyle','-','erase','none',...
            'xdata',[],'ydata',[]);
% tail=line('color','b','linestyle','-','erase','none',...
%            'xdata',[],'ydata',[]);
 for i=1:max(size(x))
  ye=[x(i) y(i)]';
  Y=[ye Y(:,1:L-1)];
  set(head,'xdata',Y(1,1),'ydata',Y(2,1));
  set(body,'xdata',Y(1,1:2),'ydata',Y(2,1:2));
%  set(tail,'xdata',Y(1,L-1:L),'ydata',Y(2,L-1:L));
  drawnow;
 end;
% el robot evoluciona segun en angulo que se vaya dando:



 % variables de interes desde el punto de vista
 % del sistema neuro-fuzzy.


    [x1 y1]=x2xp(0,[xorig yorig],[x(dimt) y(dimt)]);


   % nuevas variables.

    [u0new,u1new]=x2vr([x1 y1],a2);
```

B.2.5 sigxpob1.m

```
  function [u0new,u1new]=sigxpob1(u0,u1,u2,a,rot,origen)
%
%  simula el paso de una posicion a otra segun dinamica del
%  robot con velocidad v=1. Para objeto1.
%
%     rot     angulo de rotacion
%     origen  [xorig yorig]  origen del sistema al obstaculo
 if nargin==4
  rot=0;
  origen=[0 0];
 end
 xorig=origen(1);
 yorig=origen(2);

 t=0:0.1:10;
 dimt=max(size(t));
 X=[];
 Y1=[];
 U2=[];
 [xp0 yp0]=vr2xpob1(u0,u1,a);
 [x0 y0]=xp2x(rot,[xorig yorig],[xp0 yp0]);
```

```
 X=[X x0];
 Y1=[Y1 y0];
 plot(x0,y0,'X');
 U2=[U2 u2];

 v=1;
 x=x0+v*cos(u2).*t;
 y=y0+v*sin(u2).*t;
 ye=[x0 y0]';
 L=10; % numero de pasos
 Y=ye*ones(1,L);

head=line('color','r','linestyle','.','markersize',50,'erase','xor',..
.
            'xdata',ye(1),'ydata',ye(2));
 body=line('color','y','linestyle','-','erase','none',...
            'xdata',[],'ydata',[]);
% tail=line('color','b','linestyle','-','erase','none',...
%            'xdata',[],'ydata',[]);
 for i=1:max(size(x))
  ye=[x(i) y(i)]';
  Y=[ye Y(:,1:L-1)];
  set(head,'xdata',Y(1,1),'ydata',Y(2,1));
  set(body,'xdata',Y(1,1:2),'ydata',Y(2,1:2));
%  set(tail,'xdata',Y(1,L-1:L),'ydata',Y(2,L-1:L));
  drawnow;
 end;
% el robot evoluciona segun en angulo que se vaya dando:



 % variables de interes desde el punto de vista
 % del sistema neuro-fuzzy.
   [xp yp]=x2xp(rot,origen,[x(dimt) y(dimt)]);
   [u0new u1new]=xp2vrob1([xp yp],a);
```

B.2.6 sigen.m

```
%  simula el paso de una posicion a otra segun dinamica del
%  robot con velocidad v=1
%

 t=0:0.1:10;
 dimt=max(size(t));
 X=[];
 Y1=[];
 U2=[];
 x0=25*(1-u1);
 y0=-u0;

 X=[X x0];
 Y1=[Y1 y0];
 plot(x0,y0,'X');
 U2=[U2 u2];

 v=1;
 x=x0+v*cos(u2).*t;
 y=y0+v*sin(u2).*t;
```

```
 ye=[x0 y0]';
 L=10; % numero de pasos
 Y=ye*ones(1,L);

head=line('color','r','linestyle','.','markersize',50,'erase','xor',..
.
            'xdata',ye(1),'ydata',ye(2));
 body=line('color','y','linestyle','-','erase','none',...
            'xdata',[],'ydata',[]);
% tail=line('color','b','linestyle','-','erase','none',...
%            'xdata',[],'ydata',[]);
 for i=1:max(size(x))
  ye=[x(i) y(i)]';
  Y=[ye Y(:,1:L-1)];
  set(head,'xdata',Y(1,1),'ydata',Y(2,1));
  set(body,'xdata',Y(1,1:2),'ydata',Y(2,1:2));
%  set(tail,'xdata',Y(1,L-1:L),'ydata',Y(2,L-1:L));
  drawnow;
 end;
% el robot evoluciona segun en angulo que se vaya dando:



 % variables de interes desde el punto de vista
 % del sistema neuro-fuzzy.


   u0new=-y(dimt);
   u12=x(dimt);
   u11=x(dimt)-25;
   u1new=u11./(u11-u12);
```

B.2.7 sigen1.m

```
  function [u0new,u1new]=sigen1(u0,u1,u2,xorig,yorig,a2)
%
%  simula el paso de una posicion a otra segun dinamica del
%  robot con velocidad v=1. El sistema de refencia donde
%  se han dado las variables tiene origen (xorig,yorig).
%

 t=0:0.1:10;
 dimt=max(size(t));
 X=[];
 Y1=[];
 U2=[];

      % coordenadas respecto a sistema (xorig,yorig).

 a2=20;
 [x1 y1]=vr2xp(u0,u1,a2);

      % coordenadas respecto a sistema inercial.

 [x0 y0]=xp2x(90,[xorig yorig],[x1 y1]);
```

```
 X=[X x0];
 Y1=[Y1 y0];
 plot(x0,y0,'X');
 U2=[U2 u2];

 v=1;
 x=x0+v*cos(pi/2-u2).*t;
 y=y0+v*sin(pi/2-u2).*t;
 ye=[x0 y0]';
 L=10; % numero de pasos
 Y=ye*ones(1,L);

head=line('color','r','linestyle','.','markersize',50,'erase','xor',..
.
            'xdata',ye(1),'ydata',ye(2));
 body=line('color','y','linestyle','-','erase','none',...
            'xdata',[],'ydata',[]);
% tail=line('color','b','linestyle','-','erase','none',...
%            'xdata',[],'ydata',[]);
 for i=1:max(size(x))
  ye=[x(i) y(i)]';
  Y=[ye Y(:,1:L-1)];
  set(head,'xdata',Y(1,1),'ydata',Y(2,1));
  set(body,'xdata',Y(1,1:2),'ydata',Y(2,1:2));
%  set(tail,'xdata',Y(1,L-1:L),'ydata',Y(2,L-1:L));
  drawnow;
 end;
% el robot evoluciona segun en angulo que se vaya dando:



 % variables de interes desde el punto de vista
 % del sistema neuro-fuzzy.


    [x1 y1]=x2xp(90,[xorig yorig],[x(dimt) y(dimt)]);


   % nuevas variables.

    [u0new,u1new]=xp2vr([x1 y1],a2);
```

B.2.8 simnf.m

```
 function [sy]=simnf(U,m,sv,sigma,alfa)


% function [sy]=simnf(U,m,sv,sigma,alfa)
%
%  sistema neuro-fuzzy ya entrenado. Extrae valores para un caso
%  concreto.
%
% presentation phase

  [N2,N3]=size(sv);
  [N1,N2]=size(sigma);
```

```
  [N1,N2]=size(m);



  for i=1:N1
   for j=1:N2
    P(i,j)=exp(-((alfa*U(i)-m(i,j))^2)/(sigma(i,j)^2));
   end
  end

  for j=1:N2
   ro(j)=min(P(:,j));
  end

  for k=1:N3
   sy(k)=sum(sv(:,k).*ro')/sum(ro);
  end
```

B.2.9 vr2x.m

```
 function [x,y]=vr2x(u0,u1,a1)
%
%  [x,y]=vr2x(u0,u1,a1)
%
% Cambio de variables relativas al obstáculo esquina con origen
% en (0,0) y sistema sin rotar.

