CENTRO DE INVESTIGACIÓN Y DE
ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO
NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE COMPUTACIÓN

COORDINACIÓN VISUOMOTORA DE ROBOTS BASADA EN REDES NEURONALES

Tesis que presenta
ING. CARLOS ALBERTO CAMPOS BRACHO

Para obtener el Grado de
MAESTRO EN CIENCIAS

En la especialidad de
INGENIERÍA ELÉCTRICA

Director de Tesis

Dra. Ana María Martínez Enriquez

Dr. Juan Manuel Ibarra Zannatha

INDICE

AGRADECIMIENTOS

A mi esposa Tania y mi hijo Luis Alberto:

Por ser la inspiración de mi vida y por haberme permitido sacrificar su tiempo para terminar este trabajo de tesis... gracias.

A mis padres :

Por infinitas cosas que he recibido de ustedes pero sobre todo, por haberme puesto en el camino correcto de la vida... muchas gracias.

A mis hermanos Micky y Julio :

Por el apoyo incondicional que siempre me han dado.. gracias.

A mi hermano Axel :

Por representar un ser muy especial en mi vida.

A Toya :

Por haberme cuidado en los primeros años de mi vida.. gracias Toya.

A todo el resto de mi familia :

Por haberme apoyado en alguna determinada circunstancia.. mil gracias.

A mis suegros :

Por todos sus consejos y apoyo recibido desde que los conocí... muchisimas gracias..

A mis cuñados :

Por toda su ayuda prestada en todo momento que lo he necesitado... muchisimas gracias..

A la Dra. Ana María :

Por sus valiosos comentarios realizados en el desarrollo de este trabajo de tesis.

Al Dr. Juan Manuel Ibarra :

Por haberme propuesto este interesante trabajo de tesis y por ser un excelente motivador en el desarrollo del conocimiento científico.

A todos, en general, los profesores del CINVESTAV :

Por forjar un espíritu de investigación en la comunidad estudiantil.

A Marco y David :

Por compartir las mismas penas y glorias durante nuestra estancia en el CINVESTAV.

A Sofí :

Por haberme tratado como su hijo.

A todos, en general, mis compañeros del CINVESTAV :

Por haberme permitido formar parte de ese gran equipo de trabajo y estudio.

Resumen

En este trabajo de tesis, se aborda el problema de coordinar una tarea de manipulación robotizada de objetos a partir de la posición y orientación de estos, obtenida a través de un sistema de visión artificial (VA). El entorno de trabajo está formado por un robot industrial tipo ***SCARA***, una tarjeta digitalizadora de imágenes *Oculus 200*, una cámara de video del tipo CCD y una PC. El objetivo principal de nuestro trabajo, es el diseño e implantación de un kernel de un sistema de ensamble de piezas basado en Redes Neuronales Artificiales (KERNA), en donde la información de posicionamiento pinza-objeto, proveniente de un sistema de VA, es la información fuente de una red neuronal artificial (RNA), cuya salida generada calcula las señales de control adecuadas a los actuadores del robot.

En este sentido, se analizan las potencialidades y propiedades que exhibe un *perceptrón multicapas*, en la solución de problemas de aproximación de funciones cuyos modelos matemáticos son difíciles de obtener de manera analítica ó requieren de parámetros que no se pueden medir. En particular, este modelo de RNA utilizado, tiene como propósito aproximar la Cinemática Inversa de nuestro sistema de Robot-Visión. Cabe mencionar que para la simulación de este modelo neuronal se propone una plataforma de simulación basada en las librerias de Matlab®, cuyo objetivo es experimentar con diferentes configuraciones neuronales. Además, para la implantación de esta RNA en un lenguaje de programación orientado a objetos (C++), se propone un modelo de clases en UML, cuyo propósito es definir cualquier topología neuronal usando un esquema de programación orientada a objetos.

Por otro lado, nuestros resultados tienen como propósito contribuir en la investigación y aplicación del control inteligente de robots.

Lista de figuras

Lista de tablas

Capítulo I. Introducción

Uno de los principales problemas en la robotización de tareas de ensamble, es el alto costo y baja confiabilidad de todos los accesorios mecánicos que deben integrar el entorno del robot, tales como despachadores, alimentadores, contenedores, vibradores, etc., cuya misión es suministrar al robot todas las piezas por ensamblar en la posición y orientación en el momento adecuado. Además, en este tipo de aplicaciones se necesita programar las tareas con mucho detalle, definir las posiciones con precisión y contar con los sensores de sincronía y detección de presencia que generen la información requerida.

La alternativa más viable para resolver el problema de costo, confiabilidad y programabilidad amigable de este tipo de sistemas robotizados, es el uso de Visión Artificial (VA). Esto permite tener una mayor flexibilidad en el puesto de trabajo robotizado (cambio de aplicación mediante una simple reprogramación), una menor complejidad mecánica del puesto de trabajo y una mayor seguridad al posibilitar la introducción de nuevas funciones tales como anticolisión, detección de anomalías, etc.

El interés por utilizar técnicas de VA en los robots industriales comenzó al inicio de los años 70's [1]. En aquel entonces, la VA se utilizó para la detección de objetos, identificación de partes, así como para inspección y control de calidad. Además, durante ese mismo tiempo, se iniciaron algunos esfuerzos importantes para utilizar la VA como sensor en el lazo de control, en particular como generador de consignas fuera de línea en la opción denominada *fixed-eye* (cámara fija).

Por otro lado, en lo que concierne a la opción *eye-in-hand* (cámara embarcada en el brazo del robot), existen importantes trabajos de aplicación a la manipulación de objetos fijos y de seguimiento de objetos móviles desde mediados de los 80's.

Sin embargo, a pesar de sus ventajas, el uso de VA dentro de la robotización de tareas de ensamble requiere considerar distintos factores, difíciles de resolver, entre los cuales destacan:

- Desarrollo de modelos cinemáticos exactos.
- Procedimiento de calibración de cámara precisos.
- Potencia de cálculo al procesar las imágenes.

Dada la complejidad de las soluciones analíticas a estos problemas (calibración de cámara, inversión de modelos cinemáticos geométrico y variacional, entre otros), un método que se plantea como solución a dichos problemas son las Redes Neuronales Artificiales (RNA) [2], basados en las propiedades que éstas presentan:

- Capacidad de aproximación.
- Capacidad de generalización.
- Capacidad de adaptación.
- Potencia de cálculo, entre otras.

