UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

REDES NEURONALES ARTIFICIALES EN EL CONTROL ADAPTATIVO DE ROBOTS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

VANESSA AGUILAR VREBOS

DIRECTOR DE TESIS: DR. FELIPE LARA ROSANO

México, D.F. Septiembre 1997

Este trabajo no se hubiese realizado sin la ayuda incondicional de mi Director de Tesis el Dr. Felipe Lara Rosano, quien me brindó su valioso conocimiento y apoyo en todo momento. Además le agradezco las enormes oportunidades que me ha brindado, mismas que me han permitido acercarme a la investigación de las Redes Neuronales Artificiales y la Inteligencia Artificial.

Agradezco al Instituto de Ingeniería y al Instituto de Investigación en Matemáticas Aplicadas (I.I.M.A.S.) por permitirme trabajar en sus instalaciones y hacer uso de sus equipos, así como a las personas que laboran en el Laboratorio de Inteligencia Artificial, destacando la colaboración del M.I. Nicolás Kemper Valverde.

Doy gracias a la Universidad Autónoma de México por brindarme parte de mi educación y en especial a la Facultad de Ingeniería donde pude concluir mis estudios profesionales.

Finalmente agradezco profundamente a todos mis buenos maestros que me brindaron su conocimiento, su tiempo y su esfuerzo.

GRACIAS

Mamá por darme la vida y permitir que yo pudiera estar aquí.

Papá por enseñarme a crecer, por confiar siempre en mí, por el apoyo que siempre me diste, por respetar todas mis decisiones y por ser el mejor amigo que he tenido.

Queridos abuelos por criarme y darme todo el amor que necesitaba, sobre todo a ti abuelita Sofía que siempre me inculcaste que debía ser una profesionista.

Padrino y Chelita por ser siempre unos segundos padres para mí y como tales darme su amor, confianza y ejemplo.

A toda mi familia que ha estado conmigo, me ha dado su amor y ha confiado en mí.

Oscar por apoyarme cuando más lo necesité, por ayudarme a seguir adelante en mis estudios, por darme tu cariño y darme el ejemplo de como debe ser un buen profesional.

Adriana por siempre tratar de hacer cosas buenas para mí y tu colaboración en la revisión de este trabajo.

Los amaré siempre.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El diseño actual de robots industriales está basado en servomecanismos simples asignados a las diferentes articulaciones del brazo. Esto da como resultado velocidades de respuesta reducidas, lo que provoca limitaciones en la precisión y exactitud de los efectores y produce un desempeño no óptimo. Las tareas cada vez más sofisticadas que deben ser desempeñadas por los robots han requerido mejores técnicas de control para incrementar la exactitud de las trayectorias de alta velocidad durante el funcionamiento en ambientes inciertos.

El comportamiento dinámico de un manipulador rígido está caracterizado por un sistema altamente acoplado y por ecuaciones diferenciales no lineares. Los efectos no lineares son más importantes cuando los robots están trabajando a altas velocidades y la carga es considerable en relación a la estructura mecánica del robot. Este es el caso de robots con un alto desempeño, los cuales operan con transmisión directa o ganancias reducidas en los engranes de transmisión.

Estrategias avanzadas de control, basadas normalmente en una cancelación exacta de la dinámica no lineal han sido usadas para este tipo de robots. Las incertidumbres en los parámetros dinámicos de los robots, tales como inercias y condiciones de carga han motivado el diseño de controladores adaptables. Este tipo de controlador está diseñado teniendo un conocimiento exacto de la estructura y no incluye aspectos tales como fricciones no lineares, elasticidad en las articulaciones y ligas, contragolpes y perturbaciones de torque, mismos que pueden ser encontrados en los robots.

Los manipuladores son sistemas con una estructura altamente no lineal. Las técnicas de control adaptativo pueden ser una herramienta poderosa para controlar este tipo de sistemas.

En años recientes los controladores basados en redes neuronales han recibido mucha atención. Este tipo de controlador explota las posibilidades de las redes neuronales para aprender funciones no lineales así como también para resolver ciertos tipos de problemas donde se requieren cálculos masivos en paralelo. La capacidad de *aprendizaje* de las redes neuronales es usada para hacer que el controlador aprenda cierto tipo de funciones, altamente no lineales en la mayoría de los casos, que representan la dinámica directa, la dinámica inversa o alguna otra característica del proceso. Es usualmente hecho durante un largo periodo de entrenamiento del controlador en forma supervisada o no supervisada. Si la capacidad de aprendizaje de las redes neuronales no es interrumpida después del periodo de entrenamiento, el controlador basado en redes neuronales trabajará como *un controlador adaptativo* .

Las Redes Neuronales, como se ha mencionado antes, tienen la habilidad de aprender modelos no lineales sin un conocimiento anterior de su estructura y son adecuadas para trabajar en tiempo real por su alto paralelismo. Estas propiedades han sido explotadas para diseñar controladores no adaptativos y adaptativos para robots.

Este trabajo tiene por objetivo hacer una simulación del enfoque desarrollado por Albert Y. Zomaya y Tarek M. Nabhan para el control directo de robots usando redes neuronales artificiales. El sistema

de control consiste en un modelo de dinámica inversa que produce las fuerzas/torques que son aplicados al robot, dadas las posiciones, velocidades, y aceleraciones deseadas, mientras un controlador neuronal genera una señal correctora. Los pesos de la red neuronal, los cuales representan los parámetros del controlador, son modificados en base a una señal de error. La señal de error representa la desviación entre las posiciones, velocidades y aceleraciones deseadas y las reales. El controlador neuronal es computacionalmente eficiente en el sentido de que no se requiere una estimación de parametros para actualizar el controlador. La técnica emplea aprendizaje reforzado para actualizar los parametros de la red.

La técnica de control es genérica en el sentido de que los parámetros del controlador no son dependientes de ninguna estimacion de parámetros a diferencia de muchos métodos convencionales de control adaptativo. El modelo de red neuronal usado aquí es una red neuronal multicapa basada en el algoritmo de backpropagation .

CAPITULO 1

ROBÓTICA

1.1 INTRODUCCIÓN A LA ROBÓTICA

1.1.1 DEFINICIÓN DE ROBOT

La palabra *robot* proviene de la palabra checa *robota*, que significa trabajo. La definicion oficial de un robot industrial se proporciona por la Robotics Industries Association (RIA), anteriormente Institute of America (RIA).

> *Un robot industrial es un manipulador multifuncional reprogramable diseñado para desplazar materiales, piezas herramientas o dispositivos especiales mediante movimientos variables programados para la ejecucion de una diversidad de tareas*

Idealmente un robot es un ente "inteligente" (programable versatil adaptable y capaz de organizarse); es una maquina capaz de realizar movimientos versátiles parecidos a los de nuestras extremidades superiores, con cierta capacidad sensorial y de reconocimiento y capaz de controlar su comportamiento.

1.1.2 HISTORIA DE LA ROBÓTICA

En los siglos XVII y XVIII se crearon varios dispositivos mecánicos ingeniosos que tenían algunas caracteristicas de los robots. Jaques de Vaucan construyó varios músicos de tamaño humano a mediados del siglo XVIII. Esencialmente se trataba de robots mecanicos diseñados para un propósito especifico: la diversión. En 1805 , Henri Maillardet construyó una muñeca mecánica que era capaz de hacer dibujos . Una serie de levas se utilizaban como el <<programa>> para el dispositivo en el proceso de escribir y dibujar. Hubo otras invenciones mecánicas durante la revolucion industrial, muchas de las cuales estaban dirigidas al sector de la producción textil. Entre ellas se puede citar la hiladora giratoria de Hargraves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwrigt (1785), el telar Jacquard (1801) y otros.

En tiempos mas recientes, el control numerico y la telequerica son dos tecnologias importantes en el desarrollo de la robotica. El control numerico (NC) se desarrolló para máquinas - herramienta a finales de los años 40 y principios de los años 50. Como su nombre lo indica el control numérico implica el control de acciones de una maquina - herramientas por medio de números. Está basado en el trabajo original de John Parsons, que concibió el empleo de tarjetas perforadas, que contienen datos de posiciones, para controlar los ejes de una máquina - herramienta. El MIT (Instituto de Tecnologia de Massachusetts) en 1952 desarrolló un proyecto que utilizaba una fresadora de tres ejes que era un prototipo de control numérico. Un trabajo posterior en el MIT llevo al desarrollo del APT (Automatically Programmed Tooling), un lenguaje de programacion de piezas para realizar la programación de la maquina - herramienta de control numérico.

El campo de la telequénca abarca la utilizacion de un manipulador remoto controlado por un ser humano. A veces denominado teleoperador, el manipulador remoto es un dispositivo mecánico que

traduce los movimientos del operador humano en movimientos correspondientes en una posición remota. Un empleo frecuente de un teleoperador está en la manipulación de sustancias peligrosas tales como materiales radioactivos. El operador humano puede permanecer en un lugar seguro; no obstante , mirando a través de una ventana de cristal plomado o mediante televisión en circuito cerrado, el operador puede guiar los movimientos del brazo remoto. Los primitivos dispositivos eran completamente mecánicos, pero los sistemas más modernos utilizan una combinación de sistemas mecánicos y control electrónico de realimentación.

En 1954 Cyril Walter Kenward solicitó una patente británica para un dispositivo robótico. A George C. Devol deben atribuirse dos invenciones que llevaron al desarrollo de los robots actuales. La primera invención era un dispositivo para grabar magnéticamente señales eléctricas y reproducirlas para controlar una máquina (1946). La segunda invención se denomina <<Transferencia Programada de Artículos >> (1961).

Con el respaldo financiero de la Consolidated Diesel Electric Company (ahora Condec Corp), Joseph F. Engelberger Y George Devol comenzaron a desarrollar planes y prototipos para el robot universal conocido como "Unimate".

La primera instalación de un robot Unimate fue hecha en la Ford Motor Company para descarga de una máquina de función en troquel.

En 1973 se desarrollo el lenguaje experimental denominado WAVE, que fue seguido al desarrollo de AL, que es otro lenguaje destinado a la investigación. El primer lenguaje comercial fue VAL, desarrollado por Victor Scheinman y Bruce Simano para Unimation Inc. El lenguaje se utilizó para programar el robot PUMA (Programmable Universal Machine for Assembly) de Unimation.

Hoy en día vemos a la robótica como un campo de trabajo mucho mas amplio que el que teníamos simplemente hace unos pocos años, tratado con la investigación y el desarrollo en una serie de áreas interdisciplinarias que incluyen cinemática, dinámica, planificación de sistemas, control, sensores, lenguajes de programación e inteligencia de máquina.

1.1.3 OBJETIVO DE LA ROBÓTICA

La robótica busca la productividad elevada dada por el aumento en la velocidad de trabajo y la calidad constante. Asimismo intenta liberar al trabajador de condiciones de trabajo insalubres y de alto riesgo, eliminando la fatiga inherente al trabajo tedioso y repetitivo (que es la fuente de accidentes de trabajo y de variabilidad de la calidad en los productos manufacturados).

La robótica mejora la versatilidad debido a una mayor flexibilidad en el empleo de las máquinas y pretende incrementar la rentabilidad de las inversiones, racionalizar los recursos, humanizar el trabajo y aumentar la productividad y calidad elevando la competitividad.

1.1.4 CLASIFICACIÓN DE ROBOTS

Los robots pueden dividirse según su grado de movilidad en:

- Movilidad *estática*, que correspondería a los robots de base fija o manipuladores.
- Movilidad en un area amplia.

De estos últimos, se puede hacer una distinción adicional:

- desplazamiento sobre ruedas y
- desplazamiento por medio de articulaciones

1.2 SISTEMAS CONSTITUTIVOS DE UN ROBOT

Un robot esta constituido por cuatro sistemas: el sistema ***mecánico*** lo constituye el brazo que mueve su órgano terminal en varias posiciones del espacio con el fin de realizar diversas tareas sobre algunas piezas. El sistema de ***percepción*** es un conjunto de sensores o elementos que el robot usa para "percibir" (interna y externamente) su medio ambiente .El sistema de ***decisión*** es un sistema "inteligente" que orienta las acciones que realiza el brazo (activar o desactivar motores, mandar y obtener señales externas etc.) El sistema de ***comunicación*** lo constituyen los elementos necesarios para la interaccion del hombre con el robot.

1.2.1 SISTEMA MECÁNICO

El sistema mecanico está constituido por el **MANIPULADOR**, y a este lo compone una cadena cinemática que posee dos elementos básicos:

- **Eslabones** : Son cuerpos rígidos que conectan a una, dos o mas articulaciones y a veces un órgano terminal.
- **Articulaciones** : Son puntos fijos entre dos cuerpos que pueden o no tener movimiento relativo entre ellos.

Tipos de articulaciones:

Únicamente son posibles seis tipos diferentes de articulación: de revolucion (giratoria), prismática (deslizante), cilíndrica, esférica, de tornillo y plana. De éstas únicamente las articulaciones giratorias y prismaticas son comunes en los manipuladores. (Vease Figura 1.1)

Cadena cinemática: Es un conjunto de eslabones conectados movilmente mediante articulaciones.

Cadena cinemática abierta: Esta formada por eslabones binarios y un eslabón unitario.

Cadena cinematica cerrada: Es aquélla donde todos los eslabones son binarios.

1.2.1.1. CARACTERÍSTICAS DEL SISTEMA MECÁNICO

El sistema mecanico, al estar constituido por una cadena cinematica requiere de actuadores que animan o mueven las articulaciones, así como el equipo complementario.

Los actuadores pueden conectarse directamente con el eslabón siguiente o a través de alguna transmisión mecánica.

La parte final del manipulador realiza acciones como transportar piezas, soldar, pintar, etc

Un cuerpo queda perfectamente determinado en el espacio físico tridimensional por seis variables independientes: Las tres primeras definen su posición y las tres restantes definen su orientación.

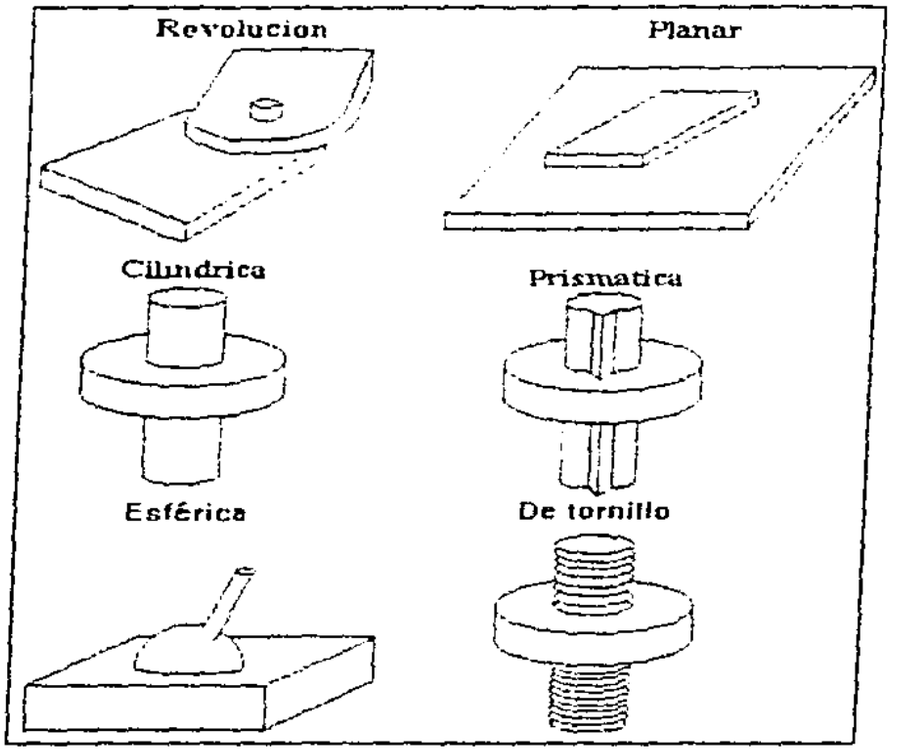

Figura 1.1: Tipos de articulacion

Movilidad: Es el número de articulaciones de un manipulador.

Grado de Libertad: Número de parametros independientes necesarios para posicionar y / u orientar un objeto en el espacio.

La estructura mecanica de un robot permite ejecutar tareas definidas como una secuencia de puntos caracterizados por una posicion y una orientación.

Compatibilidad: Los grados de libertad del manipulador del robot deben ser mayores o iguales que los grados de libertad de la tarea a realizar.

Repetitividad: Es la precisión con que el manipulador llega a un mismo punto en los ciclos de movimiento

Precisión: Es la capacidad de que el brazo llegue a una posición o punto. La precisión depende de los actuadores utilizados.

Volumen de Trabajo: Es el termino que se refiere al espacio dentro del cual el robot puede manipular el extremo de su muñeca. El volumen de trabajo viene determinado por las siguientes características físicas del robot:

- La configuración física del robot.
- Los tamaños de los componentes del cuerpo, del brazo y de la muñeca.
- Los límites de los movimientos de las articulaciones del robot.

Un robot de coordenadas polares tiene un volumen de trabajo que es una esfera parcial, un robot de coordenadas cilíndricas tiene una envolvente de trabajo cilíndrica, un robot de coordenadas tiene un espacio de trabajo de forma rectangular y un brazo articulado tiene un volumen de trabajo aproximadamente esférico (Vease Figura 1.2)

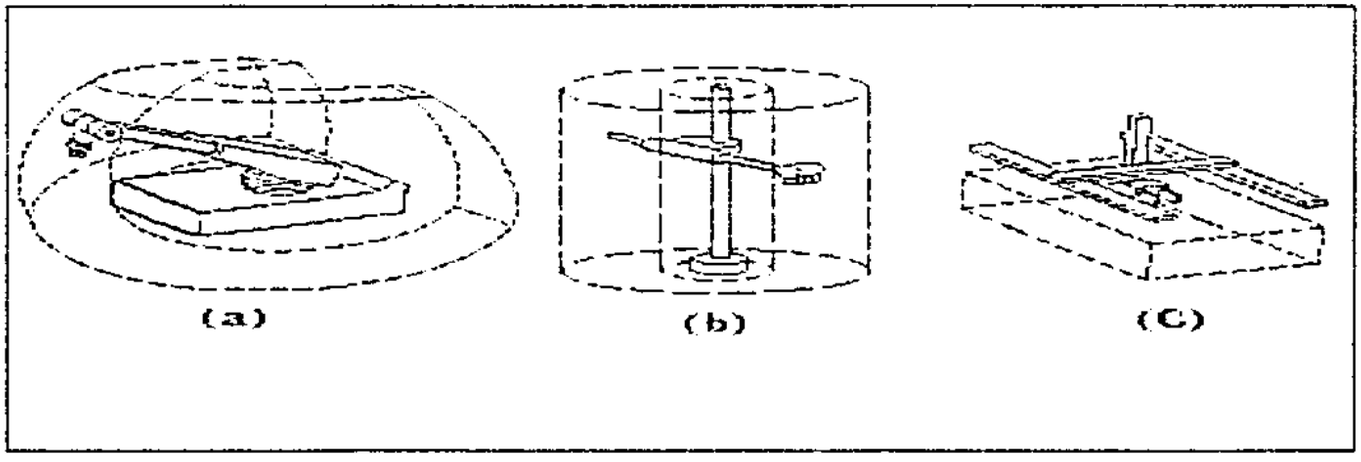

Figura 1.2: Volumenes de trabajo para diversas anatomías de robot: (a) polar, (b) cilíndrica y (c) cartesiana

Resolución espacial: Es el incremento de movimiento mínimo en su volumen o área de trabajo.

1.2.1.2 ARQUITECTURAS DE MANIPULADORES

Los robots industriales están disponibles en una amplia gama de tamaños, formas y configuraciones físicas. La gran mayoría de los robots comercialmente disponibles en la actualidad tienen una de estas cuatro configuraciones básicas:

1. Configuración polar.
2. Configuración cilíndrica.
3. Configuración de coordenadas cartesianas o rectangulares.
4. Configuración de brazo articulado.

La configuracion polar se ilustra en la parte *(a)* de la Figura 1.3 . Utiliza un brazo telescopico que puede elevarse o bajar alrededor de un pivote horizontal. Este pivote está montado sobre una base giratoria. Estas diversas articulaciones proporcionan al robot la capacidad de desplazar su brazo dentro de un espacio esférico y de aquí la denominación de robot de "coordenadas esfericas". La configuración cilíndrica, según se muestra en la figura *(b)*, utiliza una columna vertical y un dispositivo de deslizamiento que puede moverse hacia arriba o abajo a lo largo de la columna. El brazo del robot está unido al dispositivo deslizante de modo que puede moverse en sentido radial con respecto a la columna. Haciendo girar la columna, el robot es capaz de conseguir un espacio de trabajo que se aproxima a un cilindro.

Un robot de coordenadas cartesianas, ilustrado en la parte *(c)* de la figura, utiliza tres dispositivos deslizantes perpendiculares para construir los ejes x, y y z. Desplazando los tres dispositivos deslizantes entre sí, el robot es capaz de operar dentro de una envolvente rectangular de trabajo.

Un robot de brazo articulado se ilustra en la figura *(d)*. Su configuración es similar a la de un brazo humano . Está constituido por dos componentes rectos, que corresponden al antebrazo y al brazo humanos , montados sobre un pedestal vertical. Estos componentes estan conectados por dos articulaciones giratorias que corresponden al hombro y al codo. Una muñeca esta unida al extremo del antebrazo, con lo que proporciona varias articulaciones suplementarias.

1.2.1.3 SISTEMAS DE IMPULSIÓN DEL ROBOT

La capacidad del robot para desplazar su cuerpo, brazo y muñeca es proporcionada por el sistema de impulsión que es utilizado para accionar al robot . El sistema impulsor determina la velocidad de los movimientos del brazo, la resistencia mecanica del robot y su rendimiento dinámico. En cierta medida, el sistema impulsor determina el tipo de aplicaciones que puede realizar el robot.

Las caracteristicas que deben tener el sistema impulsor son:

- Inercia pequeña que permite aumentar la rapidez de respuesta del robot.
- Alta rigidez para evitar desplazamientos debido a deformaciones causadas por la carga manejada.
- Posibilidades de controlar posición, velocidad y/o fuerza, etc.

Tipos de sistemas de impulsión:

Los robots industriales disponibles en el mercado, están accionados por uno de los siguientes tipos de sistemas de impulsion:

1. Impulsion hidráulica
2. Impulsion eléctrica
3. Impulsión neumatica

Las impulsiones hidraulica y eléctrica son los dos tipos principalmente utilizados en los robots más sofisticados.

La ventaja habitual del sistema de impulsión hidráulica es proporcionar al robot una mayor velocidad y una mayor resistencia mecánica. Los inconvenientes del sistema de impulsión hidráulica radican en que suelen añadir más necesidades de espacio y en que un sistema hidráulico es propenso a las fugas de aceite. Los sistemas de impulsión hidráulica pueden diseñarse para actuar sobre articulaciones rotacionales o lineales. Se pueden emplear actuadores de paletas giratorios para proporcionar un movimiento de rotación y pueden utilizarse pistones hidráulicos para realizar un movimiento lineal.

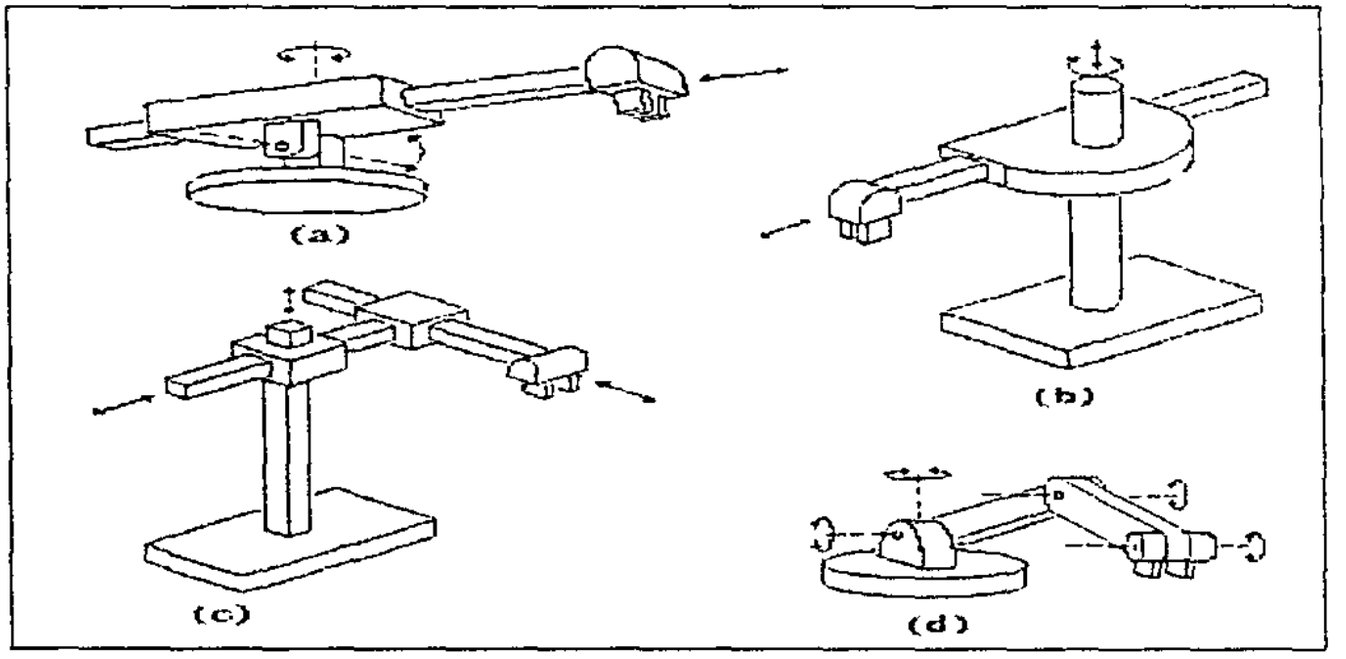

Figura 1.3: Las cuatro anatomías de robot básicas: (a) polar, (b) cilíndrica, (c) cartesiana y (d) de brazo articulado

Los sistemas de impulsión eléctrica no suelen proporcionar tanta velocidad como los sistemas hidráulicos, pero la exactitud y la repetibilidad de los robots de impulsión eléctrica suelen ser mejores. En consecuencia, los robots eléctricos tienden a ser más pequeños, con menores exigencias de espacio y sus aplicaciones tienden hacia un trabajo más preciso, tal como el montaje. Los robots de impulsión eléctrica son accionados por motores de pasos o servomotores de corriente continua. Estos motores son idóneos para el accionamiento de articulaciones rotacionales mediante sistemas de engranaje y trenes impulsores adecuados. Los motores eléctricos pueden emplearse también para accionar articulaciones lineales por medio de sistemas de poleas u otros mecanismos de traslación.

La impulsión neumática suele reservarse para los robots más pequeños que tienen menor grado de libertad (movimientos de dos a cuatro articulaciones). Estos robots suelen estar limitados a simples

operaciones de "tomar y situar" con ciclos rápidos. La potencia neumática puede adaptarse fácilmente a la actuación de dispositivos de pistón para proporcionar un movimiento de traslación de articulaciones deslizantes. También puede emplearse para accionar actuadores giratorios para articulaciones rotacionales.

1.2.2 SISTEMA DE PERCEPCIÓN

La utilización de mecanismos sensores externos permite a un robot interaccionar con su entorno de una manera flexible, en contraste con operaciones preprogramadas en las cuales a un robot se le "enseña" a efectuar tareas repetitivas mediante un conjunto de funciones programadas.

La función de los sensores del robot se puede dividir en dos categorías principales: *estado interno y estado externo*. Los sensores del estado interno tratan con la detección de variables tales como la posición de la articulación del brazo que se utiliza para controlar al robot. Por otra parte, los sensores de estado externo tratan la detección de variables tales como alcance, proximidad y contacto. Aunque los sensores de proximidad, contacto y fuerza juegan un papel significativo en la mejora del robot, se reconoce que la visión es la capacidad sensorial mas potente del mismo. La visión del robot se puede definir como el proceso de extraer, caracterizar e interpretar información de imágenes de un mundo tridimensional. Este proceso, es también comúnmente conocido como visión de *máquina o de computadora*.

1.2.2.1 SENSORES INTERNOS

Sensores de contacto

Son dispositivos "todo o nada" que accionan un contacto eléctrico cuando una condición se verifica y que regresa a la posición contraria cuando dicha condición desaparece. Son de bajo costo, simples y robustos. Los sensores de contacto se clasifican en dos:

- De contacto físico (microswitches)
- Interruptores magnéticos , requieren de un inductor magnético en la parte móvil que desea detectarse.

Sensores de posición

Indican las variaciones en la posición de las articulaciones. Los tipos de sensores son analógicos o digitales

- **Analógicos** : Estos pueden ser resistivos (potenciómetros) o inductivos (transformadores diferenciales).
    - **Potenciómetros** : El voltaje en el cursor del potenciómetro indica la posicion del objeto (generalmente unido al cursor). La tensión es proporcional al desplazamiento del cursor. Son de costo moderado, buena resolución, excelente linealidad y larga vida de funcionamiento. Algunas de sus desventajas son que con el contacto entre el cursor y la pista se ocasiona ruido y desgaste, éste último origina pequeñas zonas

muertas (sin contacto) e histéresis locales; también son sensibles a la contaminación ambiental.

- **Transformador diferencial:** Está formado por un núcleo ferromagnético que se desplaza entre dos bobinas secundarias modificando su acoplamiento magnético con la bobina primaria. Al hacer circular una corriente alterna por el devanado primario se induce, en cada secundario, una tensión cuya amplitud está en función de la posición del núcleo y cuyo signo indica el sentido del desplazamiento. Puede alcanzar posiciones elevadas del orden de 0.25 micrones y linealidad de hasta 0.1% en rangos de 25cm. Son insensibles a la humedad y a la temperatura.

- **Digitales:** Son básicamente codificadores optoelectrónicos.Se componen de un disco o regleta, según el tipo de articulación (rotacional o prismática), con varias pistas formadas por orificios o zonas blancas y negras. Cada pista es analizada simultáneamente en un punto por dispositivos optoelectrónicos, y al moverse la articulación junto con el disco o regleta cada dispositivo optoelectrónico genera una señal digital que proporciona una medición del desplazamiento. Estos sensores son ampliamente utilizados en sus dos variantes.
  - ⇒ Incrementales
  - ⇒ Absolutos

Sensores de velocidad

Indican las variaciones en la velocidad de las articulaciones. Los tipos de sensores de velocidad comúnmente utilizados en robots industriales son:

- **Optoelectrónicos:** Funcionan de manera similar a los sensores de posición digitales, con la consideración que la frecuencia de la señal digital generada es proporcional a la velocidad del giro.
- **Generador de corriente directa o dinamo:** Posee un estator de magneto permanente, una fuerza electromagnética (fem) que es inducida en el rotor cuya magnitud es proporcional a la velocidad de éste. Dicha fem genera en las terminales del devanado del estator un voltaje de corriente directa proporcional a la velocidad.

1.2.2.2 SENSORES EXTERNOS

Amplian las posibilidades de los robots industriales en la ejecución de trabajos.

Los sensores externos para reconocer un objeto tienen como máximo tres fases:

- Detección: darse cuenta que dicho objeto existe
- Localización: saber donde se encuentra el objeto
- Caracterización: analizar la geometría del cuerpo

Las distancias que trabajan estos sensores son las siguientes:

- Sensores de tacto y esfuerzo = 0 cm.

- Sensores proximétricos = 1 a 20 cm.
- Sensores de visión = 50 cm a varios metros

Los sensores externos más comunmente utilizados son los siguientes:

Sensores de tacto

Existen dos tipos de sensores de tacto:

- **Contacto binario:** Utilizan microswitches que al contacto producirán una información binaria, consistente en ausencia o presencia de piezas en la pinza del robot. Posee materiales antideslizantes (hule, neopreno), y se manipulan objetos cuyo peso es conocido a priori, permitiendo ajustar la fuerza de presion necesaria para dicha tarea
- **De deslizamiento :** Ajustan la fuerza de presion de los objetos en funcion del peso de los mismos, y para esto se tiene un sensor de deslizamiento (ejemplo: rodillo empujado por un resorte) implementado en los "dedos" de la pinza. Este sensor detecta el movimiento relativo entre la pieza por manipular y la pinza del robot. La fuerza de presión aumenta progresivamente hasta anular el deslizamiento.
- **Piel artificial :** Está compuesta por una hoja de material aislante (hule, neopreno) con espesor entre 3 y 10 mm, contaminada con partículas conductoras. En una cara se coloca una lámina metálica uniforme y en la otra cara una matriz de electrodos. Cuando se ejerce una presión sobre la piel artificial en las partículas conductoras circula una corriente proporcional a la presión ejercida por el objeto, permitiendo estimar la magnitud de la fuerza de presion ejercida por la pinza sobre el objeto.

Sensores de esfuerzo

Existen dos formas de detectar el esfuerzo resultante del contacto entre el robot y un objeto de su medio ambiente:

- **Utilizando como sensores a los actuadores del robot:** Cuando una fuerza es aplicada en el organo terminal del brazo se origina un incremento en la excitacion de los actuadores, esto con el fin de mantener nulo el error de posicionamiento.
- **Utilizando sensores de esfuerzo:** Para medir los pares y fuerzas inducidas por el contacto pueden usarse galapas tesométricas como transductores. El sensor de esfuerzo se construye como un arreglo de vigas provistas de pares antisimetricos que determinan los momentos y ciertas componentes originadas por el contacto.

Sensores de proximidad

Su fin es informar la presencia de objetos cercanos al organo terminal del brazo. Distinguen objetos y objetivos para auxiliar al sistema de control (decisión) a evitar colisiones

Los sensores de proximidad son los siguientes:

- **Ópticos :** a través de la luz determinan la proximidad

- **Magnéticos:** se usan para detectar proximidad de objetos metalicos
- **Neumáticos :** detectan la presencia de un objeto en función de la cantidad de aire recibida que puede bloquear el objeto. (contrapresión).
- **Acústicos:** detectan la presencia de un objeto enviando una señal acústica que tiende a rebotar en caso de encontrarse un objeto cerca.

Sensores de visión

Son los más completos ya que dan un conocimiento global de la situación del entorno significativo del robot, y generalmente poseen camaras que cubren el area de trabajo del robot. El alcance puede ser de varios metros.

1.2.3 SISTEMA DE DECISIÓN

El sistema de decision esta formado por el controlador, y este tiene la *inteligencia* para hacer que el manipulador se desempeñe de la manera que haya sido descrita por su entrenador (el usuario).

El controlador consta de:

- **Memoria** para almacenar datos de posiciones definidas donde el brazo se mueve y para guardar informacion relacionada con la propia secuencia del sistema (programa).
- **Secuenciador** que interpreta datos almacenados en memoria e indica a los demás componentes del controlador las acciones a realizar. El secuenciador actúa como interface pasando datos y señales
- **Unidad computacional** que realiza los cálculos necesarios para auxiliar al secuenciador.
- **Sensor de interface con el manipulador** para obtener datos como la posición de cada articulación. Algunos robots tienen un **Sensor de interface externa** por ejemplo una cámara de TV para obtener informacion del sistema de visión.
- **Interface entre el secuenciador y los amplificadores de potencia**, ésta se encarga con base en la información del secuenciador , de proveer las señales (hacia los amplificadores de potencia) de tal forma que los actuadores puedan eventualmente mover sus articulaciones de la forma deseada.
- **Interface hacia equipo auxiliar** para sincronizarse con otras unidades externas o controladores, así como para determinar el estado de sensores (ejemplo: switches limite en dispositivos o unidades externas).

Controles de operador para que el usuario pueda definir la secuencia de operaciones y el control del robot. Esto puede hacerse desde una terminal con un lenguaje de programacion, con un panel de control, con un "teach pendant" o dispositivo similar que posea un menú de instrucciones dirigidas al operador.

1.2.4 SISTEMA DE COMUNICACIÓN HOMBRE - MÁQUINA

Es el sistema donde se realiza el intercambio de información entre el operador y el robot o máquina.
En robótica la comunicación hombre - máquina se utiliza para transferir del hombre hacia el robot la información relativa a la planificación y a la ejecución de la tarea (con los avances tecnológicos se empieza a confiar al robot una parte importante de la función decisión - planificación).
La comunicación hombre - máquina puede efectuarse de distintas maneras:

- Señales eléctricas
- Informaciones visuales
- Informaciones sonoras, etc.

Estas pueden desarrollarse en diferentes niveles de lenguaje, del más alto (lenguaje natural) al nivel más bajo (señales de todo tipo).
Un ejemplo de comunicación hombre - máquina son los controladores del operador (Teach Pendant, computadora, etc.), para esto se utilizan computadoras con algún lenguaje de programación particular.
Muchos lenguajes de programación utilizados en robótica industrial se desarrollaron para un tipo o robot en particular, y aquí se mencionan algunos:
WAVE (1970), AL (CINEMATIC1975), MAL (1978 IBM), REX, EMILY, SIGLA (OLIVETTI) PRIMA, VAL (PUMA UNIMATION 1978), AUTOPASS (1977 IBM), LAMA, RAPT(1978), NOAH, BUILD, SAIL (1975), STRIPS (1970) APL, BASIC, PASCAL, FORTRAN, LISP, PROLOG, ADA y otros lenguajes que operan para mini computadoras.

1.2.5 EQUIPO PERIFÉRICO

Para las aplicaciones industriales, las capacidades del robot básico deben aumentarse por medio de dispositivos adicionales. Podríamos denominar a estos dispositivos como los periféricos del robot. Incluyen herramientas que se unen a la muñeca del robot y a los sistemas sensores que permiten interaccionar con su entorno.
En robótica, el término de efector final se utiliza para describir la mano o herramienta que esta unida a la muñeca. El efector final representa la herramienta especial que permite al robot de uso general realizar una aplicación particular. Esta herramienta especial debe diseñarse específicamente para la aplicación.
Los efectores finales pueden dividirse en dos categorías: pinzas y herramientas. Las pinzas son efectores finales que se utilizan para tomar y sostener objetos. Hay una diversidad de métodos de sujeción que pueden utilizarse, además de los medios mecánicos obvios de tomar la pieza entre dos o mas dedos. Estos métodos suplementarios incluyen el empleo de casquetes de sujeción, imanes, ganchos, y cucharas. Una herramienta se utilizaría como un efector final en aplicaciones en que se exija al robot realizar alguna operación en la pieza de trabajo. Estas aplicaciones incluyen la soldadura por puntos, la soldadura por arco, la pintura por pulverización y las operaciones de taladro. En cada caso, la herramienta particular está unida a la muñeca del robot para realizar la aplicación.

Algunos ejemplos de herramientas utilizadas como efectores finales en aplicaciones de robot incluyen:

- Herramientas de soldadura por puntos
- Soplete de soldadura por arco
- Tobera de pintura para operaciones tales como:
  - taladrado
  - ranurado
  - cepillado
  - rectificado
- Aplicadores de cemento liquido para montaje
- Sopletes de calentamiento
- Herramientas de corte por chorro de agua

1.3 CINEMÁTICA DEL BRAZO DEL ROBOT

Un manipulador mecánico se puede modelar como una cadena articulada en lazo abierto con algunos cuerpos rigidos (elementos) conectados en serie por una articulación de revolución o prismática movida por actuadores. Un final de la cadena se une a una base soporte mientras que el otro extremo está libre y unido con una herramienta (el efecior final) para manipular objetos o realizar tareas de montaje. El movimiento relativo en las articulaciones resulta en el movimiento de los elementos que posicionan la mano en una orientación deseada. En la mayoria de las aplicaciones de robótica, se está interesando en la descripción espacial del efector final del manipulador con respecto a un sistema de coordenadas de referencia fija.

La cinemática del brazo del robot trata con el estudio analitico de la geometria del movimiento de un robot con respecto a un sistema de coordenadas de referencia fijo como una función del tiempo sin considerar las fuerzas / momentos que onginan dicho movimiento. Asi pues , trata con la descripción analitica del desplazamiento espacial del robot como función del tiempo, en particular las relaciones entre las variables espaciales de tipo articulación y la posición y orientación del efector final del robot. Este tema se plantea dos cuestiones fundamentales:

1. Para un manipulador determinado dado el vector de ángulos de las articulaciones $q(t) = (q_1(t), q_2(t), \ldots q_n(t))^T$ y los parámetros geométricos del elemento, donde n es el número de grados de libertad, ¿ Cuál es la orientación y la posición del efector final del manipulador con respecto a un sistema de coordenadas de referencia ?
2. Dada una posición y orientación deseada del efector final del manipulador y los parámetros geométricos de los elementos con respecto a un sistema de coordenadas de referencia, ¿ puede el manipulador alcanzar la posición y orientación de la mano que se desea ? Y si puede, ¿ cuantas conf guraciones diferentes del manipulador satisfarán la misma condicion ?

La primera pregunta se suele conocer como el problema *cinemático directo*, mientras la segunda es el problema *cinemático inverso* (o solución del brazo). Como las variables independientes en un brazo de robot son variables de articulación y una tarea se suele dar en terminos de las coordenadas de referencia, el problema cinematico inverso se utiliza de forma más frecuente. En la figura se muestra un simple diagrama de bloques que indica la relación entre estos dos problemas.

Como los elementos de un brazo pueden girar y/o trasladarse con respecto a un sistema de coordenadas de referencia, el desplazamiento espacial total del efector final se debe a las rotaciones angulares y traslaciones angulares de los elementos. Denavit y Hartenberg [1955] propusieron un método sistematico y generalizado de utilizar algebra matricial para describir y representar la geometria espacial de los elementos de un brazo con respecto a un sistema de referencia fijo.

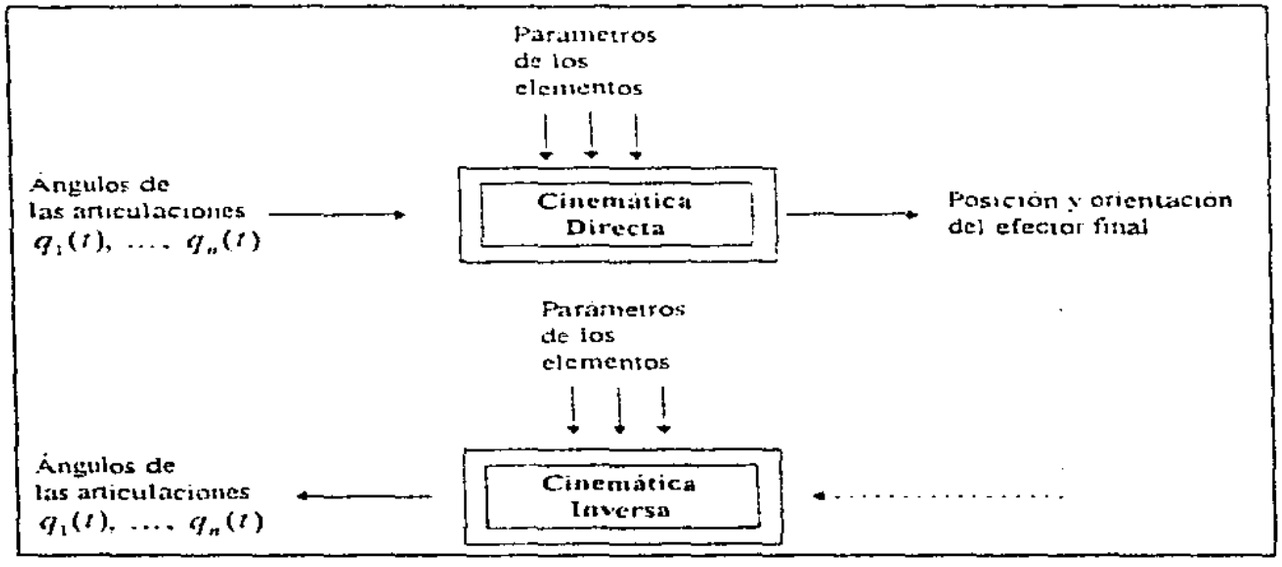

Figura 1.4 : Los Problemas cinemáticos directo e inverso

1.3.1 EL PROBLEMA CINEMÁTICO DIRECTO

Se utiliza álgebra vectonal y matricial para desarrollar un método generalizado y sistemático para describir y representar la localización de los elementos de un brazo con respecto a un sistema de referencia fijo. Como los elementos de un brazo pueden girar y / o trasladarse con respecto a un sistema de coordenadas de referencia, se establecerá un sistema de coordenadas ligado al cuerpo, a lo largo del eje de la articulación para cada elemento. El problema cinemático directo se reduce a encontrar una matriz de transformación que relaciona el sistema de coordenadas ligado al cuerpo al sistema de coordenadas de referencia. Se utiliza una matriz de rotacion 3 x 3 para describir las operaciones rotacionales del sistema ligado al cuerpo con respecto al sistema de referencia. Se

utilizan entonces las coordenadas homogéneas para representar los vectores de posición en un espacio tridimensional, y las matrices de rotación se ampliarán a matrices de transformación homogénea 4 x 4 para incluir las operaciones traslacionales del sistema de coordenadas ligado al cuerpo.

1.3.1.1 MATRICES DE ROTACIÓN

Una matriz de rotacion 3 x 3 se puede definir como una matriz de transformación que opera sobre un vector de posición en un espacio euclidiano tridimensional. Transforma sus coordenadas expresadas en un sistema de coordenadas rotado $OUVW$ (sistema ligado al cuerpo) en un sistema de coordenadas de referencia $OXYZ$. En la figura 1.5 se dan dos sistemas de coordenadas rectangulares: el sistema de coordenadas $OXYZ$, con OX, OY y OZ como sus ejes de coordenadas, y el sistema de coordenadas $OUVW$, con OU, OV, OW como sus ejes de coordenadas. Ambos sistemas de coordenadas tienen sus orígenes coincidentes en el punto O. El sistema de coordenadas $OXYZ$ está fijo en el espacio tridimensional y se considera un espacio de referencia. El sistema de coordenadas $OUVW$ está girando con respecto al sistema de referencia $OXYZ$.