%      argumentos:
%                  u0,u1   variables relativas esquina1
%                   a1     anchura esquina1.
%      salida:
%                 x,y    coordenadas respecto a sistema
%                        esquina1.
   y=-u0;
   x=a1*(1-u1);
```

B.2.10 x2vr.m

```
 function [x,y]=vr2x(u0,u1,a1)
%
%  [x,y]=vr2x(u0,u1,a1)
%
%% Cambio de variables relativas al obstáculo esquina con origen
% en (0,0) y sistema sin rotar.

%      argumentos:
%                  u0,u1   variables relativas esquina1
%                   a1     anchura esquina1.
%      salida:
%                 x,y    coordenadas respecto a sistema
%                        esquina1.
   y=-u0;
   x=a1*(1-u1);
```

B.2.11 x2xp.m

```
 function [x1,y1]=x2xp(tita,d01,p)

%
```

```
%  [x1,y1]=xp2x(tita,d01,p)
%
%            transforma variables de sistema base a sistema
%            rotado un angulo tita y trasladado d01.
%  entrada:
%             tita  angulo de rotacion del sistema de referencia.
%             en grados.
%             d01   [xorig,yorig] vector de traslacion.
%             p     [xp,yp] punto en el sistema base.
%
%  salida: [x1,y1] coordenadas sistema rotado y trasladado.
%
tita=tita*pi/180;
R=[cos(tita) -sin(tita) 0
   sin(tita) cos(tita)  0
   0         0          1];

 % comprueba d01 vector columna

[fil,co]=size(d01);
if co~=1
  d01=d01';
end
D=[d01;0];

 % comprueba d01 vector columna

[fil,co]=size(p);
if co~=1
  p=p';
end
P=[p;0;1];

H1=[R' -R'*D;zeros(1,3) 1];
PP=sparse(H1)*sparse(P);
PP=full(PP);
x1=PP(1);
y1=PP(2);
```

B.2.12 xp2x.m

```
 function [x,y]=xp2x(tita,d01,pp)

%
%  [x,y]=xp2x(tita,d01,pp)
%
%            transforma variables de sistema rotado un angulo
%            tita y trasladado d01 a sistema base
%  entrada:
%             tita  angulo de rotacion del sistema de referencia.
%             en grados.
%             d01   [xorig,yorig] vector de traslacion.
%             pp    [xp,yp] punto en el sistema rotado y trasladado.
%
%  salida: [x,y] coordenadas en sistema base.
%
tita=tita*pi/180;
R=[cos(tita) -sin(tita) 0
   sin(tita) cos(tita)  0
```

```
   0         0          1];

 % comprueba d01 vector columna

[fil,co]=size(d01);
if co~=1
  d01=d01';
end
D=[d01;0];

 % comprueba d01 vector columna

[fil,co]=size(pp);
if co~=1
  pp=pp';
end
PP=[pp;0;1];

H=[R D;zeros(1,3) 1];
P=sparse(H)*sparse(PP);
P=full(P);
x=P(1);
y=P(2);
```

B.2.13 vr2xp.m

```
 function [x1,y1]=vr2xp(u0,u1,a2)
%
%  [x1,y1]=vr2xp(u0,u1,a2)
%
%      argumentos:
%                  u0,u1   variables relativas esquina2
%                   a2     anchura esquina2.
%      salida:
%                 x1,y1    coordenadas respecto a sistema
%                          esquina2.
   y1=u0;
   x1=a2*(1-u1);
```

B.2.14 xp2vr.m

```
   function [u0,u1]=xp2vr(pp,a2)

%
%   [u0,u1]=xp2vr(pp,a2)
%
%          convierte coordenadas sistema esquina2 a coordenadas
%          relativas.
%
%          pp   coordenadas (x,y) sistema esquina2
%          u0,u1 variables relativas.
%          anchura esquina.
    x=pp(1);
    y=pp(2);
    u0=y;
    u11=-x+a2;
    u12=-x;
    u1=u11./(u11-u12);
```

B.2.15 vr2xpob1.m

```
   function [x,y]=vr2xob1(u0,u1,a)
%
% function  [x,y]=vr2xob1(u0,u1,a)
%
%       calcula variables relativas para obtaculo1 a partir de
%       coordenadas dadas con referencia al sistema de coordenadas
%       base.
%
%        (x,y) correspondiente a la posicion robot.
%        a     ancho de obstaculo.
 y=-u0;
 x=(u1*a);
```

B.2.16 xp2vrob1.m

```
   function [u0,u1]=x2vrob1(x,y,a)
%
% function  [u0,u1]=x2vrob1(x,y,a)
%
%       calcula variables relativas para obtaculo1 a partir de
%       coordenadas dadas con referencia al sistema de coordenadas
%       base.
%
%        (x,y) correspondiente a la posicion robot.
%        a     ancho de obstaculo.
 u0=-y;
 u11=-x;
 u12=a.*ones(size(x))-x;
 u1=u11./(u11-u12);
```

B.3 SISTEMA CONTROL NEURO-FUZZY SOBRE SULTAN I.

Listados en C e Interactive C para controlar el robot Sultan I mediante un enlace de Radio frecuencia. El programa en C se ejecuta en el PC mientras que el realizado en Interactive C lo hace en el prototipo. Obsérvese que el programa movrobo2.c se convierte en una función mex, de tal forma que únicamente se ha de introducir en el script p2ob2.m como una función en sustitución del código correspondiente a la simulación del robot de cara a aplicar la estrategia neuro-fuzzy.

B.3.1 movrobo2.c

```
/*
 *  movrobo2.C
 *
```

```
 *
 *  The calling syntax is:   [x,y] = movrobo2(grados,xref,yref,dist)
 *
 * grados ángulo en radianes que debe girar el robot.
 * xref,yref  coordenadas del sistema de referencia definido por el
robot frente al sistema de referencia definido en simulación.
 * dist  distancia a recorrer por el robot en un movimiento.
 * x,y coordenadas según sistema de referencia utilizado en simulación
de la posición final del robot.
Nota: compilado con Microsoft C versión 6.00.

Must be compiled with /Alfw and /Gs option to generate proper code.
 */

#include <math.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <sys\stat.h>
#include <stdio.h>
#include <io.h>
#include <bios.h>
#include <dos.h>
#include "cmex.h"


                      /* definicion de constantes */


#define Nsensores 12    /* numero de bytes recibidos */
#define CAR_CONTROL '^'  /* caracter de control */
#define MASCARA 0X07    /* mascara para valor absoluto motores*/
#define MASCARA1 0X0F  /* mascara tomar niddle bajo*/
#define Nmotores 4     /* numero de bytes enviados */
#define Nveces 3 /* numero de veces se envian comandos motores*/
#define COTA_GIRO 10 /* offset giro (§) */

#define NENVIA 1 /* numero de veces enviando datos a los motores */

       /* coordenadas de origen del robot respecto al origen
                                                 sistema neuro-fuzzy */

#define PI (3.14159265358979)


#define MINDELAY 50 /* numero de mseg que espera para emitir */


                      /* definicion de estructuras */
 typedef struct {
             unsigned char control;
             unsigned char izquierda;
             unsigned char centro;
             unsigned char derecha;
             short int x;
             short int y;
             short int grados;
             char paridad;
             char stop;
           } SENSORES;
```