En nuestro caso particular, el uso de Redes Neuronales Artificiales (RNA), queda plenamente justificado con las ventajas de que éstas evitan el problema de las singularidades y la estimación de algunos parámetros que no se pueden medir y que son necesarios en el modelo cinemático Inverso de nuestro sistema de Robot-Visión. Además, una vez entrenada la RNA, su uso implica un menor error de aproximación y una menor complejidad computacional en comparación con las soluciones analíticas.

En este sentido, dentro del contexto de este trabajo, es de particular interés analizar las potencialidades y propiedades que exhibe el *perceptrón multicapas* así como tambien el diseño e implantación del mismo, dentro de un kernel de un sistema de ensamble de piezas basado en RNA (KERNA), donde la información fuente de posicionamiento relativo pinza-objeto, proveniente de un sistema de VA, es la información de entrada a la RNA cuya salida generada, calcula las señales de control adecuadas a los actuadores del robot *Unimate S-103*. Cabe mencionar que los objetos considerados en este trabajo son objetos cilíndricos que pueden ser sujetados por la muñeca del robot y además presentan la misma altura.

Además, para la implantación de esta RNA, se propone un modelo de clases en UML, cuyo propósito es definir cualquier topología neuronal dentro del paradigma de programación orientada a objetos. Al respecto, el uso de un lenguaje de modelado de sistemas así como el uso de la programación orientada a objetos (POO), en este trabajo está justificado por los beneficios que estas presentan [34]:

- Consistencia de modelos.
- Estabilidad en presencia de cambios de requerimientos.
- Reutilización de código.
- Escalabilidad.
- Modelación estática y dinámica de objetos.
- Soporte para organización de código fuente.
- Soporte para concurrencia.
- Empaquetamiento de clases, entre otras.

Con base en los anteriores párrafos, el contenido de este trabajo es el siguiente:

El capítulo **II** describe las áreas de investigación relacionadas con el mismo: visión artificial (VA), redes neuronales artificiales (RNA) y control visuomotor (CV).

Dentro del capítulo **III** se describe el modelo Cinemático directo y los problemas que se presentan al tratar de resolver de una forma analítica el modelo Cinemático inverso del robot *Unimate S-103*. También, se describen las características de control, programación y comunicación de dicho robot.

El capítulo **IV** analiza las diferentes etapas que conforman nuestro sistema de Visión Artificial: captura, filtraje de ruido, segmentación e identificación de objetos presentes en la imagen.

Por otro lado, el capitulo **V** explica los aspectos más importantes de las RNA relacionados con este trabajo: elementos de procesamiento, topología y capacidad de aproximación de funciones. Además, muestra la construcción e implantación de KERNA correspondiente a nuestro sistema integrado Robot-Visión.

Finalmente, en él capitulo **VI** se resumen las conclusiones obtenidas en este trabajo, así como también los trabajos futuros a desarrollar.

Capítulo II. Antecedentes

II.1 Visión Artificial

El interés por utilizar técnicas de Visión Artificial (VA), en los robots industriales, comenzó al inicio de los años 70's[1]. Básicamente, el objetivo de los sistemas de VA aplicados en Robótica, es transformar el conjunto de datos contenidos en una imagen digitalizada, en una descripción de los objetos que se encuentran dentro del espacio de trabajo del robot.
Existen diversas aplicaciones potenciales de la VA en el área de la Robótica, entre otras, las siguientes [1]:

- Control de operaciones de los Robots.
- Aplicaciones médicas.
- Exploración de medio ambiente desconocido.
- Inspección y control de calidad.
- Detección y diagnóstico de anomalías.

En la manufactura por ejemplo, la VA es indispensable en el ensamble de piezas así como en la manipulación e inspección de los materiales debido a la necesidad que se tiene de identificar y localizar los objetos, en cualquier parte del espacio de trabajo del robot sin importar su posición u orientación.

En este sentido, para efectuar el reconocimiento de un objeto, un sistema de VA debe procesar la imagen efectuando cálculos numéricos, verificaciones y comparaciones sobre la imagen de la escena en cuestión.

De manera general, se considera que las etapas que conforman el proceso de reconocimiento de un objeto sobre una imagen son:

- *Formación.*- Es la transformación mediante procedimientos ópticos del mundo tridimensional en una imagen bidimensional.

- *Captación.*- Este proceso obtiene una señal eléctrica que contiene la información de la imagen así como su conversión A/D que permite tenerla disponible en la memoria de la computadora.

- *Filtraje.*- Consiste en mejorar la imagen digital obtenida, eliminando el ruido producido por sombras, exceso de luz, etc. También incluye el realzado de las características de la imagen necesarias en fases posteriores.

- *Segmentación.*- Es un proceso que se encarga de particionar una imagen dada en regiones correspondientes a objetos. Una segmentación simple, en el caso de imágenes bien contrastadas, consiste en pasar la imagen a través de un umbral de brillantez, creando así, una imagen binaria en la cual se distinguen dos tipos de regiones: región objeto y región fondo.

- *Descripción.*- Es el proceso de modelar los objetos presentes en la imagen.

- *Reconocimiento.*- El reconocimiento es una tarea, cuya función es obtener, para cada objeto segmentado, las características ó atributos que modelan al objeto en cuestión.

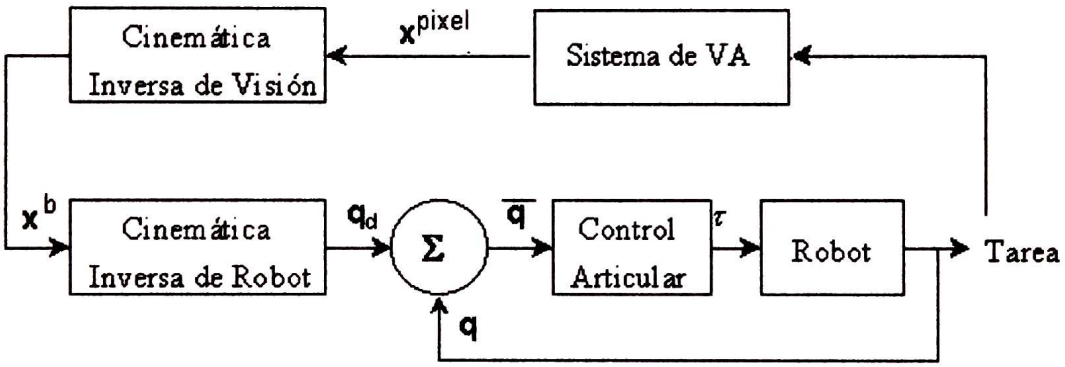
Para propósitos de este trabajo, el sistema de VA tiene como finalidad proporcionar la identificación (a través del cálculo de ciertos atributos geométricos invariables a rotación, traslación y cambios de escala del objeto en cuestión) y la localización (a través de la determinación del pixel que corresponde al baricentro del objeto dado en términos de coordenadas *pixels x,y*), de las piezas que se desean ensamblar.