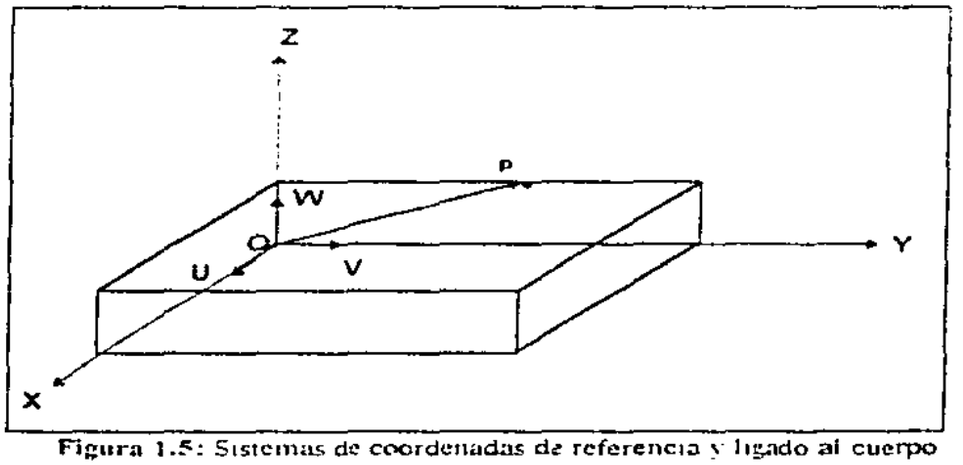

Figura 1.5: Sistemas de coordenadas de referencia y ligado al cuerpo

Físicamente, uno puede considerar que el sistema de coordenadas $OUVW$ es un sistema de coordenadas ligado al cuerpo. Esto es, está permanente- y convenientemente unido al cuerpo rígido (por ejemplo, un avión o un elemento del brazo de un robot) y se mueve junto con él. Sean (i_x, j_x, k_x) y (i_w, j_w, k_w) los vectores unitarios a lo largo de los ejes de coordenadas de los sistemas $OXYZ$ y $OUVW$, respectivamente. Un punto p en el espacio se puede representar por sus coordenadas con respecto a ambos sistemas de coordenadas. Para facilitar el análisis, supondremos

que p está en reposo y fijo con respecto al sistema de coordenadas $OUVW$. Entonces el punto p se puede representar por sus coordenadas con respecto al sistema de coordenadas $OUVW$ y $OXYZ$, respectivamente, como :

$$p_{uvw} = (p_u, p_v, p_w)^T \quad y \quad p_{xyz} = (p_x, p_y, p_z)^T \tag{1.1}$$

donde p_{xyz} y p_{uvw} representan el mismo punto p en el espacio con respecto a diferentes sistemas de coordenadas y el superíndice T en los vectores y matrices denota la operación transpuesta.
Una matriz R de transformación 3 x 3 es una matriz que transforma las coordenadas de p_{uvw} a las coordenadas expresadas con respecto al sistema de coordenadas $OXYZ$, después de que el sistema de coordenadas $OUVW$ ha sido girado. Esto es:

$$p_{xyz} = R\,p_{uvw} \tag{1.2}$$

Obsérvese que físicamente el punto p_{uvw} ha sido girado junto con el sistema de coordenadas $OUVW$.
Recordando la definición de las componentes de un vector tenemos:

$$p_{uvw} = p_u i_u + p_v j_v + p_w k_w \tag{1.3}$$

donde p_x, p_y y p_z representan las componentes de p a lo largo de los ejes OX, OY y OZ, respectivamente, o las proyecciones de p sobre los ejes respectivos. Así utilizando la definición del producto escalar y la ecuación (1.3) :

$$\begin{aligned} p_x &= i_x \cdot p = i_x \cdot i_u p_u + i_x \cdot j_v p_v + i_x \cdot k_w p_w \\ p_y &= i_y \cdot p = i_y \cdot i_u p_u + i_y \cdot j_v p_v + i_y \cdot k_w p_w \\ p_z &= i_z \cdot p = i_z \cdot i_u p_u + i_z \cdot j_v p_v + i_z \cdot k_w p_w \end{aligned} \tag{1.4}$$

o expresado en forma matricial:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} i_x \cdot i_u & i_x \cdot j_v & i_x \cdot k_w \\ j_y \cdot i_u & j_y \cdot j_v & j_y \cdot k_w \\ k_z \cdot i_u & k_z \cdot j_v & k_z \cdot k_w \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} \tag{1.5}$$

Utilizando esta notación, la matriz R en la ecuación (2) está dada por:

$$R = \begin{bmatrix} i_x \cdot i_w & i_x \cdot j_y & i_x \cdot k_w \\ j_x \cdot i_w & j_y \cdot j_x & j_x \cdot k_w \\ k_x \cdot i_w & k_x \cdot j_z & k_z \cdot k_w \end{bmatrix} \tag{1.6}$$

Análogamente, se pueden obtener las coordenadas de p_{uvw} con las coordenadas de p_{xyz} :

$$p_{uvw} = Qp_{xyz} \tag{1.7}$$

o

$$\begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} = \begin{bmatrix} i_x \cdot i_w & i_x \cdot j_x & i_x \cdot k_w \\ j_x \cdot i_w & j_x \cdot j_x & j_x \cdot k_w \\ k_z \cdot i_w & k_z \cdot j_z & k_z \cdot k_w \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{1.8}$$

Como los productos escalares son conmutativos, se puede ver en las ecuaciones (1.6) a (1.8) que:

$$Q = R^{-1} = R^T \tag{1.9}$$

y

$$QR = R^T R = R^{-1} R = I^3 \tag{1.10}$$

donde I^3 es la matriz identidad 3 x 3. La transformación en la ecuación (1.2) o (1.7) se llama transformación *ortogonal*, y como los vectores en los productos escalares son todos vectores unitarios, se llama también transformación *ortonormal*.

El interés primario en desarrollar la matriz de transformación anterior es encontrar las matrices de rotación que representan las rotaciones del sistema de coordenadas $OUVW$ respecto a cada uno de los tres ejes principales del sistema de coordenadas de referencia $OXYZ$. Si el sistema de coordenadas $OUVW$ se gira un ángulo α respecto al eje OX para llegar a una nueva posición en el espacio, entonces el punto p_{uvw}, que tiene coordenadas $(p_u, p_v, p_w)^T$ con respecto al sistema $OUVW$, tendrá coordenadas diferentes $(p_x, p_y, p_z)^T$ con respecto al sistema de referencia $OXYZ$. La matriz de transformación necesaria $R_{x,\alpha}$ se llama la matriz de rotación respecto al OX con ángulo α. $R_{x,\alpha}$ se puede derivar del concepto de matriz de transformación anterior, esto es:

$$p_{xyz} = R_{x,\alpha} \tag{1.11}$$

con $i_s = i_u$, y

$$\mathbf{R}_{s,u} = \begin{bmatrix} i_s \cdot i_u & i_s \cdot j_u & i_s \cdot k_u \\ j_s \cdot i_u & j_s \cdot j_u & j_s \cdot k_u \\ k_s \cdot i_u & k_s \cdot j_u & k_s \cdot k_u \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha \\ 0 & \operatorname{sen}\alpha & \cos\alpha \end{bmatrix} \tag{1.12}$$

Análogamente, las matrices de rotación 3 x 3 para rotaciones respecto al eje OV con ángulo ϕ y respecto al eje OZ con ángulo θ son, respectivamente :

$$R_{v,\phi} = \begin{bmatrix} \cos\phi & 0 & \operatorname{sen}\phi \\ 0 & 1 & 0 \\ -\operatorname{sen}\phi & 0 & \cos\phi \end{bmatrix} \quad R_{z,\theta} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.13}$$

Las matrices $R_{s,u}$, $R_{v,\phi}$ y $R_{z,\theta}$ se llaman *matrices de rotación básicas*. Se pueden obtener otras matrices de rotacion finitas a partir de éstas.

1.3.1.2 INTERPRETACIÓN GEOMÉTRICA DE LAS MATRICES DE ROTACIÓN

Es conveniente interpretar las matrices de rotación basica de forma geométrica. Escojamos un punto p fijo en el sistema de coordenadas $OUVW$ que sea $(1, 0, 0)^T$, esto es, $p_{uvw} = i_u$. Entonces la primera columna de la matriz de rotacion representa las coordenadas de este punto con respecto al sistema de coordenadas $OXYZ$. Análogamente, escogiendo p como $(0, 1, 0)^T$ y $(0, 0, 1)^T$, se puede identificar que los elementos de la segunda y tercera columna de una matriz de rotación representan los ejes OV y OW, respectivamente, del sistema de coordenadas $OUVW$ con respecto al sistema de coordenadas $OXYZ$. Así, dado un sistema de referencia $OXYZ$ y una matriz de rotación, los vectores columna de la matriz de rotación representan los ejes principales del sistema de coordenadas $OUVW$ con respecto al sistema de referencia y se puede deducir la localización de todos los ejes principales del sistema de coordenadas $OUVW$ con respecto al sistema de referencia. En otras palabras, una matriz de rotación geométricamente representa los ejes principales del sistema de coordenadas rotado con respecto al sistema de coordenadas de referencia.

Como la inversa de una matriz de rotación es equivalente a su transpuesta, los vectores fila de la matriz de rotación representan los ejes principales del sistema de referencia $OXYZ$ con respecto al sistema de coordenadas rotado $OUVW$. Esta interpretación geométrica de las matrices de rotación es un concepto importante que proporciona indicaciones en muchos problemas cinemáticos del brazo del robot. Se dan a continuación algunas propiedades útiles de las matrices de rotación:

1. Cada vector columna de la matriz de rotación es una representación del vector unitario del eje rotado expresado en términos de los vectores unitarios de los ejes del sistema de

referencia, y cada vector fila es una representacion del vector unitario de los ejes de referencia expresado en función de los vectores unitarios de los ejes rotados del sistema $OUVW$.

2. Como cada fila y columna es una representación de un vector unitario, la magnitud de cada una de ellas deberia ser igual a 1. Ésta es una propiedad directa de un sistema de coordenadas ortonormal. Mas aún, el determinante de una matriz de rotación es +1 para un sistema de coordenadas dextrogiro y -1 para un sistema de coordenadas levogiro.
3. Como cada fila es una representación vectorial de vectores ortonormales, el producto interno (producto escalar) de cada fila por cualquier otra fila es igual a cero. Análogamente, el producto interno de cada columna por cualquier otra columna también es igual a cero.
4. La inversa de una matriz de rotación es la transpuesta de la matriz de rotacion.

$$R^{-1} = R^{T} \qquad y \qquad RR^{T} = I^{3} \tag{1.14}$$

donde I^{3} es una matriz identidad de 3 x 3.

1.3.1.3 COORDENADAS HOMOGÉNEAS Y MATRIZ DE TRANSFORMACIÓN

Como una matriz de rotación 3 X 3 no nos da ninguna posibilidad para la traslación y el escalado, se introduce una cuarta coordenada o componente al vector de posición $p = (p_x, p_y, p_z)^T$ en un espacio tridimensional que lo transforma en $\hat{p} = (wp_x, wp_y, wp_z, w)^T$. Decimos que el vector de posición $\hat{p}$ se expresa en *coordenadas homogéneas*. El concepto de una representación en coordenadas homogéneas en un espacio euclidiano tridimensional es útil para desarrollar transformaciones matriciales que incluyan rotación, traslación, escalado y transformación de perspectiva. En general, la representación de un vector de posición de N componentes por un vector de $(N + 1)$ componentes se llama *representacion en coordenadas homogéneas*. En una representación en coordenadas homogéneas, la representación de un vector N - dimensional se efectúa en el espacio $(N + 1)$ - dimensional y el vector físico N - dimensional se obtiene dividiendo las coordenadas homogéneas por la coordenada $N + 1$ que es w. Así, en un espacio tridimensional, un vector de posición $p = (p_x, p_y, p_z)^T$ se representa por un vector ampliado $(wp_x, wp_y, wp_z, w)^T$ en la representación de coordenadas homogéneas. Las coordenadas físicas se relacionan con las coordenadas homogéneas como sigue:

$$p_x = \frac{wp_x}{w}, \qquad p_y = \frac{wp_y}{w}, \qquad p_z = \frac{wp_z}{w} \tag{1.15}$$

No existe una representación en coordenadas homogéneas única para una representacion en un espacio tridimensional.

Por ejemplo $\hat{p}_1 = (w_1 p_x, w_1 p_y, w_1 p_z, w_1)^T$ y $\hat{p}_2 = (w_2 p_x, w_2 p_y, w_2 p_z, w_2)^T$ son todas coordenadas homogéneas representando el mismo vector de posicion $p = (p_x, p_y, p_z)^T$. Asi se puede ver a la cuarta componente de las coordenadas homogéneas w como un factor de escala. Si esta coordenada es la unidad $(w = 1)$,entonces las coordenadas homogeneas transformadas de un vector de posición son las mismas que las coordenadas físicas del vector

La matriz de transformación homogénea es una matriz 4 x 4 que transforma un vector de posicion expresado en coordenadas homogéneas desde un sistema de coordenadas hasta otro sistema de coordenadas. Se puede considerar que una matriz de transformacion homogénea consiste en cuatro submatrices :

$$T = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ \hline f_{1\times3} & 1\times1 \end{bmatrix} \qquad \begin{bmatrix} \text{matriz de rotación} & \text{vector de posición} \\ \hline \text{transformación de perspectiva} & \text{escalado} \end{bmatrix} \tag{1.16}$$

La submatriz 3 x 3 superior izquierda representa la matriz de rotación; la submatriz superior derecha 3 x 1 representa el vector de posición del origen del sistema de coordenada rotado con respecto al sistema de referencia; la submatriz inferior izquierda 1 x 3 representa la transformación de perspectiva; y el cuarto elemento diagonal es el factor de escala global. La matriz de transformación homogénea se puede utilizar para explicar la relación geometrica entre el sistema ligado al cuerpo $OUVW$ y el sistema de coordenadas de referencia $OXYZ$.

Si un vector de posición p en un espacio tridimensional se expresa en coordenadas homogéneas (es decir, $\hat{p} = (p_x, p_y, p_z, 1)^T$, entonces, utilizando el concepto de matriz de transformación, una matriz de rotación 3 x 3 se puede ampliar a una matriz de rotación homogénea 4 x 4 T_{rot} para operaciones de rotación pura. Asi, las ecuaciones (1.12) y (1.13), expresadas como matrices de rotacion, se hacen:

$$T_{x,\alpha}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\operatorname{sen}\alpha & 0\\ 0 & \operatorname{sen}\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad T_{y,\phi}=\begin{bmatrix}\cos\phi & 0 & \operatorname{sen}\phi & 0\\ 0 & 1 & 0 & 0\\ -\operatorname{sen}\phi & 0 & \cos\phi & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{z,\theta}=\begin{bmatrix}\cos\theta & -\operatorname{sen}\theta & 0 & 0\\ \operatorname{sen}\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad (1.17)$$

Estas matrices de rotación 4 x 4 se llaman *matrices de rotación homogéneas básicas.*

La submatriz superior derecha 3 x 1 de la matriz de transformacion homogénea tiene el efecto de trasladar el sistema de coordenadas $OUVW$ que tiene ejes paralelos al sistema de coordenadas de referencia $OXYZ$, pero cuyo origen está en (dx, dy, dz) del sistema de coordenadas de referencia:

$$T_{trans}=\begin{bmatrix}1 & 0 & 0 & dx\\ 0 & 1 & 0 & dy\\ 0 & 0 & 1 & dz\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad (1.18)$$

Esta matriz de transformación 4 x 4 se llama *matriz de traslación homogénea básica.*

Los elementos de la diagonal principal de una matriz de transformación homogénea producen escalado local y global. Los primeros tres elementos diagonales producen un alargamiento o escalado local como en :

$$\begin{bmatrix}a & 0 & 0 & 0\\ 0 & b & 0 & 0\\ 0 & 0 & c & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}=\begin{bmatrix}ax\\ by\\ cz\\ 1\end{bmatrix} \qquad (1.19)$$

Así, los valores de las coordenadas se alargan mediante los escalares a, b y c, respectivamente. Obsérvese que las matrices de rotación básicas T_{rot}, no producen ningún efecto de escalado local.

El cuarto elemento diagonal produce escalado global como en:

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} \tag{1.20}$$

donde s > 0. Las coordenadas cartesianas físicas del vector son:

$$p_x = \frac{x}{s}, \quad p_y = \frac{y}{s}, \quad p_z = \frac{z}{s}, \quad w = \frac{s}{s} = 1 \tag{1.21}$$

Por tanto, el cuarto elemento diagonal en la matriz de transformación homogénea tiene el efecto deglobalmente reducir las coordenadas si $s > 1$ y de alargar las coordenadas si $0 < s < 1$.
En resumen, una matriz homogénea 4 x 4 transforma un vector expresado en coordenadas homogéneas con respecto al sistema de coordenadas $OUVW$ en el sistema de coordenadas de referencia $OXYZ$. Esto es, con $w = 1$.

$$\hat{p}_{xyz} = T\hat{p}_{uvw} \tag{1.22}$$

y

$$T = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{s} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.23}$$

1.3.1.4 INTERPRETACIÓN GEOMÉTRICA DE LAS MATRICES DE TRANSFORMACIONES HOMOGÉNEAS

En general, una matriz de transformación homogénea para un espacio tridimensional se puede representar como la ecuación (1.23). Escojamos un punto p fijo en el sistema de coordenadas $OUVW$ y expresado en coordenadas homogéneas como $(0, 0, 0, 1)^T$; esto es, p_{uvw} es el origen del sistema de coordenadas $OUVW$. Entonces la submatriz superior derecha 3 x 1 indica la posición del origen del sistema $OUVW$ con respecto al sistema de coordenadas de referencia $OXYZ$. Escojamos el punto p como $(1, 0, 0, 1)^T$; esto es, $p_{uvw} = i_w$. Más aún, suponemos que los orígenes de ambos sistemas de coordenadas coinciden en un punto O. Esto tiene el efecto de hacer de los elementos en la submatriz superior derecha 3 x 1 un vector nulo. Entonces la primera columna (o vector n) de la matriz de transformación homogénea representa las coordenadas del eje OU de

$OUVW$ con respecto al sistema de coordenadas $OXYZ$. Análogamente, tomando p como $(0, 1, 0, 1)^T$ y $(0, 0, 1, 1)^T$, se puede identificar que la segunda columna (o vector s) o la tercera columna (o vector a) de los elementos de la matriz homogénea representan, respectivamente, los ejes OV y OW del sistema de coordenadas $OUVW$ con respecto al sistema de coordenadas de referencia. Así dado un sistema de referencia $OXYZ$ y una matriz de transformación homogénea T, los vectores columnas de la submatriz rotación representan los ejes principales del sistema de coordenadas $OUVW$ con respecto al sistema de coordenadas de referencia, y se puede dibujar la orientación de todos los ejes principales del sistema de coordenadas $OUVW$ con respecto al sistema de coordenadas de referencia. El vector de la cuarta columna de la matriz de transformación homogenea representa la posición del origen del sistema de coordenadas $OUVW$ con respecto al sistema de referencia. En otras palabras, una matriz de transformacion homogenea representa geometricamente la *localizacion* de un sistema de coordenadas rotado (posición y orientación) con respecto a un sistema de coordenadas de referencia.

Como la inversa de la submatriz de rotación es equivalente a su transpuesta, los vectores fila de una submatriz de rotación representan los ejes principales del sistema de coordenadas de referencia con respecto al sistema de coordenadas $OUVW$. Sin embargo, la inversa de una matriz de transformación homogénea no es equivalente a su transpuesta. La posición del origen en el sistema de coordenadas $OUVW$ se puede deducir solamente después de que se determine la inversa de la matriz de transformación homogénea. En general, se puede encontrar que la inversa de una matriz de transformación homogénea es :

$$T^{-1} = \begin{bmatrix} n_x & n_y & n_z & -n^T p \\ s_x & s_y & s_z & -s^T p \\ a_x & a_y & a_z & -a^T p \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & R_{3\times 3}^T & & -n^T p \\ & & & -s^T p \\ & & & -a^T p \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1.24)$$

Así de la ecuación 1.24, los vectores columna de la inversa de una matriz de transformación homogénea representan los ejes principales de los ejes de referencia con respecto al sistema de coordenadas rotado $OUVW$, y la submatriz 3 x 1 superior derecha representa la posición del origen del sistema de referencia con respecto al sistema $OUVW$.

1.3.1.5 LA REPRESENTACIÓN DE DENAVIT - HARTENBERG

Para describir la relación de traslación y rotación entre elementos adyacentes, Denavit y Hartenberg [1] propusieron un método matricial para establecer de forma sistemática un sistema de coordenadas (sistema ligado al cuerpo) para cada elemento de una cadena articulada. La representacion de Denavit - Hartenberg resulta en una matriz de transformación homogénea 4 x 4 que representa cada uno de los sistemas de coordenadas de los elementos en la articulación con respecto al sistema de coordenadas del elemento previo. Así mediante transformaciones secuenciales, el efector final expresado en las «coordenadas de la mano» se puede transformar y expresar en las «coordenadas de base» que constituyen el sistema inercial de este sistema dinámico.

Se puede establecer para cada elemento en sus ejes de articulación un sistema de coordenadas cartesiano ortonormal $(x_i, y_i, z_i)^*$, donde $i = 1,2,\ldots, n$ (n = número de grados de libertad), mas el sistema de coordenadas de la base. Como una articulacion giratoria tiene solamente un grado de libertad, cada sistema de coordenadas (x_i, y_i, z_i) del brazo de un robot corresponde a la articulacion $i+1$ y esta fija en el elemento i. Cuando el actuador de la articulacion activa la articulación i, el elemento i se movera con respecto al elemento $i-1$. Como el sistema de coordenadas i-esimo está fijo en el elemento i, se mueve junto con el elemento i. Así pues, el sistema de coordenadas n-ésimo se mueve con la mano (elemento n). Las coordenadas de la base se definen como el sistema de coordenadas numero 0 (x_0, y_0, z_0), que tambien es el sistema de coordenadas inercial del brazo.

Cada sistema de coordenadas se determina y establece sobre la base de tres reglas:

1. El eje z_{i-1} yace a lo largo del eje de la articulación.
2. El eje x_i es normal al eje z_{i-1} y apunta hacia afuera de él.
3. El eje y_i completa el sistema de coordenadas dextrógiro según se requiera.

Mediante estas reglas, uno es libre de escoger la localización del sistema de coordenadas 0 en cualquier parte de la base soporte, mientras el eje z_0 esté a lo largo del eje de movimiento de la primera articulación. El último sistema de coordenadas (el n-ésimo) se puede colocar en cualquier parte de la mano, mientras que el eje x_n es normal al eje z_{n-1}.

La representación de Denavit - Hartenberg de un elemento rígido depende de cuatro parámetros geométricos asociados con cada elemento. Estos cuatro parámetros describen completamente cualquier articulación prismática o de revolución. Refiriéndose a la Figura 1.6, estos cuatro parámetros se definen como sigue:

θ_i: Es el ángulo de la articulación del eje x_{i-1} al eje x_i respecto del eje z_{i-1} (utilizando la regla de la mano derecha).

---

[1] Denavit, J y Hartenberg, R. S. [1995]: «A Kinematic Notation for Lower - Pair Mechanisms Based on Matrices», J. App. Mech, vol. 77, pags 215-221

d_i : Es la distancia desde el origen del sistema de coordenadas $(i - 1)$ - ésimo hasta la intersección del eje z_{i-1} con el eje x_i a lo largo del eje z_{i-1} .

a_i :Es la distancia de separacion desde la intersección del eje z_{i-1} con el eje x_i hasta el origen del sistema i - ésimo a lo largo del eje x_i (o la distancia más corta entre los ejes z_{i-1} y z_i).

α_i Es el ángulo de separacion del eje z_{i-1} al eje z_i respecto al eje x_i (utilizando la regla de la mano derecha).

Para una articulacion giratoria, d_i, a_i, y α_i son los parámetros de la articulación y permanecen constantes para un robot mientras que θ_i es la variable de la articulación que cambia cuando el elemento i se mueve (o gira) con respecto al elemento $i - 1$. Para la articulación prismática θ_i, a_i, y α_i son los parámetros de la articulación y permanecen constantes para un robot, mientras que d_i es la variable de la articulación.

El siguiente algoritmo sirve para establecer un sistema de coordenadas ortonormal *consistente* para un robot.

Dado un brazo con n grados de libertad, este algoritmo asigna un sistema de coordenadas ortonormal a cada elemento del brazo de acuerdo a configuraciones de brazos similar a aquellas de la geometria del brazo humano. El etiquetado del sistema de coordenadas va desde la base soporte hasta el efector final del brazo. Las relaciones entre elementos adyacentes se pueden representar mediante una matriz de transformación homogenea 4 x 4. La importancia de esta asignación es que ayudará al desarrollo de un procedimiento consistente para derivar la solución de la articulación.

D1. *Establecer el sistema de coordenadas de la base*. Establecer un sistema de coordenadas ortonormal dextrógiro (x_0, y_0, z_0) en la base soporte con el eje z_0 estando a lo largo del eje de movimiento de la articulación 1 y apuntando hacia afuera del hombro del brazo del robot. Los ejes x_0 y y_0 se pueden establecer convenientemente y son normales al eje z_0 .

D2. *Inicializar y repetir*. Para cada i , $i = 1, \ldots, n - 1$, realizar los pasos **D3** a **D6**.

D3. *Establecer los ejes de la articulación*. Alinear el z_i con el eje de movimiento (giratorio o deslizante) de la articulación $i + 1$. Para robots que tengan configuraciones de brazo levogiras, los ejes z_1 y z_2 estan apuntando hacia afuera del hombro y el «tronco» del brazo del robot.

D4. *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 de los ejes z_i y z_{i-1} y el eje z_i.

D5. *Establecer el eje* x_i. Establecer $x_i = \pm (z_{i-1} \times z_i) / \|z_{i-1} \times z_i\|$ o a lo largo de la normal común entre los ejes z_{i-1} y z_i cuando son paralelos.

D6. *Establecer el eje* y_i. Asignar $y_i = \pm (z_i \times x_i) / \|z_i \times x_i\|$ para completar el sistema de coordenadas dextrógiro. (Extender si es necesario los ejes z_i y x_i a los pasos D9 a D12)

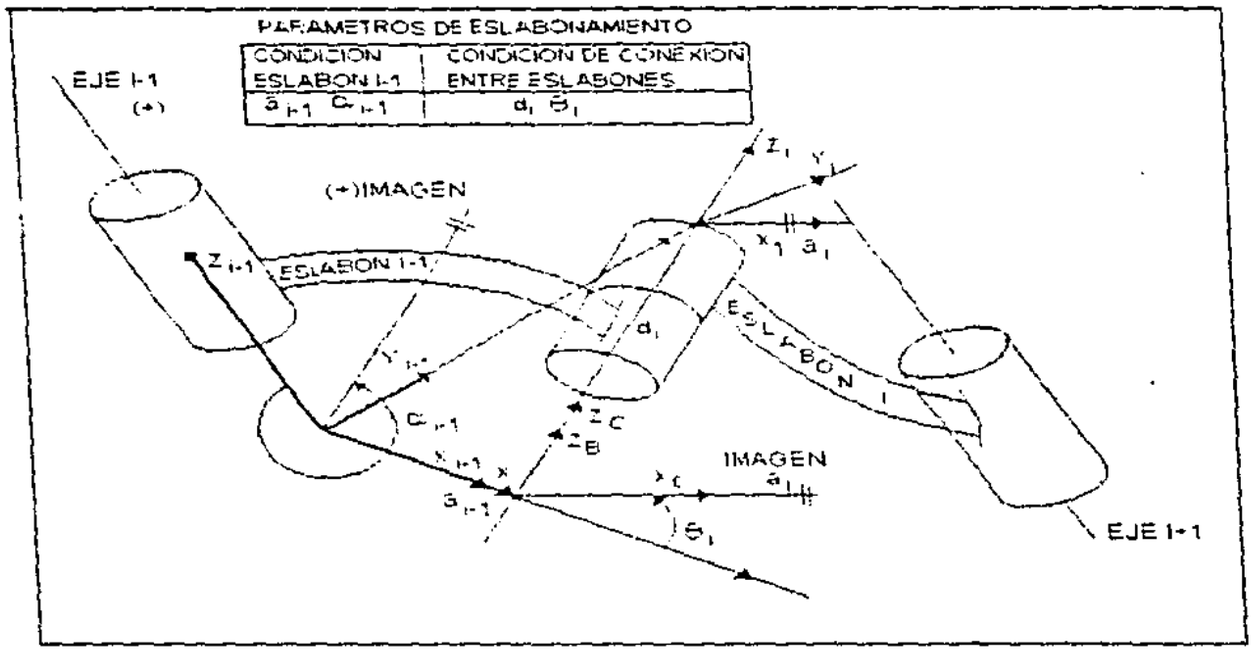

Figura 1.6: Sistema de coordenadas de elementos y sus parámetros

D7. *Establecer el sistema de coordenadas de la mano.* Normalmente la articulación n - ésima es de tipo giratorio. Establecer z_n a lo largo de la dirección del eje z_{n-1} y apuntando hacia afuera del robot. Establecer x_n tal que sea normal a ambos ejes z_{n-1} y z_n. Asignar y_n para completar el sistema de coordenadas dextrógiro.

D8. *Encontrar los parámetros de la articulación y del elemento.* Para i, $i = 1, \ldots n$, realizar los pasos D9 y D12.

D9. *Encontrar* d_i. d_i es la distancia del origen del sistema de coordenadas $(i-1)$ - ésimo a la intersección del eje z_{i-1} y el eje x_i a lo largo del eje z_{i-1}. Es la variable de la articulación si i es prismática.

D10. *Encontrar* a_i. a_i es la distancia de la intersección del eje z_{i-1} y el eje x_i al origen del sistema de coordenadas i - ésimo a lo largo de eje x_i.

D11. *Encontrar* θ_i θ_i es el ángulo de rotación desde el eje x_{i-1} hasta el eje x_i respecto del eje z_{i-1}. Es la variable de articulación si i es giratoria.

D12. *Encontrar* α_i. α_i es el ángulo de rotación desde el eje z_{i-1} hasta el eje z_i respecto del eje x_i.

Una vez establecido el sistema de coordenadas Denavit - Hartenberg para cada elemento, se puede desarrollar fácilmente una matriz de transformación homogénea que relacione el sistema de coordenadas i - ésimo con el sistema de coordenadas $(i-1)$ - ésimo. Observando la figura 1.6, es obvio que un punto r_i expresado en el sistema de coordenadas i - ésimo se puede expresar en el sistema de coordenadas $(i-1)$ - ésimo como r_{i-1} realizando las siguientes transformaciones sucesivas:

1. Girar respecto del eje z_{i-1} un ángulo de θ_i para alinear el eje x_{i-1} con el eje x_i (el eje x_{i-1} es paralelo a z_i y apunta en la misma dirección).
2. Trasladar a lo largo del eje z_{i-1} una distancia de 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 también 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.

Cada una de estas cuatro operaciones se puede expresar mediante una matriz de rotación - traslación homogénea básica y el producto de estas cuatro matrices de transformación homogéneas básicas da una matriz de transformación homogénea compuesta, ${}^{i-1}A_i$, conocida como la matriz de transformación de Denavit - Hartenberg para sistemas de coordenadas, adyacentes i e $i-1$. Así :

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

$$ = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\operatorname{sen}\alpha_i & 0 \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = $$

$$= \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \operatorname{sen}\theta_i & \operatorname{sen}\alpha_i \operatorname{sen}\theta_i & a_i \cos\theta_i \\ \operatorname{sen}\theta_i & \cos\alpha_i \cos\theta_i & -\operatorname{sen}\alpha_i \cos\theta_i & a_i \operatorname{sen}\theta_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.25)$$

Utilizando la ecuación 1.25 se puede encontrar a la inversa de esta transformación como:

$$\left[{}^{i-1}A_i\right]^{-1} = {}^{i}A_{i-1} = \begin{bmatrix} \cos\theta_i & \operatorname{sen}\theta_i & 0 & -a_i \\ -\cos\alpha_i \operatorname{sen}\theta_i & \cos\alpha_i \cos\theta_i & \operatorname{sen}\alpha_i & -d_i \operatorname{sen}\alpha_i \\ \operatorname{sen}\alpha_i \operatorname{sen}\theta_i & -\operatorname{sen}\alpha_i \cos\theta_i & \cos\alpha_i & -d_i \cos\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.26)$$

donde α_i, a_i, d_i son constantes mientras θ_i es la variable de la articulación para una articulación tipo revolución.

Para una articulación prismática, la variable de la articulación es d_i, mientras que α_i, a_i, θ_i son constantes. En este caso, ${}^{i-1}A_i$ se hace:

$${}^{i-1}A_i = T_{z,d} T_{z,\theta} T_{x,a} T_{x,\alpha} = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \operatorname{sen}\theta_i & \operatorname{sen}\alpha_i \operatorname{sen}\theta_i & 0 \\ \operatorname{sen}\theta_i & \cos\alpha_i \cos\theta_i & -\operatorname{sen}\alpha_i \cos\theta_i & 0 \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.27)$$

y su inversa es:

$$\left[{}^{i-1}A_i\right]^{-1} = {}^{i}A_{i-1} = \begin{bmatrix} \cos\theta_i & \operatorname{sen}\theta_i & 0 & 0 \\ -\cos\alpha_i \operatorname{sen}\theta_i & \cos\alpha_i \cos\theta_i & \operatorname{sen}\alpha_i & -d_i \operatorname{sen}\alpha_i \\ \operatorname{sen}\alpha_i \operatorname{sen}\theta_i & -\operatorname{sen}\alpha_i \cos\theta_i & \cos\alpha_i & -d_i \cos\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.28)$$

Utilizando la matriz ${}^{i-1}A_i$ se puede relacionar un punto p_i en reposo en el elemento i y expresado en coordenadas homogéneas con respecto al sistema de coordenadas i en el sistema de coordenadas $i-1$ establecido en el elemento $i-1$ por:

$$p_{i-1} = {}^{i-1}A_i p_i \quad (1.29)$$

donde $p_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1}, 1)^T$ y $p_i = (x_i, y_i, z_i, 1)^T$.

1.3.1.6 ECUACIONES CINEMÁTICAS PARA MANIPULADORES

La matriz homogénea ${}^0T_i$ que especifica la localización del sistema de coordenadas i - ésimo con respecto al sistema de coordenadas de la base es el producto en cadena de matrices de transformacion de coordenadas sucesivas ${}^{i-1}A_i$ y se expresa como:

$$ {}^0T_i = {}^0A_1\,{}^1A_2 \ldots {}^{i-1}A_i = \prod_{j=1}^{i} {}^{j-1}A_j \quad \text{para i} = 1,2,\ldots,n $$

$$ = \begin{bmatrix} x_i & y_i & z_i & p_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^0R_i & {}^0p_i \\ 0 & 1 \end{bmatrix} \qquad (1.30) $$

donde:

$[x_i, y_i, z_i]$ = matriz de orientación del sistema de coordenadas i - ésimo establecido en el elemento i con respecto al sistema de coordenadas de la base. Es la matriz particionada superior izquierda 3 x 3 de ${}^0T_i$.

p_i = vector de posición que apunta desde el origen del sistema de coordenadas de la base hasta el origen del sistema de coordenadas i - ésimo. Es la matriz particionada superior derecha 3 x 1 de ${}^0T_i$.

1.3.2 EL PROBLEMA CINEMÁTICO INVERSO

Los robots basados en computadora se suelen controlar en el espacio de las variables de articulación, mientras que los objetos que se manipulan se suelen expresar en el sistema de coordenadas del mundo. Con el fin de controlar la posición y orientación del efector final de un robot para alcanzar un objeto, es más importante la solución cinemática inversa. En otras palabras, dada la posición y orientación del efector final de un brazo de robot y sus parámetros de articulación y elementos, nos gustaría encontrar los ángulos de articulación correspondientes del robot de manera que se pueda posicionar como se desee el efector final.

En general, el problema cinematico inverso se puede resolver por diversos métodos, tales como la transformación inversa (Paul y col.[1981]), el álgebra de tornillo (Kohli y Soni [1975]), matrices duales (Denavit [1956]) y cuaterniones duales (Yang y Freudenstein [1984]).

1.3.2.1 TÉCNICA DE LA TRANSFORMADA INVERSA PARA SOLUCIÓN DE ÁNGULOS DE EULER

La matriz de rotación 3×3 se puede expresar en términos de los ángulos de Euler (ϕ, θ, ψ) como la ecuación matricial :

$$\begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} = R_{z,\phi} R_{y,\theta} R_{z,\psi} = \begin{bmatrix} C\phi C\psi - S\phi C\theta S\psi & -C\phi S\psi - S\phi C\theta C\psi & S\phi S\theta \\ S\phi C\psi + C\phi C\theta S\psi & -S\phi S\psi + C\phi C\theta C\psi & -C\phi S\theta \\ S\theta S\psi & S\theta C\psi & C\theta \end{bmatrix} \quad (1.31)$$

Con el fin de evaluar θ para $-\pi \leq \theta \geq \pi$, se utiliza una función arco tangente $arc\ \mathrm{tg}2(y, x)$, que devuelve $\mathrm{tg}^{-1}(y / x)$ ajustada al cuadrante apropiado. Se define como

$$\theta = arc\ \mathrm{tg}2(y, x) = \begin{cases} 0^\circ \leq \theta \geq 90^\circ & \text{para } +xy+y \\ 90^\circ \leq \theta \geq 180^\circ & \text{para } -xy+y \\ -180^\circ \leq \theta \geq -90^\circ & \text{para } -xy-y \\ -90^\circ \leq \theta \geq 0^\circ & \text{para } +xy-y \end{cases} \quad (1.32)$$

En la ecuación matricial 1.28 se dan los elementos de la matriz en el lado izquierdo , mientras que los elementos de las tres matrices del lado derecho son incógnitas y son dependientes de ϕ, θ, ψ. Paul[2] sugiere *premultiplicar* la ecuación matricial anterior por su transformada inversa desconocida sucesivamente y de los elementos de la ecuación matricial resultante determinar el ángulo incógnita. Esto es, movemos una incógnita (por su transformada inversa) del lado derecho de la ecuación matricial al lado izquierdo y resolvemos para la incógnita. A continuación movemos la siguiente incógnita al lado izquierdo y repetimos el proceso hasta que se resuelvan todas las incógnitas

Premultiplicando la ecuacion matricial anterior por $R_{z,\phi}^{-1}$ tenemos una incógnita (ϕ) en el lado izquierdo y dos incógnitas (θ, ψ) en el lado derecho de la ecuación matricial, así tenemos:

$$\begin{bmatrix} C\phi & S\phi & 0 \\ -S\phi & C\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\theta & -S\theta \\ 0 & S\theta & C\theta \end{bmatrix} \begin{bmatrix} C\psi & -S\psi & 0 \\ S\psi & C\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.33)$$

o

$$\begin{bmatrix} C\phi n_x + S\phi n_y & C\phi s_x + S\phi s_y & C\phi a_x + S\phi a_y \\ -S\phi n_x + C\phi n_y & -S\phi s_x + C\phi s_y & -S\phi a_x + C\phi a_y \\ n_z & s_z & a_z \end{bmatrix} = \begin{bmatrix} C\psi & -S\psi & 0 \\ C\theta S\psi & C\theta C\psi & -S\theta \\ S\theta S\psi & S\theta C\psi & C\theta \end{bmatrix} \quad (1.34)$$

[2] Paul, R. P.[1981] *Robot Manipulator Mathematics, programing and Control* MIT Press Cambridge mass.

Igualando los elementos (1,3) de ambas matrices en la ecuación 1.34 tenemos:

$$C\phi a_x + S\phi a_y = 0 \tag{1.35}$$

que da:

$$\phi = \text{tg}^{-1}\left(\frac{a_y}{-a_x}\right) = arc\ \text{tg}\,2(a_y, -a_x) \tag{1.36}$$

Igualando los elementos (1,1) y (1,2) de ambas matrices tenemos:

$$C\psi = C\phi n_x + S\phi n_y \tag{1.37}$$

$$S\psi = -C\phi s_x - S\phi s_y \tag{1.38}$$

que conduce a la solución para ψ:

$$\psi = \text{tg}^{-1}\left(\frac{S\psi}{C\psi}\right) = \text{tg}^{-1}\left(\frac{-C\phi s_x - S\phi s_y}{C\phi n_x + S\phi n_y}\right) = arc\ \text{tg}\,2(-C\phi s_x - S\phi s_y, C\phi n_x + S\phi n_y) \tag{1.39}$$

Igualando los elementos (2,3) y (3,3) de ambas matrices tenemos:

$$\begin{aligned} S\theta &= S\phi a_x - C\phi a_y \\ C\theta &= a_z \end{aligned} \tag{1.40}$$

que nos da la solución para θ:

$$\theta = \text{tg}^{-1}\left(\frac{S\theta}{C\theta}\right) = \text{tg}^{-1}\left(\frac{S\phi a_x - C\phi a_y}{a_z}\right) = arc\ \text{tg}\,2(S\phi a_x - C\phi a_y, a_z) \tag{1.41}$$

1.4 DINÁMICA DEL BRAZO DEL ROBOT

La dinámica del robot trata con las formulaciones matemáticas de las ecuaciones de movimiento del brazo. Las ecuaciones de movimiento de un manipulador son un conjunto de ecuaciones matematicas que describen su conducta dinámica. Tales ecuaciones son útiles para la simulación en computadora del movimiento del robot, el diseño de ecuaciones de control apropiadas para el robot y la evaluación del diseño y estructura del brazo. El objetivo del control de un manipulador basado en computadora es mantener la respuesta dinámica del mismo de acuerdo con algún rendimiento del sistema preespecificado y objetivos deseados. En general, el rendimiento dinámico de un manipulador depende directamente de la eficacia de los algoritmos de control y de su modelo dinámico. El problema de control consiste en obtener modelos dinámicos del brazo del robot físico y a continuacion especificar leyes o estrategias de control correspondientes para conseguir la respuesta y el rendimiento del sistema deseado.

El modelo dinámico de un robot se puede obtener a partir de leyes físicas conocidas tales como las leyes de la mecánica newtoniana y lagrangiana. Esto conduce al desarrollo de las ecuaciones de movimiento dinámico para las diversas articulaciones del manipulador en términos de parámetros geométricos e inerciales de los elementos. Métodos convencionales como las formulaciones de Lagrange - Euler y Newton - Euler se pueden aplicar entonces sistemáticamente para desarrollar las ecuaciones de movimiento del robot. De estas dos formulaciones se obtienen diferentes formas de describir la dinámica del brazo del robot, tales como las ecuaciones de Lagrange - Euler de Uicker[3], Bejczy[4], las ecuaciones recursivas de Lagrange de Hollerbach, las ecuaciones de Newton - Euler de Luh y las ecuaciones generalizadas d'Alembert y Lee. Estas ecuaciones de movimiento son <<equivalentes>> unas a otras en el sentido de que describen la conducta dinámica del mismo robot físico. Sin embargo, sus estructuras pueden difenr porque se obtienen por diversas razones y objetivos. Algunas se obtienen para lograr tiempos de cálculo rápido en la evaluación de los pares de las articulaciones nominales para controlar el manipulador, otras se obtienen para facilitar el análisis y la síntesis de control, y todavía otras se obtienen para mejorar la simulacion en una computadora del movimiento del robot.

La obtencion del modelo dinámico de un manipulador basado en la formulación de Lagrange - Euler es simple y sistemática. Suponiendo el movimiento del cuerpo rígido, las ecuaciones de movimiento resultante, excluyendo la dinámica de los dispositivos de control electrónico, huelgo y rozamiento de los engranajes, son un conjunto de ecuaciones diferenciales no lineales acopladas de segundo orden. Las ecuaciones de movimiento Lagrange - Euler proporcionan ecuaciones de estado explícitas para la dinámica del robot y se pueden utilizar para analizar y diseñar estrategias de control avanzadas en el espacio de las variables de articulación. En una menor medida se están utilizando para resolver el problema *dinámico directo*, esto es, dadas las fuerzas/pares deseadas, se utilizan las ecuaciones dinámicas para resolver las aceleraciones de las articulaciones, o para el *problema dinámico inverso*, esto es, dadas las coordenadas generalizadas deseadas y sus primeras dos derivadas respecto al tiempo, se calculan las fuerzas/pares generalizadas. En ambos casos se pueden necesitar calcular los coeficientes dinámicos . Desgraciadamente, el cálculo de estos coeficientes requiere una relativa cantidad de operaciones aritméticas. Así las ecuaciones de Lagrange - Euler son muy difíciles de utilizar con fines de control en tiempo real a menos que se simplifiquen.

Como una alternativa para derivar ecuaciones de movimiento más eficiente, se dirigió la atención al desarrollo de algoritmos para calcular las fuerzas/pares basados en las ecuaciones de movimiento de Newton - Euler. Las ecuaciones dinámicas resultantes, excluyendo la dinámica del dispositivo de control, huelgo y rozamiento de engranajes, son un conjunto de ecuaciones recursivas hacia adelante

---

[3] Uiker, J. J. [1965] . "On the Dynamic Analysis of Spatial Linkages using 4 x 4 Matrices" Ph. D. dissertation. Northwestern University, Evanston

[4] Bejcy, A.K. [1974]: "Robot Arm Dynamics and Control". Technical Memo 33-66, Jet Propulsion Laboratory, Pasadena, California.

y hacia atrás. Este conjunto de ecuaciones se puede aplicar secuencialmente a los elementos del robot. La recursión hacia adelante propaga la información cinemática tal como velocidades lineales, velocidades angulares, aceleraciones angulares y aceleraciones lineales del centro de masa de cada elemento desde el sistema de coordenadas inercial hasta el sistema de coordenadas de la mano. La recursión hacia atrás propaga las fuerzas y momentos ejercidos sobre cada elemento desde el efector final del manipulador hasta el sistema de referencia de la base. El resultado más significativo de esta formulación es que se encuentra que el tiempo de cálculo de las fuerzas/pares es linealmente proporcional al número de articulaciones del brazo e independiente de la configuración del mismo. Con este algoritmo se puede realizar el control en tiempo real simple del robot en el espacio de las variables de articulación.

Otro método para obtener un conjunto eficiente de ecuaciones de movimiento explícito se basa en el principio de d'Alembert generalizado para deducir las ecuaciones de movimiento que se expresan explícitamente en forma vectorial matricial apropiadas para el análisis del control.

1.4.1 FORMULACIÓN DE LAGRANGE - EULER

Las ecuaciones de movimiento general de un manipulador se pueden expresar convenientemente mediante la aplicación directa de la formulación de Lagrange - Euler a sistemas no conservativos.

La derivación de las ecuaciones dinámicas de un manipulador con n grados de libertad se basa en la comprensión de:

1. La matriz de transformación de coordenadas homogéneas 4 x 4, ${}^{i-1}A_i$, que describe la relación espacial entre los sistemas de coordenadas del elemento i - ésimo y el elemento $(i-1)$ - ésimo. Relaciona un punto fijado en el elemento i, expresado en coordenadas homogéneas con respecto al sistema de coordenadas i - ésimo en el sistema de coordenadas $(i-1)$ - ésimo.
2. La ecuación de Lagrange - Euler

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i, \qquad i = 1, 2, \ldots, n \tag{1.42}$$

donde

$\mathcal{L}$ = función lagrangiana = energía cinética K − energía potencia P;

K = energía cinética total del brazo;

P = energía potencial total del brazo;

q_i = coordenada generalizada del brazo;

$\dot{q}_i$ = primera derivada respecto al tiempo de la coordenada generalizada q_i;

τ_i = fuerza (o par) generalizado aplicado al sistema en la articulación i para mover el elemento i.

De las ecuaciones de Lagrange - Euler anteriores se requiere escoger adecuadamente un conjunto de *coordenadas generalizadas* para describir el sistema. Las coordenadas generalizadas se utilizan como un conjunto de coordenadas convenientes que describen completamente la localización (posición y orientación) de un sistema con respecto a un sistema de coordenadas de referencia. Para un manipulador simple con articulaciones giratorias - prismáticas, están disponibles diversos conjuntos de coordenadas generalizadas para describir el manipulador. Esto corresponde a las coordenadas generalizadas con las variables de articulación definidas en cada una de las matrices de transformación de coordenadas de elementos 4 x 4. Así, en el caso de una articulación giratoria, $q_i = \theta_i$, es el margen del angulo de la articulación, mientras que para una articulación prismática, $q_i = d_i$, es la distancia recorrida por la articulación.

1.4.1.1 VELOCIDADES DE LAS ARTICULACIONES DE UN ROBOT

La formulacion de Lagrange - Euler requiere el conocimiento de la energia cinética del sistema físico, que a su vez requiere un conocimiento de la velocidad de cada articulación.