```
 typedef struct {
             unsigned izq_motor:3;
             unsigned der_motor:3;
             unsigned signo_izq:1;
             unsigned signo_der:1;
           } MOTORES;



double leer_grad();
movrobo2(double *x,double *y,double *grados,double *x0,double
*y0,double *d0);
leer_xy(double *x,double *y);
consigna_motores(MOTORES giro);
motorola2intel(int tam,void *preal,int unidades);



/*
 * Input Arguments
 */
#define GRADOS_IN prhs[0]
#define X0_IN     prhs[1]
#define Y0_IN     prhs[2]
#define DISTANCIA_IN prhs[3]

/*
 * Output Arguments
 */
#define X_OUT   plhs[0]
#define Y_OUT   plhs[1]




/*
 * MATLAB functions have the form
 *    [a,b,c,...] = fun(d,e,f,...)
 * where the a,b,c are "left-hand-side" arguments and the d,e,f,...
are
 * "right-hand-side" arguments.
 */

user_fcn(nlhs, plhs, nrhs, prhs)
int nlhs;       /* Number of left-hand-side arguments */
int nrhs;       /* Number of right-hand-side arguments */
Matrix *plhs[]; /* Length nlhs array for pointers to the lhs arguments
*/
Matrix *prhs[]; /* Length nrhs array of pointers to the rhs arguments
*/
{
      double *x,*y,*x0,*y0,*d0;
      double *grados;
      unsigned int m, n;

      /*
       * Check for proper number of arguments
       */
```

```
      if (nrhs != 4) {
            mex_error("movrobo2 requires four input arguments.");
      } else if (nlhs != 2) {
            mex_error("movrobo2 requires two output arguments.");
      }

      /*
       * Check the dimensions of Y.  Y can be 4 X 1 or 1 X 4.
       */
      m = GRADOS_IN->m;
      n = GRADOS_IN->n;
      if ((max(m,n) != 1) || (min(m,n) != 1)) {
            mex_error("movrobo2 requires that Grados be a 1 x 1
vector.");
      }
      m = X0_IN->m;
      n = X0_IN->n;

      if ((max(m,n) != 1) || (min(m,n) != 1)) {
            mex_error("movrobo2 requires that x0 be a 1 x 1 vector.");
      }

      m = Y0_IN->m;
      n = Y0_IN->n;

      if ((max(m,n) != 1) || (min(m,n) != 1)) {
            mex_error("movrobo2 requires that y0 be a 1 x 1 vector.");
      }

      m = DISTANCIA_IN->m;
      n = DISTANCIA_IN->n;

      if ((max(m,n) != 1) || (min(m,n) != 1)) {
            mex_error("movrobo2 requires that distancia be a 1 x 1
vector.");
      }

      /*
       * Create a matrix for the return argument
       */
      X_OUT = create_matrix(1, 1, REAL);
      Y_OUT = create_matrix(1, 1, REAL);

      /*
       * Assign pointers to the various parameters
       */
      x = X_OUT->pr;
      y = Y_OUT->pr;
      grados = GRADOS_IN->pr;
      x0= X0_IN->pr;
      y0= Y0_IN->pr;
      d0= DISTANCIA_IN->pr;
      /*
       * Do the actual computations in a subroutine
       */
      movrobo2(x,y,grados,x0,y0,d0);
}

 movrobo2(x,y,mat_grados,x0,y0,d0)
 double *mat_grados,*x,*y,*x0,*y0,*d0;
{
```

```
  /* descripcion: mueve el robot
                                    grados angulo en radianes
                                    x,y    coordenadas a las que llega

                                    */

 double grad0_robot,xrobot,yrobot,x0robot,y0robot,distancia;
 double xneufuzzy,yneufuzzy;
 double grad_angulo,g,*grados,grad_c;
 int giro,i,fp,c;
 MOTORES
girar_der,girar_izq,rapido_girar_der,rapido_girar_izq,parar,recto;
 struct timeb timeinicial,timefinal;
 grad_c=(*mat_grados);
 grados=&grad_c;

 /* giros especificados en funcion de los motores */

 girar_der.izq_motor=1;
 girar_der.der_motor=0;
 girar_der.signo_izq=0;
 girar_der.signo_der=0;

 rapido_girar_der.izq_motor=2;
 rapido_girar_der.der_motor=0;
 rapido_girar_der.signo_izq=0;
 rapido_girar_der.signo_der=0;

 girar_izq.izq_motor=0;
 girar_izq.der_motor=1;
 girar_izq.signo_izq=0;
 girar_izq.signo_der=0;

 rapido_girar_izq.izq_motor=0;
 rapido_girar_izq.der_motor=2;
 rapido_girar_izq.signo_izq=0;
 rapido_girar_izq.signo_der=0;


 parar.izq_motor=0;
 parar.der_motor=0;
 parar.signo_izq=0;
 parar.signo_der=0;

 if ((parar.signo_der==1)&&(parar.der_motor==0))
 {
  printf("error parar en la elecci¢n de signo en motores");
  exit(1);
 }
 if ((parar.signo_izq==1)&&(parar.izq_motor==0))
 {
  printf("error parar en la elecci¢n de signo en motores");
  exit(1);
 }


 recto.izq_motor=2;
 recto.der_motor=2;
 recto.signo_izq=0;
 recto.signo_der=0;
```

```
 if ((recto.signo_der==0)&&(recto.der_motor==0))
 {
  printf("error recto en la elecci¢n de signo en motores");
  exit(1);
 }
 if ((recto.signo_izq==0)&&(recto.izq_motor==0))
 {
  printf("error recto en la elecci¢n de signo en motores");
  exit(1);
 }



   /* abrir puerto serie y inicializarlo */



_bios_serialcom(_COM_INIT,0,_COM_1200|_COM_NOPARITY|_COM_STOP1|_COM_CH
R8);



   /* convertir grados al rango -180§, 180§. */

        grad_angulo=(*grados)*180/(PI);
        if ((grad_angulo>360)||(grad_angulo<-360))
        {
         printf("error sistema Neuro-Fuzzy valor angulo mas de una
vuelta");
         exit(1);
        }
        if (grad_angulo>=0)
        {
         if (grad_angulo>180)
         {
              (*grados)=grad_angulo-360;
         }
         else
         {
              (*grados)=grad_angulo;
         }
        }
        else
        {
         if (grad_angulo>-180)
         {
              (*grados)=grad_angulo;
         }
         else
         {
              (*grados)=360+grad_angulo;
         }
        }

 /*
              while ((c = getchar()) != '\n')
                   printf("%c", c);
             consigna_motores(parar);
              while ((c = getchar()) != '\n')
              printf("%c", c);
              grad0_robot = leer_grad();
```

```
            while ((c = getchar()) != '\n')
         printf("%c", c);

       consigna_motores(girar_der);

        while ((c = getchar()) != '\n')
        printf("%c", c);

       consigna_motores(girar_izq);

            while ((c = getchar()) != '\n')
         printf("%c", c);


       consigna_motores(rapido_girar_der);

            while ((c = getchar()) != '\n')
         printf("%c", c);

       return 0;*/




/* orientar el robot */

giro=COTA_GIRO+1;
while (abs(giro) > COTA_GIRO)
{


                  grad0_robot = leer_grad();
                  giro=(int) ((*grados)-grad0_robot);
                  if (abs(giro) <= COTA_GIRO) break;

  printf("grad0= %lf \n",grad0_robot);
  printf("giro= %d \n",giro);


  if (giro>0)
  {
       if (abs(giro)>40)
       {
        consigna_motores(rapido_girar_izq);
       }
       else
       {
        consigna_motores(girar_izq);
       }