En él capítulo **IV** (ver también Apéndice A), se analizan los algoritmos utilizados en la implantación de cada una de las etapas anteriores, cuya integración, conforma nuestro sistema automático de reconocimiento de objetos.

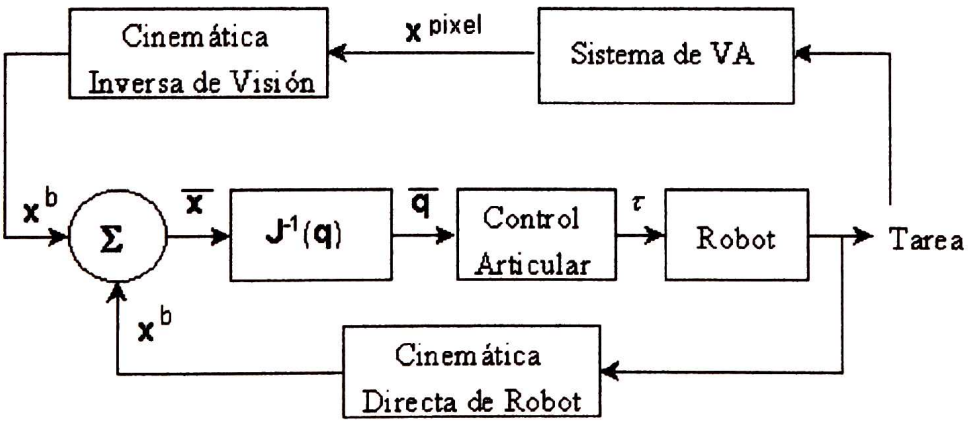
II.2 Control Visuomotor de Robots

De manera tradicional, el control de robots se hace en el espacio articular, es decir; se calcula a partir del error de posición articular, definido como la diferencia entre la posición articular deseada $\mathbf{q}_d$ (consigna) y la posición medida por los sensores del robot $\mathbf{q}$. Cuando se dispone de un sistema de VA para generar mediciones de la posición de los objetos por manipular ($\mathbf{x}^{pixel}$), se tienen tres opciones a considerar en el diseño del controlador del robot [4]:

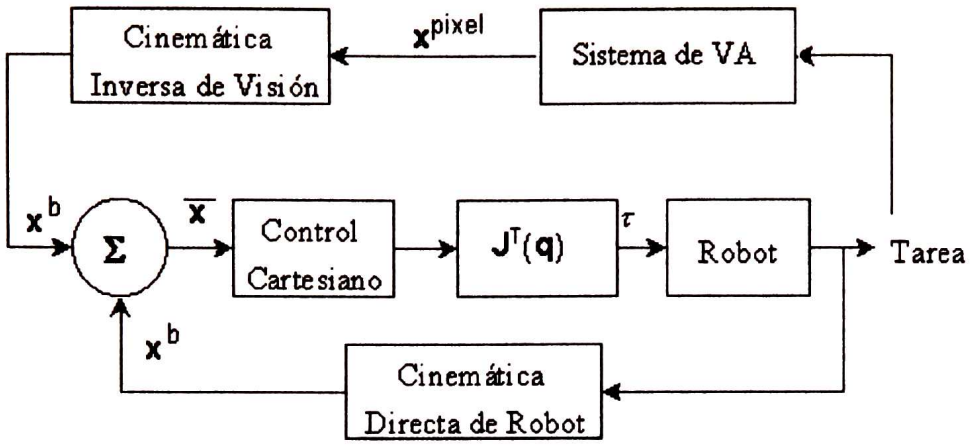
- Transformar las mediciones $\mathbf{x}^{pixel}$ generadas por el sistema de VA, en consignas $\mathbf{q}_d$ y aplicar los controladores articulares clásicos (fig. II.1a).
- Calcular el error en posición cartesiana usando directamente la medición proporcionada por el sistema de VA y la Cinemática inversa de visión, la posición cartesiana equivalente del robot ($\mathbf{x}^b$) y luego transformar en error articular mediante el Jacobiano inverso. Finalmente se aplica un controlador articular clásico. (fig. II.1b).
- La tercera alternativa utiliza el error cartesiano en un controlador cartesiano, cuya salida deberá transformarse en una variable de control articular mediante el Jacobiano transpuesto del robot (fig. II.1c).



a) Cinemática Inversa más Control Articular



b) Jacobiano Inverso más Control Articular



c) Control Cartesiano más Jacobiano Transpuesto

Fig. II.1 Esquemas de Control Visuomotor

En los primeros dos esquemas se requiere de una inversa, ya sea de la transformación cinemática del sistema de VA ó del Jacobiano del robot, mientras que en el tercer esquema se necesita la transformación cinemática directa del robot y la transpuesta de su Jacobiano.

En nuestro caso, como no es posible modificar el controlador del robot debido a que utilizamos un robot industrial cuya arquitectura computacional es cerrada, estamos obligados a utilizar un controlador articular en donde las mediciones visuales de la posición de un objeto, son transformadas en consignas articulares, transformación conocida como la cinemática inversa de un sistema de Robot-Visión.

Estrategias de Integración Visión- Robot

Dentro de los esquemas de control visuomotor que fueron presentados de manera general en el apartado anterior, se pueden adoptar diferentes estrategias de generación de información visual y de su utilización dentro del controlador. En efecto, la información visual puede usarse tan sólo para generar consignas ó como sensor en un servomecanismo visual. Desde el punto de vista del montaje de la cámara se tienen esquemas con cámara fija (*fixed-eye*), y con cámara móvil (*eye in hand*), mientras que, desde el punto de vista de los objetos, estos pueden ser estáticos ó móviles. Además, si se considera el tipo de aplicación, ésta puede requerir información visual *3D* ó utilizar tan sólo información *2D*.