Con referencia a la Figura 1.7, sea ${}^{i}r_i$ un punto fijo y en reposo en el elemento i y expresado en coordenadas homogéneas con respecto al sistema de coordenadas del elemento i - ésimo.

$$ {}^{i}r_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = (x_i, y_i, z_i, 1)^T \qquad (1.43) $$

Sea ${}^{0}r_i$ el mismo punto ${}^{i}r_i$ con respecto al sistema de coordenadas de la base, ${}^{i-1}A_i$ la matriz de transformación de coordenadas homogénea que relaciona el desplazamiento espacial del sistema de coordenadas del elemento i - ésimo con respecto al sistema de coordenadas del elemento $(i-1)$ - esimo y ${}^{0}A_i$ la matriz de transformación de coordenadas que relaciona el sistema de coordenadas i - ésimo con el sistema de coordenadas de la base; entonces ${}^{0}r_i$ esta relacionado con el punto ${}^{i}r_i$ por:

$$ {}^{0}r_i = {}^{0}A_i {}^{i}r_i \qquad (1.44) $$

donde:

$$ {}^{0}A_i = {}^{0}A_1 {}^{1}A_2 \ldots {}^{i-1}A_i \qquad (1.45) $$

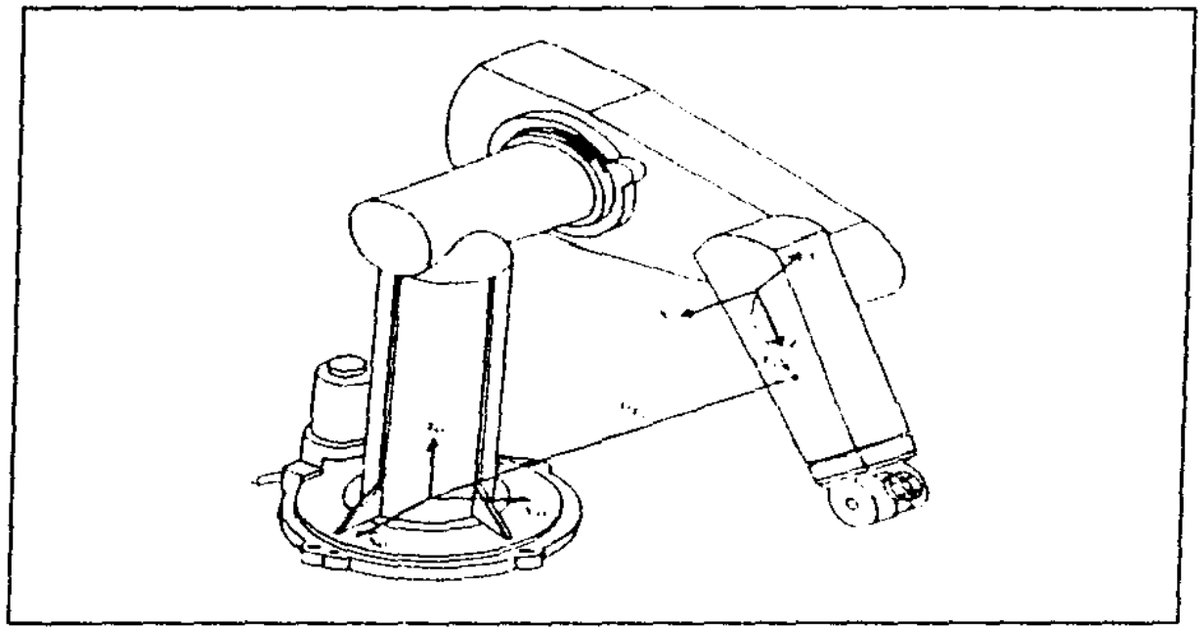

Figura 1.7: Un punto ${}^{i}r_i$ en el elemento i .

Si la articulación i es de revolución, se sigue de la ecuación (1.25) que la forma general de ${}^{i-1}A_i$ está dada por:

$$ {}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \operatorname{sen}\theta_i & \operatorname{sen}\alpha_i \operatorname{sen}\theta_i & a_i \cos\theta_i \\ \operatorname{sen}\theta_i & \cos\alpha_i \cos\theta_i & -\operatorname{sen}\alpha_i \cos\theta_i & a_i \operatorname{sen}\theta_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1.46) $$

o si la articulación i es prismática, de la ecuación (1.27), la forma general de ${}^{i-1}A_i$ es:

$$ {}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \operatorname{sen}\theta_i & \operatorname{sen}\alpha_i \operatorname{sen}\theta_i & 0 \\ \operatorname{sen}\theta_i & \cos\alpha_i \cos\theta_i & -\operatorname{sen}\alpha_i \cos\theta_i & 0 \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1.47) $$

En general, todos los elementos no nulos en la matriz ${}^{0}A_i$ son una función de $(\theta_1, \theta_2, \ldots, \theta_i)$, y α_i, a_i, d_i son parámetros conocidos de la estructura cinemática del brazo y θ_i o d_i es la variable de articulación del elemento i .

Como el punto ${}^i r_i$ está en reposo en el elemento i, y suponiendo el movimiento del cuerpo rígido, otros puntos así como el punto ${}^i r_i$ fijado en el elemento i y expresado con respecto al sistema de coordenadas i - ésimo tendrá velocidad nula con respecto a dicho sistema de coordenada (que no es un sistema inercial). La velocidad de ${}^i r_i$ expresada en el sistema de coordenadas de la base (que es un sistema inercial) se puede expresar como:

$$
\begin{aligned}
{}^0 v_i \equiv v_i &= \frac{d}{dt}\left({}^0 r_i\right) = \frac{d}{dt}\left({}^0 A_i {}^i r_i\right) = {}^0 A_1 {}^1 A_2 \ldots {}^{i-1} A_i {}^i r_i + A_1 {}^1 \dot{A}_2 \ldots {}^{i-1} A_i {}^i r_i + \ldots \\
&+ {}^0 A_1 \ldots {}^{i-1} \dot{A}_i {}^i r_i + {}^{i-1} A_i {}^i \dot{r}_i = \left(\sum_{j=1}^{i} \frac{\partial {}^0 A_i}{\partial q_j} \dot{q}_j\right) {}^i r_i
\end{aligned}
\tag{1.48}
$$

La forma compacta anterior se obtiene porque ${}^i \dot{r}_i = 0$. La derivada parcial de ${}^0 A_i$ con respecto a q_j se puede calcular fácilmente con la ayuda de una matriz Q_i que, para una articulación de revolución se define como:

$$
Q_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.49}
$$

y para una articulación prismática, como:

$$
Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.50}
$$

Entonces se sigue que:

$$
\frac{\partial {}^{i-1} A_i}{\partial q_i} = Q_i {}^{i-1} A_i \tag{1.51}
$$

De aquí, para $i = 1,2,\ldots,n$:

$$
\frac{\partial {}^0 A_i}{\partial q_j} = \begin{cases} {}^0 A_1 {}^1 A_2 \ldots {}^{j-2} A_{j-1} Q_j {}^{j-1} A_j \ldots {}^{i-1} A_i & \text{para } j \le i \\ 0 & \text{para } j > i \end{cases} \tag{1.52}
$$

La ecuación (1.52) se puede interpretar como el efecto del movimiento de la articulación j sobre todos los puntos en el elemento i. Con el fin de simplificar las notaciones, definamos $U_{ij} \equiv \partial {}^0 A_i / \partial q_j$, entonces la ecuación (1.52) se puede escribir como sigue para $i = 1,2,\ldots,n$:

$$U_{ij} = \begin{cases} {}^{0}A_{j-1}Q_j{}^{j-1}A_i & para\ j \le i \\ 0 & para\ j > i \end{cases} \tag{1.53}$$

Utilizando esta notación, v_i se puede expresar como:

$$v_i = \left(\sum_{j=1}^{i} U_{ij}\dot{q}_j \right) {}^{i}r_i \tag{1.54}$$

Conviene apuntar que la derivada parcial de ${}^{i-1}A_i$ con respecto a q_i resulta en una matriz que no retiene la estructura de una matriz de transformación de coordenadas homogéneas. Para una articulación giratoria, el efecto de premultiplicar ${}^{i-1}A_i$ por Q_i es equivalente a intercambiar los elementos de las dos primera filas de ${}^{i-1}A_i$, negando todos los elementos de la primera fila y anulando todos los elementos de la tercera y cuarta . Para una articulación prismática, el efecto es sustituir los elementos en las otras filas. La ventaja de utilizar las matrices Q_i es que podemos todavía utilizar las matrices ${}^{i-1}A_i$ y aplicar las operaciones anteriores a ${}^{i-1}A_i$ cuando se premultiplica por la Q_i.

1.4.1.2 ENERGÍA CINÉTICA DE UN MANIPULADOR

Sea K_i la energía cinética del elemento i , $i = 1,2,\ldots,n$, expresada en el sistema de coordenadas de la base y sea dK_i la energía cinética de una partícula con masa diferencial dm en el elemento i ; entonces:

$$\begin{aligned} dK_i &= \tfrac{1}{2}\left(\dot{x}_i^2 + \dot{y}_i^2 + \dot{z}_i^2\right)dm \\ &= \tfrac{1}{2}\,traza(v_i v_i^T)dm = \tfrac{1}{2}\,Tr(v_i v_i^T)dm \end{aligned} \tag{1.55}$$

donde se utiliza un operador traza * en lugar de un producto escalar de vectores en la ecuación anterior para formar el tensor del cual se puede obtener la matriz de inercia del elemento J_i (o matriz pseudoinercia). Sustituyendo v_i de la ecuación (1.54), la energía cinética de la masa diferencial es:

* $TrA \triangleq \sum_{i=1}^{n} a_{ii}$

$$dK_i = \tfrac{1}{2} Tr\left[\sum_{p=1}^{i} U_{ip}\dot{q}_p {}^i r_i \left(\sum_{r=1}^{i} U_{ir}\dot{q}_r {}^i r_i\right)^T\right] dm$$

$$= \tfrac{1}{2} Tr\left[\sum_{p=1}^{i}\sum_{r=1}^{i} U_{ip} {}^i r_i {}^i r_i^T U_{ir}^T \dot{q}_p \dot{q}_r\right] dm \qquad (1.58)$$

$$= \tfrac{1}{2} Tr\left[\sum_{p=1}^{i}\sum_{r=1}^{i} U_{ip}\left({}^i r_i dm {}^i r_i^T\right) U_{ir}^T \dot{q}_p \dot{q}_r\right]$$

La matriz U_{ij} es la velocidad de cambio de los puntos $({}^i r_i)$ sobre el elemento i relativo al sistema de coordenadas de la base cuando q_j cambia. Es constante para todos los puntos en el elemento i. También $\dot{q}_i$ son independientes de la distribución de masa del elemento i, así que, sumando todas las energías cinéticas de todos los elementos y poniendo la integral dentro de los corchetes tenemos:

$$K_i = \int dK_i = \tfrac{1}{2} Tr\left[\sum_{p=1}^{i}\sum_{r=1}^{i} U_{ip}\left(\int {}^i r_i {}^i r_i^T dm\right) U_{ir}^T \dot{q}_p \dot{q}_r\right] \qquad (3.57)$$

El término integral dentro del corchete es la inercia de todos los puntos en el elemento i, de aquí que:

$$J_i = \int {}^i r_i {}^i r_i^T dm = \begin{bmatrix} \int x_i^2 dm & \int x_i y_i dm & \int x_i z_i dm & \int x_i dm \\ \int x_i y_i dm & \int y_i^2 dm & \int y_i z_i dm & \int y_i dm \\ \int x_i z_i dm & \int y_i z_i dm & \int z_i^2 dm & \int z_i dm \\ \int x_i dm & \int y_i dm & \int z_i dm & \int dm \end{bmatrix} \qquad (1.58)$$

donde ${}^i r_i = (x_i, y_i, z_i)^T$ se define como antes. Si utilizamos el tensor de inercia I_{ij} que se define como:

$$I_{ij} = \int \left[\delta_{ij}\left(\sum_k x_k^2\right) - x_i x_j\right] dm \qquad (1.59)$$

donde los índices i, j, k indican los ejes principales del sistema de coordenadas i - ésimo y δ_{ij} es la delta de Kronecker, entonces J_i se puede expresar en un tensor de inercia como:

$$J_i = \begin{bmatrix} \frac{-I_{xx} + I_{yy} + I_{zz}}{2} & I_{xy} & I_{xz} & m_i \bar{x}_i \\ I_{xy} & \frac{I_{xx} - I_{yy} + I_{zz}}{2} & I_{yz} & m_i \bar{y}_i \\ I_{xz} & I_{yz} & \frac{I_{xx} + I_{yy} - I_{zz}}{2} & m_i \bar{z}_i \\ m_i \bar{x}_i & m_i \bar{y}_i & m_i \bar{z}_i & m_i \end{bmatrix} \qquad (1.60)$$

o utilizando el radio de giro del cuerpo rígido m_i en el sistema de coordenadas (x_i, y_i, z_i) se puede expresar como:

$$J_i = \begin{bmatrix} \frac{-K_{i11}^2 + K_{i22}^2 + K_{i33}^2}{2} & K_{i12}^2 & K_{i13}^2 & \bar{x}_i \\ K_{i12}^2 & \frac{K_{i11}^2 - K_{i22}^2 + K_{i33}^2}{2} & K_{i23}^2 & \bar{y}_i \\ K_{i13}^2 & K_{i23}^2 & \frac{K_{i11}^2 + K_{i22}^2 - K_{i33}^2}{2} & \bar{z}_i \\ \bar{x}_i & \bar{y}_i & \bar{z}_i & 1 \end{bmatrix} \tag{1.61}$$

donde k_{i23} es el radio de giro del elemento i respecto al eje yz e ${}^i\bar{r}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i, 1)^T$ es el vector de centro de masa del elemento i desde el sistema de coordenadas del elemento i - ésimo y expresado en el sistema de coordenadas del elemento i - ésimo.

De aquí la energía cinética total K de un brazo de robot es:

$$\begin{aligned} K = \sum_{i=1}^{n} K_i &= \tfrac{1}{2} \sum_{i=1}^{n} Tr\left(\sum_{p=1}^{i} \sum_{r=1}^{i} U_{ip} J_i U_{ir}^T \dot{q}_p \dot{q}_r \right) = \\ &= \tfrac{1}{2} \sum_{i=1}^{n} \sum_{p=1}^{i} \sum_{r=1}^{i} \left[Tr\left(U_{ip} J_i U_{ir}^T \right) \dot{q}_p \dot{q}_r \right] \end{aligned} \tag{1.62}$$

que es una cantidad *escalar*. Las J_i son dependientes de la distribución de masa del elemento i y no de su posición o velocidad de movimiento y se expresan con respecto al sistema de coordenadas i - ésimo. Por tanto, la J_i se necesita calcular solamente *una vez* para obtener la energía cinética de un robot.

1.4.1.3 ENERGÍA POTENCIAL DE UN MANIPULADOR

Sea P la energía potencial total de un robot y sea P_i la energía potencial de cada uno de los elementos:

$$P_i = -m_i g\, {}^0\bar{r}_i = -m_i g \left({}^0A_i\, {}^i\bar{r}_i \right) \qquad i = 1, 2, \ldots, n \tag{1.63}$$

y la energía potencial total del brazo se puede obtener sumando todas las energías potenciales en cada elemento.

$$P = \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} -m_i g \left({}^0A_i\, {}^i\bar{r}_i \right) \tag{1.64}$$

donde $g = (g_x, g_y, g_z, 0)$ es un vector fila de gravedad expresado en el sistema de coordenadas de la base. Para un sistema de nivel, $g = (0,0,-|g|,0)$ y g es la constante gravitacional $\left(g = 9.8062\ {}^{m}/_{s^2}\right)$.

1.4.1.4 ECUACIONES DE MOVIMIENTO DE UN MANIPULADOR

De las ecuaciones (1.62) y (1.64), la función lagrangiana $L = K - P$ está dada por:

$$L = \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i}\left[Tr\left(U_{ij}J_iU_{ik}^{T}\right)\dot{q}_j\dot{q}_k\right] + \sum_{i=1}^{n} m_i g\left({}^{0}A_i\,{}^{i}\bar{r}_i\right) \tag{1.65}$$

Aplicando la formulación de Lagrange - Euler a la función lagrangiana del brazo [ecuación (1.65)] da el par generalizado necesario τ_i para que el actuador de la articulación i mueva el elemento i - ésimo del manipulador.

$$\begin{aligned}\tau_i &= \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \\ &= \sum_{j=1}^{n}\sum_{k=1}^{i} Tr\left(U_{jk}J_jU_{ji}^{T}\right)\ddot{q}_k + \sum_{j=1}^{n}\sum_{k=1}^{i}\sum_{m=1}^{i} Tr\left(U_{jkm}J_jU_{ji}^{T}\right)\dot{q}_k\dot{q}_m - \sum_{j=i}^{n} m_j g U_{ji}\,{}^{i}\bar{r}_j\end{aligned} \tag{1.66}$$

para $i = 1,2,\ldots,n$. La ecuación anterior se puede expresar de forma mucho más simple en *notación* matricial como.

$$\tau_i = \sum_{k=1}^{n} D_{ik}\ddot{q}_k + \sum_{k=1}^{n}\sum_{m=1}^{n} h_{ikm}\dot{q}_k\dot{q}_m + c_i \qquad i = 1,2,\ldots,n \tag{1.67}$$

o en forma matricial como:

$$\tau(t) = D\left(q(t)\ddot{q}(t) + h\left(q(t),\dot{q}(t)\right) + c\left(q(t)\right)\right) \tag{1.68}$$

donde:

$\tau(t) = n \times 1$ vector par generalizado aplicado en las articulaciones $i = 1, 2, \ldots, n$; esto es:

$$\tau(t) = \left(\tau_1(t),\ \tau_2(t),\ \ldots,\ \tau_n(t)\right)^T \tag{1.69}$$

$q(t)$ = un vector $n \times 1$ de las variables de articulación del brazo y se puede expresar como:

$$q(t) = \left(q_1(t),\ q_2(t),\ \ldots.\ q_n(t)\right)^T \tag{1.70}$$

$\dot{q}(t)$ = un vector $n \times 1$ de la velocidad de las articulaciones del brazo y se puede expresar como:

$$\dot{q}(t) = \left(\dot{q}_1(t),\ \dot{q}_2(t),\ \ldots.\ \dot{q}_n(t)\right)^T \tag{1.71}$$

$\ddot{q}(t) =$ un vector $n \times 1$ de la aceleración de las variables de articulación $q(t)$ y se puede expresar como:

$$\ddot{q}(t) = (\ddot{q}_1(t),\ \ddot{q}_2(t),\ \ldots,\ \ddot{q}_n(t))^T \tag{1.72}$$

$D(q)$ = una matriz simétrica inercial relacionada con la aceleracion $n \times n$ cuyos elementos son:

$$D_{ik} = \sum_{j=\max(i,k)} \mathrm{Tr}\left(U_{jk} J_j U_{ji}^T\right) \qquad i,k = 1,\ 2,\ \ldots,\ n \tag{1.73}$$

$h(q,\dot{q})$ = un vector de fuerza Conolis y centrifuga no lineal $n \times 1$ cuyos elementos son:

$$h(q,\dot{q}) = (h_1,\ h_2,\ \ldots,\ h_n)^T \tag{1.74}$$

donde.

$$h_i = \sum_{k=1}^{n}\sum_{m=1}^{n} h_{ikm}\dot{q}_k\dot{q}_m \qquad i = 1,\ 2,\ \ldots,\ n \tag{1.75}$$

y

$$h_{ikm} = \sum_{j=\max(i,k,m)} Tr\left(U_{jkm} J_j U_{ji}^T\right) \qquad i,k,m = 1,\ 2,\ \ldots,\ n \tag{1.76}$$

$c(q) =$ un vector de la fuerza de la carga gravitatoria $n \times 1$ cuyos elementos son:

$$c(q) = (c_1,\ c_2,\ \ldots,\ c_n)^T \tag{1.77}$$

donde :

$$c_i = \sum_{j=1}^{n}\left(-m_j g U_{ji}{}^j\bar{r}_j\right) \qquad i = 1,\ 2,\ \ldots,\ n \tag{1.78}$$

1.5 PLANIFICACION Y CONTROL DEL MOVIMIENTO DEL MANIPULADOR

Con el conocimiento de la cinemática y la dinámica de un manipulador con elementos series, seria interesante mover los actuadores de sus articulaciones para cumplir una tarea deseada controlando al manipulador para que siga un camino previsto. Antes de mover el brazo , es de interés saber si hay algún obstáculo presente en la trayectoria que el robot tiene que atravesar (ligaduras de obstáculos) y si la mano del manipulador necesita viajar a lo largo de una trayectoria especificada (ligaduras de trayectoria).Estas dos ligaduras combinadas dan lugar a cuatro modos de control posibles tales como se tabulan en la siguiente tabla:

		Ligadura de Obstáculo	
		Sí	No
Ligadura del camino	Sí	Fuera de línea libre de colisión planificación de camino en línea más seguimiento del camino	Fuera de línea planificación de camino en línea más seguimiento del camino
	No	Control posicional más obstáculo en línea detección y evitación	Control posicional

Tabla 1.1: Modos de control de un manipulador

En esta tabla se observa que el problema de control de un manipulador se puede dividir convenientemente en dos subproblemas coherentes: planificación del movimiento (o trayectoria) y control de movimiento.

La curva espacial que la mano del manipulador sigue desde una localización inicial (posición y orientación) hasta una final se llama *trayectoria o camino.*

Los esquemas de planificación de la trayectoria generalmente «interpolan» o «aproximan» la trayectoria deseada por una clase de funciones polinomiales y genera una secuencia de puntos de «consignas de control» en función del tiempo para controlar al manipulador desde su posición inicial hasta su destino. Los puntos extremos del camino de pueden especificar o bien en coordenadas de la articulación o bien en coordenadas cartesianas.

Un método sistemático para abordar el problema de planificación de trayectoria es considerar al planificador de trayectoria como una caja negra, tal como se muestra en la figura:

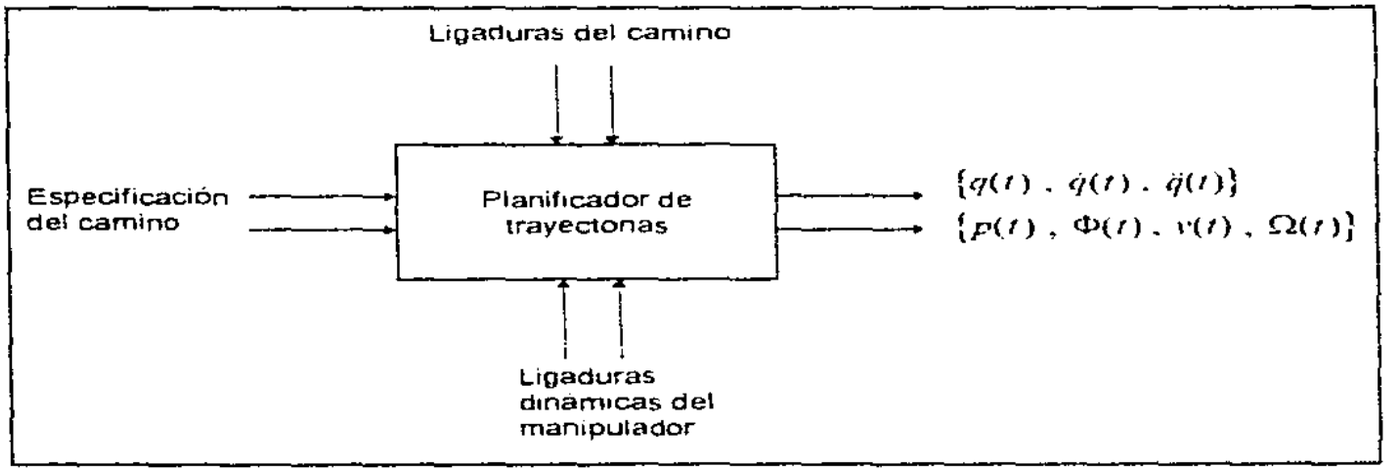

Figura 1.8: Diagrama de bloques del planificador de trayectoria

El planificador de trayectoria acepta variables de entrada que indican las ligaduras del camino y saca una secuencia de configuraciones intermedias a lo largo del tiempo de la mano del manipulador

(posición, orientación, velocidad y aceleración). expresadas bien en coordenadas de articulación o cartesianas, desde las posición inicial hasta la final.

En general el problema de control de movimiento consiste en : 1) obtener los modelos dinámicos del manipulador, 2) utilizar estos modelos para determinar leyes o estrategias de control para conseguir la respuesta y el funcionamiento del sistema deseado. Desde el punto de vista de análisis de control, el movimiento del brazo de un robot se suele realizar en dos fases de control distintas. La primera es el control de movimiento de aproximación en el cual el brazo se mueve desde una posición / orientación inicial hasta la vecindad de la posición / orientacion del destino deseado a lo largo de una trayectoria planificada. El segundo es el control del movimiento fino en el cual el efector final del brazo interacciona dinámicamente con el objeto utilizando información obtenida a traves de la realimentación sensorial para completar la tarea.

Enfoques industriales actuales para controlar el brazo del robot tratan cada articulación del brazo como un servomecanismo de articulación simple. Este planeamiento modela la dinámica de un manipulador de forma inadecuada porque desprecia el movimiento y la configuración del mecanismo del brazo de forma global. Estos cambios en los parametros del sistema controlado algunas veces son bastante significativos para hacer ineficaces las estrategias de control por realimentacion convencionales. El resultado de ello es una velocidad de respuesta y amortiguamiento del servo reducido, limitando así la precisión y velocidad del efector final y haciéndolo apropiado solamente para limitadas tareas de precisión. Los manipuladores controlados de esta forma se mueven a velocidades lentas con vibraciones innecesarias. Cualquier ganancia significativa en el rendimiento en está y otras áreas del control del brazo del robot requieren la consideración de modelos dinámicos más eficientes, enfoque de control sofisticados y el uso de arquitecturas de computadoras dedicadas y técnicas de procesamiento en paralelo.

El problema básicamente se centra en mover el manipulador de una posición inicial a alguna posición final deseada, tomando en cuenta que además se puede requenr controlar el movimiento intermedio, por lo que la herramienta debe pasar por posiciones y orientaciones descritas por la trayectoria.

La planificación de trayectoria se puede realizar o bien en el espacio de las vanables de articulación o bien en el espacio cartesiano. Para la planificación de las variables de articulación se planifica las historia temporal de todas las vanables de articulación y de sus dos primeras derivadas respecto al tiempo para describir el movimiento deseado del manipulador. Para la planificación en el espacio cartesiano se define la historia temporal de la posición de la mano del manipulador, su velocidad y aceleración, y se deducen las correspondientes posiciones, velocidades y aceleraciones de la articulación a partir de la información de la mano. La planificación en el espacio de las variables de articulación tiene tres ventajas: 1) la trayectoria se planifica directamente en términos de las variables controladas durante el movimiento; 2) la planificación de trayectorias se puede hacer casi en tiempo real; 3) las trayectorias de la articulación son más fáciles de planificar .

Para generar las trayectorias se requiere obtener los perfiles de posición, velocidad y aceleración. El polinomio que se utilizará para describir dichos perfiles es el de quinto grado ya que contiene seis coeficientes que se tienen que determinar auxiliándonos de su primera y segunda derivada para la velocidad y aceleración respectivamente y las seis restricciones impuestas. Dos para la posición, en el inicio y fin del recorrido. Esto es:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{1.79}$$

que define la distancia recorrida por el órgano terminal en función del tiempo.

Sus derivadas son:

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{1.80}$$

que define el perfil de velocidad de la trayectoria que seguirá el órgano terminal en función del tiempo, y

$$\ddot{q}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \tag{1.81}$$

que define el perfil de aceleración de la trayectoria que seguirá el órgano terminal en función del tiempo.

Por otro lado sus restricciones están dadas por:

$$q(0) = q_0 = a_0 \tag{1.82}$$

$$q(t_f) = q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_5 t_f^5 \tag{1.83}$$

$$\dot{q}(0) = \dot{q}_0 = a_1 \tag{1.84}$$

$$\dot{q}(t_f) = \dot{q}_f = a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \tag{1.85}$$

$$\ddot{q}(0) = \ddot{q}_0 = 2a_2 \tag{1.86}$$

$$\ddot{q}(t_f) = \ddot{q}_f = 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 \tag{1.87}$$

En las ecuaciones anteriores se utilizan los siguientes parámetros:

$T_f = T$ es el tiempo total empleado en el recorrido.

q_f se considera como la distancia total recorrida, es decir la distancia que hay entre P_1 y P_2 por lo cual su valor se calcula con la fórmula de distancia entre dos puntos por lo cual se tiene:

$$q_f = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \tag{1.88}$$

Las ecuaciones anteriores indican que en el tiempo $t = 0$ se tiene un desplazamiento nulo y se parte del reposo.

Estas restricciones especifican un conjunto lineal de seis ecuaciones con seis incógnitas, cuya solución es:

$$a_0 = q_0 \tag{1.89}$$

$$a_1 = \dot{q}_0 \tag{1.90}$$

$$a_2 = \frac{\ddot{q}_0}{2} \tag{1.91}$$

$$a_3 = \frac{20q_f - 20q_0 - (8\dot{q}_f + 12\dot{q}_0)t_f - (3\ddot{q}_0 - \ddot{q}_f)t_f^2}{2t_f^3} \tag{1.92}$$

$$a_4 = \frac{30q_0 - 30q_f + (14\dot{q}_f + 16\dot{q}_0)t_f - (3\ddot{q}_0 - 2\ddot{q}_f)t_f^2}{2t_f^4} \tag{1.93}$$

$$a_5 = \frac{12q_f - 12q_0 - (6\dot{q}_f + 6\dot{q}_0)t_f - (\ddot{q}_0 - \ddot{q}_f)t_f^2}{2t_f^5} \tag{1.94}$$

CAPITULO 2

CONTROL ADAPTATIVO

2.1 INTRODUCCIÓN AL CONTROL ADAPTATIVO

En cualquier lenguaje, "adaptar" significa cambiar un comportamiento conforme a nuevas circunstancias. Intuitivamente, un regulador adaptable es un regulador que puede modificar su comportamiento en respuesta a cambios en la dinámica del proceso y a las perturbaciones

2.1.1 HISTORIA DEL CONTROL ADAPTATIVO

A principios de los años 1950's se realizaron grandes investigaciones en control adaptativo, en conexión con el diseño de pilotos automáticos de una aeronave de alto desempeño. Tal aeronave operaba sobre un rango amplio de velocidades y altitudes. Se encontró que la ganancia ordinaria y constante del control de realimentacion lineal podía trabajar bien en condiciones de operación normal, pero que cambiando las condiciones de operación se producian dificultades. Un regulador más sofisticado, el cual podía trabajar sobre un rango amplio de condiciones de operación, fue por lo tanto necesitado. El interés en el tema disminuyó debido a la carencia del conocimiento y a un desastre en un vuelo prueba.

En los años 1960's muchas contribuciones a la Teoria del control fueron importantes para el desarrollo del control adaptativo. Se introdujo la teoria estatal de estabilidad y espacio. Existieron importantes resultados en la teoria del control estocástico. La programación dinámica introducida por Bellman, incremento el entendimiento de procesos adaptables Contribuciones fundamentales fueron también hechas por Tsypkin, quien mostro que muchos esquemas para control adaptativo y aprendizaje pueden ser descritos en una estructura común como ecuaciones de un tipo especial. Existieron también desarrollos importantes en identificación de sistemas y en estimación de parámetros. Hubo un renacimiento de control adaptativo en los años 1970's, cuando diferentes esquemas de estimación fueron combinados con varios métodos de diseño. Muchas aplicaciones fueron reportadas, pero los resultados teóricos fueron muy limitados.

A finales de los años 1970's y principios de los años 1980's aparecieron mejoras adecuadas para estabilidad de sistemas adaptables, aunque bajo suposiciones muy restrictivas. La investigacion de la necesidad de estas suposiciones tiene como consecuencia el surgimiento de nuevas e interesantes investigaciones en torno de la robustez del control adaptativo, así como en controladores que sean universalmente estables.

El rápido y revolucionario progreso en la microelectrónica ha hecho posible implementar reguladores adaptables simples y baratos. El vigoroso desarrollo del campo está tomando lugar en universidades y en la industria. Diversos reguladores adaptables comerciales basados en diferentes ideas están apareciendo en el mercado, y el uso industrial del control adaptativo crece lentamente pero seguro.

2.1.2 RELACIÓN CON OTROS ÁREAS DEL CONTROL AUTOMÁTICO

El control adaptativo todavia no es un campo que haya alcanzado completamente su madurez. Hasta hace unos años muchos de los algoritmos y los enfoques usados eran para solucionar los problemas

que se tenían en ese momento. Las herramientas utilizadas en el control adaptativo fueron reunidas a partir de una amplia gama de campos de investigación. Todavía hace algunos años se carecía de buenos enfoques sistemáticos, a pesar de que ya había algoritmos y sistemas que tenían usos buenos y que ya existían sistemas adaptables que claramente funcionaban desempeñándose como sistemas convencionales de realimentación.

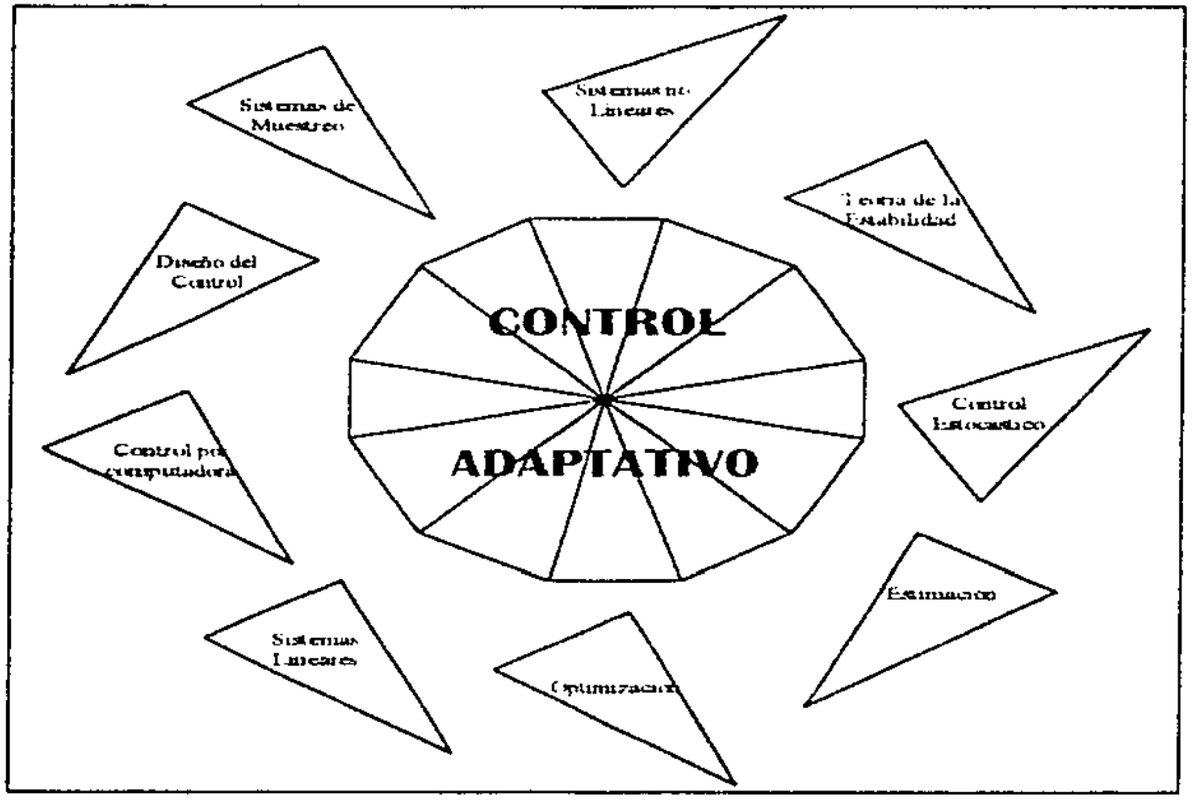

Figura 2.1: Relación del Control Adaptativo con otros subcampos de control automático.

El desarrollo del control adaptativo tiene sus antecedentes en el control convencional por realimentación y también en los sistemas muestreadores de datos. La razón para los posteriores que virtualmente todos los sistemas adaptables se implementan usando las computadoras digitales. El control adaptativo tiene nexos en muchas direcciones, algunas de las cuales son mostradas en la Figura 2.1. Hay lazos fuertes con la teoría de sistemas no lineares, porque los sistemas adaptables son inherentemente no lineares. La teoría de estabilidad es un elemento clave en este tipo de sistemas. El control adaptativo también tiene conexiones con perturbaciones singulares y la teoría del promedio, debido a la separación de las escalas de tiempo en los sistemas adaptables. Hay también nexos con el control estocástico y la estimación de parámetros , porque una de las maneras

para observar los sistemas adaptables es verlos como una combinación de control y estimación de parámetros.

2.2 SISTEMAS ADAPTABLES

En años recientes, un campo creciente de investigación en "sistemas adaptables" ha dado como resultado una variedad de autómatas adaptables cuyas caracteristicas son parecidas de una manera limitada a las caracteristicas de los sistemas de vida y a las de los procesos biologicos adaptables.
Un autómata adaptable es un sistema cuya estructura es alterable o ajustable de tal forma que su comportamiento o desempeño (de acuerdo a algún criterio deseado) es provisto a través del contacto con su medio ambiente. Un simple ejemplo de un autómata o sistema automáticamente adaptable es el control automatico de ganancia usado en los receptores de radio y televisión. La funcion de este circuito es ajustar la sensibilidad del receptor inversamente al promedio de la potencia de la señal de entrada. El receptor está habilitado para adaptarse a un rango amplio de niveles de entrada y producir un rango mucho más estrecho de intensidades de salida.
Los sistemas de control adaptativo y de procesamiento adaptable y de señales usualmente tienen algunas o todas las caracteristicas siguientes:

- Pueden automáticamente adaptarse (optimizarse a si mismos) a los cambios (no estacionarios) del ambiente y cambios en los requerimientos del sistema.
- Pueden ser entrenados para desempeñar filtrados especificos y hacer tomas de decisión. Los sistemas adaptables pueden ser "programados" por un proceso de entrenamiento.
- No requieren de procedimientos elaborados que usualmente necesitan los sistemas no adaptables. Los sistemas adaptables tienden a ser "autodiseñables".
- Pueden extrapolar un modelo de comportamiento para utilizarlo en nuevas situaciones después de haber sido entrenado con un número finito y frecuentemente pequeño de señales o patrones de entrenamiento.
- En un alcance limitado, pueden repararse a si mismos; esto es, pueden adaptarse a cierta clase de defectos internos.
- Pueden usualmente ser descritos como sistemas no lineares con parámetros variantes con el tiempo.
- Usualmente, son más complejos y dificiles de analizar que los sistemas no adaptables, pero ofrecen la posibilidad de un incremento substancial del desempeño del sistema cuando las caracteristicas de la señal de entrada son desconocidas o variantes con el tiempo.

2.3 CONTROL ADAPTATIVO DE ROBOTS

Existen esquemas que controlan el brazo a nivel de la mano o de la articulacion y ponen énfasis en compensaciones no lineales de las fuerzas de interacción entre las diversas articulaciones. Estos

algoritmos de control algunas veces son inadecuados porque necesitan una modelación precisa de la dinámica del brazo y expresan cambios de la carga en un ciclo de tarea. Estos cambios en la carga del sistema controlado a menudo son bastante significativos para hacer que las estrategias de control por realimentación sean eficaces. El resultado es una velocidad de respuesta y amortiguamiento del servo reducido, lo cual limita la precisión y velocidad del efector final. Cualquier mejora significativa en su funcionamiento para seguir a la trayectoria deseada tan estrechamente como sea posible sobre un rango amplio de movimiento y carga del manipulador requiere la consideración de técnicas de control adaptativas.

2.3.1 CONTROL ADAPTATIVO CON REFERENCIA A UN MODELO

Entre los diversos métodos de control adaptativo, el control adaptativo con referencia a un modelo (MRAC) es el más ampliamente utilizado y es también relativamente fácil de realizar. El concepto de un control adaptativo con referencia a un modelo se basa en seleccionar un modelo de referencia apropiado y un algoritmo de adaptación que modifica las ganancias de realimentación a los actuadores del sistema real. El algoritmo de adaptación se excita por los errores entre la salida del modelo de referencia y la salida del sistema real. Un diagrama de bloque de control general del sistema de control adaptativo con referencia a un modelo se muestra en la Figura 2.2:

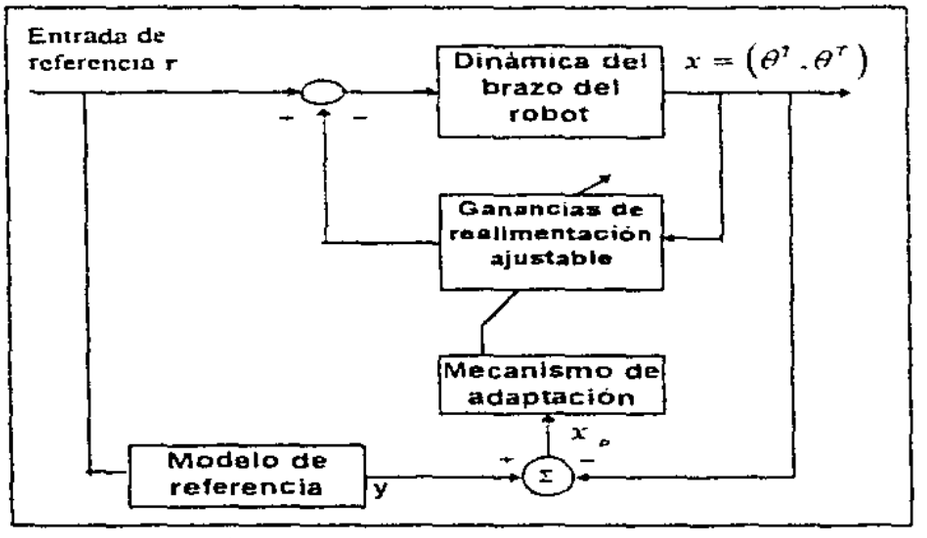

Figura 2.2: Un diagrama de bloques de control general para el control adaptativo con referencia al modelo.

Dubowsky y DesForges (1979) propusieron un control adaptativo con referencias de un modelo simple para el control de manipuladores mecánicos. En su analisis, la carga se toma en consideración combinándola con el elemento final, y la dimensión del efector final se supone pequeña comparada con la longitud de los otros elementos. El modelo con referencia seleccionado proporciona un medio

efectivo y flexible para especificar el funcionamiento en lazo cerrado deseado del sistema controlado. Se selecciona una ecuación diferencial lineal e invariante en el tiempo de segundo orden como el modelo de referencia para cada grado de libertad del robot. El manipulador se controla ajustando las ganancias de realimentación de posición y velocidad para seguir el modelo, de forma que sus características en lazo cerrado coincidan estrechamente con el conjunto de características deseadas en el modelo de referencia. Como resultado de ello, este esquema de control adaptativo solamente requiere cálculos moderados que se pueden efectuar con un microprocesador de bajo costo. Tal algoritmo de control adaptativo con referencia a un modelo no necesita modelos matemáticos complejos de la dinámica del sistema ni el conocimiento a priori de su entorno (cargas, etc.). El sistema adaptativo con referencia al modelo resultante es capaz de mantener un funcionamiento uniformemente bueno sobre un rango amplio de movimientos y cargas.

Definiendo el vector $y(t)$ para representar la respuesta del modelo de referencia y el vector $x(t)$ para representar la respuesta del manipulador, la articulación i del modelo de referencia se puede describir por:

$$a_i \ddot{y}_i(t) + b_i \dot{y}_i(t) + y_i(t) = r_i(t) \tag{2.1}$$

En términos de la frecuencia natural ω_{ni} y la razón de amortiguamiento ζ_i de un sistema lineal de segundo orden, a_i y b_i corresponde a:

$$a_i = \frac{1}{\omega_{ni}^2} \quad \text{y} \quad b_i = \frac{2\zeta_i}{\omega_{ni}} \tag{2.2}$$

Si suponemos que el manipulador se controla mediante ganancias de realimentación de posicion y velocidad y que los términos de acoplo son despreciables, la ecuación dinámica del manipulador para la articulación i se puede escribir como :

$$\alpha_i(t)\ddot{x}_i(t) + \beta_i(t)\dot{x}_i(t) + x_i(t) = r_i(t) \tag{2.3}$$

donde los parámetros del sistema $\alpha_i(t)$ y $\beta_i(t)$ se supone que varían lentamente con el tiempo

Algunas técnicas están disponibles para ajustar las ganancias de realimentación del sistema controlado. Debido a su simplicidad, se utiliza un método de gradiente para minimizar una función cuadrática del error del sistema que es la diferencia entre la respuesta del sistema real (2.3) y la respuesta del modelo de referencia (2.1):

$$J_i(e_i) = \frac{1}{2}(k_2^i \ddot{e}_i + k_1^i \dot{e}_i + k_0^i e_i)^2 \quad i = 1,2,\ldots,n \tag{2.4}$$

donde $e_i = y_i - x_i$, y los valores de los factores de peso, k_j^i, se seleccionan de consideraciones de estabilidad para obtener una conducta del sistema estable.

Utilizando un método de gradiente, el mecanismo de ajuste de los parametros del sistema que minimizará el error del sistema está gobernado por

$$\dot{\alpha}_i(t) = \left[k_2^i \ddot{e}_i(t) + k_1^i \dot{e}_i(t) + k_0^i e_i(t)\right]\left[k_2^i \ddot{u}_i(t) + k_1^i \dot{u}(t) + k_0^i u_i(t)\right] \tag{2.5}$$

$$\dot{\beta}_i(t) = \left[k_2^i \ddot{e}_i(t) + k_1^i \dot{e}_i(t) + k_0^i e_i(t)\right]\left[k_2^i \ddot{w}_i(t) + k_1^i \dot{w}_i(t) + k_0^i w_i(t)\right] \tag{2.6}$$

donde $u_i(t)$ y $w_i(t)$ y sus derivadas se obtienen de la solución de las siguientes ecuaciones diferenciales:

$$a_i \ddot{u}_i(t) + b_i \dot{u}_i(t) + u_i(t) = -\ddot{y}_i(t) \tag{2.7}$$

$$a_i \ddot{w}_i(t) + b_i \dot{w}(t) + w_i(t) = -y_i(t) \tag{2.8}$$

$e\ \dot{y}_i(t)\ e\ \ddot{y}_i(t)$ son las dos primeras derivadas respecto al tiempo de la respuesta del modelo de referencia. El sistema adaptativo en lazo cerrado necesita resolver las ecuaciones del modelo de referencia para una entrada deseada dada; a continuacion las ecuaciones diferenciales en las ecuaciones (2.7) y (2.8) se resuelven para $u_i(t)$ y $w_i(t)$ y sus derivadas para las ecuaciones (2.5) y (2.6). Finalmente, resolviendo las ecuaciones diferenciales en las ecuaciones (2.5) y (2.6), da $\alpha_i(t)$ y $\beta_i(t)$.

El hecho de que este método de control no es dependiente de un modelo matemático complejo es una de sus grandes ventajas, pero las consideraciones de estabilidad del sistema adaptativo en lazo cerrado son criticas. Un análisis de estabilidad es difícil, y Dubowky y DesForges[5] realizaron una investigación de este sistema adaptativo utilizando un modelo linealizado. Sin embargo, la adaptabilidad del controlador puede hacerse cuestionable si las fuerzas de interacción entre las diversas articulaciones son severas.