  }
  else if (giro<0)
  {

        if (abs(giro)>40)
       {
        consigna_motores(rapido_girar_der);
       }
```

```
             else
             {
              consigna_motores(girar_der);
             }



        }

      } /* fin bucle */

      for (i=0;i<NENVIA;i++)
      {
       consigna_motores(parar);

      }

  /* para depurar mostrar grados finales */

        grad0_robot = leer_grad();
        printf("grados final=%lf",grad0_robot);


      /* movimiento */

                    leer_xy(&x0robot,&y0robot); /* determina coordenadas
iniciales*/


            consigna_motores(recto);



       do{
              leer_xy(&xrobot,&yrobot);

             /* calcular distancia */

             distancia=sqrt(pow((xrobot-x0robot),2)+pow((yrobot-
y0robot),2));

             /* visualiza distancia */

                   printf("d=%lf",distancia);

            } while (distancia < (*d0));

            for (i=0;i<NENVIA;i++)
            {
             consigna_motores(parar);

            }
                    leer_xy(&xrobot,&yrobot);

             /* devolver valores funcion mex */

              /* realizar translacion sobre coordenadas para
                    adaptarse al sistema neurofuzzy */

              xneufuzzy=(*x0)+yrobot;
              yneufuzzy=(*y0)-xrobot;
```

```
              (*x)=xneufuzzy;
              (*y)=yneufuzzy;
              printf("xd:%lf",xneufuzzy);
              printf("yd:%lf",yneufuzzy);
              printf("yrobot:%lf",yrobot);
              printf("xrobot:%lf",xrobot);

}

double leer_grad()
{
 double valor;
 int fp,i,estado,lee,estado1;
 char ch;
 char cadena1[] = "aaaaaaaaaaa";
 char cadena[] = "aaaaaaaaaaaa";
 SENSORES datos,*pdatos;
 unsigned in;

  /* Descripcion: lee grados del robot colocando el valor en
              una variable tipo double.
              lee entre -180§ y +180§         */

 pdatos=&datos;
 ch='e';

 do {

                              /*Recibimos el dato en el byte mas
bajo.*/

                         in=_bios_serialcom(_COM_RECEIVE, 0, 0);
                         in=in & (0xFF);
                         ch=(char)in;


       } while ((ch) != (CAR_CONTROL));



        for (i=0;i<11;i++)
            {


            /* Recibimos el dato en el byte mas bajo. */

                         in=_bios_serialcom(_COM_RECEIVE, 0, 0);
                         in=in & (0xFF);
                         ch=(char)in;
                         cadena1[i]=ch;

            }

      cadena[0]=ch;
      cadena[1]='\0';
      memcpy(pdatos,cadena,1);
      memcpy(&(pdatos->izquierda),cadena1,11);
      motorola2intel(sizeof(pdatos->x),&(pdatos->x),3);
      valor=(double) (pdatos->grados);

/*   visualizacion de variables leidas    */
```

```
/*      printf("stdaux: %d",fileno(stdaux));
      printf("ocupa:%d",sizeof(SENSORES));*/
      printf("pdatos->control=%c",pdatos->control);
      printf("pdatos->izquierda=%c",pdatos->izquierda);
      printf("pdatos->centro=%c",pdatos->centro);
      printf("pdatos->derecha=%c",pdatos->derecha);
      printf("pdatos->x=%d",pdatos->x);
      printf("pdatos->y=%d",pdatos->y);
      printf("pdatos->grados= %lf",valor);
      printf("pdatos->paridad=%c",pdatos->paridad);
      printf("pdatos->stop=%c",pdatos->stop);


      return(valor);
}

consigna_motores(giro)
 MOTORES giro;

 {
      /* Descripcion: especifica consigna a los servomotores. */

 int estado,fp,i,k,r;
 long int j=0;
 unsigned status,in;
 unsigned out[3];
 char cadena[Nmotores+1],mizq,mder,byte_vel,sder,sizq,ch;
 struct timeb timeinicial,timefinal,time1inicial,time1final;
 char *cadenapr="ss",*final="ªª";

 status=0x00;

      sprintf(cadena,"%s","aaaa");
      cadena[0]='^';
      mizq=(giro.izq_motor) & (MASCARA);
      sizq=(giro.signo_izq) & (MASCARA1);
      sizq=sizq<<3;
      mizq=mizq|sizq;
      mizq=mizq<<4;
      mder=(giro.der_motor) & (MASCARA);
      sder=(giro.signo_der) & (MASCARA1);
      sder=sder<<3;
      mder=mder|sder;
      byte_vel=mizq|mder;



      for (i=1;i<=Nveces;i++)
      {
       cadena[i]=byte_vel;
      }

  do {


                  for (i=0;i<2;i++)
                  {
                       in=_bios_serialcom(_COM_RECEIVE, 0, 0);
                       in=in & (0xFF);
```

```
                        cadenapr[i]=(char)in;
                    }


       } while (strcmp(cadenapr,final)!= 0);
       printf("cadenapr:%c",cadenapr[0]);

             ftime(&timeinicial);
      do
      {
       ftime(&timefinal);
      } while ((timefinal.millitm-timeinicial.millitm)< MINDELAY);



           for (i=0;i<=4;i++)
           {
                 out[i]=(unsigned) cadena[i];
                 out[i]=out[i] & (0X00FF);
           }
       for (r=0;r<5;r++)
       {
             for (i=0;i<4;i++)
             {
              do
                   {
                    printf("out[%d]=%x",i,out[i]);
                    status=_bios_serialcom(_COM_SEND,0,out[i]);
                    status=_bios_serialcom(_COM_STATUS,0,0);
                    printf("status:%x",status);

                   } while ((status & 0x80)==1);
             }
                         k=0;
                         for (j=0;j<10000;j++)
                        {
                         k++;
                        }
       } /* fin bucle*/



 /* ver variables */

      printf("byte_vel=%x",byte_vel);
      printf("mizq=%x",mizq);
      printf("mder=%x",mder);
  /*  printf("cadena=%s",cadena);*/
      printf("giro->izquierda=%x",giro.izq_motor);
      printf("giro->derecha=%x",giro.der_motor);

       }


 leer_xy(x,y)
 double *x,*y;
{

 int fp,i,estado;
 char ch;
 char cadena1[] = "aaaaaaaaaaa";
```

```
 char cadena[] = "aaaaaaaaaaaa";
 unsigned in;
 SENSORES datos,*pdatos;

  /* Descripcion: lee x e y del robot colocando el valor en
              estas variables tipo double en cm.
                                                    */

 pdatos=&datos;
 ch='e';

  do {

                              /*Recibimos el dato en el byte mas
bajo.*/

                         in=_bios_serialcom(_COM_RECEIVE, 0, 0);
                         in=in & (0xFF);
                         ch=(char)in;

       } while ((ch) != (CAR_CONTROL));



        for (i=0;i<11;i++)
            {


            /* Recibimos el dato en el byte mas bajo. */

                         in=_bios_serialcom(_COM_RECEIVE, 0, 0);
                         in=in & (0xFF);
                         ch=(char)in;
                         cadena1[i]=ch;

            }

      cadena[0]=ch;
      cadena[1]='\0';
      memcpy(pdatos,cadena,1);
      memcpy(&(pdatos->izquierda),cadena1,11);
      motorola2intel(sizeof(pdatos->x),&(pdatos->x),3);
      (*x)=(double) (pdatos->x);
      (*y)=(double) (pdatos->y);