Generación de Consignas por Visión

Es la aplicación más simple de la visión en el control de movimiento. En este esquema sólo se hace la medición de la posición y orientación mediante un sistema de visión *2D* del objeto, ya sea en una opción conocida como *look and move* (objetos estáticos) ó haciendo la estimación de la trayectoria deseada para el caso de objetos móviles. Dicho de otra manera, en esta opción sólo se hace la coordinación visión-acción de los movimientos del robot. En este esquema de utilización de la visión artificial en el control de robots, las cámaras siempre son fijas.

Servomecanismo Visual

Cuando el sistema de VA se usa para la medición del error de posición en línea, se puede diseñar un control visual directo en base a dicho error. Generalmente se utilizan con cámara fija y pueden aplicarse con objetos estáticos (regulación) ó móviles (seguimiento). Cuando la cámara y los objetos se encuentran en movimiento, se tiene un problema de difícil solución en donde aparecen dificultades en el seguimiento y estabilidad del controlador y, sobretodo, de calibración dinámica ó de autocalibración. En este tipo de problemas se utilizan diversas metodologías de estimación del movimiento, entre las cuales destaca el flujo óptico y el uso de esquemas adaptables para la estimación del movimiento.

Dentro del alcance de este trabajo, consideramos el esquema *look and move* como nuestra estrategia de integración Visión- Robot (fig. II.2).

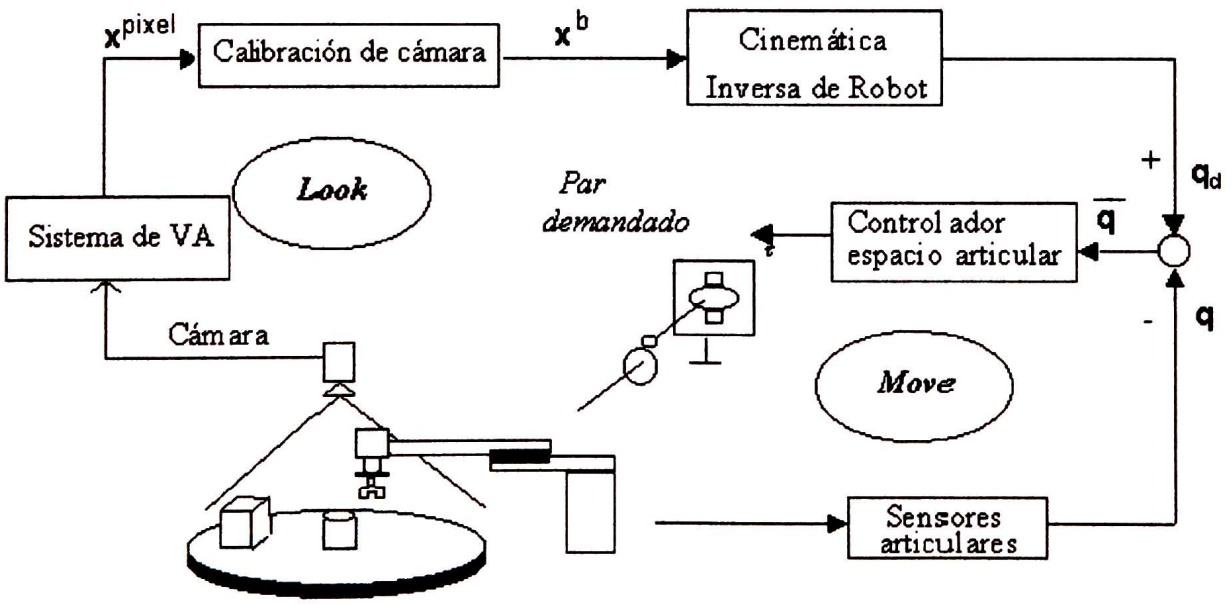


Fig. II.2 Esquema *look and move*

II.3 Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA), son redes interconectadas masivamente en paralelo, conformadas por elementos simples de procesamiento cuya propiedad es procesar información por medio de su estado como respuesta a entradas externas.
Cada elemento de procesamiento, transforma en una señal de salida, todas las señales de entrada que provienen ya sea de otros elementos de procesamiento ó del mundo exterior. La respuesta de cada unidad es una función (llamada de *transferencia ó activación*), de la suma ponderada de todas sus entradas, adicionadas de una polarización llamada *umbral*. En la figura II.3 se ilustra la arquitectura típica de una neurona artificial.

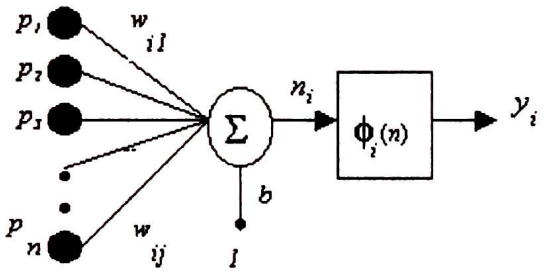


Fig. II.3 Arquitectura básica de una neurona artificial

La ecuación básica de una neurona artificial es:

$$y_i = \phi_i\left(\sum_{j=1}^{n}\left(w_{ij}^T p_j + b_i\right)\right) \qquad \text{II.1}$$

donde:

- y_i Valor de salida de la i-ésima neurona.
- ϕ_i Función de activación de la neurona.
- w_{ij} Ponderación de la conexión entre la j-ésima entrada y la i-ésima neurona .
- p_j j-ésimo componente del vector de entrada.
- b_i Entrada de polarización de la i-ésima neurona.