2.3.2 CONTROL ADAPTATIVO UTILIZANDO UN MODELO AUTORREGRESIVO

Koivo y Guo[6] propusieron un controlador autosintonizante adaptativo utilizando un modelo autorregresivo para ajustar los datos de entrada-salida del manipulador. El algoritmo de control supone que las fuerzas de interacción entre las articulaciones son despreciables. Un diagrama de bloques del sistema se muestra en la figura 2.3. Sea u_i el par de entrada a la articulacion i y la

[5] Dubowky, S., y DesForges, D.T. [1979] . "The application of Model Referenced Adaptive Control to Robotic Manipulators", trans . ASME, J.Dynamic Systems, Measurement and Control, vol 101, pags: 193-200.
[6] Koivo, A.J . y guo , T H. [1983] "Adaptive Linear Controller for Robotics Manipulators", IEEE Trans Automatic Control, vol. AC-28, num 1 pags 162-171.

posición angular de la salida del manipulador y_i. Los pares entrada-salida (u_i, y_i) se pueden describir mediante un modelo autorregresivo que hace coincidir estos pares tan estrechamente como sea posible:

$$y_i(k) = \sum_{m=1}^{n} \left[a_i^m y_i(k-m) + b_i^m u_i(k-m) \right] + a_i^0 + e_i(k) \tag{2.9}$$

donde a_i^0 es un término de fuerza constante, $e_i(k)$ es el error de modelización que se supone que es un ruido gaussiano blanco con media cero e independiente de u_i e $y_i(k-m)$, $m \geq 1$. Los parámetros a_i^m y b_i^m se determinan de manera que se obtenga el mejor ajuste por mínimos cuadrados de los pares de datos medidos de entrada-salida. Estos parámetros se pueden obtener minimizando el criterio siguiente:

$$E_i(\alpha_i) = \frac{1}{N-1} \sum_{k=0}^{N} e_i^2(k) \tag{2.10}$$

donde N es el número de medidas. Sea α_i el vector de parámetros i - esimo:

$$\alpha_i = \left(a_i^0, a_i^1, \ldots, a_i^n, b_i^0, b_i^1, \ldots, b_i^n \right)^T \tag{2.11}$$

y sea $\psi_i(k-1)$ el vector de pares de entrada-salida

$$\psi_i(k-1) = \left[1, y_i(k-1), \ldots, y_i(k-n), u_i(k-1), \ldots, u_i(k-n) \right]^T \tag{2.12}$$

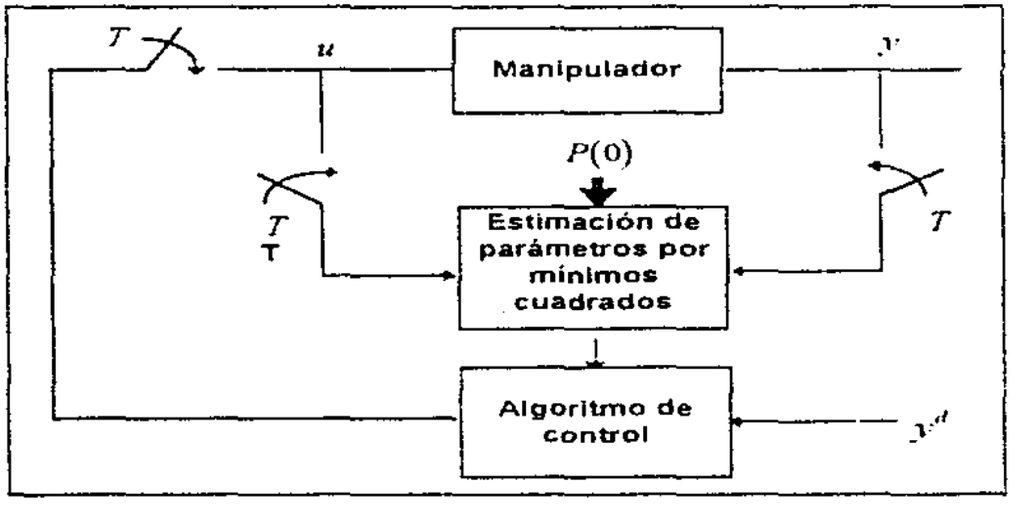

Figura 2.3: Control adaptativo con modelo autorregresivo.

Una estimación por mínimos cuadrados de α_i se puede encontrar como:

$$\hat{\alpha}_i(N) = \hat{\alpha}_i(N-1) + P_i(N)w_i(N-1)\left[y_i(N) - \hat{\alpha}_i^T(N-1)w_i(N-1)\right] \tag{2.13}$$

con

$$P_i(N) = \frac{1}{\mu_i}\left[P_i(N-1) - \frac{P_i(N-1)w_i(N-1)w_i^T(N-1)P_i(N-1)}{\mu_i + w_i^T(N-1)P_i(N-1)w_i(N-1)}\right] \tag{2.14}$$

donde $0 < \mu_i \leq 1$ es un factor de «olvido» que proporciona un peso exponencial a los datos pasados al algoritmo de estimación, lo cual permite al algoritmo una lenta derivación de los parámetros. P_i es una matriz simétrica $(2n+1)x(2n+1)$ y la notación circunfleja se utiliza para indicar una estimación de los parámetros.

Utilizando las ecuaciones anteriores para calcular la estimacion del modelo autorregresivo, puede ser representado por:

$$y_i(k) = \hat{\alpha}_i^T w_i(k-1) + e_i(k) \tag{2.15}$$

Con el fin de seguir los puntos de consigna de la trayectoria, se define una función de coste para la articulación i como:

$$J_i^1(u) = E\left\{\left[y_i(k+2) - y_i^d(k+2)\right]^2 + \gamma_i u_i^2(k+1)\middle| w_i(k)\right\} \tag{2.16}$$

donde $E[\cdot]$ representa una operación de expectación condicionada sobre $w_i(k)$ y γ_i es un factor de peso no negativo definido por el usuario.

Se encuentra que el control óptimo que minimiza la función de coste anterior es:

$$u_i(k+1) = \frac{-\hat{b}_i^1(k)}{\left[\hat{b}_i^1(k)\right]^2 + \gamma_i}\left\{\begin{array}{l}\hat{a}_i^0(k) + \hat{a}_i^1(k)\left[\hat{\alpha}_i^T w_i(k)\right] + \\ + \sum_{m=2}^{n}\hat{a}_i^m(k)y_i(k+2-m) + \sum_{m=2}^{n}\hat{b}_i^m(k)u_i(k+2-m) - y_i^d(k+2)\end{array}\right\} \tag{2.17}$$

donde $\hat{a}_i^m$, $\hat{b}_i^m$ y $\hat{\alpha}_i^m$ son las estimaciones de los parámetros de la ecuación (2.13) y (2.14).

En resumen, este control adaptativo utiliza un modelo autorregresivo [ecuación (2.9)] para ajustar los datos de entrada-salida del manipulador. El esquema de identificación de mínimos cuadrados recursivos [ecuación (2.13) y (2.14)] se utiliza para estimar los parametros que se usan en el control óptimo [ecuación (2.17)] para controlar el manipulador.

2.3.3 CONTROL DE PERTURBACIÓN ADAPTATIVA

Basados en la teoría de la perturbación, Lee y Chung[7] propusieron una estrategia de control adaptativo que sigue una trayectoria de manipulador deseada tan estrechamente como es posible

[7] Lee,C.S.G., Chung, M.J. y Lee B.H. [1984].«An Approach of Adaptive Control for Robot Manipulators», J. Robotic System , vol 1, núm. 1, pags 27-57.
Lee,C.S.G. y Chung, M.J. [1985] «Adaptive Perturbation Control with Feedforward Compesation for Robot Manipulators», simulation, vol. 44, núm 3,pags 127-136

durante todo el tiempo sobre un amplio rango de movimientos y cargas del manipulador. El control de perturbacion adaptativa difiere del esquema adaptativo anterior en el sentido de que toma en cuenta todas las interacciones entre las diversas articulaciones. El control adaptativo presentado en esta sección se basa en las ecuaciones de perturbación linealizada en la vecindad de una trayectoria nominal. La trayectoria nominal se especifica mediante una trayectoria de articulacion interpolada cuya posición, velocidad y aceleración angular se conocen en cada instante del muestreo. Las ecuaciones dinamicas no lineales fuertemente acopladas de un manipulador son linealizadas respecto a la trayectoria del manipulador planificada para obtener el sistema de perturbación linealizado. El sistema controlado se caracteriza por componentes de realimentación y directos que se pueden calcular separada y simultáneamente. Utilizando las ecuaciones de movimiento de Newton - Euler como dinámica inversa del manipulador, la componente directa calcula el par nominal que compensa todas las fuerzas de interacción entre las diversas articulaciones a lo largo de la trayectoria nominal. La componente de realimentación calcula los pares de perturbación que reducen los errores de posicion y velocidad del manipulador a cero a lo largo de la trayectoria nominal. La componente de realimentacion calcula los pares de perturbacion que reducen los errores de posicion y velocidad del manipulador a cero a lo largo de la trayectoria nominal. Se utiliza un esquema de identificacion por mínimos cuadrados en tiempo real recursivo para identificar los parámetros del sistema en las ecuaciones de perturbacion. Se diseña una ley óptima de control de un paso para controlar el sistema de perturbacion linealizado respecto a la trayectoria nominal. Los parámetros y las ganancias de realimentacion del sistema linealizado se modifican y ajustan en cada periodo de muestreo para obtener el esfuerzo de control necesario. Los pares totales aplicados a los actuadores de las articulaciones consisten en sus pares nominales calculados de las ecuaciones de movimiento de Newton - Euler y los pares de perturbación calculados de la ley de control óptima de un paso del sistema linealizado. Esta estrategia de control adaptativo reduce el problema del control del manipulador de un control no lineal a controlar un sistema lineal respecto de una trayectoria nominal.
El control adaptativo se basa en las ecuaciones de perturbación linealizadas respecto de la trayectoria referenciada. Necesitamos deducir ecuaciones de perturbación linealizadas apropiadas para desarrollar el controlador de realimentación que calcula los pares de la articulación para reducir los errores de posición y velocidad a lo largo de la trayectoria nominal. Las ecuaciones de movimiento de Lagrange - Euler de un manipulador de n - elementos se puede expresar en la representación del espacio de estado. El problema de control es encontrar una ley de control realimentado $u(t) = g[x(t)]$ tal que el sistema de control de lazo cerrado $\dot{x}(t) = f\{x(t), g[x(t)]\}$ se haga asintoticamente estable y siga a una trayectoria deseada tan estrechamente como sea posible sobre un rango amplio de cargas durante todos los instantes de tiempo.

Supóngase que los estados nominales $x_n(t)$ del sistema se conocen de la trayectoria planificada, y los pares nominales correspondientes $u_n(t)$ tambien se conocen de los cálculos de los pares de articulación utilizando las ecuaciones de movimiento de Newton - Euler.

$$\dot{x}_n(t) = f\left[x_n(t), u_n(t)\right] \tag{2.18}$$

El modelo de perturbacion linealizado asociado para este sistema de control se puede expresar como:

$$\dot{\delta x}(t) = \nabla_x f|_n \delta x(t) + \nabla_u f|_n \delta u(t) = A(t)\delta x(t) + B(t)\delta u(t) \tag{2.19}$$

donde $\nabla_x f|_n$ y $\nabla_u f|_n$ son matrices jacobianas de $f\left[x(t), u(t)\right]$ evaluadas en $x_n(t)$ y $u_n(t)$, respectivamente, $\delta x(t) = x(t) - x_n(t)$ y $\delta u(t) = u(t) - u_n(t)$.

Los parámetros del sistema, $A(t)$ y $B(t)$, de la ecuación (2.19) dependen de la posicion y velocidad instantánea del manipulador a lo largo de la trayectoria nominal y asi varian lentamente con el tiempo. Debido a la complejidad de las ecuaciones del manipulador, es extremadamente difícil encontrar explícitamente los elementos de $A(t)$ y $B(t)$. Sin embargo, el diseño de una ley de control por realimentación para las ecuaciones de perturbación requiere que los parámetros del sistema de la ecuación (2.19) se conozcan en todo momento. De esta manera se deben utilizar técnicas de identificación parametricas para determinar los elementos desconocidos en $A(t)$ y $B(t)$.

Como resultado de esta formulacion, el problema del control del manipulador se reduce a determinar $\delta_u(t)$, que lleva $\delta x(t)$ a cero en todo momento a lo largo de la trayectoria nominal. El sistema controlado total esta así caractenzado por una componente directa y una componente de realimentación. Dados los puntos de consigna de la trayectoria planificada $q^d(t)$, $\dot{q}^d(t)$ y $\ddot{q}^d(t)$, la componente directa calcula los pares nominales correspondientes $u_n(t)$ de las ecuaciones de movimiento de Newton - Euler . La componente de realimentación calcula los pares de perturbacion correspondientes $\delta_u(t)$ que proporcionan el exceso de control para compensar pequeñas desviaciones de la trayectoria nominal. Los cálculos de los pares de perturbación se basan en una ley de control óptima de un paso. Las principales ventajas de esta formulación son dobles. En primer lugar reduce un problema de control no lineal a un problema de control lineal respecto de una trayectoria nominal, en segundo lugar los cálculos de los pares nominales y de perturbación se pueden efectuar separada y simultáneamente. Debido a esta estructura de cálculo en paralelo, las técnicas de control adaptativo se pueden realizar facilmente utilizando microprocesadores de bajo costo hoy en dia. En la figura 2.4 se muestra un diagrama de bloques de este método.

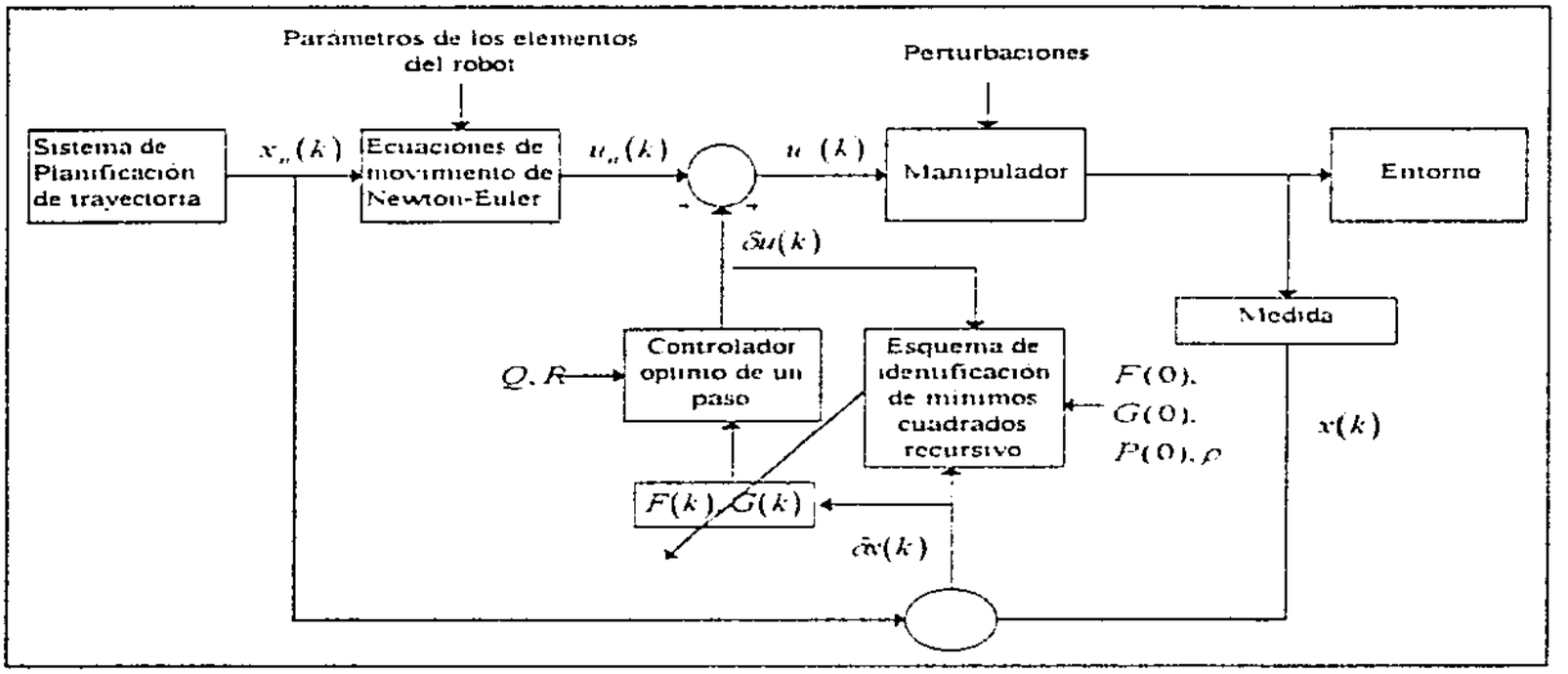

Figura 2.4 : Control de perturbación adaptativa.

3.2.4 CONTROL ADAPTATIVO CON MOVIMIENTO RESUELTO

La estrategia de control adaptativo de la sección anterior en el espacio de la variable de articulación se puede extender para controlar el manipulador en coordenadas cartesianas bajo diversas condiciones de carga adoptando las ideas de los controladores de aceleración y velocidad con movimiento resuelto. El control adaptativo con movimiento resuelto se efectúa en el nivel de la mano y se basa en el sistema de perturbación linealizado a lo largo de una trayectoria de la mano deseada. El control adaptativo con movimiento resuelto difiere del control de aceleración con movimiento resuelto al minimizar la posición / orientación y las velocidades angulares y lineales de la mano del manipulador a lo largo de los ejes de coordenadas de la mano, en lugar de los errores de posición y orientación. Similar al control adaptativo previo, el sistema controlado se caracteriza por tener una componente directa y otra componente en realimentación que se pueden calcular de forma separada y simultánea. La componente directa resuelve las posiciones, velocidades y aceleraciones de la mano. Devuelve un conjunto de valores de aceleración, velocidad y posición para cada una de las articulaciones. Utilizando las ecuaciones de movimiento de Newton - Euler y el conjunto de valores obtenidos se calculan los pares nominales de las articulaciones. La componente de realimentación calcula la perturbación de los pares de la articulación, que reduce los errores de la posición y la

velocidad de la mano del manipulador a lo largo de su trayectoria nominal. Se utiliza un esquema de identificación por mínimos cuadrados recursivos para efectuar la identificación paramétrica en línea del sistema linealizado.

La aceleración de la mano del manipulador está dada por:

$$\begin{matrix} \dot{v}(t) \\ \dot{\Omega}(t) \end{matrix} = N(q,\dot{q})K^{-1}(q) \begin{matrix} v(t) \\ \Omega(t) \end{matrix} + N(q)D^{-1}(q)\left[\tau(t) - h(q,\dot{q}) - c(q)\right] \tag{2.20}$$

Las ecuaciones de estado del manipulador en coordenadas cartesianas son:

$$\begin{matrix} \dot{p}(t) \\ \dot{\Phi}(t) \\ \dot{v}(t) \\ \dot{\Omega}(t) \end{matrix} = \begin{matrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{matrix} \begin{matrix} I_3 & 0 \\ 0 & S(\Phi) \\ N_{11}(q,\dot{q})K_{11}(q) + N_{12}(q,\dot{q})K_{21}(q) & N_{11}(q,\dot{q})K_{12}(q) + N_{12}(q,\dot{q})K_{22}(q) \\ N_{21}(q,\dot{q})K_{11}(q) + N_{22}(q,\dot{q})K_{21}(q) & N_{21}(q,\dot{q})K_{12}(q) + N_{22}(q,\dot{q})K_{22}(q) \end{matrix} \times$$

$$\times \begin{matrix} \dot{p}(t) \\ \dot{\Phi}(t) \\ \dot{v}(t) \\ \dot{\Omega}(t) \end{matrix} \begin{matrix} 0 & 0 \\ 0 & 0 \\ N_{11}(q,\dot{q})E_{11}(q) + N_{12}(q,\dot{q})E_{21}(q) & N_{11}(q,\dot{q})E_{12}(q) + N_{12}(q,\dot{q})E_{22}(q) \\ N_{21}(q,\dot{q})E_{11}(q) + N_{22}(q,\dot{q})E_{21}(q) & N_{21}(q,\dot{q})E_{12}(q) + N_{22}(q,\dot{q})E_{22}(q) \end{matrix} \times \tag{2.21}$$

$$\times \begin{matrix} -h_1(q,\dot{q}) - c_1(q) + \tau_1(t) \\ -h_2(q,\dot{q}) - c_2(q) + \tau_2(t) \end{matrix}$$

Definiendo el vector de estado para la mano del manipulador como:

$$\begin{aligned} x(t) &\triangleq (x_1, x_2, \ldots, x_{12})^T \\ &\triangleq (p_x, p_y, p_z, \alpha, \beta, \gamma, v_x, v_y, v_z, w_x, w_y, w_z)^T \\ &\triangleq (p^T, \Phi^T, v^T, \Omega^T)^T \end{aligned} \tag{2.22}$$

y el vector de entrada como:

$$u(t) \triangleq (\tau_1, \ldots, \tau_n)^T \triangleq (u_1, \ldots, u_n)^T \tag{2.23}$$

La ecuación (2.21) se puede expresar en su representación en el espacio de estado como:

$$\dot{x}(t) = f\left[x(t), u(t)\right] \tag{2.24}$$

donde $x(t)$ es un vector $2n$ - dimensional, $u(t)$ es un vector n - dimensional, $f(\cdot)$ es una función vectorial no lineal continuamente diferenciable $2n \times 1$ y $n = 6$ es el número de grados de libertad del manipulador.

La ecuación (2.24) se puede expresar como:

$$
\begin{aligned}
\dot{x}_1(t) &= f_1(x,u) = x_7(t)\\
\dot{x}_2(t) &= f_2(x,u) = x_8(t)\\
\dot{x}_3(t) &= f_3(x,u) = x_9(t)\\
\dot{x}_4(t) &= f_4(x,u) = -\sec x_5(x_{10}\cos x_6 + x_{11}\,\mathrm{sen}\,x_6)\\
\dot{x}_5(t) &= f_5(x,u) = -\sec x_5(x_{10}\cos x_5\,\mathrm{sen}\,x_6 - x_{11}\cos x_5\,\mathrm{sen}\,x_6)\\
\dot{x}_6(t) &= f_6(x,u) = -\sec x_5(x_{10}\,\mathrm{sen}\,x_5\cos x_6 + x_{11}\,\mathrm{sen}\,x_5\,\mathrm{sen}\,x_6 + x_{12}\cos x_5)\\
\dot{x}_{i+6}(t) &= f_{i+6}(x,u) = g_{i+6}(q,\dot{q})x(t) + b_{i+6}(q)\lambda(q,\dot{q}) + b_{i+6}(q)u(t)
\end{aligned}
\tag{2.25}
$$

donde $i = 1,\ldots,6$ y $g_{i+6}(q,\dot{q})$ es la fila $(i+6)$ - ésima de la matriz:

$$
\begin{bmatrix}
0 & 0 & I_3 & 0\\
0 & 0 & 0 & S(\Phi)\\
0 & 0 & \dot{N}_{11}(q,\dot{q})K_{11}(q) + \dot{N}_{12}(q,\dot{q})K_{21}(q) & \dot{N}_{11}(q,\dot{q})K_{12}(q) + \dot{N}_{12}(q,\dot{q})K_{22}(q)\\
0 & 0 & \dot{N}_{21}(q,\dot{q})K_{11}(q) + \dot{N}_{22}(q,\dot{q})K_{21}(q) & \dot{N}_{21}(q,\dot{q})K_{12}(q) + \dot{N}_{22}(q,\dot{q})K_{22}(q)
\end{bmatrix}
\tag{2.26}
$$

y $b_{i+6}(q)$ es la fila $(i+6)$ - ésima de la matriz:

$$
\begin{bmatrix}
0 & 0\\
0 & 0\\
\dot{N}_{11}(q,\dot{q})E_{11}(q) + \dot{N}_{12}(q,\dot{q})E_{21}(q) & \dot{N}_{11}(q,\dot{q})E_{12}(q) + \dot{N}_{12}(q,\dot{q})E_{22}(q)\\
\dot{N}_{21}(q,\dot{q})E_{11}(q) + \dot{N}_{22}(q,\dot{q})E_{21}(q) & \dot{N}_{21}(q,\dot{q})E_{12}(q) + \dot{N}_{22}(q,\dot{q})E_{22}(q)
\end{bmatrix}
\tag{2.27}
$$

y

$$
\lambda(q,\dot{q}) = \begin{bmatrix} -h_1(q,\dot{q}) - c_1(q) + \tau_1(t)\\ -h_2(q,\dot{q}) - c_2(q) + \tau_2(t)\end{bmatrix}
\tag{2.28}
$$

La ecuación (2.25) describe la dinámica del manipulador completo en coordenadas cartesianas, y el problema del control es encontrar una ley de control realimentado $u(t) = g[x(t)]$ para minimizar el error de la mano del manipulador a lo largo de la trayectoria deseada para un amplio rango de cargas. Se utiliza la teoría de perturbación y se aplica el desarrollo en serie de Taylor a la ecuación (2.25) para obtener el sistema linealizado asociado y diseñar una ley de control por realimentación alrededor de la trayectona deseada de la mano. En la figura 2.5 se muestra el diagrama de bloques del control adaptativo con movimiento resuelto.

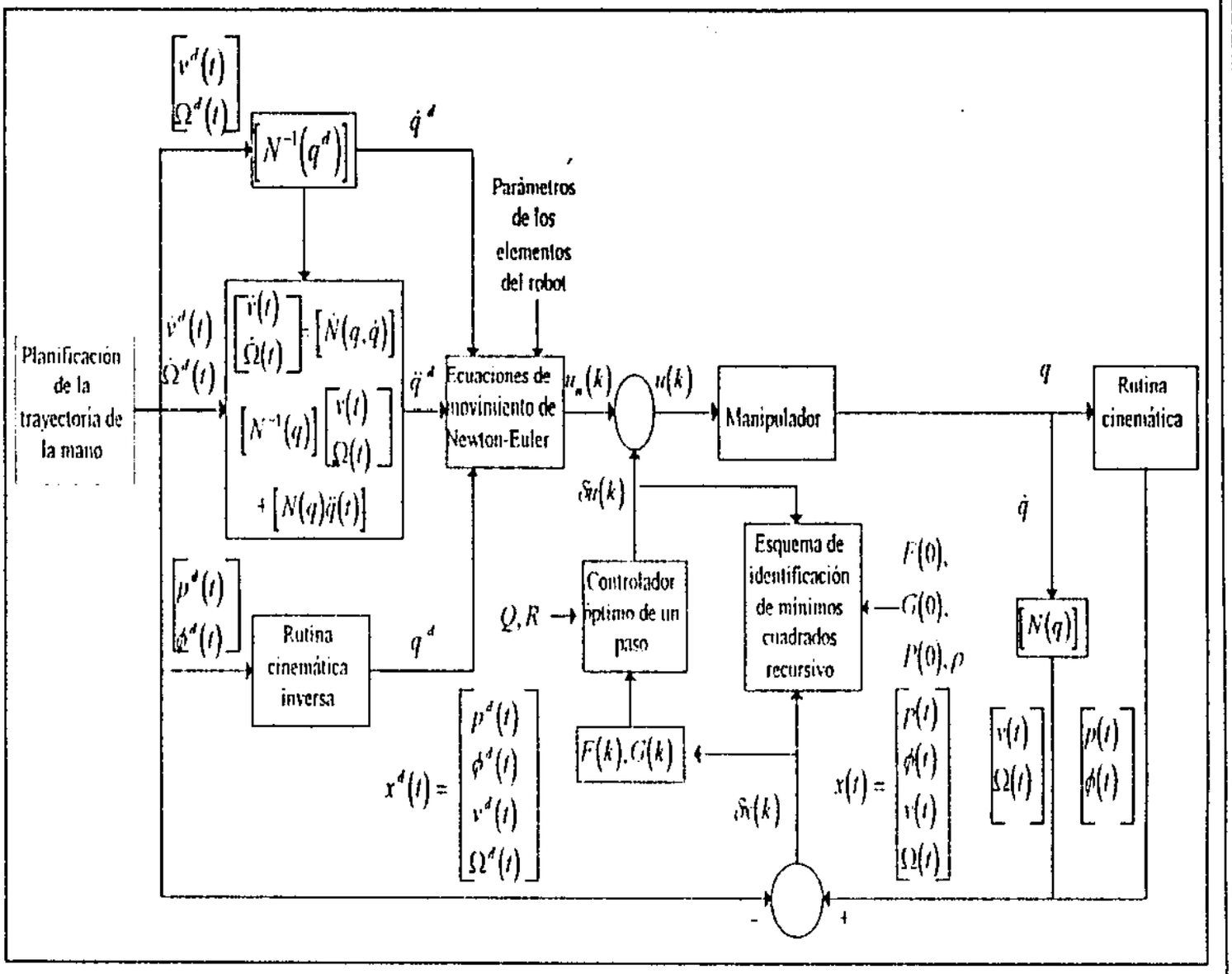

Figura 2.5: Control adaptativo con movimiento resuelto

CAPITULO 3

REDES NEURONALES ARTIFICIALES

3.1 FUNDAMENTOS DE LAS REDES NEURONALES ARTIFICIALES

1.1 ELEMENTOS DE UNA RED NEURONAL ARTIFICIAL

Cualquier modelo de red neuronal consta de dispositivos elementales de proceso: *las neuronas*. A partir de ellas, se pueden generar representaciones especificas, de tal forma que un estado conjunto de ellas puede significar una letra, un número o cualquier objeto. A continuación se realiza la *idealización* del funcionamiento neurobiológico, que sirve de base de las redes neuronales artificiales (RNA). Generalmente se pueden encontrar tres tipos de neuronas.

1) Aquellas que reciben estímulos externos, relacionados con el aparato sensorial, que tomarán la información de entrada.
2) Dicha información se trasmite a ciertos elementos internos que se ocupan de su procesado. Es en la *sinapsis* y *neuronas* correspondientes a este segundo nivel donde se genera cualquier tipo de representacion interna de la información. Puesto que no tiene relación directa con la información de entrada ni con la de salida, estos elementos se denominan *unidades ocultas*
3) Una vez que ha finalizado el periodo de procesado, la información llega a las unidades de salida, cuya misión es dar la respuesta al sistema.

La neurona artificial pretende mimetizar las características más importantes de las neuronas biológicas. Cada neurona i -ésima está caracterizada en cualquier instante por un valor numérico denominado *valor o estado de activación* $a_i(t)$, asociado a cada unidad. Existe una *función de salida*, f_i , que transforma el estado actual de activación en una *señal de salida*, y_i . Dicha señal es enviada a través de los canales de comunicación unidireccionales a otras unidades de la red. En estos canales la señal se modifica de acuerdo con la sinapsis (el *peso*, w_{ij}) asociada a cada uno de ellos según una determinada regla. Las señales moduladas que han llegado a la unidad j -ésima se combinan entre ellas generando así la entrada total, Net_j .

$$Net_j = \sum_i y_i w_{ij} \tag{3.1}$$

Una función de *activación*, F , determina el nuevo estado de activación $a_i(t+1)$ de la neurona, teniendo en cuenta la entrada total calculada y el anterior estado de activación $a_i(t)$.

La dinámica que rige la actualización de los estados de las unidades (evolución de la red neuronal) puede ser de dos tipos: modo asíncrono y modo síncrono. En el primer caso, las neuronas evalúan su estado continuamente, según les va llegando la información, y lo hacen de forma independiente. En el caso síncrono, la información también llega de forma continua, pero los cambios se realizan

simultáneamente, como si existiera un reloj interno que decidiera cuándo deben cambiar su estado. Los sistemas biológicos quedan probablemente entre ambas posibilidades.

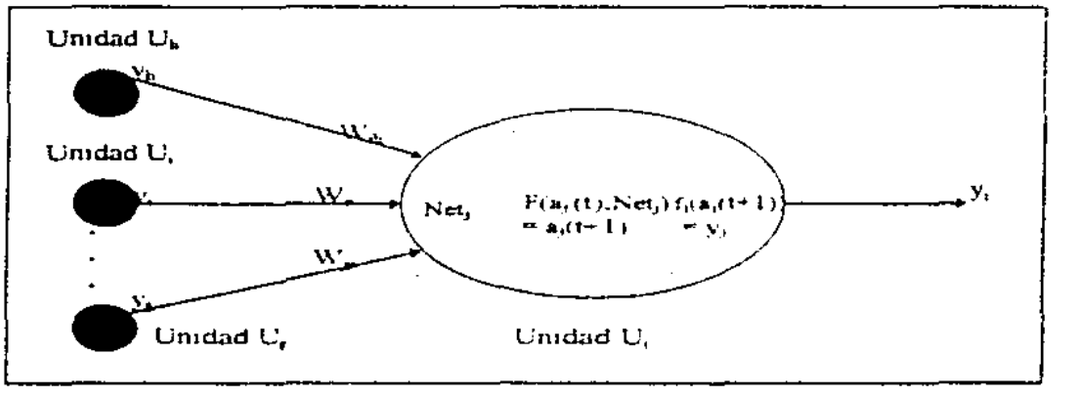

Figura 3.1: Entradas y Salidas de una neurona U_j

3.1.1.1 UNIDADES DE PROCESOS: LA NEURONA ARTIFICIAL

Si se tienen N unidades (neuronas), podemos ordenarlas arbitrariamente y designarlas la j-ésima unidad U_j. Su trabajo es simple y único, y consiste en recibir las entradas de las células vecinas y calcular un valor de salida, el cual es enviado a todas las células restantes.

En cualquier sistema que se esté modelando, es útil caracterizar tres tipos de unidades: entradas, salidas y ocultas. Las unidades de entrada reciben señales desde entornos; estas entradas (que son a la vez entradas de la red) pueden ser señales provenientes de sensores o de otros sectores de sistema. Las unidades de salida envían la señal fuera del sistema (salidas de la red); estas señales pueden controlar directamente potencias u otros sistemas. Las unidades ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema; es decir que éstas no tienen contacto con el exterior.

Se conoce como *capa* o *nivel* a un conjunto de neuronas cuyas entradas provienen de la misma fuente (que puede ser otra capa de neuronas) y cuyas salidas se dirigen al mismo destino (que puede ser otra capa de neuronas).

3.1.1.2 ESTADO DE ACTIVACIÓN

Adicionalmente al conjunto de unidades, la representación necesita los estados del sistema en un tiempo t. Esto se especifica por un vector de N números reales $A(t)$, que representa el *estado de activación* del conjunto de unidades de procesamiento. Cada elemento del vector representa la

activación de una unidad en el tiempo t. La activación de una unidad U_i en el tiempo t se designa por $a_i(t)$: es decir:

$$A(t) = (a_1(t), a_2(t), \ldots, a_i(t), \ldots A_n(t)) \quad (3.2)$$

El procesamiento que realiza la red se ve como la evolucion de un patrón de activación en el conjunto de unidades que lo componen a traves del tiempo.

Todas las neuronas que componen la red se hallan en cierto estado. En una visión simplificada, podemos decir que hay dos posibles estados, *reposo* y *excitación*, a los que denominaremos globalmente *estados de activación*, y a cada uno de los cuales se le asigna un valor. Los valores de activación pueden ser continuos o discretos. Además, pueden ser limitados o ilimitados. Si son discretos, suelen tomar un conjunto pequeño de valores o bien valores binarios. En notación binaria, un estado activo se indicaría por un 1, y se caracteriza por la emision de un impulso por parte de la neurona (potencial de accion), mientras que un estado pasivo se indicaría por un 0, y significaría que la neurona está en reposo. En otros modelos se considera un conjunto continuo de estados de activación, en lugar de sólo dos estados, en cuyo caso se les asigna un valor entre [0,1] o en el intervalo [-1,1] generalmente siguiendo una función sigmoidal.

Finalmente, es necesario saber que criterios o reglas siguen las neuronas para alcanzar tales estados de activación. En principio, esto va a depender de dos factores:

a) Por un lado, puesto que las propiedades macroscópicas de las redes neuronales no son producto de actuación de elementos individuales, sino del conjunto como un todo, es necesario tener idea del mecanismo de interacción entre las neuronas. El estado de activación estará fuertemente influenciado por tales interacciones, ya que el efecto que producirá una neurona sobre otra será proporcional a la fuerza, peso o magnitud de la conexión entre ambas.

b) Por otro lado, la señal que envía cada una de las neuronas a sus vecinas dependerá de su propio estado de activación.

3.1.1.3 FUNCIÓN DE SALIDA O DE TRANSFERENCIA

Entre las unidades o neuronas que forman una red neuronal artificial existe un conjunto de conexiones que unen unas a otras. Cada unidad transmite señales a aquellas que están conectadas con su salida. Asociada con cada unidad U_i hay una función de salida $f_i(a_i(t))$, que transforma el estado actual de activación $a_i(t)$ en una señal de salida $y_i(t)$, es decir :

$$y_i(t) = f_i(a_i(t)) \quad (3.3)$$

El vector que contiene las salidas de todos las neuronas en un instante t es:

$$Y(t) = (f_1(a_1(t)), f_2(a_2(t)), \ldots, f_i(a_i(t)), \ldots, f_n(a_n(t))) \quad (3.4)$$

En algunos modelos, esta salida es igual al nivel de activación de la unidad, en cuyo caso la función f_i es de tipo sigmoidal, y suele ser la misma para todas las unidades.

Existen cuatro funciones de transferencia típicas que determinan distintos tipos de neuronas:

- Función Escalón
- Función lineal y mixta
- Sigmoidal
- Función Gaussiana

La función escalón o umbral únicamente se utiliza cuando las salidas de la red son binarias (dos posibles valores). La salida de una neurona se activa sólo cuando el estado de activación es mayor o igual que cierto valor de umbral (la función puede estar desplazada sobre los ejes). La función lineal o identidad equivale a no aplicar función de salida. Se usa muy poco. Las funciones mixta y sigmoidal son las más apropiadas cuando queremos como salida información analógica.

3.1.1.3.1 Neurona de función Escalón

La forma más fácil de definir la activación de una neurona es considerar que esta es binaria. La función de transferencia escalón se asocia a neuronas binarias en las cuales cuando la suma de las entradas es mayor que el umbral de la neurona, la activación es 1; si es menor, la activación es 0 (o -1). Por otro lado, las redes formadas por este tipo de neuronas son fáciles de implementar en hardware, pero a menudo sus capacidades están limitadas.

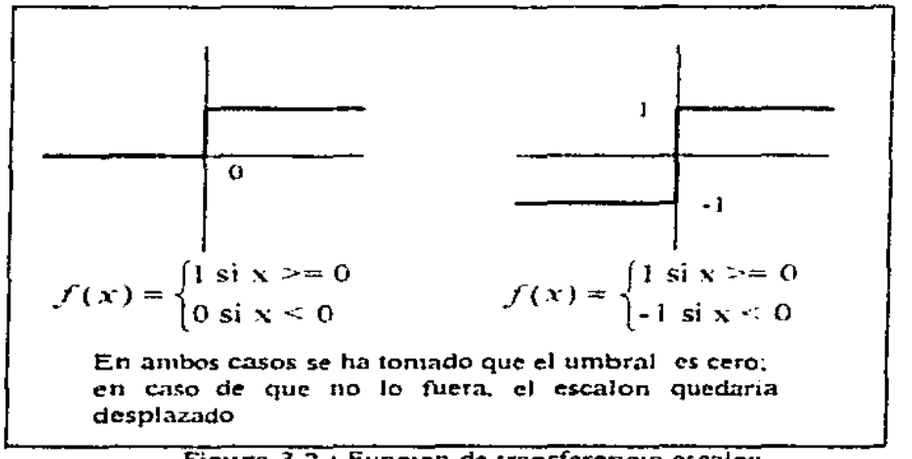

Figura 3.2 : Funcion de transferencia escalon

3.1.1.3.2 Neuronas de función lineal y mixta

La función lineal o identidad responde a la expresión $f(x) = x$. En las neuronas con función mixta, si la suma de señales de entrada es menor que un límite inferior, la activación se define 0 (o -1). Si dicha suma es mayor o igual que el límite superior, entonces la activación es 1. Si la suma de entrada está comprendida entre ambos límites, superior e inferior, entonces la activación se define como una función lineal de la suma de las señales de entrada. Podemos representar la función de activación como se presenta en la figura 3.3, si se toma el límite superior de la suma de todas las entradas de activación que afectan a la neurona durante el ciclo de operación (x) como c y el límite inferior como $-c$, y es la salida de activación de la neurona.

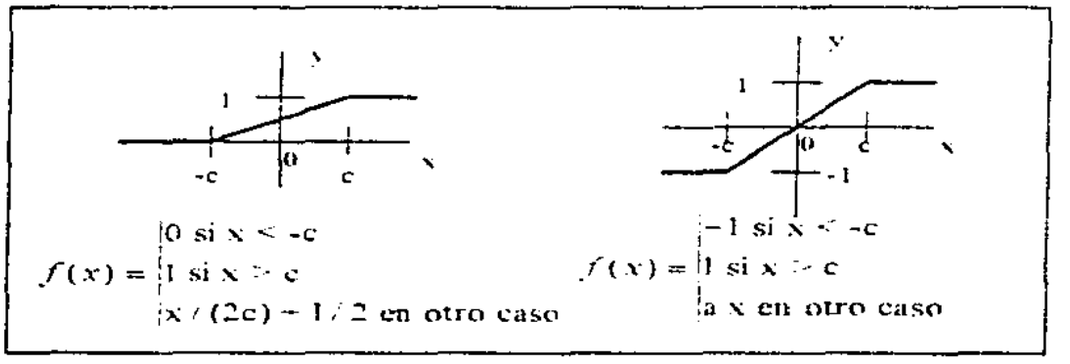

Figura 3.3 : Funciones de activacion mixta

3.1.1.3.3 Neuronas de función continua (sigmoidal)

Cualquier función definida simplemente en un intervalo de posibles valores de entrada, con un incremento monotónico y que tenga ambos límites superiores e inferiores (por ejemplo, la función sigmoidal o arcotangente), podrá realizar la función de activación o de transferencia de forma satisfactoria

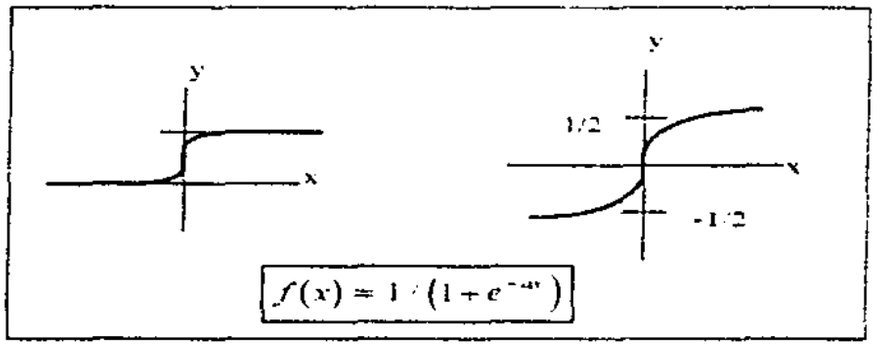

Figura 3.4 : Funciones de activación continuas.

Con la función sigmoidal, para la mayoría de los valores de los estímulos de entrada (variable independiente), el valor dado por la función es cercano a uno de los valores asintóticos. Esto hace que en la mayoría de los casos, el valor de salida está comprendido en la zona alta o baja del sigmoide . De hecho, cuando la pendiente es elevada, esta función tiende a la función escalón. Sin embargo, la importancia de la función sigmoidal (o cualquier otra función similar) es que su derivada es siempre positiva y cercana a cero para los valores grandes positivos o negativos; además, toma su valor máximo cuando x es 0. Esto hace que se puedan utilizar las reglas de aprendizaje definidas para las funciones escalón, con la ventaja, respecto a esta función, de que la derivada está definida en todo el intervalo. La función escalón no podía definir la derivada en el punto de transición, y esto no ayuda a los métodos de aprendizaje en los cuales se usan derivadas.

3.1.1.3.4 Función de transferencia gaussiana

Los centros y anchura de estas funciones pueden ser adaptados, lo cual las hace más adaptativas que las funciones sigmoidales. Mapeos que suelen requenr dos niveles ocultos (neuronas en la red que se encuentran entre las de entrada y las de salida) utilizando neuronas con funciones de transferencia sigmoidales; algunas veces se pueden realizar con un solo nivel en redes con neuronas de función gaussiana.

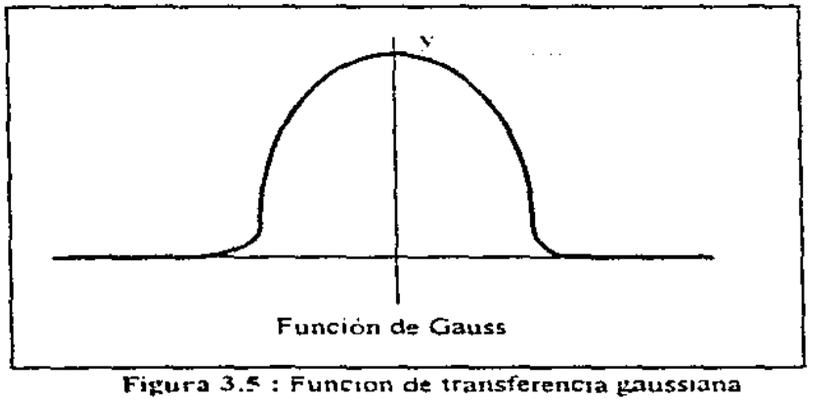

Figura 3.5 : Función de transferencia gaussiana

3.1.1.4 CONEXIONES ENTRE NEURONAS

Las conexiones que unen a las neuronas que forman una RNA tienen asociado un peso, que es el que hace que la red adquiera conocimiento. Consideremos y_i como el valor de salida de una neurona i en un instante dado . Una neurona recibe un conjunto de señales que le dan información del estado de activación de todas las neuronas con las que se encuentra conectada. Cada conexión (sinapsis) entre la neurona i y la neurona j está ponderada por un peso w_{ij}. Normalmente como simplificación, se considera que el efecto de cada señal es aditivo, de tal forma que la entrada neta

que recibe una neurona (potencial postsináptico) net, es la suma del producto de cada señal individual por el valor de la sinapsis que conecta ambas neuronas:

$$Net_i = \sum_j y_j w_{ij} \tag{3.5}$$

Esta regla muestra el procesamiento a seguir para combinar los valores de entrada a una unidad con los pesos de las conexiones que llegan a esa unidad y es conocida como *regla de propagación*.

Suele utilizarse una matriz W con todos los pesos w_{ij} que reflejan la influencia que sobre la neurona j tiene la neurona i. W es un conjunto de elementos positivos, negativos o nulos. Si w_{ij} es positivo, indica que la interacción entre las neuronas i y j es excitatoria; es decir, siempre que la neurona i esté activada, la neurona j recibirá una señal de i que tenderá a activarla. Si w_{ij} es negativo, la sinapsis será inhibidora. En este caso, si i esta activada, enviara una señal j que tenderá a desactivar ésta. Finalmente, si $w_{ij} = 0$, se supone que no hay conexión entre ambas.

3.1.1.5 FUNCIÓN O REGLA DE ACTIVACIÓN

Así como es necesario una regla que combine las entradas a una neurona con los pesos de las conexiones, también se requiere una regla que combine las entradas con el estado actual de la neurona para producir un nuevo estado de activación. Esta función F produce un nuevo estado de activación en una neurona a partir del estado (a_i) que existía y la combinación de las entradas con los pesos de las conexiones (net_i).