/*   visualizacion de variables leidas  */

/*      printf("cadena=%s",cadena);
      printf("stdaux: %d",fileno(stdaux));
      printf("ocupa:%d",sizeof(SENSORES));
      printf("pdatos->control=%c",pdatos->control);
      printf("pdatos->izquierda=%c",pdatos->izquierda);
      printf("pdatos->centro=%c",pdatos->centro);
      printf("pdatos->derecha=%c",pdatos->derecha);
      printf("pdatos->x=%d",pdatos->x);
      printf("pdatos->y=%d",pdatos->y);
      printf("pdatos->grados= %d",pdatos->grados);
      printf("pdatos->paridad=%c",pdatos->paridad);
      printf("pdatos->stop=%c",pdatos->stop);  */

}
```

```
       /* rutina de conversion de datos de motorola a intel */

motorola2intel(tam,preal,unidades)
int tam,unidades;
void *preal;
{
 /* Descripcion: Convierte un numero del formato utilizado
             por los procesadores motorola 68000 a los
             utilizados por procesadores intel o viceversa.
       parametros:
             tam       tama·o en bytes del tipo de dato
                        (real,double,int ...)
             unidades  numero de datos.
             preal     puntero a los datos a convertir */
 char *ptr_real,*ptr_buf;
 int i,j,k;

 ptr_real=(char *) preal;
 ptr_buf=(char *) malloc(tam);
 if (ptr_buf==NULL)
 {
  printf("(error10(conversion a intel)- memoria insuficiente");
  exit(1);
 }
 for (k=1;k<=unidades;k++)
 {
  j=tam-1;
  for (i=0;i<tam;i++)
  {
      *(ptr_buf+i)=*(ptr_real+j);
      j--;
  }
  for (j=0;j<tam;j++)
  {
      *(ptr_real+j)=*(ptr_buf+j);
  }
  ptr_real+=tam;
 }
}
```

B.3.2 interp2.c

```
Programa desarrollado en interactive C, para se ejecutado en Sulta I.
/*
*****************************************************************
*/
/*      PROGRAMA PARA CONVERTIR AL ROBOT EN UN
*/
/*      INTERPRETE DE COMANDOS EXTERNOS
*/
/*      INTERFAZ SERIE CON ORDENADOR EXTERNO
*/
/*****************************************************************
***/
/*     Jose Julio Rodrigo Bello           2/03/99
*/
```

```
/*
*****************************************************************
*/

int servo_dir;        /* direccion de movimiento del servo */
int jservo_pos;       /* posicion del servo */

int array_sonar[5];

int array_luz[5];
int ldr_izq,ldr_der;

int ant_dist = 0;
int ant_compass = 0;
int compass_ini = 0;

/* variables booleanas */
int lectura_bruj_iniciada = 0;

float x,y;
int dist, c;
int velocidad;

int baudios;
int ser_byte;     /* byte recibido por el puerto serie */

int motizq;       /* variables para los comandos de los motores */
int motder;

int dirizq;       /* direccion de los motores */
int dirder;



/* Este procedimiento se encarga de realizar periodicamente la lectura
*/
/* de los sensores del sonar, la brujula y el codificador incremental
*/
/* Periodicamente envia los datos a traves del puerto serie */
void Lectura_sensores()
{
      while(1)
            {
            /* movemos el servo hasta la posici¢n adecuada */
            if (jservo_pos == 1)
                  {
                  /*servo (2200);*/
                  time_var1 = 0;
                  sistema_posicionamiento();
                  }
            else
                  if (jservo_pos == 3)
                        {
                        /*servo (4000);*/
                        time_var1 = 0;
                        start_process(enviar_datos());
                        }
                  else
                        {
                        /*servo (3100);*/
                        time_var1 = 0;
```

```
                        jservo_pos == 2;
                        }

            while (time_var1 < 160)        /* esperamos 160 ms */
                  defer();

            sonar_init(0);    /* iniciamos lectura del sonar... */
            time_var1 = 0;
            while (time_var1 < 12)  /* esperamos hasta 12ms */
                  defer();

            array_sonar[jservo_pos] = (sonar_time >> 6);

            /* Preparamos las variables de colocacion del servo */
            if (servo_dir == 0)
                  {
                  jservo_pos = jservo_pos + 1;
                  if (jservo_pos > 2)
                        {
                        servo_dir = 1;
                        jservo_pos = 3;
                        }
                  }
            else
                  {
                  jservo_pos = jservo_pos - 1;
                  if (jservo_pos < 2)
                        {
                        servo_dir = 0;
                        jservo_pos = 1;
                        }
                  }
            }
}

/* Este procedimiento es el encargado de enviar el comando a los
motores */
void ejecutar_comando(int ser_byte)
{
      /* lo primero que vamos a hacer es dividir la parte del motor */
      /* izquierdo y la del derecho */
      motder = ser_byte & 0b00000111;
      motizq = (ser_byte & 0b01110000) >> 4;
      if ((ser_byte & 0b00001000) != 0)   /* comprobamos las
direcciones */
            dirder = 1;    /* marcha atras */
      else
            dirder = 0;
      if ((ser_byte & 0b10000000) != 0)
            dirizq = 1;
      else
            dirizq = 0;

      /* activamos los motores */
      _set_motor(0,dirizq,motizq);
      _set_motor(1,dirder,motder);

}

/* Esta funci¢n comprueba continuamente si se ha recibido algo por el
*/
```

```
/* puerto serie. Si se recibe algo se comprueba si es correcto */
/* (hay doble redundancia) y se envia el comando a los motores */
void Recibir_comando()
{
      while(1)
            {
            if (recibido == 1)
                  {
                  /* comprobamos si el byte es correcto con la
redundancia...*/
                  if (brx1 == brx2)
                        {
                        ser_byte = brx1;
                        ejecutar_comando(ser_byte);
                        }
                  else
                        if (brx2 == brx3)
                              {
                              ser_byte = brx2;
                              ejecutar_comando(ser_byte);
                              }
                        else
                              if (brx1 == brx3)
                                    {
                                    ser_byte = brx1;
                                    ejecutar_comando(ser_byte);
                                    }
                              else
                                    /* Error en la recepcion */
                                    beep();
                  recibido = 0;
                  }
                  defer();   /* soltamos el procesador */
            }
}


/* Esta funcion detiene al robot y da un aviso sonoro */
void Emergencia()
{
      _set_motor(0,0,0);   /* nos paramos */
      _set_motor(1,0,0);
      msleep(200L);
      beep();
      msleep(500L);
      beep();
      msleep(500L);
      beep();
      msleep(500L);
}

/* Procedimiento para leer el valor de la brujula del puerto SPI */
void brujula()
{
      bit_clear(0x0e, 4);     /* P/C  a 0*/
      msleep(12L);
      bit_set(0x0e, 4);   /* P/C  a 1*/
      while (digital(15) == 1);
      msleep(12L);
      bit_clear(0x0e, 8);     /* SS   a 0*/
      msleep(10L);
```

```
      defer();                /* para conseguir los 5 mseg de quantum
*/
      leer_bruj(0);
      bit_set(0x0e, 8);     /* SS   a 1*/
      msleep(10L);
}


/* Procedimiento para leer la brujula y el codificador incremental */
/* tambien calcula las coordenadas cartesianas X,Y del robot */
void sistema_posicionamiento()
{
      float recorrido;

      brujula();  /* llamamos a la brujula */

      /* calculamos el azimuth con respecto al eje Y inicial... */
      c = compass - compass_ini;
      if (c < -180)
            c = c + 360;
      else if (c > 180)
            c = c - 360;

      velocidad = distancia - ant_dist;    /* distancia recorrida */
      recorrido = (float)velocidad * 0.62;  /* pasamos a
centimetros... */
      ant_dist = distancia;