En la actualidad existen diversos modelos de RNA [3], pero en general, una RNA está definida mediante la siguiente 3-tupla [17]:

$$\text{RNA}=(\mathbf{S},\mathbf{A},\mathbf{T}) \qquad \text{II.2}$$

donde:

S es el conjunto de datos de entrada-salida, y está definido por:

$$\mathbf{S} = \{\mathbf{P}, \mathbf{Y}\} \qquad \text{II.3}$$

$$\mathbf{P} = \{\mathbf{p}_i \mid \mathbf{p}_i \text{ es el i-ésimo vector de entrada}\} = \{\mathbf{p}_1, \mathbf{p}_2, .., \mathbf{p}_i, .., \mathbf{p}_n\}$$

$$\mathbf{Y} = \{\mathbf{q}_i \mid \mathbf{q}_i \text{ es el i-ésimo vector de salida}\} = \{\mathbf{q}_1, \mathbf{q}_2, .., \mathbf{q}_i, ..., \mathbf{q}_n\}$$

(Para el caso en el que se pretende utilizar una RNA con algoritmo de aprendizaje supervisado [3], los conjuntos **P**, **Y** $\neq \{\phi\}$. Por otro lado, para el caso de las RNA con algoritmo de aprendizaje no supervisado [3], **P** $\neq \{\phi\}$ y **Y** $= \{\phi\}$).

T representa la topología de la RNA y está dada por:

$$\mathbf{T} = \{\mathbf{F}, \mathbf{L}\} \qquad \text{II.4}$$

$$\mathbf{F} = \{\mathbf{c}_i \mid \mathbf{c}_i \text{ es la i-ésima capa}\} = \{\mathbf{c}_1, \mathbf{c}_2, .., \mathbf{c}_i, ..., \mathbf{c}_n\}$$

$$\mathbf{L} = \{w_{i,j-->k,l} \mid w \textit{ valor} \text{ de la conexión del nodo } j - \text{capa } i \text{ y el nodo } k - \text{capa } l\}$$

(Los valores finales de $w_{i,j->k,l}$, se ajustan mediante la utilización de un algoritmo de *aprendizaje* [3], cuyo objetivo es definir los valores óptimos de $w_{i,j->k,l}$. En esta formalización de las RNA, se consideran a los valores de umbralización b_i como valores w con la excepción de que estos son multiplicados por un valor de entrada unitario).

A representa al conjunto de parámetros constantes de la RNA (i.e. velocidad de aprendizaje, momentum, etc.)

$$\mathbf{A} = \{a_1, a_2, a_3, ..., a_m\} \qquad \text{II.5}$$

II.4 Planteamiento del problema

De la manera de implementar cualquiera de los esquemas de control mencionados con anterioridad, se presentan diversos problemas a resolver entre los que destacan la calibración de cámara, el estudio de la estabilidad, etc. En particular, la calibración de cámara es un problema que se presenta en cualquier esquema de integración Visión-Robot, ya que en todos, se tiene la necesidad de calcular los valores de las variables articulares que corresponden a puntos en el espacio de las imágenes. Si bien es posible tener formas analíticas para llevar a cabo las transformaciones necesarias, sus parámetros deben ser calculados ó medidos mediante procedimientos que, generalmente, introducen algunos errores de modelado y de cómputo.
En el contexto de la presente tesis, el problema que nos proponemos resolver es el problema de la calibración de cámara y el de la Cinemática Inversa del Robot *Unimate S-103*, donde la solución propuesta se basa en las potencialidades que ofrecen las Redes Neuronales Artificiales (RNA): adaptabilidad, generalización y aproximación.

Dentro de nuestro control visuomotor propuesto, la RNA a utilizar va a aproximar el modelo Cinemático Inverso del sistema Visión- Robot (procedimiento de calibración de cámara y cinemática inversa del robot). En la figura II.4 se muestra un diagrama del sistema de coordinación visuomotora propuesto.

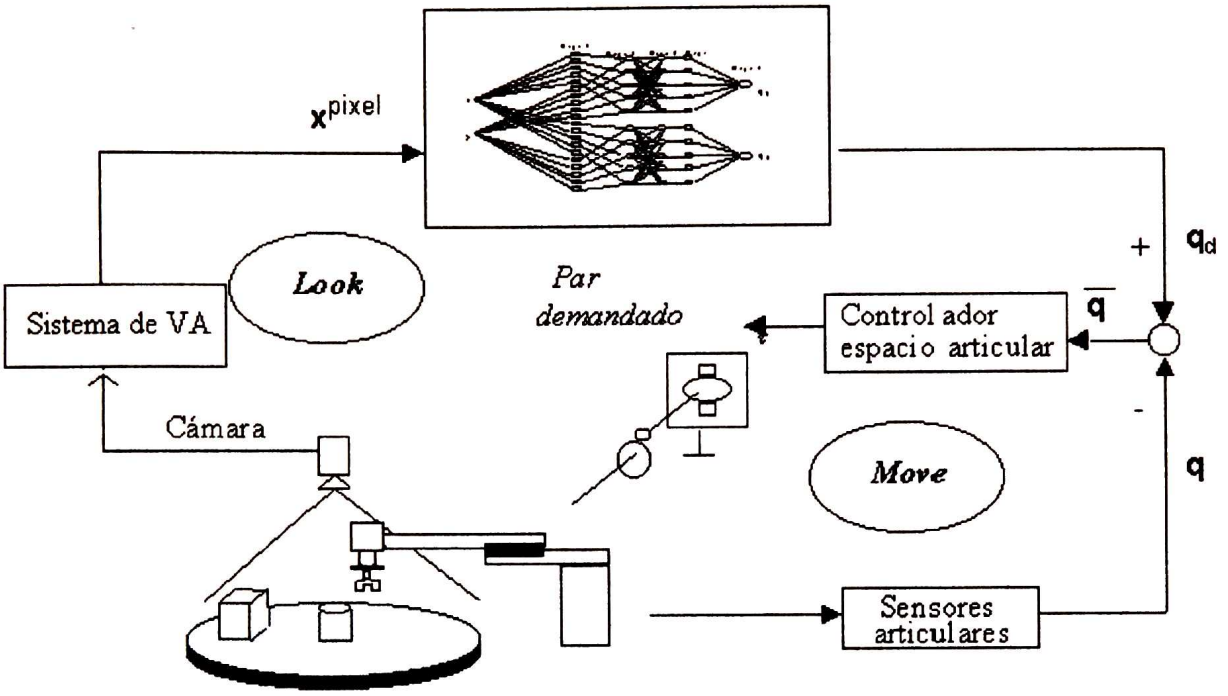


Fig. II.4 Esquema visuomotor propuesto

II.5 Conclusiones

Este capitulo presentó una retrospectiva de las áreas de investigación relacionadas con este trabajo: visión artificial, redes neuronales y control visuomotor de robots. En el área de VA, se describieron las etapas: formación, captura, filtraje, segmentación, descripción y reconocimiento. Dependiendo de la aplicación (en nuestro caso la localización e identificación de los objetos en el espacio de trabajo del robot), se seleccionan los algoritmos más adecuados a cada una de las etapas mencionadas.