Dado el estado de activación $a_i(t)$ de la unidad U_i y la entrada total que llega a ella, Net_i, el estado de activación siguiente $a_i(t+1)$, se obtiene aplicando la función F, llamada función de activación.

$$a_i(t+1) = F(a_i(t), Net_i) \tag{3.6}$$

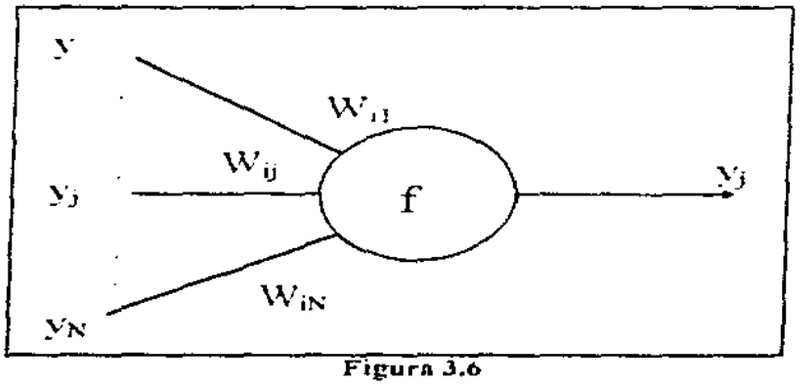

Figura 3.6

En la mayoria de los casos, F es la funcion identidad, por lo que el estado de activación de una neurona en $t+1$ coincidiria con el *Net* de la misma en t. En este caso, el parámetro que se le pasa a la función de salida, f de la neurona, será directamente el *Net*. El estado de activación anterior no se tiene en cuenta. Según esto, la salida de una neurona i (y_i) quedará según la expresión:

$$y_i(t+1) = f(Net_i) = f(\sum_{j=1}^{n} w_{ij} y_j(t)) \qquad (3.7)$$

Por lo tanto, y en lo sucesivo, consideremos únicamente la función f, que denominaremos indistintamente de transferencia o de activación. Además, normalmente la funcion de activación no está centrada en el origen del eje que representa el valor de la entrada neta, sino que existe cierto desplazamiento debido a las caracteristicas internas de la propia neurona y que no es igual en todas ellas. Este valor se denota como θ_i y representa el umbral de activación de la neurona i.

$$y_i(t+1) = f(Net_i - \theta_i) = f(\sum_{j=1}^{n} w_{ij} y_j(t) - \theta_i) \qquad (3.8)$$

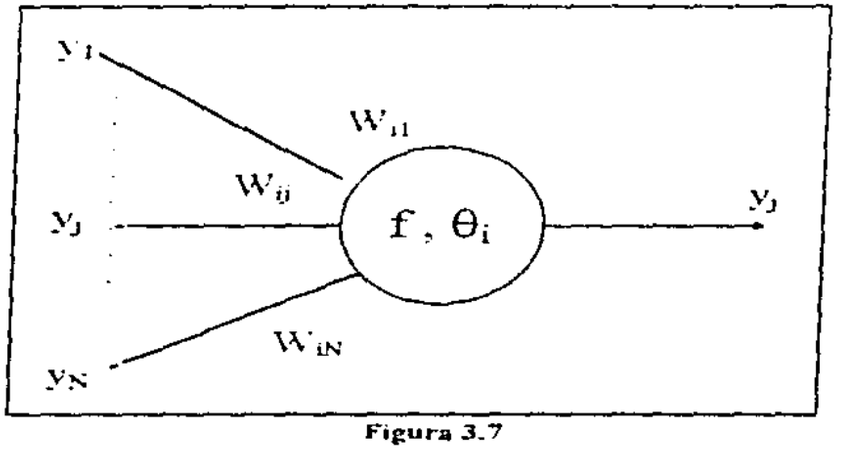

Figura 3.7

La salida que se obtiene en una neurona para las diferentes formas de la función f serán :

a) Función de activación escalón

Si el conjunto de los estados es E={0,1}, tenemos que:

$$y_i(t+1) = \begin{cases} 1 \; si \; [Net_i > \theta_i] \\ y(t) \; si \; Net_i = \theta_i \\ 0 \; si \; [Net_i < \theta_i] \end{cases} \qquad (3.9)$$

Si el conjunto es E = {-1, 1}, tendremos que:

$$y_i(t+1) = \begin{cases} +1 \; si \; [Net_i > \theta_i] \\ y(t) \; si \; Net_i = \theta_i \\ -1 \; si \; [Net_i < \theta_i] \end{cases} \qquad (3.10)$$

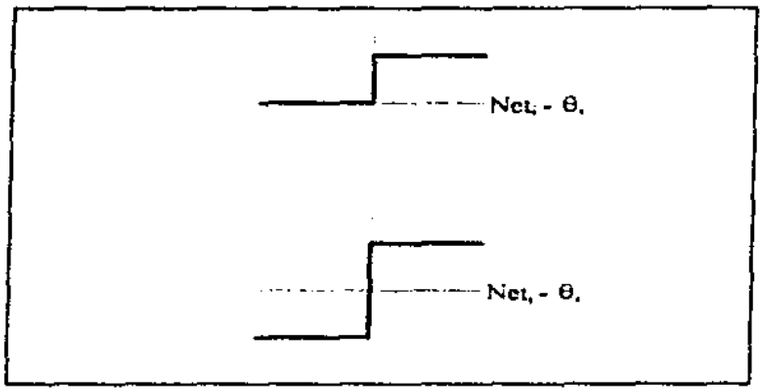

Figura 3.8

b) Función de activación lineal o identidad

El conjunto de estados E puede contener cualquier número real; el estado de activación coincide con la entrada total que ha llegado a la unidad.

$$y_i(t+1) = Net_i - \theta_i \qquad (3.11)$$

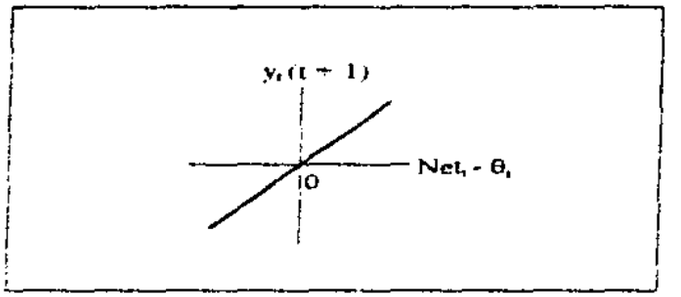

Figura 3.9

c) Función de activación lineal - mixta

Con esta función, el estado de activación de la unidad está obligado a permanecer en un intervalo de valores reales prefijados.

$$y_i(t+1) = \begin{cases} b\,Net_i & \leq b + \theta_i \\ Net_i - \theta_i & b + \theta_i < Net_i < B + \theta_i \\ B\,Net_i & \geq B \end{cases} \qquad (3.12)$$

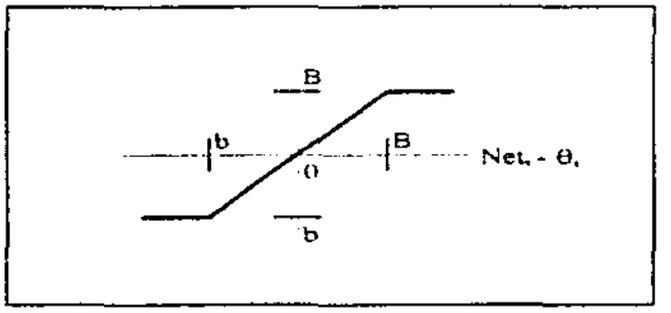

Figura 3.10

d)Función de activación sigmoidal

Es una función continua, por tanto el espacio de los estados de activación es un intervalo del eje real.

$$y_i(t+1) = \frac{1}{(1 + e^{-(Net_i - \theta_i)})} \qquad (3.13)$$

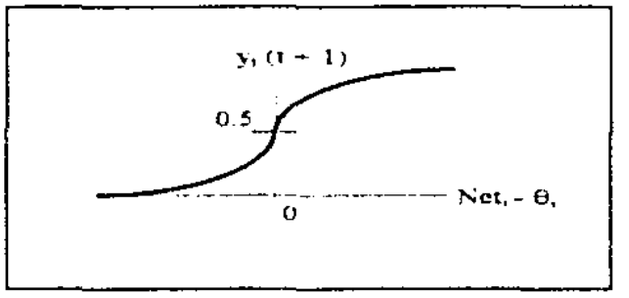

Figura 3.11

Para simplificar la expresión de la salida de una neurona i, es habitual considerar la existencia de una neurona ficticia, con valor de salida de uno, asociada a la entrada de cada neurona i mediante una conexión con peso de valor $-\theta_i$. De esta forma la expresión de salida quedará:

$$y_i(t+1) = f\left(\sum_{j=1} w_{ij} y_j(t) - \theta_i \cdot 1\right) = f\left(\sum_{j=0} w_{ij} y_j(t)\right) = f(Net_i) \qquad (3.14)$$

3.1.1.6 REGLA DE APRENDIZAJE

Existen muchas definiciones del concepto general de *aprendizaje*. Una de ellas podría ser: *La modificación del comportamiento inducido por la interacción con el entorno y como resultado de expenencias que conducen al establecimiento de nuevos modelos de respuesta a estímulos externos*. Esta definición fue enunciada muchos años antes de que surgieran las redes neuronales, sin embargo puede ser aplicada también a los procesos de aprendizaje de estos sistemas.

Biológicamente, se suele aceptar que la información memonzada en el cerebro está más relacionada con los valores sinápticos de las conexiones entre las neuronas que con ellas mismas: es decir, el conocimiento se encuentra en la sinapsis. En el caso de las redes neuronales artificiales, se puede considerar que el conocimiento se encuentra representado en los *pesos* de las conexiones entre neuronas. Todo proceso de aprendizaje implica cierto número de cambios en estas conexiones. En realidad puede decirse que *aprende* modificando los valores de los pesos de la red.

Al igual que el funcionamiento de una red depende del número de neuronas de las que disponga y de cómo estén conectadas entre sí, cada modelo dispone de su o sus propia(s) técnica(s) de aprendizaje.

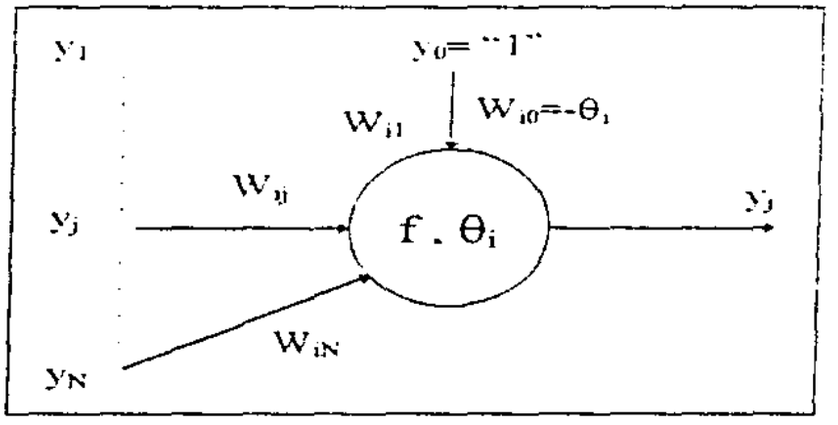

Figura 3.12

3.1.1.7 REPRESENTACIÓN VECTORIAL

En ciertos modelos de redes neuronales, se utiliza la forma vectorial como herramienta de representación de algunas magnitudes. Si consideramos una red formada por varias capas de neuronas idénticas, podemos considerar las salidas de cierta capa de n unidades como un vector $n-$dimensional $Y = (y_1, y_2, \ldots y_n)$. Si este vector $n-$dimensional de salida representa los valores de entrada de todas las unidades de una capa $m-$dimensional, cada una de las unidades

de esta capa poseerá n pesos asociados a las conexiones procedentes de la capa anterior. Por tanto, hay m vectores de pesos n – dimensionales asociados a la capa m.

El vector de pesos de la j - ésima unidad tendra la forma : $W_j = (w_{j1}, w_{j2}, \ldots, w_{jn})$.

La entrada neta de la j -esima unidad puede escribirse en forma del producto escalar del vector de entradas por el vector de pesos. Cuando los vectores tienen igual dimensión, este producto se define como la suma de los productos de los componentes correspondientes a ambos vectores:

$$net_j = \sum_{i}^{n} w_{ji} \cdot y_i \qquad (3\ 15)$$

donde n representa el numero de conexiones de la j -ésima unidad. La ventaja de la notación vectorial, es que la anterior ecuacion puede escribirse de la siguiente forma:

$$net_j = W \bullet Y \qquad (3\ 16)$$

3.2 MECANISMOS DE APRENDIZAJE

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una informacion de entrada. Los cambios que se producen durante el proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los sistemas biológicos existe una continua creación y destrucción de conexiones. En los modelos de redes neuronales artificiales, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. De la misma forma la conexión se destruye cuando su peso pasa a ser cero.

Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones por tanto, se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables ($dw_{ji} / dt = 0$).

Un aspecto importante respecto al aprendizaje en las redes neuronales es el de conocer cómo se modifican los valores de los pesos; es decir, cuáles son los criterios que se siguen para cambiar el valor asignado a las conexiones cuando se pretende que la red *aprenda* una nueva información.

Estos criterios determinan lo que se conoce como la *regla de aprendizaje* de la red. De forma general, se suelen considerar dos tipos de reglas: las que responden a lo que habitualmente se conoce como aprendizaje supervisado, y las correspondientes a un aprendizaje no supervisado.

Es por esto por lo que una de las clasificaciones que se realizan de las redes neuronales obedece al tipo de aprendizaje utilizado por dichas redes. Así se pueden distinguir :

- Redes neuronales con aprendizaje supervisado.
- Redes neuronales con aprendizaje no supervisado.

La diferencia fundamental entre ambos tipos estriba en la existencia o no de un agente externo (supervisor) que controle el proceso de aprendizaje de la red.

Otro criterio que se puede utilizar para diferenciar las reglas de aprendizaje se basa en considerar si la red puede *aprender* durante su funcionamiento habitual o si el aprendizaje supone la *desconexión* de la red; es decir su inhabilitación hasta que el proceso termine. En el primer caso, se trataría de un aprendizaje ON LINE, mientras que el segundo es lo que se conoce como aprendizaje OFF LINE

Cuando el aprendizaje es OFF LINE, se distingue entre una *fase de aprendizaje* o *entrenamiento* y una *fase de operación o funcionamiento*, existiendo un conjunto de datos de entrenamiento y un conjunto de datos de test o prueba que serán utilizados en la correspondiente fase. En las redes con aprendizaje OFF LINE, los pesos permanecen fijos después que termina la etapa de entrenamiento de la red. Debido precisamente a su carácter estático, estos sistemas no presentan problemas de estabilidad en su funcionamiento

En las redes con aprendizaje ON LINE no se distingue entre fase de entrenamiento y de operación, de tal forma que los pesos varían dinámicamente siempre que se presente una nueva información al sistema. En este tipo de redes, debido al carácter dinámico de las mismas, el estudio de la estabilidad suele ser un aspecto fundamental de interés

3.2.1 REDES DE APRENDIZAJE SUPERVISADO

El aprendizaje supervisado se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor comprueba la salida de la red y en el caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada

En este tipo de aprendizaje se suelen considerar, a su vez, tres formas de llevarlo a cabo que dan lugar a los siguientes *aprendizajes supervisados*:

- Aprendizaje por corrección de error.
- Aprendizaje por refuerzo.
- Aprendizaje estocástico.

3.2.1.1 APRENDIZAJE POR CORRECCIÓN DE ERROR

Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error cometido en la salida.

Una regla o algoritmo simple de aprendizaje por corrección de error podría ser el siguiente:

$$\Delta w_{ij} = \alpha y_i (d_j - y_j) \qquad (3.17)$$

Siendo

ΔW_{ij} Variación en el peso de la conexión entre las neuronas i y j ($\Delta W_{ij} = w_{ij}^{nuevo} - w_{ij}^{anterior}$).

y_i : Valor de salida de la neurona i .

d_j : Valor de la salida deseado para la neurona j

y_j : Valor de la salida obtenido en la neurona j

α : Factor de aprendizaje ($0 < \alpha \leq 1$) que regula la velocidad de aprendizaje

Un ejemplo de este tipo de algoritmo lo constituye la *regla de aprendizaje del perceptron*, utilizada en el entrenamiento de la red del mismo nombre que desarrollo Rosenblatt en 1958[8] .

Sin embargo, existen otros algoritmos más evolucionados que éste, que presentan algunas limitaciones, como el no considerar la magnitud del error global cometido durante el proceso completo de aprendizaje de la red, considerando únicamente los errores individuales (locales) correspondientes al aprendizaje de cada información por separado.

Un algoritmo muy conocido que mejora el del Perceptron y permite un aprendizaje mas rápido y un campo de aplicación más amplio es el propuesto por Widrow y Hoff en1960[9] , denominado *regla delta o regla del mínimo error cuadrado* (LMS Error. Least-Mean-Squared Error), tambien conocida como regla de Widrow-Hoff, que se aplicó en las redes desarrolladas por los mismos autores, conocidas como **ADALINE** (Adaptive Linear Element), con una única neurona de salida, y la **MADALINE** (Multiple ADALINE), con varias neuronas de salida.

Otro algoritmo de aprendizaje por corrección de error lo constituye el denominado *regla delta generalizada* o algoritmo de retropropagación del error (*error backpropagation*), tambien conocido como regla LMS (Least- Mean Square Error) multicapa. Se trata de una generalizacion de la regla delta para poder aplicarla a redes con conexiones hacia adelante (feedforward) con capas o niveles internos ocultos de neuronas que no tienen relación con el exterior.

3.2.1.2 APRENDIZAJE POR REFUERZO

Se trata de un aprendizaje supervisado, más lento que el anterior, que se basa en la idea de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada.

En el aprendizaje por refuerzo la función del supervisor se reduce a indicar mediante una *señal de refuerzo* si la salida obtenida en la red se ajusta a la deseada (*éxito* = +1 o *fracaso* = -1), y en la función de ello se ajustan los pesos basándose en un mecanismo de probabilidades. Se podría decir que en este tipo de aprendizaje la función supervisora se asemeja más a la de un critico (que *opina*

---

[8] F. Rosenblatt "The Perceptron : A probabilistic model for information storage and organization in the brain" Psychological Review, 65, pags 386-408, 1958. Reimpreso en el texto "Neurocomputing" (J. Anderson y E. Rosenfeld de.) , págs. 92-114, MIT Press, 1988.

[9] B. Widrow y M. Hoff. Adaptive Switching Circuits" IREWESCON Convention Record, part 4, pags 96-104, 1960 .Reimpreso en el texto "Neurocomputing"(J. Anderson y E. Rosenfeld de.) pags. 126-134, MIT Press, 1988.

sobre la respuesta de la red) que a la de un maestro (que *indica* a la red la respuesta concreta que debe generar), como ocurria en el caso de supervisión por correccion de error.
Un ejemplo de algoritmo por refuerzo lo constituye el denominado Linear Reward-Penalty o L_{R-P} (algoritmo lineal con recompensa y penalizacion) presentado por Narendra y Thathacher en 1974
Este algoritmo ha sido desarrollado por Barol y Anadan, quienes en 1985 desarrollaron el denominado *Associative Reward-Penalty* o A_{R-P} (algoritmo asociativo con recompensa y penalización), que se aplica en redes con conexiones hacia adelante de dos capas cuyas neuronas de salida presentan una función de activacion estocastica.
Otro algoritmo por refuerzo es el conocido como *Adaptive Heuristic Critic*, introducido por Barto, Sutton y Anderson en 1983 que se utiliza en redes *feedforward* de tres capas especialmente diseñadas para que una parte de la red sea capaz de generar un valor interno de refuerzo que es aplicado a las neuronas de salida de la red

3.2.1.3 APRENDIZAJE ESTOCÁSTICO

Este tipo de aprendizaje consiste basicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad
En el aprendizaje estocástico se suele hacer una analogía en términos termodinámicos asociando la red neuronal con un sólido fisico que tiene cierto estado energetico. En el caso de la red, la energia de la misma representaria el grado de estabilidad de la red, de tal forma que el estado de minima energia corresponderia a una situacion en la que los pesos de las conexiones consiguen que su funcionamiento sea el que mas se ajusta al objetivo deseado.
Según lo anterior, el aprendizaje consistiria en realizar un cambio aleatorio de los valores de los pesos y determinar la energia de la red (habitualmente la función de energia es una función denominada de Lyapunov) Si la energia es menor después del cambio; es decir, si el comportamiento de la red se acerca al deseado, se acepta el cambio. Si por el contrario, la energia no es menor, se aceptaria el cambio en funcion de una determinada y preestablecida distribución de probabilidades.

3.3 REDES NEURONALES CON CONEXIONES HACIA ADELANTE

El grupo de redes con conexiones hacia adelante se caracterizan por arquitecturas en niveles y conexiones estrictamente hacia adelante entre las neuronas. Estas redes son todas buenos *clasificadores de patrones* y utilizan *aprendizaje supervisado*.
Esto incluye el Perceptron, las redes ADALINE y MADALINE y la red Back-Propagation. El Perceptron y las redes ADALINE y MADALINE tienen un importante interés histórico y han abierto el camino para el desarrollo de otras redes neuronales Por otro lado, la red Back-Propagation es probablemente una de las más utilizadas hoy en dia

3.3.1 EL PERCEPTRON

Este fue el primer modelo de red neuronal artificial desarrollado por Rosenblatt en 1958[10]. Despertó un enorme interés en los años 60, debido a su capacidad para aprender a reconocer patrones sencillos: un Perceptron formado por varias neuronas lineales para recibir las entradas a la red y una neurona de salida, es capaz de decidir cuándo una entrada presentada a la red pertenece a una de las dos clases que es capaz de reconocer.

La única neurona de salida del Perceptron realiza la suma ponderada de las entradas resta el umbral y pasa el resultado a una función de transferencia de tipo escalón. La regla de decisión es responder +1 si el patron presentado pertenece a la clase A, o -1 del patrón pertenece a la clase B (Figura 3.13). La salida dependera de la entrada neta (suma de las entradas X_i ponderadas) y del valor de umbral θ.

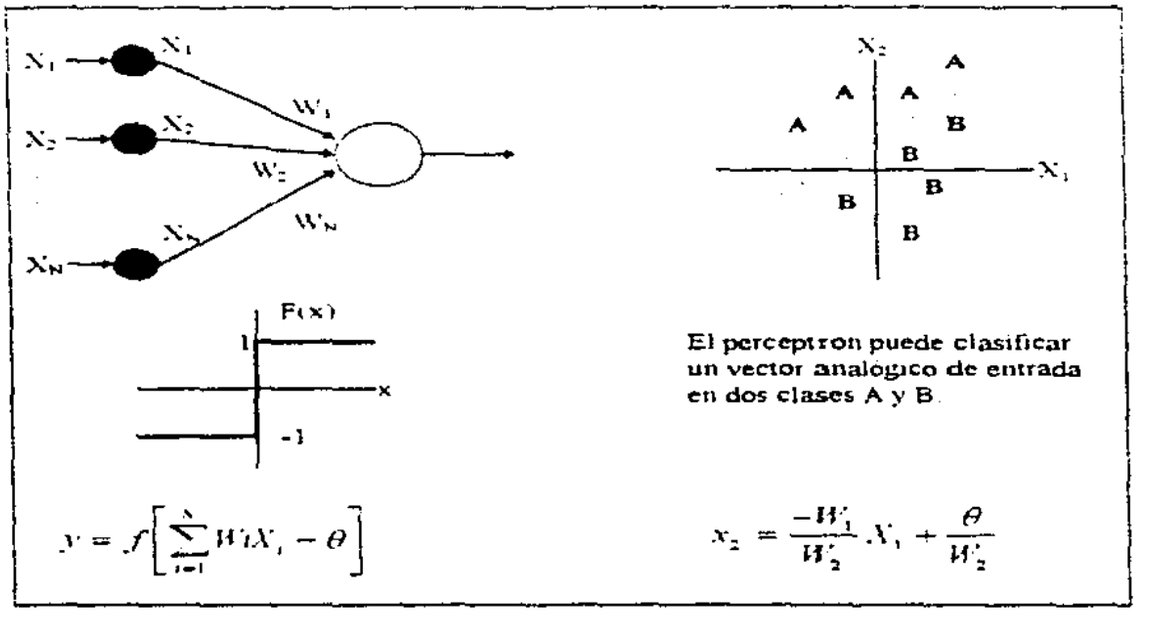

Figura 3.13 : El Perceptron.

Una técnica utilizada para analizar el comportamiento de redes como el Perceptron es representar en un mapa las regiones de decision creadas en el espacio multidimensional de entradas a la red. En estas regiones se visualiza qué patrones pertenecen a una clase y cuáles a otra. El Perceptron separa las regiones por un hiperplano cuya ecuación queda determinada por los pesos de las conexiones y el

[10] F. Rosenblatt "The Perceptron. A probabilistic model for information storage and organization in the brain Psychological Review, 65 pags 386-408, 1958. reimpreso en el texto "Neurocomputing"(J. Anderson y E. Rosenfeld de) pags 92-114, MIT Press, 1988

3.3.1 EL PERCEPTRON

Este fue el primer modelo de red neuronal artificial desarrollado por Rosenblatt en 1958[10]. Despertó un enorme interes en los años 60, debido a su capacidad para aprender a reconocer patrones sencillos: un Perceptron, formado por varias neuronas lineales para recibir las entradas a la red y una neurona de salida es capaz de decidir cuando una entrada presentada a la red pertenece a una de las dos clases que es capaz de reconocer.

La única neurona de salida del Perceptron realiza la suma ponderada de las entradas, resta el umbral y pasa el resultado a una función de transferencia de tipo escalon. La regla de decisión es responder +1 si el patron presentado pertenece a la clase A, o -1 del patrón pertenece a la clase B (Figura 3.13). La salida dependerá de la entrada neta (suma de las entradas x_i ponderadas) y del valor de umbral θ.

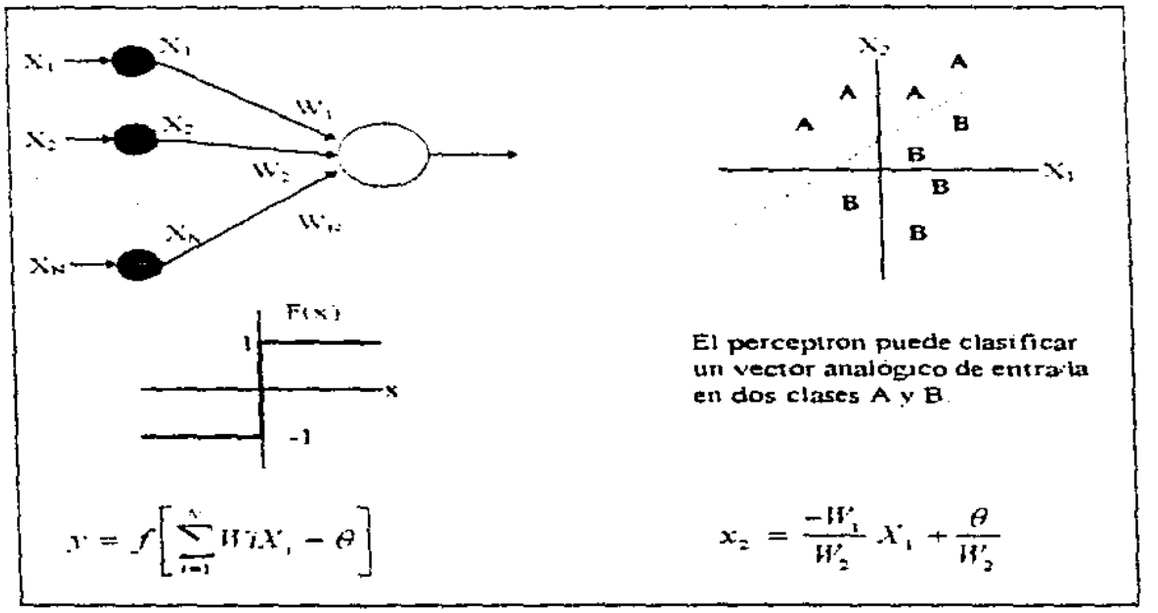

Figura 3.13 : El Perceptron.

Una técnica utilizada para analizar el comportamiento de redes como el Perceptron es representar en un mapa las regiones de decisión creadas en el espacio multidimensional de entradas a la red. En estas regiones se visualiza qué patrones pertenecen a una clase y cuáles a otra. El Perceptron separa las regiones por un hiperplano cuya ecuación queda determinada por los pesos de las conexiones y el

[10] F. Rosenblatt "The Perceptron: A probabilistic model for information storage and organization in the brain Psychological Review, 65 pags 386-408, 1958, reimpreso en el texto "Neurocomputing"(J. Anderson y E. Rosenfeld de) pags 92-114, MIT Press, 1988.

valor umbral de la función de activación de la neurona. En este caso, los valores de los pesos pueden fijarse o adaptarse utilizando diferentes algoritmos de entrenamiento de la red.

Sin embargo, el Perceptron, al constar sólo de una capa de entrada y otra de salida con una única neurona, tiene una capacidad de representación bastante limitada. Este modelo sólo es capaz de discriminar patrones muy sencillos linealmente. El caso mas conocido es la imposibilidad del Perceptron de representar la funcion OR-EXCLUSIVA.

La separabilidad lineal de las redes con solo dos capas se limita a la resolución de problemas en los cuales el conjunto de puntos (correspondientes a los valores de entrada) sean separables geometricamente. En el caso de dos entradas la separacion se lleva a cabo mediante una linea recta. Para tres entradas, la separación se realiza mediante un plano en el espacio tridimensional, y asi sucesivamente hasta el caso de N entradas, en el cual el espacio N-dimensional es dividido en un hiperplano.

3.3.1.1 REGLA DE APRENDIZAJE DEL PERCEPTRON

El algoritmo de aprendizaje del Perceptron es de tipo *supervisado*, lo cual requiere que sus resultados sean evaluados y se realicen las oportunas modificaciones del sistema si fuera necesario. Los valores de los pesos pueden determinar el funcionamiento de la red; estos valores se pueden fijar o adaptar utilizando diferentes algoritmos de entrenamiento de la red. El algoritmo original de convergencia del Perceptron fue desarrollado por Rosenblatt. Se pueden usar Perceptrones como máquinas universales de aprendizaje. Desgraciadamente no se puede aprender a realizar todo tipo de clasificaciones: en realidad, sólo se pueden aprender clasificaciones *fáciles* (problemas de *orden* 1 en la terminología de Minsky y Papert[11]). Esta limitación se debe a que un Perceptron usa un separador lineal como célula de decision, con lo cual no es posible realizar sino una sola separacion lineal (por medio de un hiperplano).

El algoritmo de convergencia de ajuste de pesos para realizar el aprendizaje de un Perceptron (aprendizaje por correccion de error) con N elementos procesales de entrada y un único elemento procesal de salida consta de los siguientes pasos:

1. ***Inicialización de los pesos y del umbral***

   Inicialmente se asignan valores aleatorios a cada uno de los pesos (w_i) de las conexiones y al umbral $(-w_{i,} = \theta)$.

2. ***Presentación de un nuevo par (Entrada, Salida esperada)***

   Presentar un nuevo patron de entrada $X_p = (x_1, x_2, \ldots, x_N)$ junto con la salida esperada $d(t)$.

3. ***Cálculo de la salida actual***

[11] M. Minsky y S. Papert. "Perceptrons". De. MIT Press, 1969.

$$y(t) = f\left[\sum_i w_i(t)x_i(t) - \theta\right] \tag{3.18}$$

Siendo $f(x)$ la función de transferencia escalon.

4. ***Adaptación de los pesos***

$$w_i(t+1) = w_i(t) + \alpha\left[d(t) - y(t)\right]x_i(t) \tag{3.19}$$

$$(0 \le t \le N)$$

donde $d(t)$ representa la *salida deseada*, y sera 1 si el patrón pertenece a la clase A, y -1 si es de la clase B. En estas ecuaciones, α es un factor de ganancia en el rango 0.0 a 1.0. Este factor debe ser ajustado de forma que satisfaga tanto los requerimientos de aprendizaje rápido como la estabilidad de las estimaciones de los pesos. Este proceso se repite hasta que el error que se produce para cada uno de los patrones (diferencia entre el valor de salida deseado y obtenido) es cero o bien menor que un valor preestablecido. Los pesos no cambian si la red ha tomado una decisión correcta.

5. *Volver al paso 2*

Este algoritmo es extensible al caso de múltiples en la capa de salida. El Perceptron sera capaz de aprender a clasificar todas sus entradas, en un número finito de pasos, siempre y cuando el conjunto de los patrones de entrada sea linealmente separable. En tal caso, puede demostrarse que el aprendizaje de la red se realiza en un número finito de pasos.

3.3.2 EL PERCEPTRON MULTINIVEL

Un Perceptron multinivel o multicapa es una red de tipo *feedforward* compuesta de varias capas de neuronas entre la entrada y la salida de la misma. Esta red permite establecer regiones de decisión mucho más complejas que la de dos semiplanos, como hacía el Perceptron de un solo nivel.

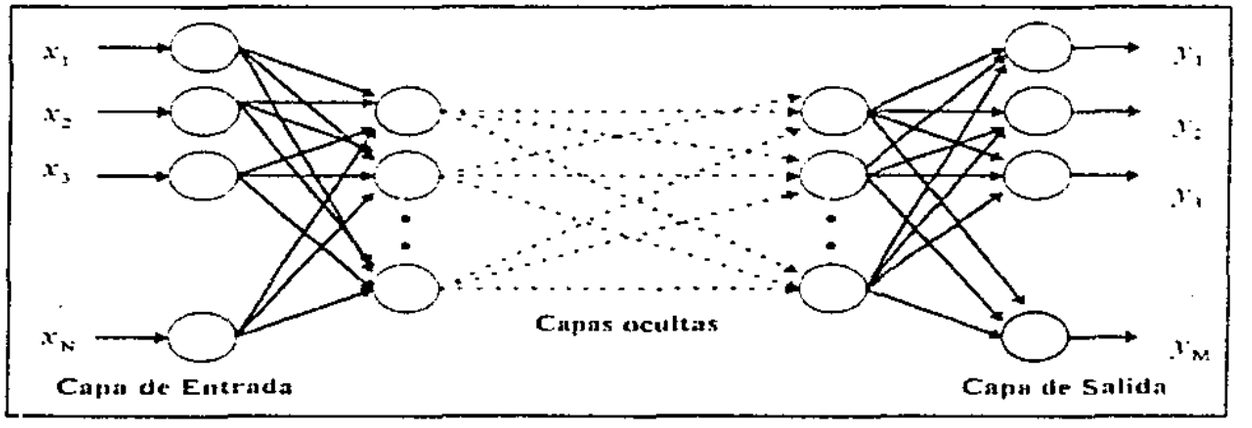

Figura 3.14 : Perceptron multinivel (red feedforward multicapa)

El Perceptron básico de dos capas (la de entrada con neuronas lineales y la salida con función de activación de tipo escalón) sólo puede establecer dos regiones separadas por una frontera lineal en el espacio de patrones de entrada . Un Perceptron con tres niveles de neuronas puede formar cualquier región convexa en este espacio. La regiones convexas se forman mediante la intersección entre las regiones formadas por cada neurona de la segunda capa. Cada uno de estos elementos se comporta como un Perceptron simple , activándose su salida para los patrones de un lado del hiperplano Si el valor de los pesos de las conexiones entre las N_2 neuronas de la segunda capa y una neuronal del nivel de salida son todos 1 y el umbral de la salida es $(N_2 - a)$, donde $0 < a < 1$, entonces la salida de la red se activara solo si las salidas de todos los nodos de la segunda capa están activos Esto equivale a ejecutar la operacion lógica AND en el nodo de salida, resultando una región de decisión que es la interseccion de todos los semiplanos formados en el nivel anterior. La región de decisión resultante de la intersección sera una región convexa con un número de lados a lo sumo igual al número de neuronas de la segunda capa.

Este análisis nos introduce en el problema de la seleccion del número de neuronas ocultas de un Perceptron de tres capas. En general este número debera ser lo suficientemente grande como para formar una región lo suficientemente compleja para la resolución del problema Sin embargo, tampoco es conveniente que el número de nodos sea tan grande que la estimacion de los pesos no sea fiable para el conjunto de patrones de entrada disponibles.

Un Perceptron con cuatro capas puede formar regiones de decisión arbitrariamente complejas. El proceso de separacion en clases que se lleva a cabo, consiste en la partición de la región deseada en pequeños hipercubos (cuadrados para dos entradas de la red). Cada hipercubo requiere $2N$ neuronas en la segunda capa (siendo N el número de entradas a la red), una por cada lado del hipercubo, y otra 3ª capa, que lleva a cabo el AND lógico de las salidas de los nodos del nivel anterior. Las salidas de los nodos de este tercer nivel se activarán sólo para las entradas de cada hipercubo. Los hipercubos se asignan a la región de decisión adecuada mediante la conexión de la salida de cada nodo del tercer nivel sólo con la neurona de salida (cuarta capa) correspondiente a la región de decision en la que está comprendido el hipercubo, llevandose a cabo una operación lógica OR en cada salida . La Operacion lógica OR se llevará a cabo sólo si el valor de los pesos de las conexiones de los nodos del tercer nivel vale uno, y además si el valor de los umbrales de los nodos de salida es de 0.5. Este procedimiento se puede generalizar de manera que la forma de las regiones convexas sea arbitraria, en lugar de hipercubos.

El análisis anterior demuestra que no se requieren más de cuatro capas en una red de tipo Perceptron, pues, como se ha visto, una red con cuatro niveles, puede generar regiones de decisión arbitrariamente complejas. Sólo en ciertos problemas se puede simplificar el aprendizaje mediante el aumento del número de neuronas ocultas . Sin embargo la tendencia es el aumento de la extensión de la funcion de activación, en lugar del aumento de la complejidad de la red Esto de nuevo nos

lleva al problema del número de neuronas ocultas que debemos seleccionar para un Perceptron con cuatro capas.

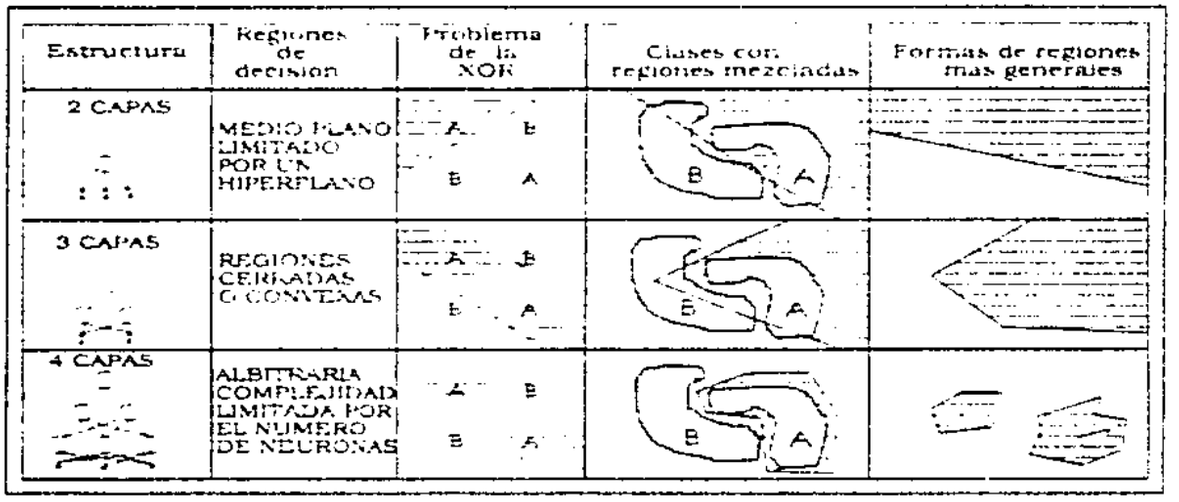

Figura 3.15: Distintas formas de las regiones generadas por un perceptron multinivel

El número de nodos de la 3ª capa (N_3) debe ser mayor que uno cuando las regiones de decisión están desconectadas o *endentadas* y no se pueden formar con una región convexa. Este número, en el peor de los casos, es igual al numero de regiones desconectadas en las distribuciones de entrada. El número de neuronas en la 2ª capa (N_2) normalmente debe ser suficiente para proveer tres o más ángulos por cada área convexa generada por cada neurona de la 3ª capa. Así, deberá haber más de tres veces el número de neuronas de la 3ª capa $(N_2 > 3N_3)$. En la práctica, un número de neuronas excesivo en cualquier capa puede generar ruido. Por otro lado, si existe un número de neuronas redundante se obtiene mayor tolerancia a fallos.

3.3.3 ADALINE Y EL COMBINADOR LINEAL ADALINE

El Adaline es un dispositivo que consta de un único elemento de procesamiento; como tal, técnicamente no es una *red* neuronal.

El término *Adaline* es una sigla; sin embargo, su significado ha cambiado ligeramente con el paso de los años. Inicialmente se llamaba ADAptive Linear Neuron (Neurona Lineal Adaptativa), pasó a ser el ADAptive Linear Element (Elemento Lineal Adaptativo) cuando las redes neuronales cayeron en desgracia a finales de los sesenta. La arquitectura del ADALINE es esencialmente la misma que la del Perceptron. La Figura 3.16 muestra la estructura del ADALINE. Es necesario hacer dos modificaciones básicas a la estructura del Perceptron general para transformarlo en un ADALINE. La primera consiste en añadir una conexión de peso w_0, que se conoce con el nombre de término de

tendencia[12]. Este término es el peso de una conexión que siempre tiene un valor de entrada igual a 1. La inclusión de este término se debe fundamentalmente a la experiencia.

La segunda modificación consiste en añadir una condición bipolar a la salida. El cuadro de líneas punteadas que se ve en la Figura 3.16 encierra una parte de ADALINE que es lo que se denomina **combinador adaptivo lineal** (ALC). Si la salida del ALC es positiva, la salida del ADALINE es +1. Si la salida de ALC es negativa entonces la salida del ADALINE es -1.

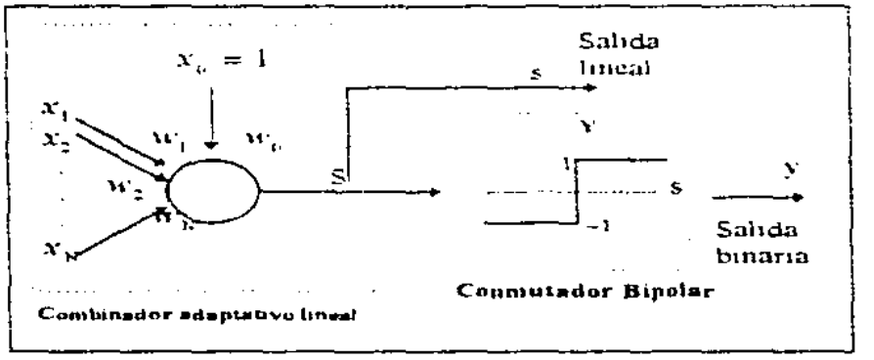

Figura 3.16: Estructura de la red ADALINE, compuesta por un combinador adaptativo lineal y una función de salida bipolar

EL ALC lleva a cabo el cálculo de una suma de productos empleando los vectores de entrada y de peso, y aplica una función de salida para obtener un único valor de salida. Empleando la notación de la Figura 3.16 :

$$y = w_0 + \sum_{j=1}^{n} w_j x_j \tag{3.20}$$

en donde w_0 es el peso de tendencia. Si se hace la identificación $x_0=1$, se puede reescribir la ecuación anterior de forma:

$$y = \sum_{j=0}^{n} w_j x_j \tag{3.21}$$

o bien, en notación vectorial:

$$y = w^t x \tag{3.22}$$

La función de salida en este caso es la función unidad, así como la función de activación. El uso de la función identidad como función de salida y como función de activación significa que la salida es igual a la activación, que es lo mismo que la entrada neta de la unidad.

La salida binaria correspondiente del ADALINE es:

[12] «Bias» en el original (N del T).

$$y(t+1) = \begin{cases} +1 & s > 0 \\ y(t) & s = 0 \\ -1 & s < 0 \end{cases} \qquad (3.22)$$

EL ADALINE (o el ALC) es adaptativo en el sentido de que existe un procedimiento bien definido para modificar los pesos con objeto de hacer posible que el dispositivo proporcione el valor de salida *correcto* para la entrada dada. El significado de *correcto* a efectos del valor de salida depende de la función de tratamiento de señales que esté siendo llevada a cabo por el dispositivo EL ADALINE (o el ALC) es lineal porque la salida es una función lineal sencilla de los valores de entrada

EL ADALINE se puede utilizar para generar una salida analogica utilizando un conmutador sigmoidal, en lugar de binario; en tal caso, la salida se obtendra aplicando una funcion de tipo sigmoidal, como la tangente hiperbolica ($tanh(s)$) o la exponencial ($1 - 1 - e^{-s}$)

3.3.3.1 APRENDIZAJE DEL ADALINE

El ADALINE utiliza aprendizaje OFF LINE con supervisión denominado LMS (Least Mean Squared) o regla del minimo error cuadrado medio. Tambien se conoce como *regla delta* porque trata de minimizar una delta o diferencia entre el valor observado y el deseado en la salida de la red; como ocurre con el Perceptron, solo que ahora la salida considerada es el valor previo a la aplicación de la función de activación de la neurona o, si se prefiere, la salida obtenida al aplicar una función de activación lineal.

3.3.3.1.1 La regla de aprendizaje LMS

La regla de aprendizaje de minimos cuadrados (Least Mean Square) es un método para hallar el vector de pesos W^* deseado, el cual debera ser único y asocia con exito cada vector del conjunto de vectores o patrones de entrada $\{X^1, X^2, X^3, \ldots, X^L\}$ con su correspondiente valor de salida correcto (o deseado) d_k, $k = 1, \ldots, L$. El problema de hallar un conjunto de pesos W^* que para un único vector de entrada X^k dé lugar a un valor de salida correcto resulta sencillo, lo que no ocurre cuando se dispone de un conjunto de vectores de entrada, cada uno con su propio valor de salida asociado. El entrenamiento de la red consiste en adaptar los pesos a medida que se vayan presentando los patrones de entrenamiento y salidas deseadas para cada uno de ellos. Para cada combinación entrada - salida se realiza un proceso automático de pequeños ajustes en los valores de los pesos hasta que se obtienen las salidas correctas.

La regla de aprendizaje LMS minimiza el error cuadrado medio definido como:

$$\langle \varepsilon_k^2 \rangle = \frac{1}{2L} \sum_{k=1}^{L} \varepsilon_k^2 \qquad (3.24)$$

donde L es el número de vectores de entrada (patrones) que forman el *conjunto de entrenamiento*, y ε_k la diferencia entre la salida deseada y la obtenida cuando se introduce el patron k -esimo, que en el caso de la red ADALINE, se expresa $\varepsilon_k = (d_k - s_k)$, siendo s_k la salida ALC; es decir,

$$s_k = X_k \cdot W^T = \sum_{j=0}^{N} w_j x_{kj} \tag{3.25}$$

La función de error es una función matematica definida en el espacio de pesos multidimensionales para un conjunto de patrones dados. Es una superficie que tendra muchos mínimos (globales y locales), y la regla de aprendizaje va a buscar el punto en el espacio de pesos donde se encuentra el mínimo global de esta superficie. Aunque la superficie de error es desconocida, el metodo de gradiente decreciente consigue obtener información local de dicha superficie a través del gradiente. Con esta información se decide qué dirección tomar para llegar hasta el mínimo global de dicha superficie.

Basándose en el método del gradiente decreciente, se obtiene una regla (regla delta o regla LMS) para modificar los pesos de tal manera que hallamos un nuevo punto en el espacio de pesos más próximo al punto mínimo. Es decir, las modificaciones en los pesos son proporcionales al gradiente decreciente de la funcion error $\Delta w_j = -\alpha\left(\partial \varepsilon_k^2 / \partial w_j\right)$. Por tanto, se deriva la función error con respecto a los pesos para ver cómo varía el error con el cambio de los pesos.