      /* por £ltimo calculamos las coordenadas cartesinas de la */
      /* posici¢n del robot con respecto al centro de coordenadas
inicial */
      y = y + recorrido * cos ( (float)c * 0.0174533);
      x = x - recorrido * sin ( (float)c * 0.0174533);

}

/* Convierte el entero a BCD y lo envia por el puerto serie */
void num_to_serie(int num)
{
      int signo;
      int centenas;
      int decenas;
      int unidades;
      int aux;

      aux = num;
      signo = 0;

      if (aux<0)
            {
            signo = 1;
            aux = -1*aux;
            }
      centenas = aux / 100;
      decenas = (aux - centenas*100) / 10;
      unidades = aux - centenas*100 - decenas*10;

      if (signo==1)
            serout('-');
      else
            serout('+');
```

```
      serout(centenas+'0');
      serout(decenas+'0');
      serout(unidades+'0');
}


/* Envia los datos por el puerto serie */
void enviar_datos()
{
      int valorizq;
      int valorder;
      int aux;
      int basura;

      /* lo primero que haremos ser  deshabilitar la Recepcion de
datos */
      /* porque la comunicaci¢n va a ser Half Duplex */
      RXenable = 0;

      if (dirizq == 0)
            valorizq = motizq;
      else
            valorizq = -motizq;
      if (dirder == 1)
            valorder = motder;
      else
            valorder = -motder;

      /* mostramos los valores principales en el LCD */
      printf("\nC:%d X:%d Y:%d   MI:%d
MD:%d",c,(int)x,(int)y,valorizq,valorder);

      serout('^');               /* byte de comienzo */

      serout(array_sonar[1]);    /* 3 bytes del sonar I,C,D */
      serout(array_sonar[2]);
      serout(array_sonar[3]);
                                                  /* Ahora
mandamos la posici¢n X,Y */
      aux = ((int)x) >> 8;
      serout(aux);             /* byte alto de X */
      aux = (int)x;
      serout(aux);                     /* byte bajo de X */
      aux = ((int)y) >> 8;
      serout(aux);             /* byte alto de Y */
      aux = (int)y;
      serout(aux);                     /* byte bajo de Y */

                                                  /* enviamos el
valor de la brujula */
      aux = c >> 8;
      serout(aux);                    /* byte alto de la brujula */
      serout(c);              /* byte bajo de la brujula */

      serout('ª');                           /* 2 bytes finales :
codigo 170 */
      serout('ª');

      /* ahora habilitamos la Recepci¢n */
      basura = serin();     /* vaciamos el buffer de recepcion */
      RXenable = 0xFFFF;
```

```
}


/* Funci¢n para quitar el control del puerto serie al Interactive C */
void disable_pcode_serial()
{
      poke(0x3c,1);
}

/* Funci¢n para devolver el control del puerto serie al Interactive C
*/
void reenable_pcode_serial()
{
      poke(0x3c,0);
}

/* Funci¢n para enviar un byte por el puerto serie */
void serout(int c)
{
      while (!(peek(0x102e) & 0x80)); /* esperamos hasta que se vacie
el */
                                                      /* buffer de
transmisi¢n */
      poke(0x102f,c);   /* enviamos el byte al registro TDR */
}

/* Funci¢n para recibir bytes por el puerto serie */
/* primero comprueba si hay algun byte que leer */
int serin()
{
      if (peek(0x102e) & 0x20)
            {
            ser_byte = peek(0x102f);
            return 1;
            }
      else
            return 0;   /* no hay ningun caracter que recibir */
}

/* Funci¢n para ver los baudios*/
void verbaudios()
{
      baudios = peek(0x102b);
}

/* Colocamos la UART a 9600 baudios */
void baudios9600()
{
      poke(0x102b,0x0030);
}

/* Colocamos la UART a 1200 baudios */
void baudios1200()
{
      poke(0x102b,0x0033);
}

/* Colocamos la UART a 300 baudios */
void baudios300()
{
      poke(0x102b,0x0035);
```

```
}


void main()
{

      sonar_on(0);     /* activamos los distintos modulos (ficheros
.ICB) */
      servo_on();
      dist_on(0);
      bruj_on(0);

      disable_pcode_serial();  /* tomamos el control del puerto serie
*/

      baudios1200();     /* pasamos a 1200 baudios (UART 68HC11) */

      RXenable = 0;      /* deshabilitamos la recepci¢n al inicio...
*/
      /*servo (3100);*/      /* posicion inicial del servo */
      jservo_pos = 2;
      servo_dir = 0;
      time_var3 = 0;

      printf("\n Pulse START... ");
      start_press();
      msleep(500L);
      printf("\n\n");

      /* pedimos al usuario que situe el robot en 0,0 y sobre el eje Y
*/
      printf("\nOrigen de Coordenadas   Pulse START");
      start_press();
      printf("\n\n");
      brujula();
      compass_ini = compass;      /* inicializamos el eje Y = 0§
azimuth */
      x = 0.0;
      y = 0.0;

      msleep(500L);
      beep();

      /* enviamos la cadena 'ok' por el puerto serie */
      serout('o');serout('k');

      start_process(Lectura_sensores());
      start_process(Recibir_comando());