En el caso de los esquemas de control visuomotor, se describieron por una parte los enfoques clásicos (Cinemática inversa más control articular, Jacobiano inverso más control articular y Jacobiano transpuesto más control cartesiano) así como los problemas inmersos en ellos: calibración de cámara y Cinemática inversa del Robot.

Con respecto a las RNA, se hizo una breve descripción del elemento principal de toda RNA: la neurona. Además, se describió una formalización de las RNA así como la utilización de éstas dentro de nuestro esquema visuomotor propuesto.

En los capítulos sucesivos se analizan en más detalle cada uno de estos dominios enfatizando la metodología seguida en el diseño e implantación de KERNA.

Capítulo III. Sistema Robot

III.1 Introducción

El robot utilizado en este trabajo es un robot industrial (*Unimate S-103),* de arquitectura mecánica tipo *SCARA* (Selective Compliance Assembly Robot Arm), apropiado para labores de ensamble. Este tipo de robots se caracteriza por tener tres articulaciones rotacionales servo-controladas y una prismática de tipo on-off. En la figura III.1 se muestra un diagrama del robot mencionado en donde se ilustran sus articulaciones.

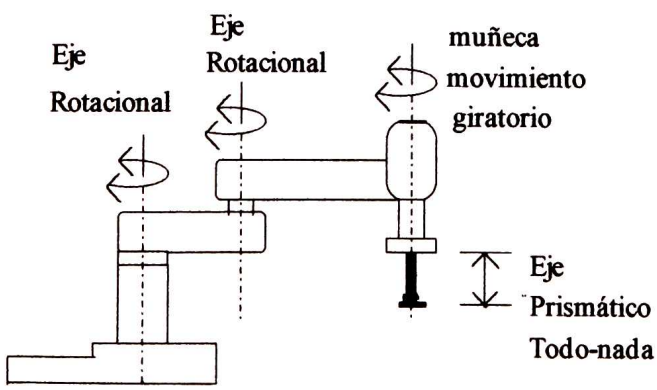


Fig. III.1 Tipo de articulaciones del robot *Unimate S-103*

Básicamente, los elementos que conforman a dicho robot son [5]: el brazo propiamente dicho, el controlador, el sistema de comunicación hombre máquina *(teach pendant)* y el software de control.
Para propósitos de este trabajo, son de particular interés la Cinemática del robot y la comunicación con el robot.
La Cinemática del robot se refiere al estudio analítico de la geometría del movimiento de un robot, con respecto a un sistema de coordenadas de referencia fijo como una función del tiempo sin considerar las fuerzas que originan dicho movimiento [6]. En este sentido, se ocupa de la descripción analítica del desplazamiento espacial del robot como función del tiempo, en particular se enfoca en las relaciones que existen entre las variables espaciales de tipo articular, posición y orientación del efector final del robot.
En cuanto a la comunicación con el robot, es de mencionarse que ésta se lleva a cabo entre el puerto serie de la PC y el puerto serie auxiliar del robot. Dicha comunicación se ajusta al protocolo de comunicación establecido por el sistema operativo del robot (C/ROS), cuya función es controlar todas las funciones del robot.
Este capítulo comprende estas dos cuestiones teórico-prácticas fundamentales en el desarrollo de esta tesis. En el siguiente apartado se desarrolla el *modelo Cinemático directo* y posteriormente, se describen los problemas inmersos en la solución analítica de la Cinemática *Inversa* del robot. Finalmente se presenta el protocolo de comunicación establecido por el sistema operativo del robot *C/ROS.*

III.2 Modelo Cinemático Directo del robot *Unimate S-103*

El modelo cinemático permite encontrar la posición (P_x, P_y, P_z) y la orientación (α,γ,η) del órgano terminal (pinza) del robot a partir de los valores de sus coordenadas articulares (q_1, q_2,...,q_n). La manera más utilizada para representar el modelo cinemático de un robot es mediante una Transformación Homogenea $\mathbf{T}(\mathbf{q})$.

$$\mathbf{T}_0^n(\mathbf{q}) = \begin{bmatrix} \alpha_x & \gamma_x & \eta_x & p_x \\ \alpha_y & \gamma_y & \eta_y & p_y \\ \alpha_z & \gamma_z & \eta_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{III.1}$$

El método a emplear para la obtención del *Modelo Cinemático Directo* es el propuesto por *Denavit-Hartenberg (D-H)* [6]. Para obtener esta transformación es necesario calcular la matriz de paso generalizado $\mathbf{A}_i$ que permite representar el referencial asociado al eslabón i+1 con respecto al eslabón i. Para obtener esta matriz, en la metodología D-H se comienza por definir los parámetros del eslabón (longitud a_i y torcedura α_i) y los de articulación (rotación Θ_i y translación d_i).
Después se determinan los ejes articulares y se asocia un referencial a cada uno de los eslabones de acuerdo con las siguientes reglas:

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

Cada eslabón está caracterizado por dos parámetros:

- a_i (longitud del eslabón): Es la distancia de separación 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 : Ángulo de torcedura del eslabón del eje z_{i-1} al eje z_i respecto el eje x_i (utilizando la regla de la mano derecha).