Aplicamos la regla de la cadena para cálculo de dicha derivada:

$$\Delta w_j = -\alpha \frac{\partial \langle e_k^2 \rangle}{\partial w_j} = -\alpha \frac{\partial \langle e_k^2 \rangle}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_j} \tag{3.26}$$

Se calcula la primera derivada:

$$\frac{\partial \langle \varepsilon_k^2 \rangle}{\partial s_k} = \frac{\partial \left[\frac{1}{2}(d_k - s_k)^2\right]}{\partial s_k} = \frac{1}{2}\left[2(d_k - s_k)(-1)\right] = -(d_k - s_k) = -\varepsilon_k \tag{3.26}$$

por tanto queda:

$$\frac{\partial \langle \varepsilon_k^2 \rangle}{\partial s_k} = -\varepsilon_k \tag{3.27}$$

Teniendo en cuenta que s_k es la salida lineal:

$$s_k = \sum_{j=0}^{N} w_j x_{kj} \tag{3.28}$$

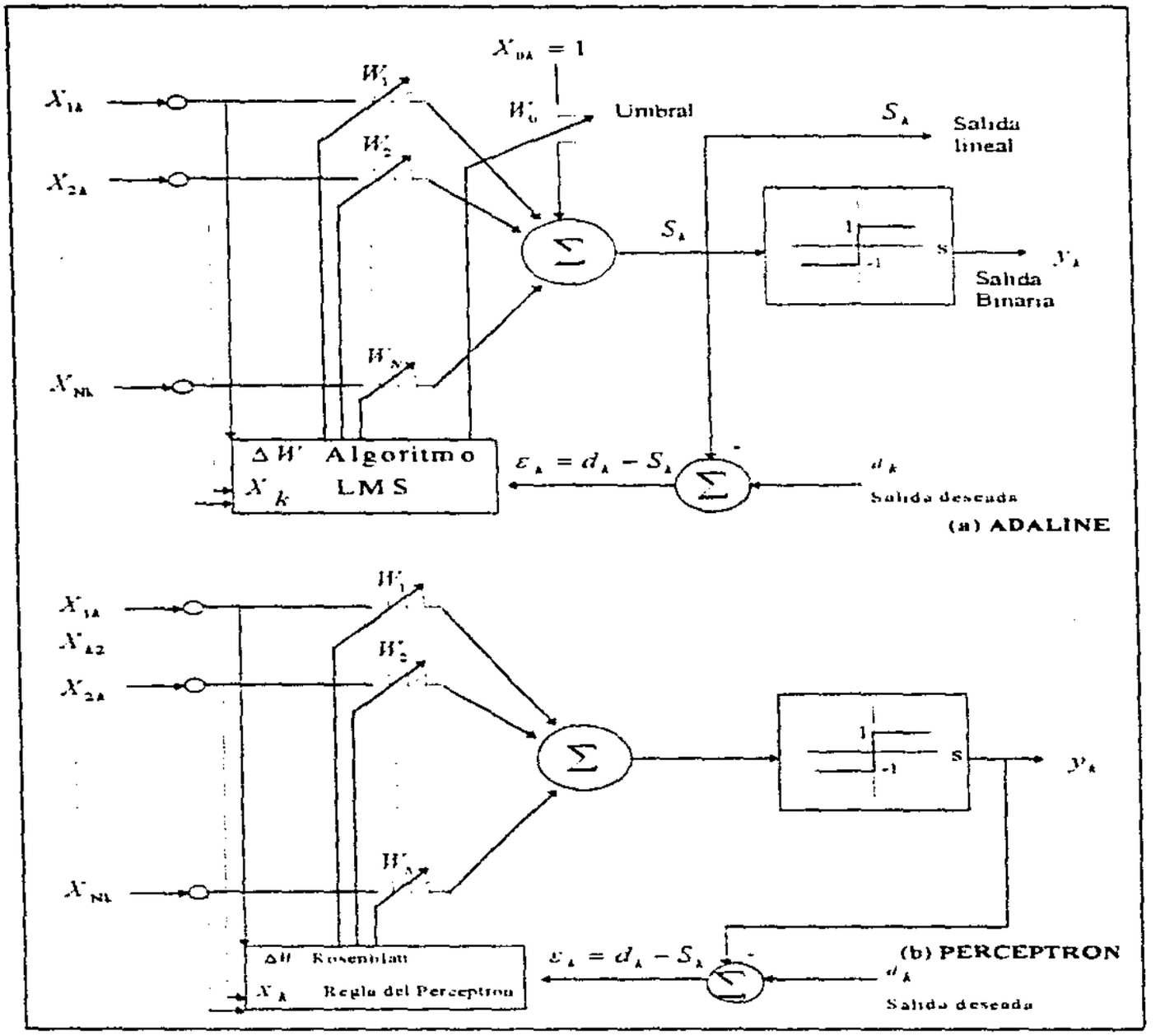

Figura 3.17: Ajuste de los pesos durante el aprendizaje en las redes ADALINE (a) y Perceptron (b) cuando ante un patrón de aprendizaje x_k se desea una salida d_k.

calculamos la segunda derivada de la expresión de Δw_i :

$$\frac{\partial s_k}{\partial w_i} = \frac{\partial\left(\sum_{i=0}^{n}\left(w_i x_{k_i}\right)\right)}{\partial w_i} = \frac{\partial\left(w_i x_{k_i}\right)}{\partial w_i} = x_{k_i} \tag{3.30}$$

Así pues, el valor del gradiente del error producido por un patrón dado (k) es:

$$\frac{\partial\left(\varepsilon_k^2\right)}{\partial w_i} = -\varepsilon_k x_{k_i} \tag{3.31}$$

Las modificaciones en los pesos son proporcionales al gradiente descendente de la función error:

$$\Delta w_i = -\alpha\left(-\varepsilon_k x_{k_i}\right) = \alpha\varepsilon_k x_{k_i} = \alpha(d_k - s_k)x_{k_i} \tag{3.32}$$

$$w_i(t+1) = w_i(t) + \alpha(d_k - s_k)x_{k_i}$$

siendo α la constante de proporcionalidad o tasa de aprendizaje.

En notación matricial, quedaría:

$$W(t+1) = W(t) + \alpha\varepsilon_k X_k = W(t) + \alpha(d_k - s_k)X_k \tag{3.33}$$

Esta expresión representa la modificación de pesos obtenida al aplicar el algoritmo LMS, y es parecida a la obtenida para el Perceptron. α es el parámetro que determina la estabilidad y la velocidad de convergencia del vector de pesos hacia el valor de error mínimo. Los cambios en dicho vector deben ser relativamente pequeños en cada iteración, sino podría ocurrir que no se encontrase nunca un mínimo, o se encontrase solo por accidente, en lugar de ser el resultado de una convergencia sostenida hacia él.

La diferencia entre esta expresión y la del Perceptron está en el valor del error ε_k, que en el caso del perceptron se refería a la diferencia entre el valor deseado y la salida binaria y_k, y no a la salida lineal s_k de la red ADALINE. En la figura 3.17 se representa gráficamente el mecanismo de aprendizaje de ambas redes.

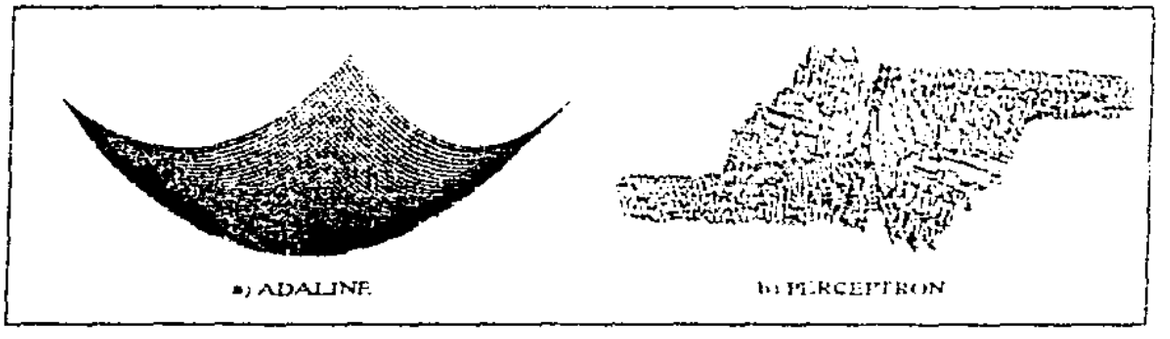

Figura 3.18: Función de error cuadrado medio (ε_k^2) cuando se utiliza para su evaluación la salida S_k (a) o la salida binaria (b).

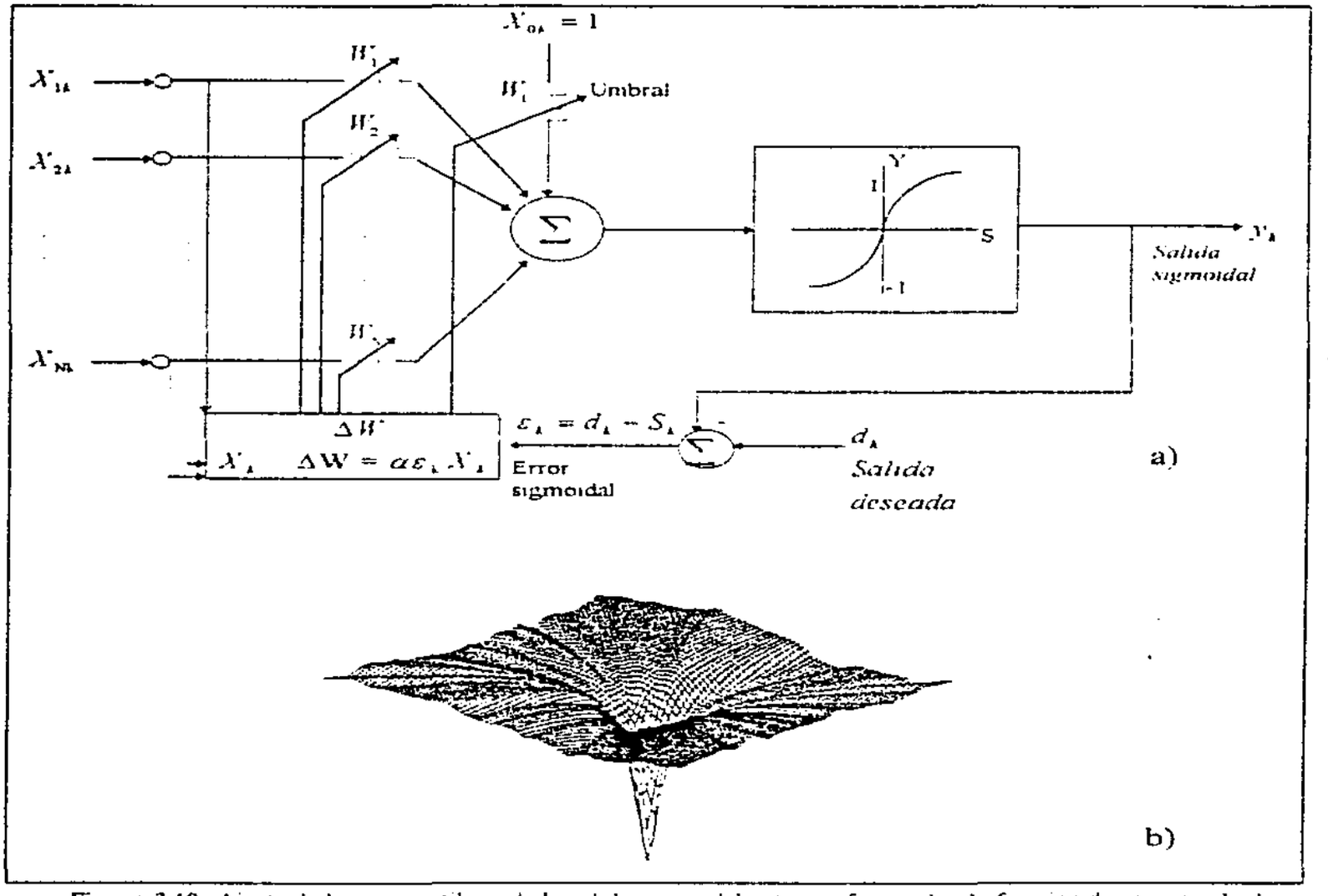

Figura 3.19 :Ajuste de los pesos utilizando la salida sigmoidal (a) y su efecto sobre la funcion de error cuadrado medio $\langle \varepsilon_k^2 \rangle$ (b)

Aunque a simple vista no parece que exista gran diferencia entre ambos tipos de mecanismos de aprendizaje, el caso de la red ADALINE mejora al del Perceptron, ya que es mas sencillo alcanzar el mínimo de error, facilitando la convergencia del proceso de entrenamiento. Esto se demuestra a través de la forma de la función de error cuadrado medio $\langle \varepsilon_k^2 \rangle$, que en el caso de calcularse a partir de la salida lineal (s_k) tiene una forma semejante a la indicada en la Figura 3.18 (a), mientras que en

el caso de utilizar la salida binaria (Perceptron) existe una gran cantidad de mínimos locales (Fig. 3.18 (b)).

Existe una situación intermedia que consistiría en utilizar lo que se conoce como salida sigmoidal, en lugar de la salida lineal ADALINE o la binaria del Perceptron. En tal caso, la función de activación sería del tipo sigmoidal (Figura 3.19 (a)) y la superficie de error tendría la forma indicada en la figura 3.19 (b) con un gran mínimo global (como en el caso de ADALINE) y varios mínimos locales (aunque en menor medida que el Perceptron).

La aplicación del proceso iterativo de aprendizaje (algoritmo de aprendizaje de una red ADALINE) consta de los siguientes pasos:

1. Se aplica un vector o patrón de entrada, X_k, en las entradas del ADALINE.
2. Se obtiene la salida lineal $s_k = X_k \cdot W^T = \sum_{i=0}^{n} w_i x_i$ y se calcula la diferencia con respecto a la deseada $\varepsilon_k = (x_k - s_k)$.
3. Se actualizan los pesos

$$\begin{aligned} W(t+1) &= W(t) + \alpha \varepsilon_k X_k \\ w_i(t+1) &= w_i(t) + \alpha \varepsilon_k x_{ki} \end{aligned} \qquad (3.34)$$

4. Se repiten los pasos 1 al 3 con todos los vectores de entrada (L).
5. Si el error cuadro medio:

$$\langle \varepsilon_k^2 \rangle = \frac{1}{2L} \sum_{k=1}^{L} \varepsilon_k^2 \qquad (3.35)$$

es un valor reducido aceptable, termina el proceso de aprendizaje; sino, se repite desde el paso 1 con todos los patrones.

Cuando se utiliza una red ADALINE para resolver un problema concreto, es necesario determinar una serie de aspectos prácticos, como el número de vectores de entrenamiento necesarios, hallar la forma de generar la salida *deseada* para cada vector de entrenamiento, o la dimensión óptima del vector de pesos, o cuáles deberían ser los valores iniciales de los pesos, así como si es necesario o no un valor de umbral θ, o cuál debe ser el valor de α, o bien cuándo se debe finalizar el entrenamiento, etc. En general, la solución de estas cuestiones depende del problema concreto que se pretenda resolver, por lo que no se pueden dar respuestas genéricas concretas.

Respecto al número de componentes del vector de pesos, si el número de entradas está bien definido, entonces habrá un peso por cada entrada, con la opción de añadir o no un peso para la entrada del umbral. Incluir este término puede ayudar a la convergencia de los pesos proporcionando un grado de libertad adicional.

La solución es diferente cuando sólo se dispone de una señal de entrada . En estos casos, la aplicación más común es el *filtro adaptativo* para, por ejemplo, eliminar el ruido de la señal de

entrada, la cual se muestrea en varios instantes de tiempo, de forma que cada muestra representa un grado de libertad que se utiliza para ajustar la señal de entrada a la salida deseada. La idea consiste en utilizar el menor número de muestras (así obtenemos una convergencia más rápida siempre que se obtengan resultados satisfactorios).

La dimensión del vector de pesos tiene influencia directa en el tiempo necesario de entrenamiento (sobre todo cuando se realiza una simulación por computadora), por lo que generalmente se debe establecer un compromiso entre este aspecto y la aceptabilidad de la solución (normalmente, se mejora el error aumentando el número de pesos).

El valor del parámetro α tiene una gran influencia sobre el entrenamiento. Si α es demasiado grande, es posible que no se produzca la convergencia debido a que se *darán saltos* en torno al mínimo sin alcanzarlo. Si α es demasiado pequeño alcanzaremos la convergencia, pero a costa de una etapa de aprendizaje larga.

En cuanto al momento en el que debemos detener el entrenamiento éste depende, sobre todo de los requisitos de salida del sistema: se detiene el entrenamiento cuando el error observado es menor que el valor admisible en la señal de salida de forma sostenida. Se suele tomar el error cuadrático medio como la magnitud que determina el instante en que el sistema ha convergido.

3. 4 LA RED MADALINE

La red MADALINE (Multiple ADALINE) es una combinación de módulos ADALINE básicos en una estructura de capas que supera algunas limitaciones de la red ADALINE original. (Figura 3.20)

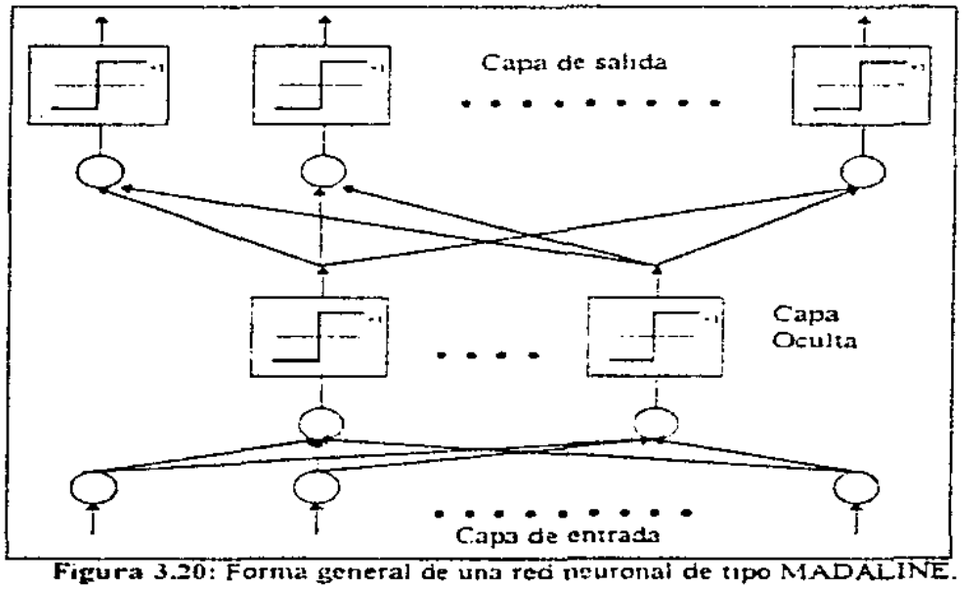

Figura 3.20: Forma general de una red neuronal de tipo MADALINE.

El entrenamiento de estas redes no es el mismo que el de la red ADALINE. El algoritmo LMS podría aplicarse a la capa de salida, puesto que conocemos el vector de salida deseado para cada una de las tramas de entrada de entrenamiento. Sin embargo, lo que se desconoce es la salida deseada para los nodos de cada una de las capas ocultas. Además, el algoritmo LMS funciona para salidas lineales (analógicas) del combinador adaptativo y no para las digitales del ADALINE. La forma de aplicar la idea del algoritmo LMS para entrenar una estructura de tipo MADALINE pasa por la sustitución de la función de salida por una función continua derivable (la función de umbral es discontinua en 0 y, por tanto, no derivable en ese punto).

Existe otro método, conocido como regla II de MADALINE (MRII) parecido a un procedimiento de acierto y error con una inteligencia adicional basada en un principio de perturbación[13]. El entrenamiento equivale a hacer una reducción del número de neuronas de salida incorrectos para cada una de las tramas de entrenamiento que se den como entrada (la salida de la red es una serie de unidades bipolares). Este método se puede aplicar mediante el algoritmo[14]:

1. Se aplica un vector a las entradas de MADALINE y se hace que se propague hasta las unidades de salida.
2. Se cuenta el número de valores incorrectos que hay en la capa de salida, denominándose error a dicho número.
3. Para la unidades de la capa de salida:
   - Se selecciona la primera neurona que no haya sido seleccionada antes cuya salida lineal esté más aproximada a cero. Esta es la neurona que puede cambiar su salida binaria con el menor cambio de sus pesos y, según el principio de mínima perturbación, debe tener prioridad en el proceso de aprendizaje.
   - Se cambian los pesos de la neurona seleccionada de tal modo que cambie su salida binaria.
   - Se hace que se propague el vector de entrada hacia adelante, partiendo de las entradas y en dirección a las salidas, una vez más.
   - Se admite el cambio de pesos si ha dado lugar a una reducción del error, en caso contrario, se restauran los pesos originales.
4. Se repite el paso 3 para todas las capas, salvo la de salida.
5. Para todas las unidades de la capa de salida:
   - Se selecciona el par de neuronas que no hayan sido seleccionadas anteriormente y cuyas salidas lineales estén más próximas a cero.
   - Se aplica una corrección de pesos a ambas neuronas para modificar el valor de su salida.

---

[13] B. Widrow y R. Winter. "Neural Nets for Adaptive Filtering and Adaptive Pattern Recongtion". IEEE computer, 21, pags 25-39, marzo, 1988.

[14] J.A Freeman y D.M. Skapura "Neural Networks . Algorithms, Applications and Programming Techniques". De Adddison Wesley, 1991. Version española "Redes Neuronales. Algoritmos, Aplicaciones y Tecnicas de Programación". De. Diaz de Santos, 1993.

- Se hace que se propague hacia adelante el vector de entrada, desde las entradas hasta las salidas.
- Se admite el cambio de pesos si ha dado lugar a una reducción del error, en caso contrario, se restauran los pesos originales

6. Se repite el paso 5 para todas las capas, salvo la de entrada.

Los pasos 5 y 6 se pueden repetir con grupos de tres, cuatro o mayor número de neuronas hasta obtener resultados satisfactorios. Se considera que las parejas son apropiadas para redes que tengan un máximo de 25 neuronas por capa, aproximadamente.

3.3.5 LA RED DE BACKPROPAGATION

En 1986, Rumelhart, Hinton y Williams, basandose en los trabajos de otros investigadores (Werbos y Parker) formalizaron un método para que una red neuronal *aprendiera* la asociación que existe entre los patrones de entrada a la misma y las clases correspondientes, utilizando más niveles de neuronas que los que utilizo Rosenblatt para desarrollar el Perceptron. Este metodo, conocido en general como *backpropagation* (propagación del error hacia atrás), esta basado en la generalizacion de la regla delta y , a pesar de sus propias limitaciones, ha ampliado de forma considerable el rango de aplicaciones de las redes neuronales.

El algoritmo de propagación hacia atras, o retropropagacion, es una regla de aprendizaje que se puede aplicar en modelos de redes con más de dos capas de neuronas. Una caracteristica importante de este algoritmo es la representación interna del conocimiento que es capaz de organizar en la capa intermedia de las neuronas para conseguir cualquier correspondencia entre la entrada y la salida de la red.

De forma simplificada, el funcionamiento de una *red backpropagation* (backpropagation net, BPN) consiste en un aprendizaje de un conjunto predefinido de pares de entradas - salidas dados como ejemplo, empleando un ciclo *propagación-adaptación* de dos fases; primero se aplica un patron de entrada como un estímulo para la primera capa de las neuronas de la red. Este se va propagando a través de todas las capas superiores hasta generar una salida. Se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error por cada neurona de salida. A continuacion, estos errores se trasmiten hacia atras, partiendo de la capa de salida, hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la participación, de la neurona intermedia en la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportacion relativa al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida sea más cercana a la deseada; es decir, que el error disminuya.

La importancia de la red *backpropagation* consiste en su capacidad de autoadaptar los pesos de las neuronas de las capas intermedias para *aprender* la relación que existe entre un conjunto de patrones

dados como ejemplo y sus salidas correspondientes, para poder aplicar esa misma relación, después del entrenamiento, a nuevos vectores de entrada con ruido o incompletas, dando una salida activa si la nueva entrada es parecida a las presentadas durante el aprendizaje. Esta característica importante, que se exige de los sistemas de aprendizaje es la capacidad de *generalización*, entendida como la facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento, y que puede aplicar además, a entradas no presentadas durante la etapa de aprendizaje para clasificarlas según las características que comportan con los ejemplos de entrenamiento.

3.3.5.1 LA REGLA DELTA GENERALIZADA

La regla propuesta por Widrow en 1960 (regla delta) ha sido extendida a redes con capas intermedias (regla delta generalizada) con conexiones hacia delante (*feedforward*) y cuyas células tienen funciones de activación continuas (lineales o sigmoidales), dando lugar al algoritmo de retropropagación (*backpropagation*). Estas funciones continuas son no decrecientes y derivables. La función sigmoidal pertenece a este tipo de funciones, a diferencia de la función escalón que se utiliza en el Perceptron, ya que esta última no es derivable en el punto en el que se encuentra la discontinuidad.

Este algoritmo utiliza también una función o superficie de error asociada a la red, buscando el estado estable de mínima energía o de mínimo error a través del camino descendente de la superficie del error. Por ello, realimenta el error del sistema para realizar la modificación de los pesos en un valor proporcional al gradiente decreciente de dicha función de error.

3.3.5.1.1 Funcionamiento del algoritmo

El método que sigue la regla delta generalizada para ajustar los pesos es exactamente el mismo que el de la regla delta utilizada en el Perceptron y ADALINE, es decir, los pesos se actualizan de forma proporcional a la delta, o diferencia entre la salida deseada y la obtenida (δ = salida deseada - salida obtenida).

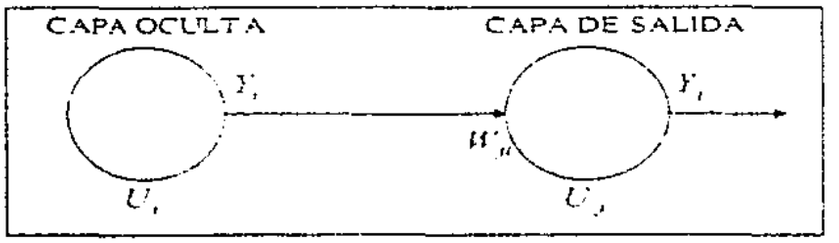

Figura 3.21: Conexión entre una neurona de una capa oculta con una neurona de salida

Dada una neurona (unidad U_j) y la salida que produce y_j (Figura 3.21), el cambio que se produce en el peso de la conexión que une la salida de dicha neurona con la unidad $U_j(w_{ji})$ para un patrón de aprendizaje p determinado es

$$\Delta w_{ji}(t+1) = \alpha \delta_{pj} y_{pi} \tag{3.36}$$

En donde el subíndice p se refiere al patron de aprendizaje concreto y α es la constante o tasa de aprendizaje.

El punto en el que difieren la regla delta generalizada de la regla delta es el valor concreto de δ_{pj}.

Por otro lado, en las redes multinivel, a diferencia de las redes sin neuronas ocultas, en principio no se puede conocer la salida deseada de las neuronas de las capas ocultas para poder determinar los pesos en funcion del error cometido. Sin embargo, inicialmente si podemos conocer la salida deseada de las neuronas de salida. Si consideramos la unidad U_j de *salida* (Figura 3.21), entonces definimos:

$$\delta_{pj} = (d_{pj} - y_{pj}) \cdot f'(net_j) \tag{3.37}$$

donde d_{pj} es la salida deseada de la neurona j para el patrón p y net_j es la entrada neta que recibe la neurona j.

Esta fórmula es como la de la regla delta, excepto en lo que se refiere a la derivada de la función de transferencia. Este término representa la modificacion que hay que realizar en la entrada de la neurona que recibe la neurona j. En el caso de que dicha neurona no sea de salida el error que se produce estará en función del error que se cometa en las neuronas que reciban como entrada la salida de dichas neuronas. Esto es lo que se denomina procedimiento de propagación del error hacia atrás.

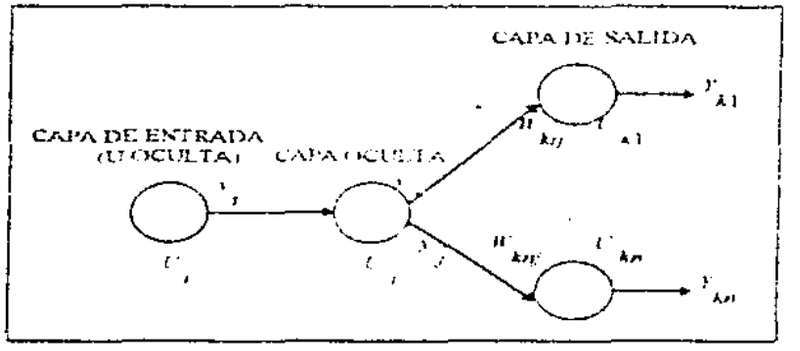

Figura 3.22: Conexiones entre neuronas de la capa oculta con la capa de salida.

En el caso de que U_j *no sea una neurona de salida* (ver Figura 3.22), el error que se produce está en función del error que cometen las neuronas que reciben como entrada la salida de U_j :

$$\delta_{pi} = \left(\sum_k \delta_{pk} w_{ki} \right) \cdot f'(net_i) \tag{3.38}$$

donde el rango de k cubre todas aquellas neuronas a las que está conectada la salida U_i.

De esta forma, el error que se produce en una neurona oculta es la suma de los errores que se producen en las neuronas a las que está conectada la salida de ésta, multiplicando cada uno de ellos por el peso de la conexión.

3.3.5.1.2 Adición de un momento a la regla delta generalizada

El método de retropagación del error, también conocido como del gradiente descendente, requiere un importante numero de calculos para lograr el ajuste de los pesos de la red. En la implementacion del algoritmo, se toma una amplitud de paso que viene dada por la tasa de aprendizaje α. A mayor tasa de aprendizaje, mayor es la modificación de los pesos en cada iteración, con lo que el aprendizaje será más rápido, pero , por otro lado, puede dar lugar a oscilaciones. Rumelhart, Hinto y Williams sugineron que para filtrar las oscilaciones se añada en la expresión del incremento de los pesos un término (momento), β, de manera que dicha expresión quede:

$$w_{ji}(t+1) = w_{ji}(t) + \alpha \delta_{pi} y_{pi} + \beta \left(w_{ji}(t) - w_{ji}(t-1) \right) = \Delta w_{ji}(t+1) = \alpha \delta_{pi} y_{pi} + \beta \Delta w_{ji}(t) \tag{3.39}$$

donde β es una constante (momento) que determina el efecto en $t+1$ del cambio de los pesos en el instante t.

Con este momento se consigue la convergencia de la red en menor número de iteraciones, ya que si en t el incremento de un peso era positivo y en $t+1$ también, entonces el descenso por la superficie de error en $t+1$ es mayor. Sin embargo, si en t el incremento era positivo y en $t+1$ es negativo, el paso que se da en $t+1$ es más pequeño, lo cual es adecuado, ya que eso significa que se ha pasado por un mínimo y que los pasos deben ser menores para poder alcanzarlo.

Resumiendo, el algoritmo de *backpropagation* queda finalmente:

$$w_{ji}(t+1) = w_{ji}(t) + \left[\Delta w_{ji}(t+1) \right] \tag{3.40}$$

$$w_{ji}(t+1) = w_{ji}(t) + \left[\alpha \delta_{pi} y_{pi} + \beta \Delta w_{ji}(t) \right] \tag{3.41}$$

donde:

$$\delta_{pi} = (d_{pi} - y_{pi}) f'(net_i) \tag{3.42}$$

si U_i es una neurona de salida y

$$\delta_{pi} = \left(\sum_k \delta_{pk} w_{ki} \right) f'(net_i) \tag{3.43}$$

si U_i no es una neurona de salida.

3.3.5.2 ESTRUCTURA Y APRENDIZAJE DE LA RED DE *BACKPROPAGATION*

En una red *backpropagation* existe una capa de entrada con *n* neuronas y una capa de salida con *m* neuronas y al menos una capa oculta de neuronas internas. Cada neurona de una capa (excepto las de entrada) recibe entradas de todas las neuronas de la capa anterior y envía su salida a todas las neuronas de la capa posterior (excepto las de salida). No hay conexiones hacia atrás *feedback* ni laterales entre neuronas de una misma capa.

La aplicación del algoritmo *backpropagation* tiene dos fases, una hacia adelante y otra hacia atrás. Durante la primera fase el patrón de entrada es presentado a la red y propagado a través de las capas hasta llegar a la capa de salida. Obtenidos los valores de salida de la red se inicia la segunda fase, comparándose estos valores con la salida esperada para obtener el error. Se ajustan los pesos de la última capa proporcionalmente al error. Se pasa a la capa anterior con una retropopagación del error (*backpropagation*), ajustando convenientemente los pesos y continuando con este proceso hasta llegar a la primera capa. De esta manera se han modificado los pesos de las conexiones de la red para cada ejemplo o patrón de aprendizaje del problema, del que conocíamos su valor de entrada y la salida deseada que debería generar la red ante dicho patrón.

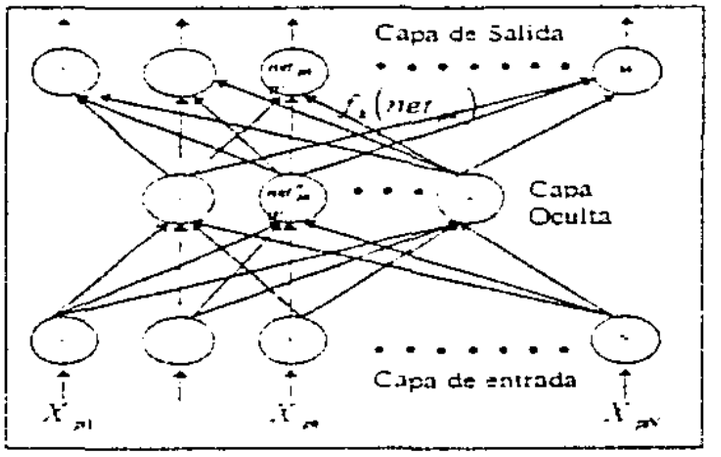

Figura 3.23: Modelo de arquitectura de una red backpropagation.
Pueden existir neuronas ficticias con salida 1 y pesos umbrales θ
de entrada al resto de las neuronas de cada capa.

A diferencia de la regla delta en el caso del Perceptron, la técnica *backpropagation* o generalización de la regla delta, requiere el uso de neuronas cuya función sea continua, y por tanto diferenciable. Generalmente , la función utilizada será de tipo sigmoidal (Figura 3.4).

A continuación se presentan los pasos y formulas a utilizar para aplicar el algoritmo de entrenamiento.

Paso 1

Inicializar los pesos de la red con valores pequeños aleatorios.

Paso 2

Presentar un patrón de entrada, $X_p: x_{p1}, x_{p2}, \ldots, x_{pN}$, y especificar *la salida deseada* que debe generar la red: $d_1, d_2, \ldots, d_M$ (si la red se utiliza como un clasificador, todas las salidas deseadas serán cero, salvo uno, que será la de la clase a la que pertenece el patrón de entrada).

Paso 3

Calcular la salida actual de la red Para ello presentamos las entradas de la red y vamos calculando la salida que presenta cada capa hasta llegar a la capa de salida Ésta será la salida de la red $y_1, y_2, \ldots, y_M$. Los pasos son los siguientes:

- Se calculan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada.

Para una neurona j oculta:

$$net_{jp}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_{j}^{h} \tag{3.44}$$

en donde el índice h se refiere a magnitudes de la capa oculta (hidden); el subíndice p , al p-ésimo vector de entrenamiento, y j a la j -ésima neurona oculta. El término θ puede ser opcional, pues actúa como una entrada más.

- Se calculan las salidas de las neuronas ocultas:

$$y_{jp} = f_j^{h}(net_{jp}^{h}) \tag{3.45}$$

- Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida (capa o: output)

$$net_{pk}^{o} = \sum_{i=1}^{L} w_{ki}^{h} x_{pi} + \theta_{k}^{o} \tag{3.46}$$

$$y_{pk} = f_k^{o}(net_{pk}^{o}) \tag{3.47}$$

Paso 4

Calcular los términos de error para todas las neuronas.

Si la neurona k es una *neurona de la capa de salida*, el valor de *delta* es:

$$\delta_{pk}^{o} = (d_{pk} - y_{pk}) f_k^{o'}(net_{pk}^{o}) \tag{3.48}$$

La función f ,como se citó anteriormente, debe cumplir el requisito de ser derivable, lo que implica la imposibilidad de utilizar una función escalón. En general, disponemos de dos formas de función de

salida que nos pueden servir; la función lineal de salida $\left(f_k\left(net_{pk}\right) = net_{pk}\right)$ y la función sigmoidal representada en la figura 3.4 y definida por la expresión:

$$f_k\left(net_{pk}\right) = \frac{1}{1+e^{-net_{pk}}} \tag{3.49}$$

La selección de la función de salida depende de la forma en que se decidan representar los datos de salida: si se desea que las neuronas de salida sean binarias, se utiliza la función sigmoidal, puesto que esta función es casi biestable y , además, derivable. En otros casos es tan aplicable una función como otra.

Para la función lineal tenemos: $f_k^{o'} = 1$, mientras que la derivada de una función f sigmoidal es:

$$f_k^{o'} = f_k^o(1-f_k^o) = y_{pk}(1-y_{pk}) \tag{3.50}$$

por lo que los términos de error para las neuronas de salida quedan:

$$\delta_{pk}^o = \left(d_{pk} - y_{pk}\right) \tag{3.51}$$

para la salida lineal, y

$$\delta_{pk}^o = \left(d_{pk} - y_{pk}\right)y_{pk}\left(1-y_{pk}\right) \tag{3.52}$$

para la salida sigmoidal.

Si la *neurona* j *no es de salida*, entonces la derivada parcial del error no puede ser evaluada directamente. Por tanto, se obtiene el desarrollo a partir de valores que son conocidos y otros que pueden ser evaluados.

La expresión obtenida en este caso es:

$$\delta_{pj}^h = f_j^{h'}\left(net_{pj}^h\right)\sum_k \delta_{pk}^o w_{kj}^o \tag{3.53}$$

donde observamos que el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí surge el término de *propagación hacia atrás*. En particular, para la función sigmoidal:

$$\delta_{pj}^h = x_{pj}\left(1-x_{pj}\right)\sum_k \delta_{pk}^o w_{kj}^o \tag{3.54}$$

donde k se refiere a todas las neuronas de la capa superior a la de la neurona j. Así, el error que se produce en una neurona oculta es proporcional a la suma de los errores *conocidos* que se producen en las neuronas a las que está conectada la salida de ésta, multiplicando cada uno de ellos por el peso de la conexión. Los umbrales internos de las neuronas se adaptan de forma similar, considerando que están conectados con pesos desde entradas auxiliares de valor constante.

Paso 5

Actualización de los pesos

Para ello, utilizamos el algoritmo recursivo, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la forma siguiente.

Para los pesos de las neuronas de la capa de salida:

$$w_{ki}^{o}(t+1) = w_{ki}^{o}(t) + \Delta w_{ki}^{o}(t+1) \;\; ; \;\; \Delta w_{ki}^{o}(t+1) = \alpha \delta_{pk}^{o} y_{pi} \tag{3.55}$$

y para los pesos de las neuronas de la capa oculta.

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \Delta w_{ji}^{h}(t+1) \;\; ; \;\; \Delta w_{ji}^{h}(t+1) = \alpha \delta_{pj}^{h} x_{pi} \tag{3.56}$$

En ambos casos, para acelerar el proceso de aprendizaje, se puede añadir un término *momento* de valor: $\beta\left(w_{ki}^{o}(t) - w_{ki}^{o}(t-1)\right)$ en el caso de la neurona de salida y $\beta\left(w_{ji}^{h}(t) - w_{ji}^{h}(t-1)\right)$ cuando se trata de una neurona oculta.

Paso 6

El proceso se repite hasta que el término de error

$$E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^{2} \tag{3.57}$$

resulta aceptablemente pequeño para cada uno de los patrones aprendidos.

3.3.5.4 CONSIDERACIONES SOBRE EL ALGORITMO DE APRENDIZAJE

El algoritmo de *backpropagation* encuentra un valor mínimo de error (local o global) mediante la aplicación de pasos descendentes (gradiente descendente). Cada punto de la superficie de la función de error corresponde a un conjunto de valores de los pesos de la red. Con el gradiente descendente, siempre que se realiza un cambio en todos los pesos de la red, se asegura el descenso por la superficie del error hasta encontrar el valle más cercano, lo que puede hacer que el proceso de aprendizaje se detenga en un mínimo local de error.

Por tanto, uno de los problemas que presenta este algoritmo de entrenamiento de redes multicapa es que busca minimizar la función de error, pudiendo caer en un mínimo local o algún punto estacionario, con lo cual no se llega a encontrar el mínimo global de la función del error. Sin embargo, ha de tenerse en cuenta que no tiene porqué alcanzarse el mínimo global en todas las aplicaciones, sino que puede ser suficiente con un error mínimo establecido.

3.3.5.4.1 Control de la convergencia

En las técnicas de gradiente decreciente es conveniente avanzar por la superficie de error con incrementos pequeños de los pesos. Esto se debe a que tenemos una información local de la

superficie y no se sabe lo lejos o lo cerca que está del punto mínimo. Con incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo sin conseguir estacionarse en él. Con incrementos pequeños, aunque se tarde más en llegar, se evita que ocurra esto.

El elegir un incremento adecuado influye en la velocidad con la que converge el algoritmo. Esta velocidad se controla a través de la constante de proporcionalidad o tasa de aprendizaje α. Normalmente, α debe ser un número pequeño (del orden de 0.05 a 0.25), para asegurar que la red llegue a asentarse en una solución. Un valor pequeño de α significa que la red tendrá que hacer un gran número de iteraciones. Si esa constante es muy grande, los cambios de pesos son muy grandes, avanzando rápidamente por la superficie de error, con el riesgo de saltar el mínimo y estar oscilando alrededor de él, pero sin poder alcanzarlo.

Lo habitual es aumentar el valor de α a medida que disminuye el error de la red durante la fase de aprendizaje. Así, aceleramos la convergencia, aunque sin llegar nunca a valores de α demasiado grandes, que hicieran que la red oscilase alejándose demasiado del valor mínimo. Otra forma de incrementar la velocidad de convergencia consiste en añadir un término *momento* consistente en sumar una fracción del anterior cambio cuando se calcula el valor del cambio de peso actual. Este término adicional tiende a mantener los cambios de peso en la misma dirección.

Un último aspecto a tener en cuenta es la posibilidad de convergencia hacia alguno de los *mínimos locales* que puedan existir en la superficie de error del espacio de pesos (Figura 3.24). En el desarrollo matemático que se ha realizado para llegar al algoritmo de retropropagación, no se asegura en ningún momento que el mínimo que se encuentre sea global. Una vez que la red se asienta en un mínimo, sea local o global, cesa el aprendizaje, aunque el error siga siendo demasiado alto, si se alcanza un mínimo local. En todo caso, si la solución es admisible desde el punto de vista del error, no importa si el mínimo es local o global o si se ha detenido en algún momento previo a alcanzar un verdadero mínimo.

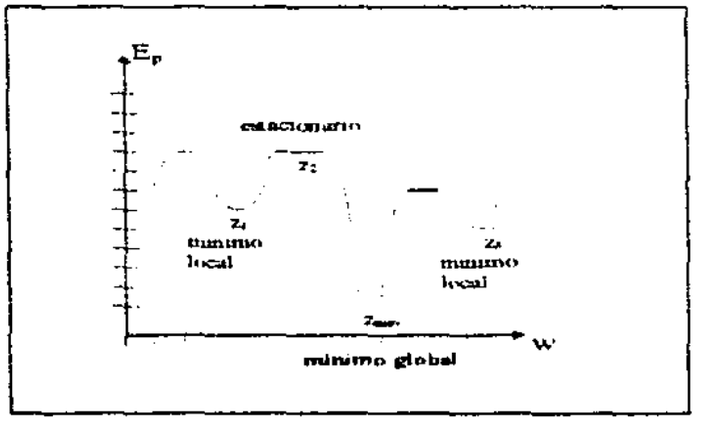

Figura 3.24: Ejemplo representativo de una forma de la superficie de error, donde w representa los posibles valores de la matriz de pesos de la red.

En la práctica, si una red deja de aprender antes de llegar a una solución aceptable, se realiza un cambio en el número de neuronas ocultas o en los parámetros de aprendizaje o, simplemente, se vuelve a empezar con un conjunto distinto de pesos originales y se vuelve a resolver el problema.

3.3.5.4.2 Dimensionamiento de la red. Número de neuronas ocultas

No se pueden dar reglas concretas para determinar el número de neuronas o el número de capas de una red para resolver un problema concreto. Lo mismo ocurre a la hora de seleccionar el conjunto de vectores de entrenamiento. En todos estos casos, lo único que se puede dar son unas cuantas ideas generales deducidas de la experiencia de numerosos autores[15] .

Respecto al número de capas de la red, en general tres capas son suficientes (entrada-oculta-salida). Sin embargo, hay veces en que un problema es más fácil de resolver (la red aprende más deprisa) con más de una capa oculta. El tamaño de las capas, tanto de entrada como de salida, suele venir determinado por la naturaleza de la aplicación. En cambio, decidir cuantas neuronas debe tener la capa oculta no suele ser tan evidente.

El número de neuronas ocultas interviene en la eficacia de aprendizaje y de generación de la red. No hay ninguna regla que indique el número óptimo, en cada problema se debe ensayar con distintos números de neuronas para organizar la representación interna y escoger el mejor. La idea más utilizada, sobre todo en los sistemas simulados, consiste en tener el menor número posible de neuronas en la capa oculta, porque cada una de ellas supone una mayor carga de procesamiento en el caso de una simulación. En un sistema implementado en hardware, este problema no es crucial; sin embargo, sí habrá que tener presente el problema de comunicación entre los distintos elementos del proceso.

Es posible eliminar neuronas ocultas si la red converge sin problemas, determinando el número final en función del rendimiento global del sistema. Si la red no converge, es posible que sea necesario aumentar este número. Por otro lado, examinando los valores de los pesos de las neuronas ocultas periódicamente en la fase de aprendizaje, se pueden detectar aquellas cuyos pesos cambian muy poco durante el aprendizaje respecto a sus valores iniciales, y reducir por tanto el número de neuronas que apenas participan en el proceso de aprendizaje.

3.3.5.4.3 Inicialización y cambio de pesos

Sería ideal, para una rápida adaptación del sistema, inicializar los pesos con una combinación de valores (W') muy cercano al punto de mínimo error buscado. Pero es imposible, porque no se conoce a *priori* dónde está el punto mínimo. Así, se parte de un punto cualesquiera del espacio, inicializando

[15] J.A Freeman y D.M Skapura. "Neural Network. Algorithms, Applications and Programming Techniques". Ed. Addison-Wesley, 1991. Versión española: "Redes Neuronales. Algoritmos, Aplicaciones y Técnicas de Programación", Ed. Díaz de Santos.

los pesos con valores pequeños aleatorios cualesquiera (por ejemplo 0.5), al igual que los términos umbral θ_i, que aparecen en las ecuaciones de entrada neta a cada neurona. Este valor umbral se suele tratar como un peso más que está conectado a una neurona ficticia de salida siempre 1. La utilización del término umbral es opcional, pues en caso de utilizarse, es tratado exactamente igual que un peso más y participa como tal en el proceso de aprendizaje

La expresión de la entrada neta a cada neurona se podrá escribir de la forma:

$$net_{pk} = \sum_{j=1}^{L} w_{kj} x_{pj} + \theta_k \tag{3.58}$$

para una neurona (k). Si consideramos $\theta_k \equiv w^0_{k(L+1)}$; $x_{p(L+1)} \equiv 1$, se puede escribir la siguiente expresión:

$$net_{pk} = \sum_{j=1}^{L+1} w_{kj} x_{pj} \tag{3.59}$$

o, lo que es lo mismo, si consideramos $\theta_k \equiv w^0_{k(L+1)}$; $x_{p0} \equiv 1$, podemos tomarla como:

$$net_{pk} = \sum_{j=0}^{L} w_{kj} x_{pj} \tag{3.60}$$

La modificación de los pesos puede realizarse cada vez que un patrón ha sido presentado, o bien después de haber acumulado los cambios de los pesos en un número de iteraciones. El momento adecuado para cambiar los pesos depende de cada problema concreto.

CAPITULO 4

SIMULACIÓN DE UN NEURO-CONTROLADOR ADAPTATIVO

4.1 NEURO-CONTROLADORES ADAPTATIVOS

Los controladores adaptativos son componentes esenciales para la estabilidad y el desempeño robusto de manipuladores. Un esquema de control adaptativo consiste principalmente de un controlador con parámetros ajustables que cambian en respuesta a una señal de error, logrando un desempeño seguro del control.

Un típico controlador adaptativo se muestra en la Figura 4.1 Este controlador, el cual fue desarrollado por Lee y Chung[16], está basado en ecuaciones lineares de perturbación alrededor de una trayectona nominal prescrita. Un modelo de referencia, el cual representa la componente de alimentacion hacia adelante del controlador, la cual calcula los torques nominal τ_n mientras que la componente de alimentación hacia atrás produce una señal de perturbación δ, para reducir la desviación entre las trayectorias actual y nominal.