}
```

Bibliografía

BIBLIOGRAFIA

[ACO96] L. Acosta, Marichal G. N., A. Hamilton, Méndez J.A. , Moreno L. *Design of a Dynamical Neural Network structure for system Identification*.IIA'96/SOCO'96 Proccedings of International ICSC Symposia on Intelligent Industrial Automation and Soft Computing. Editors: P. G. Anderson / K. Warwick. University of Reading, U.K. pp. A77-A81. 1996.

[ACO97a] L. Acosta, G.N. Marichal, A. Hamilton, J.A. Méndez, L. Moreno. *Design of a Dynamical Neural Network Structure for System Identification*. Automatica. (En segunda revisión). 1997.

[ACO97b] L. Acosta , Moreno L., Dimitrova M., A. Hamilton, Méndez J.A., Marichal G. N. *A Genetic Algorithm Approach To Robot Modelling* Proceedings of Second International ICSC Symposium on Soft Computing SOCO'97, pp. 238-242. Nimes, 1997.

[ACO98] L. Acosta, G.N. Marichal, L. Moreno, J. J. Rodrigo, A. Hamilton, J.A. Méndez. *An Robotic system based on neural network controllers*. Artificial Intelligence in Engineering. (aceptado para publicación)

[AMA80] S.-I. Amari. *Topographic organization of nerve fields*. Bull. Math. Biology, vol. 42, pp. 339-364,1980.

[ANT90] Panos J. Antsaklis. *Neural Networks in Control Systems*. Control System Magazine, April 1990.

[AST80] Aström K. J. *Self-tuning Regulators. Design Principles and Applications*. Editores: Narendra K. S., Monopoli R. V. Applications of Adaptive Control. Academic Press. 1980.

[BER93] R. H. Berry, G. D. Smith. *Using a genetic algorithm to investigate taxation-induced interactions in capital budgeting*. In Proceedings of the International Conference on Neural Networks and Genetic aAlgorithms. Springer-Verlag, Innsbruck.1993

[BOR92] S. Bornholdt, D. Graudenz. *General asymmetric neural networks and structure design by genetic algorithms*. Neural Networks, Vol 5, pp. 327-334. 1992.

[CHE90] D. L. Chester. *Why two hidden layers are better than one*. In Proceedings of the International Joint Conference on Neural Networks, vol. 1, pp. 265-268. Erlbaum, 1990

[CYB89] G. Cybenko. *Approximation by superpositions of a sigmoidal functio*n. Mathematics of Control, Signals, and Systems, Vol. 2, pp. 303-314, 1989

[DAY93] Shawn P. Day, Michael R. Davenport. *Continuous-Time Temporal Back-Propagation with Adaptable Time Delays*. IEEE Transactions on Neural Networks, Vol. 4, No. 2, pp 348-354. 1993.

[DEM93] Howard Demuth, Mark Beale. *Neural Network Toolbox. User's Guide*. The MathWorks, Inc. 1993

[DRA92] Gian Paolo Drago, Sandro Ridella. *Statistically Controlled Activation Weight Initialization (SCAWI)*. IEEE Transactions on Neural Networks, Vol. 3, NO. 4, pp. 627-631,1992.

[FRE93] James A. Freeman , David M. Skapura. *Redes neuronales Algoritmos, aplicaciones y técnicas de programación*. Addison-Wesley/ Diaz de Santos. 1993.

[FU87] K. S. Fu, R. C. Gonzalez , C. S. G. Lee. *Robótica. Control, Detección, Visión e Inteligencia*. Mc. Graw Hill. 1987

[FUE93] J. L. De La Fuente O'Connor. *Tecnologías computacionales para sistemas de ecuaciones, optimización lineal y entera*, Editorial Reverté, S. A., 1993.

[FUN93] Ken-Ichi Funahashi, Juichi Nakamura. Approximation of Dynamical Systems by Continuos Time Recurrent Neural Networks. Neural Networks, Vol. 6, pp. 801-806, 1993

[GOL89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine learning*. Addison- Wesley, Reading, Mass.

[GOO84] Goodwin G. G., Sin, K. S. *Adaptive Filtering Prediction and Control*. Prentice Hall. 1984.

[GOR90] M. Gori, G. Soda. *Temporal pattern recognition using EBPS*. EURASIP, 1990

[GRA84] R. M. Gray. *Vector quantization*. IEEE ASSP Magazine, vol. 1, pp. 4-29, 1984.

[GUP94] M. M. Gupta, D. H. Rao. *On the Principle of Fuzzy Neural Networks. Fuzzy Sets and Systems*, vol. 61, pp. 1-18. 1994

[HAM95] F. Hamilton, Tesis Doctoral: *Estrategias de control óptimo basadas en programación dinámica y redes neuronales para sistemas MIMO continuos no lineales*. Dpto. de Física Fundamental y Experimental. Universidad de La Laguna, 1995.

[HAR91] S. A. Harp, T. Samad. *Genetic synthesis of neural network architecture*. In Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York. pp. 202-221. 1991.

[HER91] J. Hertz, A. Krogh, R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

[HEW92] *HP-UX Reference. Release 9.0, HP 9000. Sections 1,2,3,4 and 1M*. Hewlett-Packard. 1992.

[HOL75] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI. 1975

[HOP82] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the National Academy of Science USA, vol. 79 pp. 2554-2558, 1982

[HOP84] J. J. Hopfield. *Neurons with a graded response have collective computational properties like those of two-state neurons*. Proceedings of the National Academy of Science USA, vol. 81, pp. 3088-3092, May 1984

[HOR89] K. Hornik, M. Stinchcombe, H. White. *Multi-layer feedforward networks are universal approximators*. Neural Networks, vol. 2, no.2, pp. 359-366, 1989.

[HUN92] K. J. Hunt, D. Sbarbaro, R. Zbikowski, P. J. Gawthrop. *Neural Networks for Control Systems – A Survey*. Automatica, Vol. 28, No. 6, pp. 1083-1112,1992.

[HUS93] D. R. Hush and B.G. Horne. *Progress in Supervised Neural Networks*, IEEE Signal Processing Magazine, No. 1, pp. 8-39, 1993.

[ISI89] A. Isidori, *Nonlinear Control Systems: An Introduction*. Springer-Verlag. Berlin. 1989.

[KAP93] Kapsalis, V. J. Rayward-Smith, G. D. Smith. *Solving the graphical Steiner tree problem using genetic algorithms*. Journal of the Operational Research Society. Vol 44, pp. 397-406. 1993

[KER88] B. W. Kernighan, D. M. Ritchie. *The C programming Language*. Second Edition. Pretince-Hall. 1988

[KOH87] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag. Berlin. 1987.

[KOH90] Teuvo Kohonen. *The Self-Organizing* Map. Proceedings of the IEEE, Vol. 78, No. 9, pp. 1464-1480, 1990

[LAN90] K. J. Lang, A.H. Waibel, G. E. Hinton. *A time-delay neural network architecture from isolated word recognition*. Neural Networks, Vol. 3, No. 1, pp. 23-44, 1990.

[LAP87] Lapedes, R. Farber. *Nonlinear signal processing using neural networks: Prediction and signal modeling*. Technical Report LA-UR-87-2662, Los Alamos National Laboratories, Los Alamos, New Mexico, 1987.

[LEV93] A. V. Levin , K. S. Narendra. *Control of Nonlinear dynamical systems using neural networks: Controllability and Stabilization*. IEE transactions on neural networks, Vol. 4, No. 2, pp. 192-206, March 1993

[LIN80] Y. Linde, A. Buzo, R. M. Gray. *An algorithm for vector quantization*. IEEE Trans. Communication, vol. COM-28 pp 84-95, 1980.

[LIN94] C. T. Lin and C. S. Lee: *Supervised and unsupervised learning with fuzzy similarity for neural network-based fuzzy logic control systems*. In R. R. Yager and L. A. Zadeh (Eds.): Fuzzy Sets, Neural Networks, and Soft Computing. Van Nostrand Reinold, New York, pp. 126-165. 1994

[LIP87] R. Lippmann. *An introduction to computing with neural nets. IEEE Acoustics, Speech and Signal Processing* Magazine, Vol. 4, pp. 4-22, April 1987.

[MAR91] S. J. Marshall, R. F. Harrison. *Optimization and training of feedforward neural networks by genetic algorithms*. Proc. 2nd IEE International Conference on Artificial Neural Networks, pp 39-43. 1991.

[MAR94] G. N. Marichal, Memoria de Licenciatura: *Estudio de la modelización de sistemas mediante redes neuronales: Aplicación a la modelización de un robot antropomórfico (manipulador)*. Facultad de Física, Universidad de La Laguna. 1994.

[MAR98] R. L. Marichal, Memoria de Licenciatura: *Adquisición y Análisis de Potenciales Evocados mediante Redes Neuronales*. Facultad de Física, Universidad de La Laguna, 1998.