Mientras que las articulaciones se caracterizan por otros dos parámetros:

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

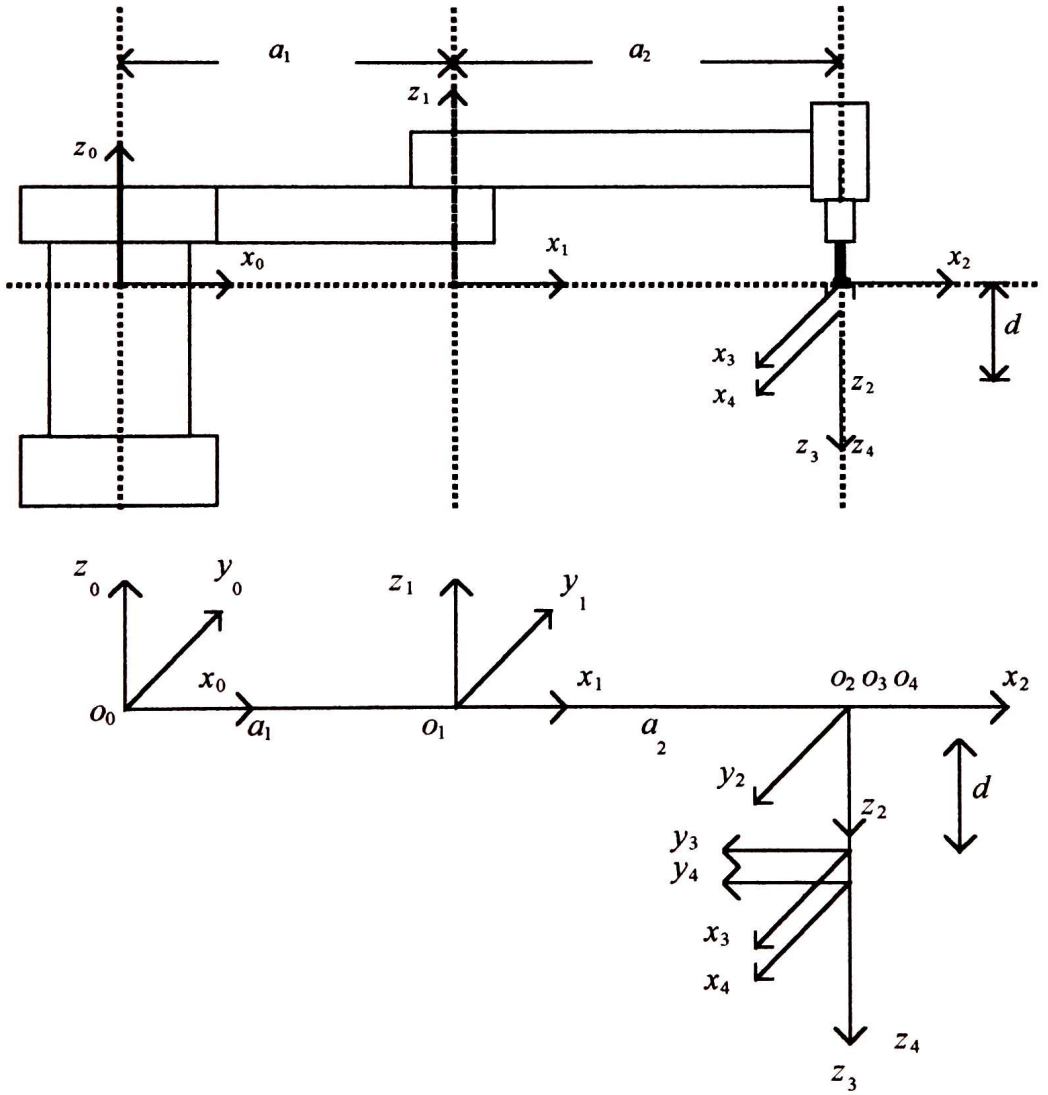


Fig.III.2 Estructura cinemática del robot *Unimate S-103*.

En este método, la matriz $\mathbf{A}_i$ es una transformación homogénea que se obtiene mediante el producto de cuatro transformaciones *básicas* que permiten confundir dos referenciales sucesivos: rotación de α_i grados alrededor del eje x_i traslación de a_i ó d_i centímetros a lo largo del eje x o z y rotación de Θ_i grados alrededor del eje x.

De esta manera se obtiene la siguiente matriz:

$$\mathbf{A}_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{III.2}$$

En la Figura III.2 se muestra un esquema del robot *Unimate S-103* con la ubicación de los referenciales asociados a cada uno de sus cuatro eslabones. Los parámetros cinemáticos de este robot aparecen en la tabla III.1, de los cuales los tres primeros son rotacionales y el cuarto desplazamiento del robot es del tipo todo-o-nada, lo que permite al robot trabajar en dos planos horizontales diferentes: uno alto para desplazar horizontalmente el órgano terminal y otro bajo para tomar ó soltar los objetos en un plano fijo en el cual se encuentra la mesa de ensamble.

Articulación	a_i	α_i	d_i	θ_i
1	a_1	0°	0	θ_1
2	a_2	180°	0	θ_2
3	0	0°	D	0
4	0	0°	0	θ_3

Tabla III.1 Parámetros de las articulaciones.

Una vez derivadas las *n* matrices de paso generalizada, en donde *n* es el número de grados de libertad (gdl) del robot, el modelo Cinemático directo se calcula mediante la multiplicación de estas *n* matrices, de la siguiente manera:

$$\mathbf{T}_0^n = \prod_{i=1}^{n} \mathbf{A}_i \qquad \text{III.3}$$

Aplicando la ecuación III.1 a los datos de la tabla III.1 obtenemos:

$$\mathbf{A}_1 = \begin{bmatrix} c1 & -s1 & 0 & a_1 c1 \\ s1 & c1 & 0 & a_1 s1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{III.4}$$

$$\mathbf{A}_2 = \begin{bmatrix} c2 & s2 & 0 & a_2c2 \\ s2 & -c2 & 0 & a_2s2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{III.5}$$

$$\mathbf{A}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{III.6}$$

$$\mathbf{A}_4 = \begin{bmatrix} c3 & -s3 & 0 & 0 \\ s3 & c3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{III.7}$$

Utilizando estas matrices y la ecuación III.2 tenemos:

$$\mathbf{T}_0{}^4 = \begin{bmatrix} c12c3 + s12s3 & -c12s3 + s12c3 & 0 & a_1c1 + a_2c12 \\ s12c3 - c12s3 & -s12s3 - c12c3 & 0 & a_1s1 + a_2s12 \\ 0 & 0 & -1 & -D \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{III.8}$$

donde:

- $sij = \sin(\theta_i + \theta_j), si = \sin(\theta_i)$
- $cij = \cos(\theta_i + \theta_j), ci = \cos(\theta_i)$

La matriz III.8 representa la matriz de transformación de cualquier punto en el espacio de trabajo del robot dada la configuración articular de éste, en posición y orientación de la pinza del robot referido a la base del mismo. En este sentido, el vector de posición del efector final está dado por (ver Fig.III.3):

$$x = a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \quad \text{III.9}$$

$$y = a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \quad \text{III.10}$$