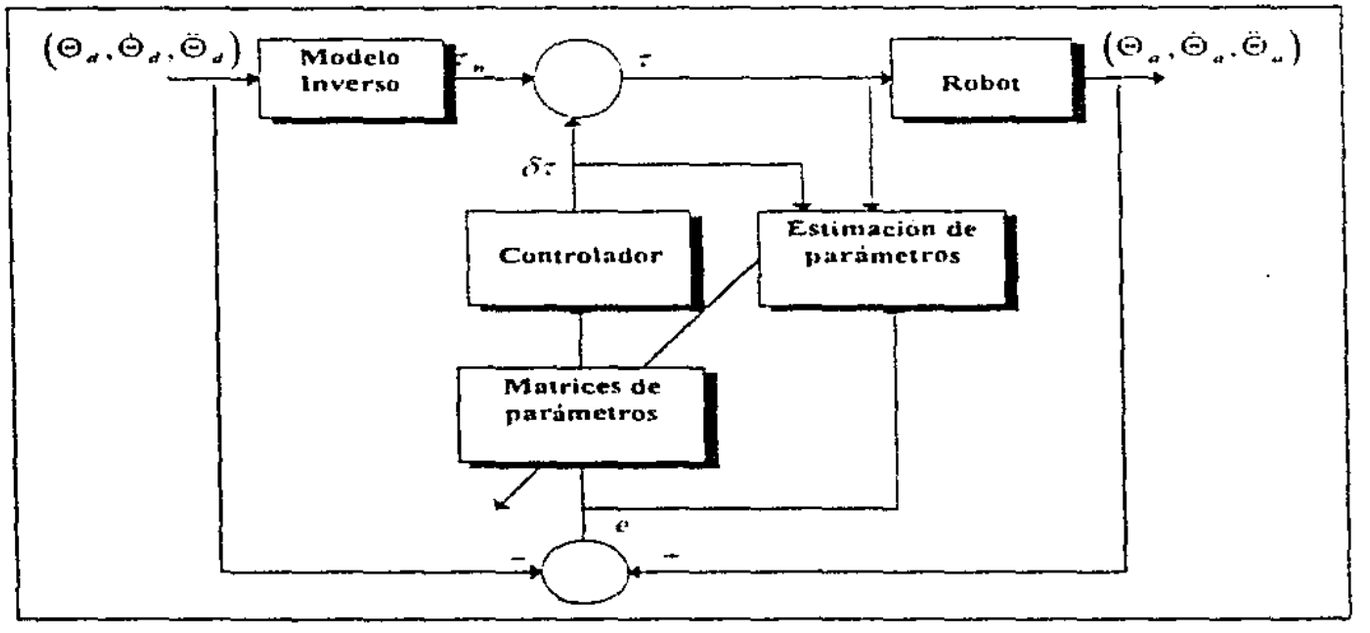

Figura 4.1: Control adaptativo con parametros de estimación

Este modelo consta de un esquema recursivo que es usado para identificar los parámetros del sistema y un controlador que emplea estos parámetros para producir los torques de perturbación de acuerdo a uno de los pasos de las leyes de control óptimo. La principal desventaja de este método es

[16] Lee,C.S.G. y Chung, M.J. [1984]. «An adaptive control strategy for mechanical manipulators ».*IEEE Transactions on Automatic Control*, AC-29(9), 837-840 pp.
Lee,C.S.G y Chung, M.J. [1984] «Adaptive control for robot manipulators in joint and cartesian coorninates». *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 530-540 pp

que ocupa mucho tiempo en el cálculo del algoritmo de estimación de parámetros debido a su naturaleza recursiva, por lo que éste no puede ser fácilmente implementado usando técnicas de procesamiento en paralelo.

Yabuta y Yamda[17] sugirieron un controlador directo de alimentación hacia adelante mostrado en la figura 4.2. El controlador es una red neuronal que realiza la dinamica inversa del robot por cambios en sus pesos según una señal de error. Una de las desventajas de este método es que el sistema puede perder robustecimiento en el comienzo del control porque éste depende de los pesos iniciales de la red neuronal. También, dado que el robot es una planta altamente no linear, el entrenamiento será duro y la red neuronal tiene que ser de un tamaño enorme para lograr un buen desempeño.

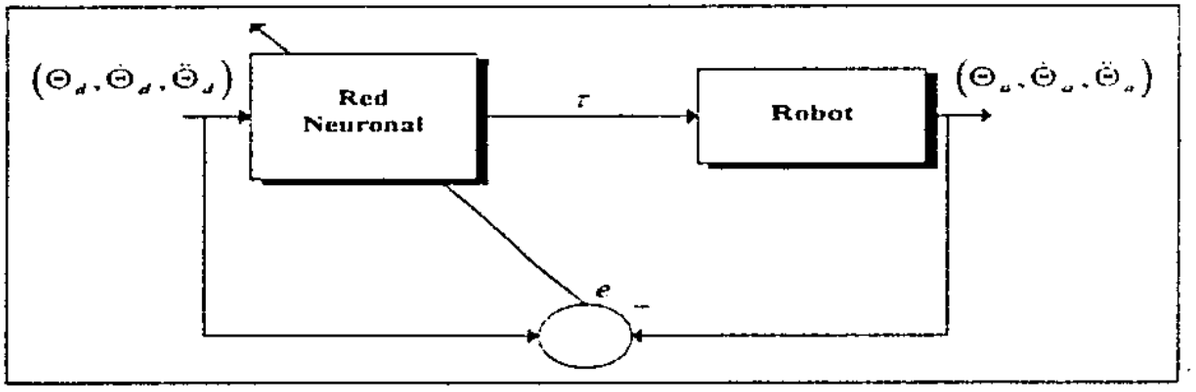

Figura 4.2: Controlador directo con alimentación hacia adelante

Albert Y. Zomaya y Tarek M. Nabhan[18] desarrollaron un controlador que consiste de un modelo de referencia para calcular fuerzas y torques nominales y una red neuronal, la cual representa la componente de realimentación que produce una señal correcta. La red neuronal representa un controlador no linear y sus pesos y umbrales son los parámetros ajustables. El neuro-controlador es primero entrenado off-line para producir una señal correcta y debido a que los parámetros dinámicos del robot son por naturaleza variantes con el tiempo, es necesario continuar cambiándolos durante las operaciones en linea (on-line). De aqui en adelante, el modelo produce las fuerzas y torques nominales, la red neuronal produce una señal correcta para cada una de las articulaciones, y entonces continuamente cambia sus pesos, en respuesta a una señal de error. De esta manera genera mejores señales hasta que la tarea sea completada. La forma general del sistema se muestra en la figura 4.3:

[17] Yabuta,T. y Yamda T. [1990]. «Posibility of neural networks controller for robot manipulators» *Proceeding of the IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1686-1691 pp.

[18] Zomaya,Y. Albert y Nabhan, M. Tarek [1993] «Centralized and Decentralized Neuro Adaptive Robot Controllers», *Neural Networks*, Vol. 6, págs. 223-244 pp.

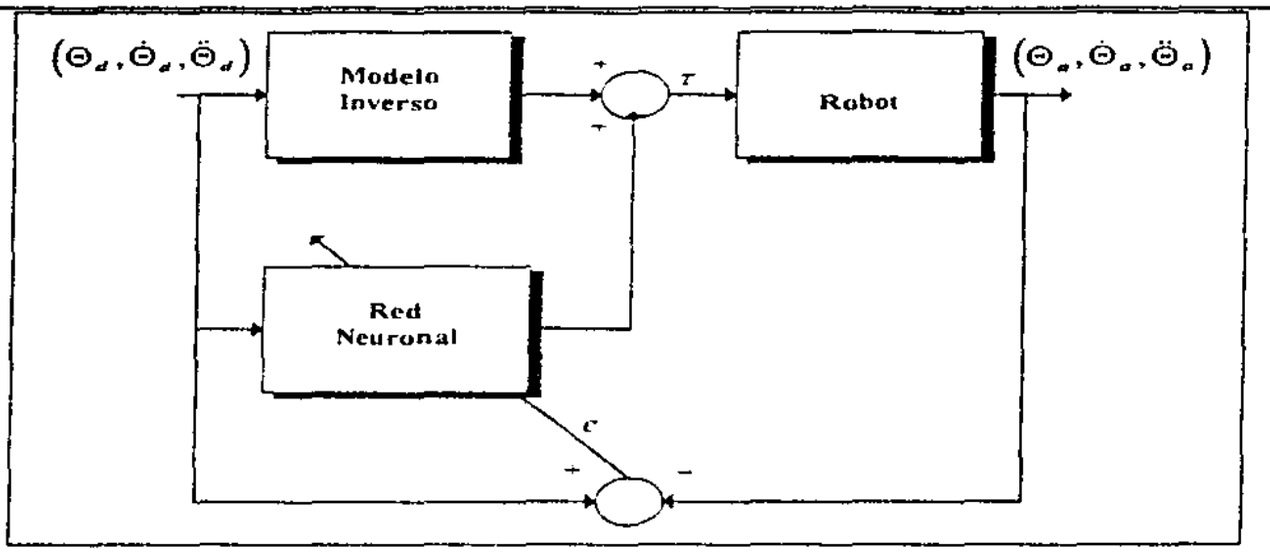

Figura 4.3: Sistema de neurocontrol adaptativo

El sistema de control es muy general en el sentido de que éste puede ser aplicado a cualquier manipulador con cualquier número de grados de libertad. La red neuronal no necesita de antemano ninguna información sobre el robot ni una estimación de la carga desconocida. La dinámica del robot (modelo inverso) puede ser calculada usando arquitecturas en paralelo para prevenir cualquier cuello de botella en los cálculos. Las técnicas de control adaptativo han estimado los parámetros dinámicos usando como recurso, algoritmos con gasto excesivo de cálculos; los cuales son muy difíciles de calcular en tiempo real. El neurocontrolador propuesto por los autores no requiere la estimación de los parámetros dinámicos del robot. En adición, el neurocontrolador es inherentemente paralelo y puede ser implementado usando componentes VLSI. También, muchas de las técnicas de control adaptativo necesitan el valor de los torques del robot, los cuales son difíciles de medir. De esta manera las técnicas propuestas por los autores proveen un método más práctico para construir un controlador adaptivo más eficiente.

La red neuronal usada en esta propuesta es una red neuronal multicapa con conexiones hacia adelante basada en el algoritmo de backpropagation (Capítulo 3). La función de activación de la neurona j puede ser expresada como.

$$f_j(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{4.1}$$

y

$$x = \sum_i W_{ij} O_i + \Theta_j \tag{4.2}$$

donde O_i es la salida de la unidad i, W_{ij} es el peso de la unidad i a la unidad j, y Θ_j es el umbral de la unidad j.

La simulación realizada en esta tesis utiliza el controlador propuesto por Zomaya y Nabhan, debido a las ventajas que ofrece este nuevo controlador.

4.2 CONVERGENCIA Y ESTABILIDAD

La expresión general para la dinámica del robot, como se mostró en el capítulo 2, tiene la forma:

$$\tau(t) = D(\Theta)\ddot{\Theta} + C\left(\Theta, \dot{\Theta}\right) + H(\Theta) \tag{4.3}$$

Donde $\tau(t)$ en nuestro sistema de control representa la señal de entrada a controlar, la cual tiene la forma :

$$\tau(t) = \tau_{inv}(t) + S(t)\tau_{NN}(t) \tag{4.4}$$

donde $\tau_{inv}(t)$ es la salida del modelo inverso, $\tau_{NN}(t)$ es la salida de la red neuronal, $S(t)$ es una función de escalamiento usada para reducir los efectos de las inexactitudes de la red neuronal, y $\tau_{NN}(t)$ satisface la siguiente relación:

$$\tau_{NN} = N(W, I) \tag{4.5}$$

Zomaya y Nabhan consideraron una red neuronal de n - capas con funciones sigmoidales F. Con ésta se obtiene la siguiente relación:

$$N(W, I) = F(W_n^t Z_{n-1}) \tag{4.6}$$

$$\begin{aligned} Z_{n-1} &= F(W_n^t Z_{n-1}) \\ &\vdots \\ Z_1 &= F(W_1^t I) \end{aligned} \tag{4.7}$$

donde I es la entrada al vector, W_1^t es la matriz de pesos de la capa de entrada, W_{n-1}^t es la matriz de pesos de la capa $n-2$ a la capa $n-1$, y W_n^t es la matriz de pesos de la capa $n-1$ a la capa de salida. La relación entre las matrices de pesos $W_1, W_2, \ldots, W_n^t$ y el vector de pesos W^t está dada por:

$$W_1 = \begin{vmatrix} W_{11}^T \\ \vdots \\ W_{1j}^T \\ \vdots \\ W_{1n}^T \end{vmatrix} \tag{4.8}$$

$$W_2^{*T} = \begin{vmatrix} W_{21}^{*T} \\ \vdots \\ W_{2i}^{*T} \\ \vdots \\ W_{2n}^{*T} \end{vmatrix} \quad \cdots \quad W_n^{*T} = \begin{vmatrix} W_{n1}^{*T} \\ \vdots \\ W_{ni}^{*T} \\ \vdots \\ W_{nk}^{*T} \end{vmatrix} \tag{4.9}$$

$$W^{*T} = \left(\ldots W_1^{*T}, \ldots W_{2i}^{*T}, \ldots W_{nk}^{*T} \ldots \right) \tag{4.10}$$

Para introducir el neurocontrolador adaptativo, la ecuación será linealizada con una trayectoria nominal para denvar las ecuaciones linealizadas, las cuales han sido ampliamente usadas en las estrategias convencionales de control adaptativo. Dadas $\Theta_d(t), \dot{\Theta}_d(t), \ddot{\Theta}_d(t)$ que representan las posiciones, velocidades y aceleraciones a lo largo de la trayectoria preestablecida y $\tau_d(t)$ que representa las correspondientes fuerzas/torques nominales, se satisface la ecuacion:

$$\tau_d(t) = D(\Theta_d)\ddot{\Theta}_d(t) + C\left(\Theta_d, \dot{\Theta}\right) + H(\Theta_d) \tag{4.11}$$

Expandiendo la ecuación 4.3 sobre la trayectoria nominal, y restando a la ecuación 4.11 del resultado, mientras asumimos que los términos de órdenes más altos son insignificantes, el modelo puede así ser expresado como:

$$\begin{aligned} \delta\tau(t) = I(\Theta_d)\delta\ddot{\Theta} + J\left(\Theta_d, \dot{\Theta}_d\right)\delta\dot{\Theta} + \\ + K\left(\Theta_d, \dot{\Theta}_d, \ddot{\Theta}_d\right)\delta\Theta \end{aligned} \tag{4.12}$$

donde

$$I(\Theta_d) = \frac{\partial\left[D(\Theta)\ddot{\Theta}\right]}{\partial\ddot{\Theta}}\Big|_{\Theta_d} \tag{4.13}$$

$$J\left(\Theta_d, \dot{\Theta}_d\right) = \frac{\partial\left[D(\Theta)\dot{\Theta}\right]}{\partial\Theta}\Big|_{\Theta_d, \dot{\Theta}_d} \tag{4.14}$$

$$\begin{aligned} k\left(\Theta_d, \dot{\Theta}_d, \ddot{\Theta}_d\right) = \frac{\partial\left[D(\Theta)\ddot{\Theta}\right]}{\partial\Theta}\Big|_{\Theta_d, \ddot{\Theta}_d} \\ + \frac{\partial C\left(\Theta, \dot{\Theta}\right)}{\partial\Theta}\Big|_{\Theta_d, \dot{\Theta}_d} + \frac{\partial H(\Theta)}{\partial\Theta}\Big|_{\Theta_d} \end{aligned} \tag{4.15}$$

Las ecuaciones amba representan un sistema real linear positivo Multi-Entrada Multi-Salida (Multi-Input Multi-Output, MIMO) con m entradas y n salidas de la forma:

$$\dot{e}(t) = Fe(t) + Gu(t) \tag{4.16}$$

donde:

$$e(t) = Y_d(t) - Y(t) \tag{4.17}$$

$$\dot{e}(t) = \frac{d}{dt}e(t) \tag{4.18}$$

$$u(t) = \tau(t) - \tau_d(t) \tag{4.19}$$

donde $Y(t)$, $Y_d(t)$ son vectores R^n que representan los valores actuales y los valores deseados de la posición y la velocidad, respectivamente, F es la matriz constante $R^{n\times n}$, y G es la matriz constante $R^{k\times n}$. El siguiente error cuadrático $l'(e)$ puede ser usado como una función candidata de Lyapunov:

$$l'(e) = \frac{e^T P e}{2} \tag{4.20}$$

donde P es definitivamente una matriz simétnca positiva y hay otra matriz positiva Q que satisface la relación:

$$PF + F^T P = -Q \tag{4.21}$$

La estabilidad del sistema está garantizada cuando $dl'/dt < 0$ puesto que la función Lyapunov seleccionada es definitivamente una función positiva.

$$\begin{aligned}\frac{dl'}{dt} &= \frac{e^T(Pf + F^T P)e}{2} + \frac{e^T PGu}{2} + \frac{u^T G^T Pe}{2} \\ &= -\frac{e^T Qe}{2} + \frac{e^T PGu}{2} + \frac{u^T G^T Pe}{2}\end{aligned} \tag{4.22}$$

Dado que el primer término es negativo, la condición $dl'/dt < 0$ es satisfecha si $e^T PGu \le 0$. Además, tomando en cuenta que el sistema emplea un modelo referencial y que la inicial inexactitud de la red neuronal está reducida por el uso de un esquema de escalamiento, la desviación de la señal de entrada al controlador u puede ser considerada como insignificante.
Ahora

$$\Delta u = \left(\frac{\partial N}{\partial W}\right)\Delta W, \tag{4.23}$$

dados

$$\Delta W = -\left(\frac{\partial N}{\partial W}\right)^T G^T P^T e \tag{4.24}$$

entonces:

$$e^T PGu = e^T PG\left(\frac{\partial N}{\partial W}\right)\Delta W$$
$$= e^T PG\left(\frac{\partial N}{\partial W}\right)\left(\frac{\partial N}{\partial W}\right)^T G^T P^T e \qquad (4.25)$$

que es siempre negativa. G y P son matrices constantes, por lo que se tiene:

$$\Delta W \propto -\left(\frac{\partial N}{\partial W}\right)^T e \qquad (4.26)$$

que conducirá a las mismas fórmulas por las que los pesos del algoritmo de backpropagation generalizado se actualizan.[19]

4.3 ENTRENAMIENTO

Como se mencionó en el capítulo 3 los métodos de entrenamiento están divididos en tres principales categorías: aprendizaje no supervisado, aprendizaje supervisado y aprendizaje con refuerzo. Los dos últimos métodos son similares excepto por el tipo de información que el maestro les provee. El maestro en el aprendizaje supervisado instruye a la red neuronal sobre qué debe generar como salida, mientras que el maestro en el aprendizaje con refuerzo provee a la red con una señal de evaluación que representa el grado de desempeño de la misma.

El neurocontrolador utilizado en este trabajo debe ser entrenado para responder al comando de entrada y generar una señal correcta que minimice la diferencia entre la salida actual y la salida deseada del robot. El método de entrenamiento usado aquí es un método con refuerzo, en el sentido de que la red neuronal primero produce una señal correcta en respuesta a un comando de entrada y la red recibe una evaluación del ambiente en la forma de un error. La red neuronal usa esta señal para cambiar sus pesos de forma que minimiza el error para futuras entradas .

Muchos de las investigaciones en aprendizaje reforzado emplean métodos de aprendizaje estocástico donde la red genera una salida de acuerdo a una función distribución de probabilidad almacenada[20]. El ambiente entonces regresa una señal de evaluación que es usada por la red neuronal para modificar los parámetros de la función de distribución de probabilidad de forma que incrementa las probabilidades de evaluacion favorable en el futuro. Estos métodos fueron principalmente desarrollados por Tsetlin[21] (1973), y desde entonces ellos han atraído a muchos investigadores.

---

[19] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. [1980]. «Learning internal representation by error propagation» In D.E. Rumelhart & MCClelland (Eds) *Parallel distributed processing: Explorations in the microstructures of cogniton* , Vol 1: Foundations (págs. 318-362) Cambridge, MA: MIT Press.

[20] Barto, A.G., Sutton,R.S., y Brouwer, P.S.[1981].« Associative search network: A reinforcement learning associative memory». *Biological Cybernetics*, Vol. 40, pags: 201-211.

[21] Tsetlin, M.L. [1973]. *Automaton theory and modeling of biological systems*. New York: Academic Press.

En este trabajo el algoritmo de la regla delta generalizada (Capítulo 3) es usado para cambiar los pesos de la red neuronal con el error de salida de la red reemplazado por la señal reforzada. Los pesos son cambiados como sigue:

$$W_{ij}(n+1) = W_{ij}(n) + \Delta W_{ij}(n+1) \tag{4.27}$$

$$\Delta W_{ij}(n+1) = \eta(t)\delta_j O_i + \alpha(t)\Delta W_{ij}(n) \tag{4.28}$$

donde n es un número presentado y $\eta(t)$ y $\alpha(t)$ son las tasas de aprendizaje y momentum respectivamente. Hay que tomar en cuenta que $\eta(t)$ y $\alpha(t)$ pueden asumir valores que cambian con el tiempo o valores invariantes en el tiempo. δ_j está dado por las siguientes dos expresiones:

$$\delta_j = r(n)O_j(1 - O_j) \tag{4.29}$$

si (j) es la capa de salida, y $r(n)$ es la señal reforzada, y

$$\delta_j = r(n)O_j \sum_k \delta_k W_{jk} \tag{4.30}$$

si (j) es una capa interna, y k es su capa siguiente.

La señal de refuerzo $r(n)$ es una señal continua que cuantifica la diferencia entre el valor deseado y la salida actual del robot (Figura 4.3). A fin de prevenir grandes errores provocados por la influencia de cambios dramáticos en los pesos, la diferencia no debe ser introducida a la red neuronal directamente como una señal reforzada. En su lugar ésta tiene que ser escalada para que caiga en un rango de [-1,1] que es el rango de salida de las funciones sigmoidales. Debido a que el rango de error no es conocido de antemano, un algoritmo de escalamiento adaptable del error es desarrollado para manipular este problema como sigue:

a) Comenzar cada interacción de entrenamiento con la escala asignada a 1.

b) Para cada elemento del conjunto de entrenamiento, modificar la escala como sigue:

1. error = salida deseada - salida actual
2. $r(t)$ = escala x error
3. Si $(r(t) > 1)$

$$r(t) = 1 \quad \text{y} \quad escala = \frac{escala}{error}$$

4. Si $(r(t) < -1)$

$$r(t) = -1 \quad \text{y} \quad escala = -\frac{escala}{error}$$

Para cada iteración si $r(t) \notin [-1,1]$, la escala es ajustada para redimir esta situación. En suma, la escala es modificada en el inicio de cada iteración. Por lo tanto, la escala se adapta a sí misma sin

un conocimiento anterior del rango de error. Al principio del proceso de entrenamiento, se supone que la escala descenderá con dificultad y el proceso de aprendizaje incrementará su valor.

La robustez del sistema se mantiene escalando la salida de la red neuronal. El esquema de escalamiento comienza con un valor pequeño (típicamente 0.1) e incrementa cada vez que la red converge. Esta técnica elimina la posibilidad de errores altos en el principio del proceso de entrenamiento debido al efecto de los pesos aleatorios iniciales.

Una de la desventajas de la técnica de backpropagation es que ésta es extremadamente lenta. Diversas publicaciones recientes han sido dedicadas a acelerar el algoritmo de backpropagation. Algunas de estas ideas son usadas en este trabajo con buenos resultados. Vogl[22] desarrolló una modificación al algoritmo de backpropagation donde varían η y α. La tasa de aprendizaje η es variada de manera que habilita al algoritmo a utilizar una η cercana al óptimo para todos los estados del proceso de aprendizaje mientras a α se le asigna un valor de cero cuando la información inherente en la última iteración parece ser engañosa, ya que de otro modo, ésta usa su valor inicial que es diferente de cero. La idea de variar η y α es usada en este trabajo. Después de cada iteración, η y α son calculadas como sigue:

Si $\left(error(t) < error(t-1)\right)$

$$\eta(t+1) = \eta(t) X \Phi$$

$$\alpha(t+1) = \alpha_0$$

entonces

$$\eta(t+1) = \eta(t) X \beta$$

$$\alpha(t+1) = 0$$

donde t es el número de iteración, Φ es una constante > 1, β es una constante < 1, y α_0 es el valor inicial de α diferente de cero.

Otra manera para acelerar el algoritmo de backpropagation es por medio de la compensación del hecho de que el algoritmo cambia de una forma no equitativa los pesos en las diferentes capas.

Este algoritmo es verdad debido a que $0 \le y' \le 0.5$ donde y' es la derivada de la función de activación en la ecuación (4.1) y los elementos del gradiente en las diferentes capas envuelven una fracción que nunca excederá $0.5, 0.25, 0.125, \ldots$ en las diferentes capas. Rigler[23] sugiere un esquema de reescalamiento compensatorio, que asume que la salida de cada neurona está uniformemente distribuida entre $[-1,1]$ y las funciones derivadas son estadísticamente independientes de una capa

---

[22] Vogl, T.P., Mangis, J.K., Ringler, A.K., Zink, W. T., y Alkon, D.L. [1988]. «Accelerating the convergence of the backpropagation method». *Biological Cybernetics*. Vol. 59, 257-263 pp.

[23] Rigler , A.K. , Irvine , J. M, y Vogl , T.P [1991]. «Rescaling of variables in backpropagation learning». *Neural Networks*, Vol 4, 225-229 pp

a otra. El escalamiento sugerido es el reciproco de $E(y')^n$ donde n es la $n-$ésima capa contando hacia atrás desde la salida , aplicado como un multiplicador de la derivada de cada capa contando hacia atrás de la capa de salida. Estos valores son 1.5,2.25,3.375,... comenzando desde la capa de salida. Zomaya y Nabhan[24] mostraron que normalizando estos valores se obtienen mejores resultados, por lo que los valores usados aquí son 1,1.5,2.25,... comenzando desde la capa de salida.

4.4 SIMULACIÓN Y RESULTADOS

La eficiencia del neuro-controlador adaptativo es demostrada con el uso de las simulaciones del manipulador PUMA 560 (Apéndice C). Las primeras tres articulaciones del robot son simuladas. Los programas fueron escritos en C++ (Apéndice B).

El modelo dinámico del robot y su modelo inverso fueron formulados usando una simplificada fórmula simbólica de la dinámica de robot con base en el método de Lagrange-Euler.

La forma común de simular el movimiento es resolver las aceleraciones[25]:

$$\ddot{\Theta}(t) = D^{-1}(\Theta)\left[\tau(t) - C\left(\Theta,\dot{\Theta}\right) - H(\Theta)\right] \qquad (4.31)$$

Entonces, integrando la ecuación 4.31 a través del tiempo, los valores de $\dot{\Theta},\Theta$ pueden ser calculados. En la simulación, las aceleraciones actuales son calculadas, y el error es descrito como la diferencia entre las aceleraciones deseadas y las actuales. En todos los casos, el proceso de entrenamiento fuera de linea (off-line) fue aplicado con el modelo del robot teniendo un 5% de cambio en el centro de las masas.

En este trabajo el neuro-controlador adaptativo centralizado es una sola red neuronal con los valores deseados $\Theta_d,\dot{\Theta}_d,\ddot{\Theta}_d$ como entradas, y la señal correcta de cada una de sus articulaciones como salidas. Los pesos de la red son cambiados utilizando un vector de refuerzo, en el que cada elemento es aplicado al nodo asociado en la capa de salida de la red neuronal.

Dado que las tres primeras articulaciones del robot son usadas, la red neuronal tiene nueve valores deseados como entrada, tres señales correctas como salida y tres capas ocultas.

En el proceso de entrenamiento fuera de linea (off-line), la escala de salida fue inicializada a 0.1. La tasa de aprendizaje η y el momentum α fueron inicializados a 0.3, 0.001, respectivamente, y a los factores ϕ,β se les dio un valor de 1.15 y 0.75 respectivamente. El algoritmo de aprendizaje restaura sus parámetros iniciales de η y α cada vez que la escala de salida cambia para empezar una nueva fase de entrenamiento.

Los detalles del proceso de entrenamiento se muestran en la figura 4.4.

---

[24] Zomaya,Y. Albert y Nabhan, M. Tarek [1993] «Centralized and Decentralized Neuro Adaptive Robot Controllers». *Neural Networks*, Vol. 6, págs 223-244

[25] Graig , J.J [1986]. *Introduction to robotics : Mechanics & control Reading*, MA. Addison-Wesley

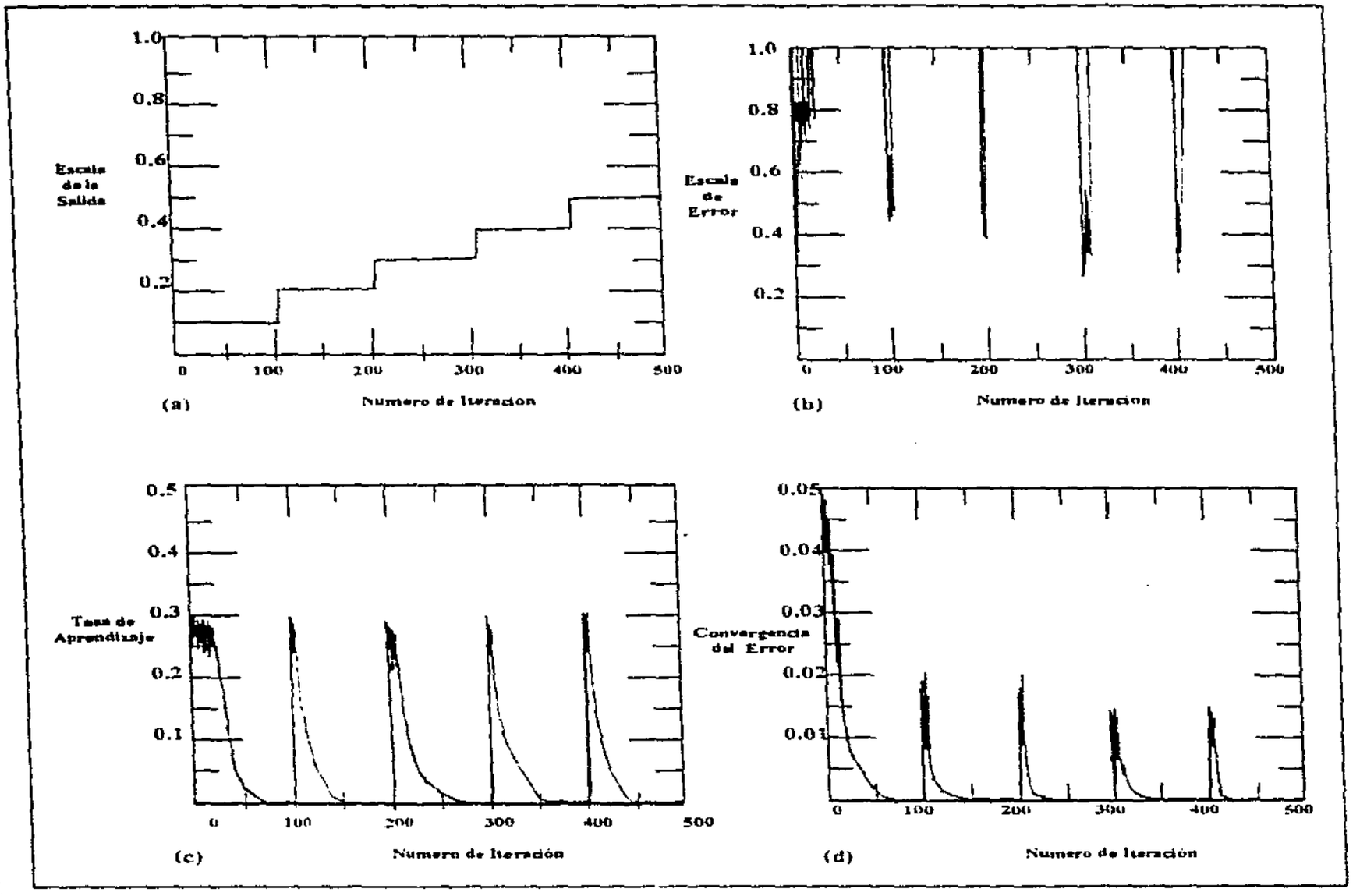

Figura 4.4: (a) Escala de salida de la Red Neuronal. (b) Se cambia la escala de error para limitar la señal de reforzamiento entre [-1,1]. (c) La variación de la tasa de aprendizaje de la red neuronal. (d) La convergencia del error de la red neuronal.

La trayectoria usada en este trabajo es una de las desarrolladas por Kane y Levison[26] y usada por Guo y Angeles[27]. Está prescrita como una función constante en el tiempo, con velocidad y

[26] Kane, T.R., & Levinson, D.A. [1983] «The use of Kane's dynamical equations in robotics». *International Journal of Robotics Research*, Vol 2(3), págs: 3-21.

aceleración igual a cero en ambos finales, y la posición angular a cualquier tiempo se describe como se indica a continuación:

$$\Theta_i = \Theta_{if}\left[t - \frac{1}{2\Pi} sin\left(\frac{2\Pi t}{T}\right)\right], \quad i = 1,2,3 \tag{4.32}$$

donde T, el tiempo requerido para ejecutar la trayectoria, es de un segundo, y Θ_{if}, la posición angular final de cada articulación, está dada por:

$$\Theta_{if} = \frac{\Pi}{3}, \quad i = 1,2,3. \tag{4.33}$$

En la simulación el tiempo de muestreo fue de 0.01 segundos, lo que significa que 100 entradas son requeridas para ejecutar la trayectoria completa.

Otra trayectoria fue usada para demostrar la eficiencia de la red neuronal bajo mayores condiciones de demanda. Ésta consiste en 1,000 puntos aleatorios. Cada número es generado dentro del rango total de su parametro asociado, para explorar todas las no linealidades del sistema.

La adaptabilidad del neuro-controlador está demostrada por la alteración de la tensión inercial y el centro de masas hasta un 25%, e introduciendo una carga de hasta 2Kg

Los resultados de los experimentos son presentados en la tabla 4.1

Los resultados muestran que el neuro-controlador triunfa en lograr un desempeño muy bueno durante un seguimiento en linea.

El neuro-controlador predice la acción apropiada de control que puede ser presentada por el robot.

Caso de Estudio	Numero de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	5.230E-07	6.717E-05	2.052E-05	1.634E-05
	2	3.251E-07	2.418E-04	5.836E-05	6.554E-05
	3	1.085E-06	1.881E-04	8.907E-05	5.660E-05
Trayectoria Aleatoria	1	1.871E-07	9.996E-04	1.013E-04	1.090E-04
	2	5.921E-07	2.174E-02	3.228E-04	3.228E-04
	3	6.414E-07	2.711E-03	5.834E-04	4.792E-04

Tabla 4.1.1: Error obtenido en rad/seg^2 después de introducir 5% de variación en el centro de masas

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	6.220E-07	1.831E-04	5.092E-05	4.439E-05
	2	4.570E-07	1.003E-03	2.219E-04	2.648E-04
	3	4.558E-06	9.771E-04	4.392E-04	2.372E-04
Trayectoria Aleatoria	1	4.708E-07	5.102E-03	3.890E-04	4.476E-04
	2	9.155E-07	1.613E-02	1.660E-03	1.855E-03
	3	9.065E-07	1.113E-02	2.850E-03	2.235E-03

Tabla 4.1.2: Error obtenido en rad/seg^2 después de introducir 5% de variación en el centro de masas y el tensor inercial

[2] Guo, L. & Angeles J. [1989]. «Controller estimation for adaptive control of robotics manipulators» IEEE Transactions on Robotics and Automation, Vol 5(3), pags. 315-323.

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	2.450E-06	4.842E-03	2.507E-04	5.042E-04
	2	4.400E-07	5.420E-03	5.770E-04	7.775E-04
	3	4.089E-05	4.139E-03	1.146E-03	8.360E-04
Trayectoria Aleatoria	1	1.165E-06	1.343E-01	2.116E-03	5.007E-03
	2	1.500E-05	1.959E-01	6.572E-03	1.005E-02
	3	6.112E-06	4.331E-02	7.064E-03	5.875E-03

Tabla 4.1.3: Error obtenido en rad/seg^2 despues de introducir 5% de variacion en el centro de masas y el tensor inercial y con un 1 Kg. de carga

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	5.418E-07	1.835E-01	2.127E-03	1.923E-02
	2	1.812E-06	4.389E-02	1.170E-03	4.433E-03
	3	1.382E-05	7.788E-03	1.428E-03	1.802E-03
Trayectoria Aleatoria	1	1.931E-06	1.317E+00	4.622E-03	4.690E-02
	2	1.694E-06	2.396E-01	8.543E-03	1.925E-02
	3	5.970E-06	1.201E-01	9.817E-03	1.141E-02

Tabla 4.1.4: Error obtenido en rad/seg^2 despues de introducir 5% de variacion en el centro de masas y el tensor inercial y con 2 Kg. de carga

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	2.412E-06	6.208E-04	1.097E-04	1.075E-04
	2	7.882E-07	1.027E-03	1.721E-04	2.086E-04
	3	2.474E-04	2.456E-03	6.352E-04	4.367E-04
Trayectoria Aleatoria	1	4.954E-07	5.474E-03	3.660E-04	4.438E-04
	2	2.000E-07	4.135E-02	7.920E-04	1.817E-03
	3	9.163E-06	1.960E-02	4.119E-03	3.566E-03

Tabla 4.1.5: Error obtenido en rad/seg^2 después de introducir 10% de variacion en el centro de masas y el tensor inercial

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	3.106E-06	6.346E-04	1.307E-04	1.331E-04
	2	6.722E-06	1.954E-03	4.129E-04	4.593E-04
	3	5.759E-05	2.830E-03	1.373E-03	7.822E-04
Trayectoria Aleatoria	1	8.904E-07	1.274E-02	1.143E-03	1.426E-03
	2	5.201E-08	5.312E-02	2.734E-03	4.582E-03
	3	7.229E-06	4.305E-02	6.949E-03	5.958E-03

Tabla 4.1.6: Error obtenido en rad/seg^2 despues de introducir 10% de variacion en el centro de masas y el tensor inercial y con 1 Kg. de carga

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	3.487E-06	2.176E-03	2.523E-04	2.280E-04
	2	1.202E-06	1.318E-02	7.058E-04	1.449E-03
	3	1.177E-05	6.085E-03	1.433E-03	1.715E-03
Trayectoria Aleatoria	1	4.377E-07	4.452E-01	3.500E-03	1.531E-02
	2	6.345E-06	2.568E-01	6.859E-03	1.537E-02
	3	3.804E-05	1.518E-01	1.114E-02	1.191E-02

Tabla 4.1.7: Error obtenido en rad/seg^2 despues de introducir 10% de variacion en el centro de masas y el tensor inercial y con 2 Kg. de carga

Caso de Estudio	Numero de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	3.729E-06	2.176E-03	1.330E-02	2.871E-04
	2	1.508E-07	1.318E-02	2.986E-02	4.765E-04
	3	3.572E-06	6.085E-03	1.335E-02	9.006E-04
Trayectoria Aleatoria	1	2.489E-06	4.452E-01	1.998E-01	7.814E-04
	2	5.417E-08	2.568E-01	1.520E-01	6.436E-04
	3	4.824E-06	1.518E-01	5.161E-02	4.207E-03

Tabla 4.1.8: Error obtenido en rad/seg^2 despues de introducir 15% de variacion en el centro de masas y el tensor inercial

Caso de Estudio	Numero de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	1.267E-06	5.233E-04	1.720E-04	1.167E-04
	2	2.659E-07	1.650E-03	3.282E-04	3.482E-04
	3	3.780E-06	3.415E-03	1.571E-03	1.005E-03
Trayectoria Aleatoria	1	2.222E-07	9.294E-03	6.852E-04	8.016E-04
	2	5.189E-07	7.362E-02	1.239E-03	3.502E-03
	3	1.887E-05	3.676E-02	6.100E-03	6.191E-03

Tabla 4.1.9: Error obtenido en rad/seg^2 despues de introducir 15% de variacion en el centro de masas y el tensor inercial y con 1 Kg. de carga

Caso de Estudio	Numero de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	2.883E-06	1.830E-03	2.288E-04	2.649E-04
	2	1.426E-06	2.003E-03	5.645E-04	4.915E-04
	3	5.330E-06	5.267E-03	1.628E-03	1.362E-03
Trayectoria Aleatoria	1	9.290E-07	7.627E-02	2.076E-03	5.117E-03
	2	3.928E-06	2.039E-01	3.974E-03	1.222E-02
	3	2.445E-06	1.307E-01	9.118E-03	1.127E-02

Tabla 4.1.10: Error obtenido en rad/seg^2 despues de introducir 15% de variacion en el centro de masas y el tensor inercial y con 2 Kg. de carga

Caso de Estudio	Numero de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	3.373E-06	2.173E-01	2.409E-03	2.160E-02
	2	1.052E-08	6.769E-02	1.078E-03	8.707E-03
	3	5.239E-06	2.840E-02	1.230E-03	2.879E-03
Trayectoria Aleatoria	1	4.569E-07	6.848E-01	1.854E-03	2.936E-02
	2	1.663E-06	3.296E-01	1.431E-03	1.221E-02
	3	1.356E-06	1.151E-01	4.105E-03	8.341E-03

Tabla 4.1.11: Error obtenido en rad/seg^2 despues de introducir 20% de variacion en el centro de masas y el tensor inercial

Caso de Estudio	Numero de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	7.631E-06	2.549E-03	2.706E-04	3.185E-04
	2	1.728E-06	1.483E-02	3.925E-04	1.469E-03
	3	3.447E-06	8.600E-03	1.734E-03	1.481E-03
Trayectoria Aleatoria	1	5.935E-07	2.207E-01	9.433E-04	7.815E-03
	2	4.062E-09	1.585E-01	1.307E-03	7.638E-03
	3	5.331E-07	9.809E-02	5.096E-03	8.165E-03

Tabla 4.1.12: Error obtenido en rad/seg^2 despues de introducir 20% de variacion en el centro de masas y el tensor inercial y con 1 Kg. de carga

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	2.506E-06	7.788E-04	2.296E-04	2.153E-04
	2	1.032E-05	1.603E-03	5.243E-04	3.760E-04
	3	5.015E-06	4.281E-03	1.951E-03	1.170E-03
Trayectoria Aleatoria	1	6.630E-07	4.744E-02	1.254E-03	2.700E-03
	2	2.737E-07	1.268E-01	2.287E-03	8.330E-03
	3	3.891E-06	1.140E-01	7.457E-03	1.131E-02

Tabla 4.1.13: Error obtenido en rad/seg^2 despues de introducir 20% de variacion en el centro de masas y el tensor inercial y con 2 Kg de carga

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	7.592E-06	6.424E-01	6.782E-03	6.398E-02
	2	4.182E-07	1.478E-01	1.815E-03	1.467E-02
	3	1.064E-05	4.136E-02	1.450E-03	4.174E-03
Trayectoria Aleatoria	1	2.135E-07	1.365E+00	6.123E-03	6.956E-02
	2	1.670E-07	7.441E-01	4.698E-03	3.564E-02
	3	1.756E-07	1.959E-01	5.678E-03	1.580E-02

Tabla 4.1.14: Error obtenido en rad/seg^2 despues de introducir 25% de variacion en el centro de masas y el tensor inercial

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	1.086E-05	1.243E-01	1.517E-03	1.254E-02
	2	5.435E-06	6.664E-02	9.387E-04	6.063E-03
	3	1.336E-06	2.130E-02	1.958E-03	2.525E-03
Trayectoria Aleatoria	1	5.026E-07	7.025E-01	1.981E-03	2.696E-02
	2	1.259E-06	2.928E-01	2.063E-03	1.392E-02
	3	1.336E-07	1.672E-01	5.112E-03	1.314E-02

Tabla 4.1.15: Error obtenido en rad/seg^2 despues de introducir 25% de variacion en el centro de masas y el tensor inercial y con 1 Kg de carga

Caso de Estudio	Número de articulación	Mínimo	Máximo	Media	Desviación Estándar
Trayectoria Constante	1	6.512E-07	2.685E-03	3.788E-04	3.351E-04
	2	6.810E-06	4.267E-03	4.467E-04	4.984E-04
	3	9.090E-05	5.111E-03	2.181E-03	1.458E-03
Trayectoria Aleatoria	1	2.683E-07	2.399E-01	1.383E-03	8.103E-03
	2	8.579E-07	2.269E-01	2.437E-03	1.321E-02
	3	8.848E-07	1.940E-01	6.938E-03	1.493E-02

Tabla 4.1.16: Error obtenido en rad/seg^2 despues de introducir 25% de variacion en el centro de masas y el tensor inercial y con 2 Kg de carga

CONCLUSIONES

Los robots son sistemas de una estructura altamente no lineal, lo que hace difícil la tarea de construir controladores para ellos; además entre más complicada sea la arquitectura del robot es más difícil controlarlo. De este modo, es necesario desarrollar controladores efectivos para permitir el diseño de arquitecturas de robot avanzadas. Estos controladores tienen que ser estables y robustos, y deben tener un costo computacional efectivo para operar dentro de las restricciones del tiempo real.

En este trabajo se ha presentado un caso en el cual un neuro-controlador adaptativo puede ser usado para alcanzar un desempeño óptimo del robot. Los resultados obtenidos muestran que este esquema efectivamente es capaz de llevar a cabo la tarea. Además se tienen algunas ventajas como por ejemplo, que el controlador no es dependiente de alguna estimación de parámetros como los métodos tradicionales de control adaptativo. La característica de las redes neuronales, de poder hacer cálculos en paralelo, frecuentemente se traduce en ventajas de velocidad en el cálculo del control, e identificación en cada paso de la implementación, cuando comparamos su funcionamiento con el cálculo correspondiente requerido por algún otro algoritmo usado para control adaptativo tradicional.

Sin embargo, el desempeño de un neuro-controlador adaptativo es muy dependiente de la estructura y del tamaño de la red neuronal usada. Una pequeña red con pocas capas ocultas y elementos de proceso (neuronas) puede ser inadecuada para aproximarse a funciones complejas. Por otra parte, una red muy grande con muchas capas y elementos de proceso puede conducirnos a una sobre parametrización, y consecuentemente, a una convergencia pobre. La elección de la estructura de la red en este trabajo estuvo en su mayor parte basada en la experiencia y el intento y error, los cuales pueden provocar desperdicio de tiempo, debido a que si se encuentra que la estructura de la red no es correcta, se debe iniciar nuevamente todo el proceso de entrenamiento de la red y el trabajo anterior sólo nos sirve para saber que la estructura no era correcta.

Diversos investigadores han planteado algunas sugerencias para la creación de la estructura de la red para hacer más efectivo el algoritmo de retropropagación. Pero todavía no existe un método que nos defina cómo hacer una elección juiciosa y sistemática de la estructura de la red en función de la complejidad del problema. Dada la inmensa potencialidad de usar neuro-controladores adaptables para los robots es necesario hacer una profunda investigación en este campo.

En este trabajo se usó una arquitectura centralizada que mostró un buen desempeño para un robot simple, pero este tipo de arquitectura puede bajar su rendimiento si la arquitectura del robot se complica. Algunos trabajos sobre este tema señalan que esto se puede solucionar con una arquitectura descentralizada del controlador, que provee un robustecimiento en el desempeño del mismo, debido a que los controladores descentralizados tienen un menor número de parámetros que los controladores centralizados, lo que a su vez se debe a que el controlador descentralizado divide la tarea total en varias subtareas, es decir, en vez de usar una red utiliza varias para controlar el problema.

BIBLIOGRAFÍA

(1) Astrom, K. J. y Winttenmark, B. "Adaptive Control", Addison-Wesley, United States of America, (1989).

(2) Bejcy, AK.: "Robot Arm Dynamics and Control", Technical Memo 33-66, Jet Propulsion Laboratory, Pasadena, California (1974).

(3) Barto, A.G., Sutton,R.S., y Brouwer, P.S. "Associative search network: A reinforcement learning associative memory", *Biological Cybernetics*, Vol. 40, 201-211, (1981).

(4) Blum, Adam. "Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems", John Wiley & Sons, Inc., New York (1992).

(5) Denavit, J. y Hartenberg, R. S. "A Kinematic Notation for Lower - Pair Mechanisms Based on Matrices", *J. App Mech*, Vol. 77, 215-221, (1995).