[MAT92] MATLAB. Reference Guide. The MathWorks. 1992

[MEG93] Saï d M.Megahed. Principles of robot modelling and simulation. John Wiley & Sons. 1993

[MEN98] J. A. Méndez, Tesis Doctoral: *Desarrollo de estrategias de control predictivas de alto rendimiento mediante la incorporación de técnicas de control robusto y de redes neuronales*. Dpto. de Física Fundamental y Experimental, Universidad de La Laguna, 1998.

[MIL90] W. Thomas Miller, Richard S. Sutton, and Paul J. Werbos. *Neural Networks for Control*. The MIT Press. 1990

[MIN69] M. Minsky, S. Papert. *Perceptrons*. Cambridge, MA: MIT Press. 1969

[MIT96] M. Mitchell. *An introduction to genetic algorithms*. MIT Press.1996

[MIY88] H. Miyamoto, M. Kawato, T. Setoyama, R. Suzuki. Feedback-error-learning neural network for trayectory control of a robotic manipulator. Neural Networks, vol.1, pp. 251-265, 1988.

[MON91] D. J. Montana. *Automated parameter tuning for interpretation of synthetic images*. In Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York. pp. 282-311. 1991.

[MOR97a] L. Moreno., L. Acosta, Marichal G. N., A. Hamilton, Méndez J.A. *A Neural Network Structure for modelling Dynamic of Puma 560 Robot*. Proceedings of Second International ICSC Symposium on Intelligent Industrial Automation IIA'97, pp. 63-66. Nimes, 1997.

[MOR97b] L Moreno, L. Acosta, J. J. Rodrigo, G. N. Marichal, A. Hamilton, J.A. Méndez. *Diseño de robots móviles autónomos*. XXVI Reunión Bienal de la Real Sociedad Española de Física. Universidad de Las Palmas de Gran Canaria, 1997.

[MOR98a] L Moreno, L. Acosta , Marichal G. N. , J. J. Rodrigo., A. Hamilton, J.A. Méndez. *A robotic manufacturing subsystem based on neural networks and fuzzy logic techniques.* Proceedings of . International ICSC Symposium on Engineering of Intelligent Systems. Universidad de La Laguna, pp. 189-193. Febrero 11-13, 1998.

[MOR98b] L. Moreno, L. Acosta, G.N. Marichal, A. Hamilton, J.A. Méndez. *An approach to the Inverse Kinematics of a Redundant Planar arm by Dynamic Neural Networks.* International ICSC/IFAC Symposium on Neural Computation. Technical University in Viena. pp 772-777, 1998.

[MOZ88] M. C. Mozer. *A focused back-propagation algorithm for temporal pattern recognition*. Technical Report, Departments of Psycology and Computer Science,. University of Toronto, Toronto. 1988

[NAR89] Narendra K. S., Annaswamy A. M. *Stable Adaptive Systems*. Prentice Hall. 1989.

[NAR90] K. S. Narendra, K. Parthasarathy. *Identification and Control of Dynamic Systems Using Neural Networks*. IEEE Transactions on Neural Networks, no. 1, pp. 4-27. 1990.

[NAR91] K. S. Narendra and K. Parthasarathy. *Gradient methods for the optimization of dynamic systems containing neural networks*. IEEE Transactions on Neural Networks, pp. 252-262, 1991.

[NER93] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, S. Marcos. *Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms*. Neural Computation. vol. 5, pp. 165-199, 1993.

[NGU90] D. Nguyen, B. Widrow.*Neural networks for self-learning control systems*. IEEE Control Systems Magazine, April 1990.

[NIJ90] H. Nijmeijer, A. J. Van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag. Berlin. 1990.

[PAL96] R. Palm, D. Driankov, H. Hellendoorn. Model Based Fuzzy Control. Springer-Verlag, Berlin-Heidelberg.

[PED93] W. Pedrycz. Fuzzy Control and Fuzzy Systems. Segunda Edición, John Wiley & Sons, New York, 1993.

[PIC94] Stephen W. Piché. Steepest Descent Algorithms for Neural Network Controllers and Filters. IEEE Transactions on Neural Networks , Vol 5, NO 2, pp. 198-212, March 1994.

[PIN88] F. J. Pineda. *Dynamics and architecture for neural computation*. Journal of Complexity, vol. 4, pp. 216-245, 1988.

[POD91a] P. Poddar, K. P. Unnikrishnan. *Nonlinear prediction of speech signals using memory neuron networks*. In Neural Networks for Signal Processing 1, B. H. Juang, S. Y. Kung and C. A. Kamm, Eds. IEEE Press, 1991

[POD91b] P. Poddar, K. P. Unnikrishnan. *Memory neuron networks: A prolegomenon.* General Motors Research Laboratories Report GMR-7493, October 21, 1991.

[REE91] C. R. Reeves, N. C. Steele. *A genetic algorithm approach to designing neural network architecture*. Proc. 8Th International Conference on Systems Engineering.

[REE92] C. R. Reeves, N. C. Steele. *Problem-solving by simulated genetic processes: a review and application to neural networks*. Proc. 10th IASTED Symposium on Applied Informatics. ACTA Press, Anaheim, CA. 1992

[ROS58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review. 65;386-408. 1958

[RUM86] D. E. Rumelhart, G. E. Hinton, R. J. Williams. *Learning internal representations by error propagation*. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge, MA: MIT Press, 1986, ch. 8, pp. 318-362, vol.1. 1986.

[SAS94] P. S. Sastry, G. Santharam, K. P. Unnikrishnan. *Memory Neuron Networks for Identification and Control of Dynamical Systems*. IEEE Transactions on Neural Networks, Vol. 5, No. 2, pp. 306-319, 1994.

[SLO91] J. E. Slotine, W. Li. *Applied Nonlinear Control*. Pretince-Hall. 1991

[SPO88] Mark W. Spong, M. Vidyasagard. *Robot Dynamics and control*. John Wiley & Sons. 1988.

[SUY97] Johan A. K. Suykens, Joos Vadewalle, Bart L. R. De Moor. *NLq Theory: Checking and Imposing Stability of Recurrent Neural Networks for Nonlinear Modeling*. IEEE Transactions on Signal Processing, Vol. 45. No. 11, pp 2682-2691. November 1997.

[TAK91] H. Takagi and J. Hayashi.Neural Network Driven Fuzzy Reasoning. Int. J. Approximate Reasoning, vol.5 , pp 191-212, 1991.

[TSO94] Ah Chung Tsoi, Andrew D. Back. *Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures*. Transactions on Neural Networks, Vol. 5, No. 2, 229-239, 1994

[UNN91] K. P. Unnikrishnan, J. Hopfield, D. Tank. *Connected digit speaker dependent speech recognition using a neural network with time delayed connections*. IEEE Transactions on Signal Processing, vol. 39, pp. 698-713, 1991.

[VRI92] B. de Vries and J. Principe. *The Gamma Model- A new neural model for temporal processing*. Neural Networks, vol. 5, no. 4, pp. 565-576, 1992.

[WAI89] Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. Lang. *Phonemic recognition using time delay neural networks*. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 3, pp. 328-339, 1989.

[WAN90] E. Wan. *Temporal backpropagation for FIR neural networks*. In Proc. Int. Joint Conf. Neural Networks, San Diego, June, 1990, pp. 575-580

[WAN94] Wang, Li-Xi. *Adaptive Fuzzy System and Control*, PTR Prentice-Hall, New Jersey, 1994.

[WAS89] Philip D. Wasserman, *Neural Computing. Theory and Practice*. Van Nostrand Reinhold. 1989.

[WER74] P. J. Werbos, Beyond regression: *New tools for prediction and analysis in the behavioral sciences*. Masters thesis, Harvard University. 1974.

[WER90] Paul J. Werbos. *Backpropagation through time: What is does and How to do it*. Proceedings of the IEEE, Vol. 78, No. 10, pp. 1550-1560, October 1990.

[WES92] Lodewyk F. A.Wessels, Etienne Barnard. *Avoiding False Local Minima by Proper Initialization of Connections*. IEEE Transactions on Neural Networks, Vol. 3, No. 6, pp. 899-905, 1992.

[WIL89] R. J. Williams, D. Zipser. *A learning algorithm for continually running fully recurrent neural networks*. Neural Computation. pp. 270-277. 1989.

[ZAD65] L. A. Zadeh. *Fuzzy sets*. Information And Control. Vol 8, pp. 338-353, 19965

[ZAD88] L. A. Zadeh, *Fuzzy Logic*. IEEE Computer, pp. 83-93, April 1988.

[ZAD92] L. A. Zadeh, *The Calculus of Fuzzy If/Then RULES*. AI Expert, pp. 23-27. March 1992.