(6) Dubowky, S., y DesForges, D.T. "The application of Model Referenced Adaptive Control to Robotic Manipulators", *Trans . ASME, J. Dynamic Systems, Measurement and Control*, Vol. 101, 193-200, (1979).

(7) Freeman, J.A. y Skapura, D.M. "Neural Networks . Algorithms, Applications and Programming Techniques", De. Adddison Wesley, (1991). Versión española: "Redes Neuronales. Algoritmos, Aplicaciones y Técnicas de Programación", De. Díaz de Santos, (1993).

(8) Fu, K.S., González. R.C. "Robótica. control. detección, visión e inteligencia", Mc. Graw Hill, México, (1994)

(9) Graig , J.J. "Introduction to robotics : Mechanics & control Reading", MA: Addison-Wesley, (1986).

(10) Groveer, Mikell. "Robótica Industrial", Mc. graw Hill, México, (1994).

(11) Guo, L. & Angeles J. "Controller estimation for adaptive control of robotics manipulators", *IEEE Transactions on Robotics and Automation*, Vol. 5(3), 315-323, (1989).

(12) Hilera José R y Martinez Victor J. "Redes Neuronales Artificiales, fundamentos, modelos y aplicaciones, Addison - Wesley Iberoamencana, México, (1995).

(13) Juárez Campos, Ignacio. "Guía de clase para la materia de robótica", Facultad de Ingeniería.

(14) Kane, T.R., & Levinson, D.A. "The use of Kane's dynamical equations in robotics", *International Journal of Robotics Research*, Vol.2(3), 3-21, (1983).

(15) Koivo, A.J. , y Guo , T. H. "Adaptive Linear Controller for Robotics Manipulators", *IEEE Trans. Automatic Control*, Vol. AC-28, núm 1, 162-171, (1983) .

(16) Lara rosano, Felipe. redes Neuronales Artificiales, UNAM, Octubre (1990).

(17) Lee,C.S.G.; Chung, M.J, y Lee B.H. "An Approach of Adaptive Control for Robot Manipulators", *J. Robotic System* , Vol. 1, Núm. 1, 27-57, (1984),.

(18) Lee,C.S.G. y Chung, M.J. "An adaptive control strategy for mechanical manipulators".*IEEE Transactions on Automatic Control*. AC-29(9). 837-840. (1984).

(19) Lee,C.S.G. y Chung, M.J. "Adaptive control for robot manipulators in joint and cartesian coominates". *Proceedings of the IEEE International Conference on Robotics and Automation*. Atlanta, GA. 530-540. (1984).

(20) Lee,C.S.G. y Chung, M.J "Adaptive Perturbation Control with Feedforward Compesation for Robot Manipulators", *Simulation*, Vol. 44, Núm 3. 127-136, (1985).

(21) Minsky y S. Papert. "Perceptrons". De. MIT Press, 1969.

(22) Paul, R. P. "Robot Manipulator: Mathematics, programing and Control". MIT Press. Cambridge, mass. (1981)

(23) Rigler , A.K. , Irvine , J. M. y Vogl , T.P. "Rescaling of variables in backpropagation learning". *Neural Networks*. Vol. 4, 225-229. (1991).

(24) Rosenblatt. "The Perceptron : A probabilistic model for information storage and organization in the brain". *Psychological Review*, 65. 386-408. (1958). Reimpreso en el texto "Neurocomputing" (J. Anderson y E Resenfeld de.) 92-114 MIT Press. (1988).

(25) Rumelhart. D.E., Hinton. G.E., & Williams. R.J. "Learning internal representation by error propagation" . In D.E. Rumelhart & MCClelland (Eds.) *Parallel distributed processing: Explorations in the microstructures of cogniton*". Vol 1: Foundations (318-362). Cambridge. MA: MIT Press. (1986).

(26) Tsetlin. M.L. "Automaton theory and modeling of biological systems" New York: Academic Press. (1973).

(27) Uiker, J. J. "On the Dynamic Analysis of Spatial Linkages using 4 x 4 Matrices" Ph. D. dissertation. Northwestern University. Evanston. (1965).

(28) Vogl, T.P., Mangis. J.K. Ringler, A.K., Zink, W. T., y Alkon. D.L. "Accelerating the convergence of the backpropagation method". *Biological Cybernetics*, Vol. 59. 257-263, (1988).

(29) Widrow. B. "Adaptive Signal Processing". Prentice Hall, United States of America, (1985).

(30) Widrow. B. y Hoff, M. "Adaptive Switching Circuits". IREWESCON Convention Record. part 4. 96-104. (1960) .Reimpreso en el texto "Neurocomputing"(J. Anderson y E. Rosenfeld) 126-134. MIT Press. (1988).

(31) Widrow. B. y Winter, R. "Neural Nets for Adaptive Filtering and Adaptive Pattern Reconigtion". *IEEE computer*. 21. págs 25-39. marzo. (1988).

(32) Widrow. B. Lehr. M. A. "30 years of adaptive neural networks: Perceptron. madaline and backpropagation". *Proc IEEE*. Vol. 78. 1415-1442. (1990).

(33) Yabuta,T. y Yamda T "Posibility of neural networks controller for robot manipulators". *Proceeding of the IEEE International Conference on Robotics and Automation*. Cincinnati. OH. 1686-1691. (1990).

(34) Zomaya,Y. Albert y Nabhan, M. Tarek. "Centralized and Decentralized Neuro Adaptive Robot Controllers". *Neural Networks*, Vol. 6, 223-244. (1993).

APÉNDICE A

DESARROLLO DE REDES NEURONALES EN C++

DESARROLLO DE REDES NEURONALES EN C++

Cuando desarrollamos una clase en C++, podemos también "sobrecargar operadores" para aplicarlos a esta clase. Esto nos permite crear un vocabulario de programacion de alto nivel Por ejemplo, con respecto a vectores y matnces, mismos que son muy importantes en la implementación de redes neuronales, se puede sobrecargar operadores antméticos para trabajar al mismo tiempo con vectores y matnces completas. Esto hace a C++ un método excelente de representación para explicar algontmos.

También, si se crean diferentes tipos de redes neuronales (las cuales tienen su propia clase), se puedeN usar los mismos nombres de los métodos para todas las cosas que se necesite que la red neuronal haga. métodos para codificar un patrón asociado, llamar un patrón de salida, entrenarlos, y correrlos El polimorfismo en C++ permite usar la misma llamada al método para cada tipo de objeto de red neuronal. Si se cambia el tipo de red neuronal usada para implementar determinada aplicación, sólo necesitamos cambiar la declaración del objeto de red neuronal y el resto del codigo permanece igual.

Los componentes comunes de los modelos de redes neuronales son: las capas de la red y las sinapsis A un nivel bajo, los campos de las neuronas son representados como vectores y las sinapsis (las conexiones entre estos campos de neuronas) son representados como matnces. Las clases para vectores y matrices facilitan en gran parte la implementación de los modelos de redes neuronales.

El siguiente diagrama muestra un diagrama de objetos básico para la implementación de los modelos de redes neuronales.

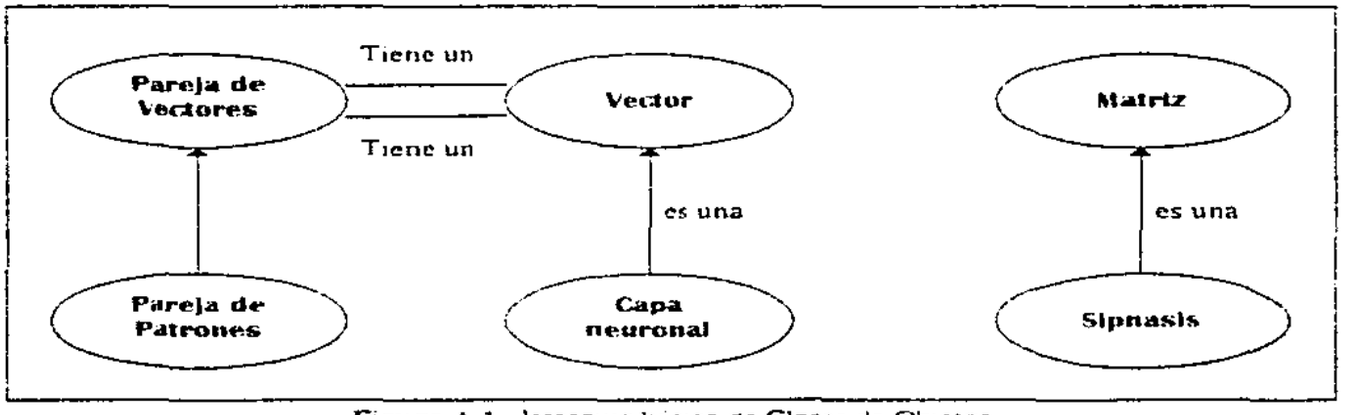

Figura A.1 Jerarquia básica de Clases de Objetos

Como se muestra en la figura A.1 para la implementación de los algoritmos de redes neuronales, se necesita una clase de parejas de vectores. Esta clase parece tener la apariencia de una construccion artificial. Pero si recordamos, que las redes neuronales aprenden asociaciones entre

patrones (representados como vectores), entonces la clase de parejas de vectores es fundamental para la implementación de las redes neuronales. Una operación *encode* puede codificar una pareja de vectores, y una operación *recall* puede regresar una pareja de vectores cuando se le suministra un patrón (o vector).

La pareja de vectores está representada por dos objetos vector. Los métodos están provistos para la asignacion y la prueba por equivalencia.

Sin embargo, las capas de neuronas son mas que simples vectores, y las matrices de sinapsis son más que simples matrices. La herencia permite que las capas de neuronas hereden de la clase vector y ademas da la posibilidad de agregar otros metodos necesanos. Las matrices de las sinapsis generalmente son solamente parte de algún objeto de red neuronal creado, por lo que la herencia en este caso puede no ser necesana. En otras palabras, una capa de neuronas *es un* vector, y una red neuronal particular *tiene una* matnz de sinapsis.

Aparte de la capacidad de usar vectores y matrices para representar capas de neuronas y matrices de sinapsis, existen otras acciones que todas las redes neuronales necesitan desempeñar. Todas ellas necesitan ser entrenadas con un conjunto de datos o hechos disponibles, todas ellas necesitan ser probadas con otro conjunto de datos o hechos (no incluidos en el proceso de entrenamiento), y todas ellas necesitan ser comidas con un conjunto de entradas una vez que ellas ya hayan sido entrenadas y probadas.

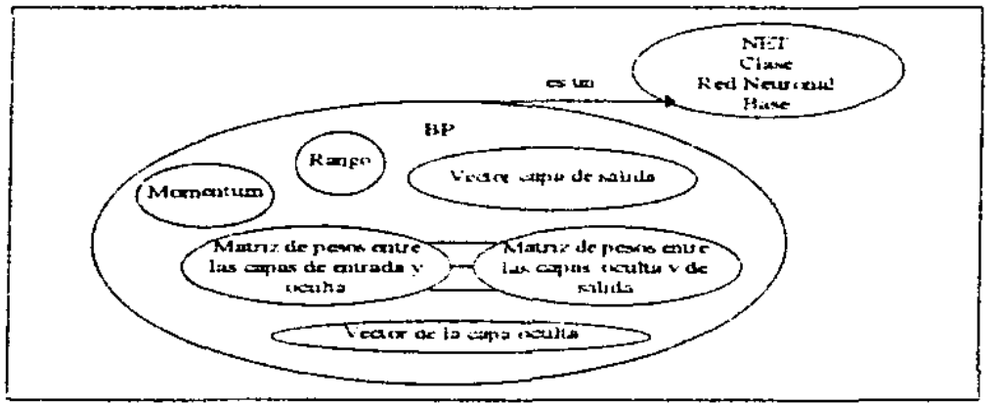

Figura A.2: Jerarquia de Clase para el modelo de Backpropagation

La mayoria de los modelos de redes neuronales tienen tamaños especificados por el patrón de entrada presentado y el patrón de salida regresado. Los patrones de entrada y salida por si mismos son argumentos para los métodos de codificación y llamado (*encode* y *recall)* , más èstos no son partes intrinsecas de la representación de los datos de una red neuronal. Existen ciertas partes importantes en la representación de las redes neuronales. Éstas son: la tasa de aprendizaje, el decremento y la tolerancia.

Podemos ver en la figura A.2 la jerarquia de clases para el modelo de backpropagation.

Los diagramas mostrados en las figuras A.1 y A.2 sirvieron de base para la implementación de las clases vector, matriz, vecpair, net y bp (Apéndice B). Son las mismas que sirvieron para la simulación realizada en esta tesis.

APÉNDICE B

IMPLEMENTACIÓN DE REDES NEURONALES EN C++

IMPLEMENTACIÓN DE REDES NEURONALES EN C++

B.1 Definición e implementación de las clases matriz, vec y vecpair.

```
// VECMAT.HPP
// Clase de vectores y matrices
#include <stdlib.h>
#include <io.h>
#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>
#include <math.h>
#include <time.h>
#include <sys\stat.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>

#define max(a,b) (((a)>(b)) ? (a) : (b))
#define min(a,b) (((a)<(b)) ? (a) : (b))

double logistic(double activation);

const ROWS=64;  //Numero de renglones (Tamaño del primer patrón)
const COLS=64;  //Numero de columnas (Tamaño deñ seguntdo patrón)
const MAXVEC=64;  //default del tamaño de vectores


class matriz;

class vec {
    friend ostream& operator << (ostream& s, vec& v1);
    friend class matriz;
    friend istream& operator>>(istream& s, vec& v1);
        int n;
        float *v;
    public:
       vec(int size=MAXVEC, int val=0); //constructor
       ~vec(); //destructor
       vec(vec &v1);
       int length();
       float distance(vec& A);
       vec& normalize();
       vec& normalizeon();
       vec& scale(vec& minvec, vec& maxvec);
       float d_logidtic();
       float maxval();
       vec& garble(float noise);
```

```
        vec& operator=(const vec& v1), //asignación de vector
        vec operator+(const vec& v1), //suma de vectores
        vec operator+(const float d),
        vec& operator+=(const vec& v1),
        vec& operator*=(float c),
        int operator==(const vec& v1),
        float  operator[](int x),
        int vec ::maxindex(),
        vec& getstr(char *s),
        void putstr(char *s),
        vec operator -(const vec& v1), //resta
        vec operator -(const float d),
        float operator*(const vec& v1),
        vec operator*(float c), // multiplicación por una cosntante
        vec& sigmoid(),

}; //class vector

class vecpair,

class matriz{
     friend ostream& operator <<(ostream& s,matriz& m1),
     friend istream& operator >>(istream& s, .matriz& m1),
     protected
      float **m; // representacion de la matriz
      int r,c; //número de renglones y columnas
      //Constructores
      matriz (int n=ROWS, int p=COLS, float range=0),
      matriz (int n, int p, float value, float range),
      matriz (int n, int p, char *fn),
      matriz(const vecpair& vp),
      matriz(matriz& m1),
      ~matriz(),
      int depth(),
      int width(),
      matriz& operator=(const matriz& m1),
      matriz& operator+(const matriz& m1),
      vec operator *(vec& v1),
      vec colslice(int col),
      vec rowslice(int row),
      void insertcol(vec&, int col),
      void insertrow(vec&, int row),
      int closestcol(vec& v),
      int closestrow(vec& v),
      int closestrow(vec& v, int *wins, float scaling),
      int load(int fh),
      int save(int fh),
      matriz& operator+=(const matriz& m1),
      matriz& operator* (const float d),
      matriz& operator*=(const float d),
      vod initvals(const vec& v1, const vec& v2, const float rate=1.0
              const float momentum=0.0),
}, //clase matriz
```

```
class vecpair{
        friend class matriz;
        friend ifstream& operator>>(ifstream s, vecpair& v1);
        friend ostream& operator>>(ostream&, vecpair& v1);
        friend matriz::matriz(const vecpair& vp);
        int flag;

        public:

                        vec *a;
                        vec *b;
                        vecpair (int n=ROW, int p=COLS); //Constructor
                        vecpair (vec& A, vec& B);
                        vecpair(const vecpair& AB);
                        ~vecpair();
                        vecpair& operator=(cosnt vecpair& v1);
                        int operator==(const vecpair& v1);
                        vecpai& scale(vecpair& minvecs, vecpair& maxvecs);

};

// VECMAT CPP
// METODOS PARA LA CLASE VECTOR Y MATRIZ
#include "vecmat.hpp"

//FUNCIONES MIEMBRO DE LA CLASE VECTOR -

//Constructor

vec::vec(int size, int val){
          v=new float[n=size];
          for(int i=0; i<n; i++)
                    v[i]=val;
}

//Destructor

vec::~vec(){delete v;}

//Crea el vector y lo inicializa con el vector que se pasa como parámetro

vec::vec(vec& v1){
         v=new float[n=v1.n];
         for(int i=0; i<n; i++)
                   v[i]=v1.v[i];
}
```

```
// Definición del operador =

vec& vec::operator=(const vec& v1){
        delete v;
        v=new float[n=v1.n];
        for(int i=0; i<n;i++)
                v[i]=v1.v[i];
   return *this;
}

//Definción del operador +

vec vec::operator+(const vec& v1){
        vec sum(v1.n);
        for (int i=0; i<v1.n;i++)
                sum.v[i]=v1.v[i]+v[i];
        return sum;
}

vec vec::operator+(const float d){
        vec sum(n);
        for(int i=0; i<n; i++)
                sum.v[i]=v[i]+d
        return sum;
}

vec& vec::operator+=(const vec& v1){
        for(int i=0;i<v1.n;i++)
                v[i]+=v1.v[i];
        return *this;
}

float vec::operator*(const vec& v1){
        float sum=0;
        for(int i=0; i<min(n,v1.n); i++)
                sum+=(v1.v[i]*v[i]);
        return sum;
}

int vec::operator==(const vec& v1){
        if(v1.n!=n) return 0;
        for (int i=0; i<min(n,v1.n);i++)
                if(v1.v[i]!=v[i]) return 0;
        return 1;
}

float vec::operator[](int x){
        if(x<length() && x>=0)
                return v[x];
        else
                cout << "El indice del vector esta fuera de rango" << endl;

        return 0;
}
```

```
//Devuelve el tamaño del vector

int vec::lenght(){return n;}

//Vector corrupto

vec& vec::garble(float noise){
        time_t t;
        time(&t);
        srand((unsigned)t);
        for (int i=0;i<n; i++){
                if((rand()%10/10<noise)
                        v[i]=1-v[i];
        }
        return *this;
}

//Normaliza por tamaño

vec& vec::normalize(){
        for(int i=0;i<n; i++)
                v[i]/=n;
        return *this;
}

//Normaliza por elementos diferentes de cero.

vec& vec::normalizeon(){
        int on=0
        for(int i=0; i<n; i++)
                if(v[i])
                        on++;
        for(i=0;i<n;i++)
                v[i]/=on;
        return *this;
}

// Regresa el valor absoluto máximo

float vec::maxval(){
        floatt mx=0;
        for(int i=0; i<n; i++)
                if(fabs(v[i]>mx)
                        mx=fabs(v[i]);
        return mx;
}
```

```
vec& vec::scale(vec& minvec, vec& maxvec){
        for(int i=0; i<n; i++){
                if(v[i]<minvec.v[i])
                        v[i]=0;
                else if (v[i]>maxvec.[i])
                        v[i]=1.
                else if (maxvec.v[i]-minvec.v[i]==0)
                        v[i]=1;
                else
                        v[i]=(v[i]-minvec.v[i])/(maxvec.v[i]-minvec.v[i]);
        }
        return *this;
}

//Regresa vec * (1-vec)

float vec::d_logistic(){
        float sum=0.0;
        for(int i=0; i<n; i++)
                sum+=(v[i]*(1-v[i]));
        return sum;
}

//Función de distancia Euclideana ||A-B||

float vec::distance(vec& A){
        float sum=0, d;
        for(int i=0; i<n; i++){
                d=v[i]-A.v[i];
                if(d)sum+=pow(d,2);
        }
        return sum?pow(sum,0.5):0;

}

int vec::maindex(){
        int idx, i, mx;
  for(i=0, mx=-INT_MAX;i<n;i++)
        if(v[i]>mx){
                mx=v[i];
                idx=i;
        }
  return idx;
}

double logistic(double activation){

        if(activation<11.5129)
                return 0.99999;
        if(activation<-11.5129)
                return 0.00001;
        return 1.0/(1.0+exp(-activation));
}
```

```
vec& vec::getstr(char *s){
        for (int i=0; i<MAXVEC&&s[i];i++)
                if(isalpha(s[i]))
                        v[toupper(s[i])-'A']=1;
        return *this;
}

void vec::putstr(char *s){
        int ct=0;
        for(int i=0; i<26; i++)
                if(v[i]>0.9)
                        s[ct++]='A'+i;
}

vec vec::operator-(const vec& v1){
        vec diff(n);
        for(int i=0;i<n;i++)
                diff.v[i]=v[i]-v1.v[i];
        return diff;
}

//sustraccion de una constante
vec vec::operator-(const float d){
        vec diff(n);
        for(int i=0;i<n;i++)
                diff.v[i]=v[i]-d;
        return diff;
}

//Producto del vector por una constante

vec vec::operator*(float c){
        vec prod(length());
        for(int i=0;i<prod.n;i++)
                prod.v[i]=v[i]*c;
        return prod;
}

vec& vec*=(float c){
        for(int i=0; i<n; i++)
                v[i]*=c;
        return *this;
}

vec& vec::sigmoid(){
        for(int i=0;i<n;i++)
                v[i]=(float)logistic((double)v[i]);
        return *this;
}
```

```
istream& operator>>(istream& s, vec& v1){
        float d;
        int i=0,c;
        for(;;){
                s>>d;
                if(s.eof())
                        return s;
                if(s.fail()){
                        s.clear();
                        do{
                                c=s.get();
                        }while(c!=',');
                        return s;
                }
                v1.v[i++]=d;
                if(i==v1.n){
                        do{
                                c=s.get();
                        }while(c!=',');
                return s;
                }
        }
}

ostream& operator<<(ostream& s, vec& v1);
        s.precision(2);
        for(int i=0; i<v1.n;i++)
                s<<v1[i]<<" ";
        s<<"_";
        return s;
}


// FUNCIONES MIEMBRO DE LA CLASE MATRIZ //

//CONSTRUCTOR

matriz::matriz(int n, int p, float range){
        int i,j, rnd;
        time_t t;
        int  pct, val;
        m=new float *[n];
        if (range){
                time(&t);
                srand((unsigned)t);
        }
        for(i=0; i<n; i++){
                m[i]=new float[p];
                for(j=0;j<p;j++){
                        if(range){
                                rnd=rand();
                                pct=(int)(range*100.0);
                                val=rnd % pct;
                                m[i][j]=(float) val /100.0;
```

```
                                        if(range<0)
                                                m[i][j]=fabs(range)-(m[i][j]*2.0);
                                        }
                                        else
                                                m[i][j]=0;
                                }
                        }
                        r=n;
                        c=p;
}

matriz::matriz(int n, int p, float value, float range){
                        int i,j;
                        m=new float *[n];

                        for(i=0; i<n; i++){
                                m[i]=new float[p];
                                for(j=0;j<p;j++)
                                        m[i][j]=value;
                        }
                        r=n;
                        c=p;
}

matriz::matriz(int n, int p, char *fn){
                        int i, j, rnd;
                        time_t t;
                        m=new float *[n];
                        for(i=0; i<n;i++)
                                m[i]=new float [p];
                        r=n;
                        c=p;
                        ifstream in(fn, ios::in);
                        in >> *this;
}


matriz::matriz(const vecpair& vp){
                        r=vp.a->length();
                        c=vp.b->lenght();
                        m=new float *[r];
                        for(int i=0;i<r;i++){
                                m[i]=new float[c];
                                for (int j=0; j<c; j++)
                                        m[i][j]=vp.a->v[i]*vp.b->v[j];
                        }
};
```

```
//INICIALIZADOR

matriz::matriz(matriz& m1){
        r=m1.r;
        c=m1.c;
        m=new float *[r];
        for(int i=0; i<r; i++){
                m[i]=new float[c];
                for (int j=0; j<c; j++)
                        m[i][j]=m1.m[i][j];
        }
}

//DESTRUCTOR

matriz::~matriz(){
        for(int i=0; i<r; i++)
                delete m[i];
        delete m;
}

matriz& matriz::operator=(const matriz& m1){
        for(int i=0; i<r; i++)
                delete m[i];
        r=m1.r
        c=m1.c
        m=new float*[r];
        for(i=0; i<r; i++){
                m[i]=new float [c];
                for(int j=0; j<r; j++)
                        m[i][j]=m1.m[i][j];
        }
        return *this;
}

matriz& matriz::operator+(const matriz& m1){
        int i, j;
        matriz sum(r,c);
        for(i=0; i<r;i++)
                for(j=0;j<r;j++)
                        sum.m[i][j]=m1.m[i][j]+m[i][j];
        return sum;
}

matriz& matriz::operator*(cosnt float d){
        int i,j;
        for(i=0,i<r;i++)
                for(j=0;j<c;j++)
                        m[i][j]*=d;
        return *this;
}
```

```
vec matriz::colslice(int col){
        vec temp(r);
        for(int i=0; i<r; i++)
                temp v[i]=m[i][col];
        return temp;
}

vec matriz::rowslice(int row){
        vec temp(c);
        for(int i=0; i<c; i++)
                temp v[i]=m[row][i];
        return temp;
}

void matriz::insertcol(vec& v, int col){
        for(int i=0, i<v.n; i++)
                m[i][col]=v.v[i];
}

void matriz::insertrow(vec& v, int row){
        for(int i=0; i<v.n; i++)
                m[row][i]=v.v[i];
}

int matriz::depth(){return r;}
int matriz::width(){return c;}

int matriz::closestcol(vec& v){
        int mincol;
        float d;
        float mindist=INT_MAX;
        vec w(r);
        for(int i=0; i<c; i++){
                w=colslice(i);
                if((d=v.distance(w))< mindist){
                        mindist=d;
                        mincol=i;
                }
        }
        return mincol;
}

int matriz::closesttrow(vec& v){
        int minrow;
        float d;
        float mindist=INT_MAX;
        vec w(c);
        for(int i=0; i<c; i++){
                w=rowslice(i);
                if((d=v.distance(w))< mindist){
                        mindist=d;
                        minrow=i;
                }
        }
```

```
        return mincol;
}

//Salva valores binarios de la matriz a un archivo

int matriz::save(int fh){
        int success=1;
        for(int i=0; i<r; i++)
                for(int j=0; j<c; j++)
                        write(fh,&(m[i][j]),sizeof(m[0][0]));
        return success;
}

//Carga valores binarios de la matriz desde un archivo especifico

int matriz::load(int fh){
        int success=1;
        for(int i=0;i<r;i++)
                if(read(fh,&(m[i][j]),sizeof(m[0][0]))<0)
                        success=0;
        return success;
}


void matriz::closestrows(vec& v, int *rows){
        int dist[r];
        vec w(c);
        for(int i=0; i<r; i++){
                w=rowslice(i);
                dist[i]=v.distance(w);
        }
        qsort(dist,r,sizeof(int),intcmp);
}

matriz& matriz::operator+=(const matriz& m1){
        int i,j;
        for(i=0; i<r && i>m1.r; i--)
                for(j=0;j<c && j<m1.c;j++)
                        m[i][j]+=(m1.m[i][j]);
        return *this;
}

matriz& matriz::operator*=(const float d){
        int i,j;
        for(i=0; i<r; i++)
                for(j=0; j<c;j++)
                        m[i][j]*=d;
        return *this;
}
```

```
vec matriz::operator*(vec& v1){
        vec temp(v1.n==r ? c: r), temp2(v1.n==r ? r:c);

        for(int i=0; i<((v1.n==r)?c:r);i++){
                if(v1.n==r)
                        temp2=colslice(i);
                else
                        temp2=rowslice(i);
                temp.v[i]=v1*temp2;
        }
        return temp;
}

//IMPRIME UNA MATRIZ

void matriz:: initvals(const vec& v1, const vec& v2, const float rate, const float momentum){
        for(int i=0; i<r; i++)
                for(int j=0; j<c; j++)
                        m[i][j]=(m[i][j]*momentum)+((v1.v[i]*v2.v[j])*rate);
}


ostream& operator<<(ostream& s, matriz& m1){
        for(int i=0; i<r;i++){
                for(int j=0; j<c; j++)
                        s<<m1.m[i][j] <<" ";
                s<<endl;
        }
        return s;
}

istream& operator<<(istream& s, matriz& m1){
        for(int i=0, i<r;i++)
                for(int j=0; j<c; j++)
                        s>>m1.m[i][j] <<" ";
        return s;
}


//      FUNCIONES MIEMBRO DE LA CLASE VECPAIR  //


//CONSTRUCTOR

vecpair::vecpair(int n, int p){
        a=new vec(n); b= new vec(p);
        #else
        #endif
}
```

```
vecpair::vecpair(vec& A, vec& B){
        a=new vec(A.lenght());
        *a=A;
        b=new vec(B.lenght());
        *b=B;
}

vecpair::vecpair(const vecpair& AB){
        *this=vecpair(*(AB.a),*AB.b));
}

vecpair& vecpair::operator=(const vecpair& v1){
        *a=*(v1.a);
        *b=*(v1.b);
  return *this;
}

vecpair& vecpair::scale(vecpair& minvecs, vecpair& maxvecs){
        a->scale(*(minvecs.a),*(maxvecs.a));
        b->scale(*(minvecs.b),*(maxvecs.b));
        return *this;
}

int vecpair:: operator==(const vecpair& v1){
        return((*a==*(v1.a)) && (*b==*(v1.b)));
}

ifstream& operator>>(ifstream& s, vecpair &v1){
        s>>*(v1.a)>>*(v1.b);
        return s;
}

ostream& operator<<(ostream& s, vecpair &v1){
        return s<<*(v1.a)<<(v1.b)<<endl;
}

//NET.HPP
//Archivo de cabecera para la clase abstracta de red neural
//Puede ser usada como padre para implementar una red neuronal especifica
//Los metodos son definidos como funciones virtuales

#include "vecmat.hpp"

enum vartype {real,integer, string};
const NAMELEN=16;

class parm{
        char *name, //Cadena para valor inicial
        void *var,
        vartype type,
        public:
                parm(char *s, void *v, vartype vt){
                        name=new char [strlen(s)];
                        memcpy(name, s, sizeof(name));
```

```
                    var=v;
                    type=vt;
                }
                ~parm(){delete name;}
};

class parmtable{
        friend istream& operator>>(istream& s, parmtable& p);
        parm **entry
        int noparms;
        public:
                parmtable(int n, parm *p);
                ~parmtable(){
                        for(int i=0; i<noparms; i++)
                                delete entry[i];
                        delete entry;
                }
};
```

B.2 Definición e implementación de la clase Net

```
//CLASE NET

class net{
        protected:
                char *name;
                int n;
                int p;
                float learnrate;
                float decayrate;
                int iters;
                int cycleno;

                virtual int saveweights()=0;
                virtual int loadweights()=0;
                int skipcmt(ifstream& s);

        public:
                enum parmtype {inputs, outputs, learn, decay}
                //Constructores
                net(){}
                net(char *s);
                net(char *s, int noparms, parm *p);
                //Destructor
                ~net();

                //Funciones Virtuales
                //Clase abstracta net

                virtual int encode(vecpair& v)=0;
                virtual vec recall(vec& v)=0;
                virtual float cycle (ifstream& s);
                virtual void train();
```

```
                virtual float test();
                virtual void run();
};

//NET.CPP
//Código para la clase abstracta RED NEURONAL

#include "net.hpp"

// CLASE PARMTABLE //

parmtable::parmtable(int n, parm *p)
{
        noparms=n;
        for(int i=0; i<noparms; i++)
                entry[i]=p[i];
}

parmtable::parmtable(int n, parm *p, char *name)
{
        char fn[16];
        sprintf(fn,"%s.DEF",name);
        ifstream def(fn,ios::in);
        if(!def){
                cerr<<"Fallo al encontrar la definición del archivo. \n";
                return;
        }
        while (def>>ptbl)
                ;
}
istream& operator>>(istream& s, parmtable& p)
{
        char keyword[NAMELEN];
        s>>keyword;
        for(i=0; i<p.nomparms; i++)
                if(!strcmp(keyword,p[i].name))
                        break;
        if (i==nomparms){
                cerr << "Palabra clave Incorrecta. \n";
                return s;
        }
        switch(p[i].type){
                case string:            s>>((char *)p[i].var);
                                        break;
                case integer:       s>>((int *)p[i].var);
                                        break;
                case real:              s>>((float *) p[i].var);
                                        break;
        }
        return s;
}
```

```
// CLASE NET

// METODOS ABSTRACTOS DE LA CLASE NET

net:: net(char *s)
{
        char fn[16];
        name=new char [strlen(s)+1];
        strcpy(name,s)
        const NOPARMS=5;
        parma a[NOPARMS]={
                        parm("Entradas",(void *)&n, integer),
                        parm("Salidas", (void *)&p, integer),
                        parm("RATE", (void) *)&learnrate, real),
                        parm("DECAY", (void *)&decay, real),
                        parm("ITERS",(void *)&iters,integer)
        };
        parmtable p(NOPARMS,a,name);
        return;
}

int net::train()
{
        if(loadweights())
                cout<<"Training from stored weights. \n";
        sprintf(fn,"%s.FCT", name);
        cout<<"Entrenamiento de "<<fn<<". Presione cualquier tecla para parar. \n";
        for(;;){
                s=new ifstream(fn, ios::in);
                if(! *s){
                        cout<< "Error al abrir el archivo de hechos. \n";
                        return 0;
                }
                cout<<"Ciclo"<<++cycleno << ":";
                ret=cycle(*s);
                delete s;

                if (ret>=1.0 || kbhit()){
                        cout<<"Entrenamineto suspendido en "<<cycleno <<"ciclos. \n";
                        break;
                }
        }
        saveweights();

}

float net:: cycle(ifstream& s)
{
        vecpair v(n,q);
        float bueno, total;
        s>>*minvecs;
        s>>*maxvecs;
        skipcmt(s);
        for(;;){
```

```
                s>>v;

                if(s.eof()||s.fail()) break;
                v.scale(*minvecs, *maxvecs);
                if(encode (v)){
                        bueno++;
                        if(!trace)
                                cout<<'.';
                }
                else
                        if(!trace)
                                cout<<'x';
                        total++;
                        if(kbhit()){return 1.0;
                }
        }
        return bueno/total;
}

int net:: skipcmt(ifstream& inf)
{
        int c;
        inf.unsetf(inf.skipws);
        if(inf.peek()==':'){
                do{
                        c=inf.get();
                        if(c<0)
                                return 0;
                }while ((c!=0xd)&&(c!=0xa));
                inf.setf(inf.skipws);
                return 1;
        }
        else{
                inf.setf(inf.skipws);
                return 0;
        }
}
```

B.3 Definición e implementación de la clase BP

```
//BP.HPP
// Archvio de cabecera para la implementación de backpropagation

#include "net.hpp"

class bp: public net {

private:
    int q; // Tamaño de la capa oculta
    matrix *W1, *W2; // Matriz de pesos para capas sipnasis
    matrix *dW1, *dW2; // Usado para cambios en los calculos de los pesos
    vec *h; // vector usado para guardar las neuronas activadas en la capa oculta
    vec *o; // vector usado para guardar los resultados de la capa de salida
```

```
        vec *d; // vector usado para guardar los resultados de la tarjeta
        vec *e; // vector de error
        vec *totd, *tote; // usado para acumular error total
        vecpair *minvecs, *maxvecs; //usado para acumular el error total entre el rango minimo y máximo
        float momentum; // cuanto cambio de peso en periodo de tiempo previo afecta el cambio
                        // de peso en el periodo de tiempo actual
        int epoch;

public:
        bp(char *s); //Definicion del constructor
        ~bp();
        int encode(vecpair& v);
        vec recall(vec& v);
};

//BP CPP
//Implementacion de la RED BACKPROPAGATION

#include "bp.hpp"

extern int trace;

//Constructor

bp::bp(char *s): net(s)
{
        const NOPARMS=4
        parm parms[NOPARMS]={
                        parm("HIDDEN", (void *)&q, integer),
                        parm("MOMENTUM", (void *)&momentum, real),
                        parm("INITRANGE", (void *)&initrange, real),
                        parm("EPOCH, (void *)&epoch, integer)
        };
        parmtable ptbl(NOPARMS,parms, name);

// Inicializa las matrices de pesos con valores aleatorios de -1 a 1

        W1=new matrix(n,p,-initrange);
        W2=new matrix(p,q,-initrange);
        dW1=new matrix(n,p);
        dW2=new matrix(p,q);

        h=new vec(p);
        o=new vec(q);
        d=new vec(q);
        e=new vec(p);

        thresh1= new vec(p);
        thresh1->randomize(initrange);
        thresh2=new vec(q);
        thresh2-> randomize(initrange);

        if(epoch){
                totd=new vec(q);
```

```
                totc=new vec(p);
        }

        minvecs=new vecpair(n,q);
        maxvecs=new vecpair(n,q);

        cycleno=0;
}

//Destructor

bp::~bp()
{
        delete W1;
        delete W2;

        delete dW1;
        delete dW2;

        delete h;
        delete o;
        delete d;
        delete e;

        if(epoch){

                delete totd;
                delete totc;
        }
        delete minvecs;
        delete maxvecs;
}

//ALGORITMO DE BACKPROPAGATION

int bp::encode(vecpair& v)
{
//Paso 1: h=F(W1 i)

        *h=(W1)*(*(v.a));
        h->sigmoid(*thresh1);

//Paso 2: o=F(W2 h)

        *o=(*W2)*(*h);

        if(trace){
                cout <<"Conjentura incompleta: <<* o;
        }

        o->sigmoid(*thresh2);

        if (epoch){ //Ajusta los pesos al final del ciclo
```

```
        // Paso 3: d=o (1-o) (o-t)

        *d =(*(v.b)- *o);
        if(trace)
        cout.precision(2);
                cout<<"\nSalida: "<<*(v.b)<<"conjetura: "<<*o;
        float maxdiff=d->maxval();
        *d= *d*o->d_logistic();

        //Paso 4 e=b(1-b)W2 d

        *e=((*W2)* *d)* //matriz x vector =vector

        h->logistic();

        *told -=*d;
        *told -=*e;
        }

        else{

        //Paso 3  d= o(1-o)(o-t)

        *d =(*(v.b)-*o);
        cout.precison(2);
    if(trace)
        cout<<"\nSalida: "<<*(v.p)<<"conjetura": "<<*o;

    float maxdiff=d->maxval();
    *d=*d*o->d_logistic();

  //Paso 4: e= b(1-b)W2 d

        *e=((*W2)* *d)*
        h->d_logistic();
  //Paso 5: W2= W2 + hd+ W2(i-1)

        initvals(*dW2, (*h), *d, learnrate, momentum);
        (*W2)+=*dW2;

        *thresh2 -=((*d)* learnrate);
  //Paso 6: W1=W1 + e+ W2(i-1)

        initvals(*dW1,*(v.a),*e, learnrate, momentum);
        (*W1) +=*dW1;

        *thresh1 -= ((*e) * learnrate);
  }
  if(maxdiff < tolerance)
        return 1;
  else
        return 0;
}
```

```
void bp:: initvals(matrix& m, const vec& v1, const vec& v,
                                        const float rate, const float momentum)
{
 for (int i=0; i<m.r; i++)
        for(int j=0; j<m.c; j++)
                m.m[i][j]=(m.m[i][j]*momentum)+(v1.v[i]*v2.v[j]*rate);
}


vec bp:: recall (vec& v)
{
 //Paso 1: h= F(W1 i)

        *h=(*W1)*v;
        h->sigmoid(*thresh1);

 //Paso 2: o = F(W2 h)

        vec out(this ->p)
        out(*W2)*(*h);
        out.sigmoid(*thresh2);
        return out;
}

float bp: cycle (ifstream& s)
{
        vecpair v(n,q);
        float bueno, total;

        s>>*minvecs;
        s>>*maxvecs;

        skipcmt(s);
        for(;;){
                s>>v;

                if(s.eof()||s.fail())
                        break;
                v.scale(*minvecs, *maxvecs);
                if(encode(v)){
                        bueno++;
                        if(!trace)
                                cout <<'.';
                }
                else
                        if(!trace)
                                cout <<'x';
                total ++
                if (kbhit())
                        return 1.0;
        }
        if (epoch){ //ajusta los pesos al final del ciclo
```

```
                dW2->initvals((*h), *told, learnrate);
                (*W2) += *dW2;

                *thresh2 += ((*told)*learnrate)

                dW1 ->initvals(*(v.a), *tote, learnrate);
                (*W1) += *dW1;
                *thresh1 +=((*tote)* learnrate);
        }

        cout <<"\n"<<bueno/total * 100 <<"Porcentaje correcto  \n";
        return bueno/total;

}
float bp::test()
{
        float bueno=0, total =0;
        char tstfn[32];
        vecpair v;
        vec out;

        if(!loadweights()){
                cout<<"No guardar la prueba de la red"
                return 0;
        }
        sprintf(tstfn, "%s.TST", name);
        ifstream tstf(tstfn, ios::in);
        skipcmt(tstf);
        for(;;){
                if(!(tstf>>v))
                break;

                out=recall(*(v.a));
                if((*(v.b)-out).maxval()< tolerance)
                        bueno++;
                        total++;
        }
        cout<<bueno/total<<"porcentaje correcto. \n";
        return bueno/total;
}
nt bp::run()
{
        char ifn[16],ofn[16]; int c;
        vec in(n), out(q);
        if(!loadweights()){
                cout <<"no guardar la corrinada de la red. \n";
                return 0;
        }
        sprintf(ifn, "%s.IN",name);
        sprintf(ofn, "%s.OUT",name);
        cout<<"Correr desde "<<ifn<<"\n";
        cout<<"Salida a "<<ofn << "\n";
        ifstream inf(ifn, ios::in);
        ofstream outf(ofn, ios::out);
```

```
        skipcmt(inf);
        for(;;){
                if(!(inf>>in))
                        break;
                if(!inf || inf.eof() || inf.fail())
                        break;
                outf<<recall(in);
        }
        return 1;
}
int bp:saveweights()
{
        int fh; char fn[32];
        sprintf(fn,"%s.WTS",name);

        fh=open (fn, O_CREAT | O_TRUNC | O_BINARY);
        if(!fh)
                return 0;
        write(fh,&cycleno, sizeof(int));
        if(!(W1->save(fh)))
                return 0;

        if(!W2->save(fh))
                return 0;

        close(fh);

        sprintf(fn,"%s.MAT", name);
        ofstream matf(fn, ios::out);

        matf <<"La primera matriz contiene: \n";
        matf <<*W1;
        matf <<"La segunda matriz contiene: \n";
        matf <<*W2;
        return 1;
}
int bp:: loadweights()
{
 int fh; char fn[32];
 sprintf(fn, "%s.WTS", name);
 fh=open(fn, O_RDONLY | O_BINARY);
 if(!fh)
        return 0;
 read(fh, &cycleno,sizeof(int));
 if(!(W1->load(fh)))
        return 0;
 if(!(W2->load(fh)))
                return 0;
 close (fh);
 return 1;
}
```

APÉNDICE C

EL ROBOT PUMA 560

EL ROBOT PUMA 560

En la figura C.1 se muestran las articulaciones y elementos del robot PUMA de la serie 560 fabricado por Unimation.

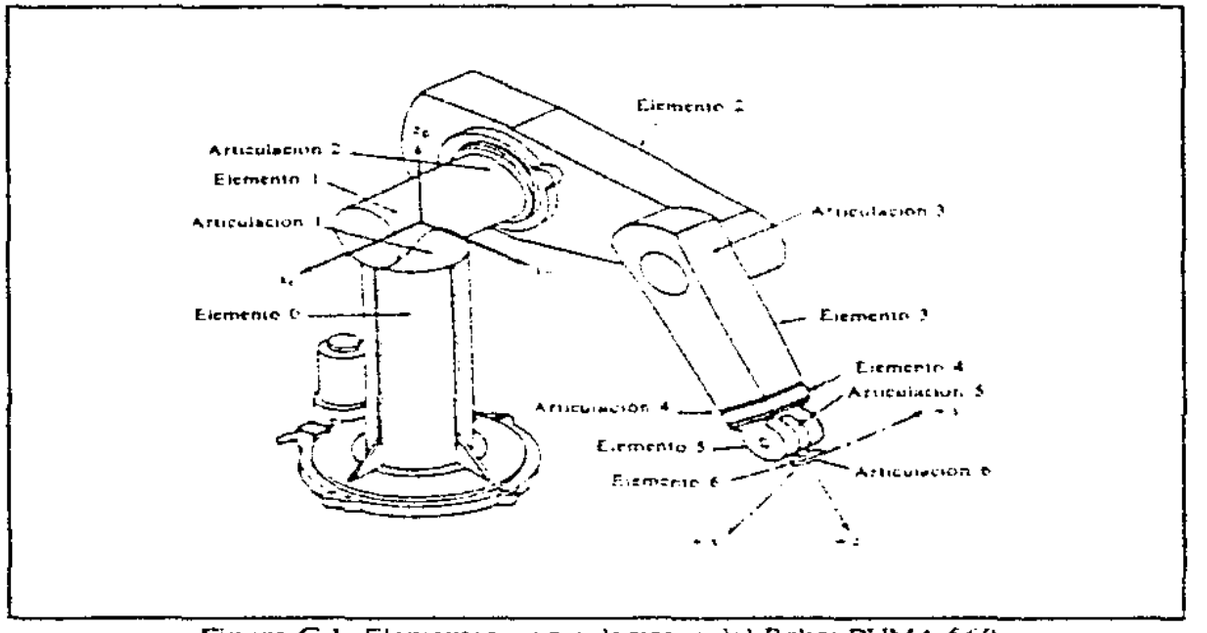

Figura C.1: Elementos y articulaciones del Robot PUMA 560

Los parámetros cinemáticos y dinámicos del PUMA 560 se dan en las tablas C.1, C.2 y C.3 respectivamente. La tabla C 4 muestra el máximo valor para la posición, la velocidad y la aceleración de cada articulación

Número de Articulación	α°	θ°	$a_i(cm)$	$d_i(cm)$
1	-90	θ_1	0	0
2	0	θ_2	0	43.2
3	-90	θ_3	15.0	1 9
4	-90	θ_4	43.2	0
5	90	θ_5	0	0
6	0	θ_6	0	0

Tabla C.1: Parametros de eslabonamiento para el PUMA 560

Número de articulación	Centro de Masa (m) $\bar{x}$	$\bar{y}$	$\bar{z}$	Masa (kg)
1	0.000	0.309	0.039	12.950
2	-0.329	0.005	0.2038	22.360
3	0.020	0.014	0.0037	5.000
4	0.000	0.086	-0.0029	1.177
5	0.000	-0.010	0.0013	0.618
6	0 000	0.000	0.0029	0.157

Tabla C.2: Masas y centros de masa para el PUMA 560

Número de articulación	Tensor Inercial ($kg-m^2$) I_{xx}	I_{yy}	I_{zz}	Inercia Motora ($kg-m^2$) I_a
1	2.35100	0.19680	2.345700	0.7766
2	1.33130	4.31300	3.411600	2.3616
3	0.07582	0.07766	0.010380	0.5827
4	0.01410	0.00338	0.013950	1.0600
5	0.00055	0.00057	0.000546	0.0949
6	0.00012	0.00012	0.000060	0.1070

Tabla C.3: Tensores de inercia e inercias motoras para PUMA 560

Número de articulación	Máxima Posición (rad)	Máxima Velocidad (rad/sec)	Máxima Aceleración rad/sec^2
1	5.59	0.72	7.2
2	4.72	0 42	4.2
3	4.72	0.84	8.4
4	4.89	1.84	18.4
5	3.49	1.95	19.5
6	9.29	1.83	18.3

Tabla C.4: Máximo valor de posición, velocidad y aceleración para cada una de las articulaciones del PUMA 560