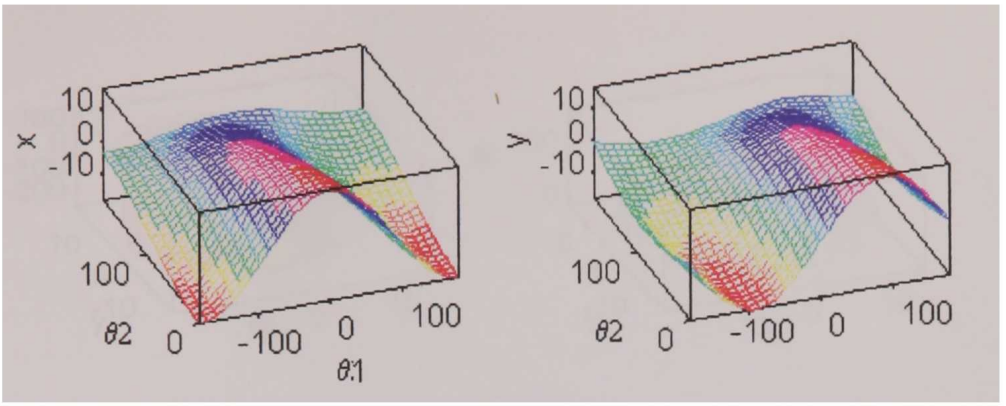


Fig.III.3 Gráfica de la cinemática directa del robot *Unimate S-103*.

III.3 Problemas en la solución analítica del Modelo Cinemático Inverso

En la sección anterior, el objetivo consistió en encontrar la posición y orientación del efector final del robot en función de las variables articulares del robot. Sin embargo, en labores de ensamble, por lo general el control de robots basado en VA se realiza en el espacio de las variables de articulación. Debido a que los objetos que se manipulan se definen en el espacio de coordenadas cartesianas, es necesario transformar las variables cartesianas asociadas a cada objeto en *n* variables articulares (problema Cinemático Inverso).

En general, la solución al problema de la Cinemática Inversa (Fig. III.4), consiste en resolver un sistema de doce ecuaciones no lineales (referidas a la matriz obtenida por el método de Denavit-Hartenberg), con *n* incógnitas (donde *n* representa el número de gdl del robot).

En nuestro caso, el problema Cinemático Inverso, considerando la matriz III.8, consiste en resolver las siguientes ecuaciones no lineales:

$$\begin{aligned} c_{12}c_3 + s_{12}s_3 &= r_{11} \\ s_{12}c_3 - c_{12}s_3 &= r_{21} \\ -c_{12}s_3 + s_{12}c_3 &= r_{12} \\ -s_{12}s_3 - c_{12}c_3 &= r_{22} \\ a_1c_1 + a_2c_{12} &= r_{14} \\ a_1s_1 + a_2s_{12} &= r_{24} \end{aligned} \qquad \text{III.11}$$

donde r_{ij} representa al elemento del i-ésimo renglón y j-ésima columna.

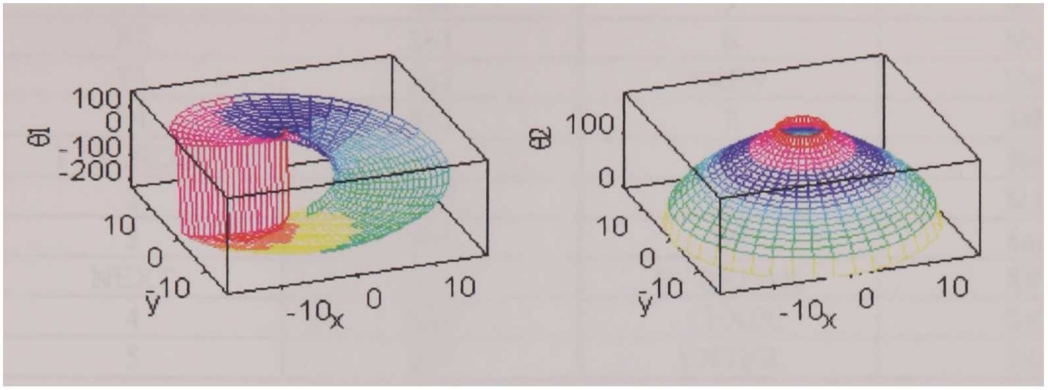


Fig.III.4 Gráfica de la cinemática Inversa del robot *Unimate S-103*.

En general, el problema de la Cinemática Inversa se puede resolver por diversos métodos analíticos [6], tales como la *transformación inversa*, *el álgebra de tornillos*, *matrices duales*, *métodos iterativos y métodos geométricos*. Sin embargo, estas metodologías presentan diversos problemas como múltiples soluciones, puntos singulares, configuraciones articulares fuera del rango de trabajo del robot y exceso de cálculos numéricos sin garantizar convergencia a la solución correcta (particularmente los métodos iterativos).
En este sentido, en el presente trabajo se propone el uso de *Redes Neuronales Artificiales* (RNA), para la solución de dicho problema con la justificación de que éstas evitan todos los problemas antes mencionados.

III.4 Comunicación PC - Robot

Es de mencionarse que este robot cuenta con su propio instrumento de comunicación conocido como *Teach Pendant* el cual se conecta a través de uno de los puertos serie del controlador del robot. Este instrumento posee 20 teclas multifuncionales que representan una matriz, cuyas funciones, al ser accionadas, pueden seleccionar el valor hexadecimal asociado a dicha tecla y enviarlo al controlador cuyo efecto sobre la operación del sistema dependerá del modo de operación en que se encuentre, según el protocolo *C/ROS* [16]. En la tabla III.2 se muestra los códigos hexadecimales asociados a cada tecla del *Teach Pendant* los cuales se utilizan en la comunicación PC-Robot y en la figura III.5 se muestra la secuencia de comunicación con el robot.

Tecla	Código (Hex)	Tecla	Código (Hex)
F1	$af	5	$b7
F2	$b1	6	$b8
F3	$b2	PREV	$b9
F4	$b3	7	$ab
1 / SCROLL	$aa	8	$bd
2	$b4	9	$b0
3	$b5	BS	$ae
NEXT	$b6	0 / BREAK	$8ª
4	$ad	- / EXEC	$a0
5	$b7	ENTER	$8d

Tabla III.2 Códigos hexadecimales asociados a las teclas frontales del *teach pendant*

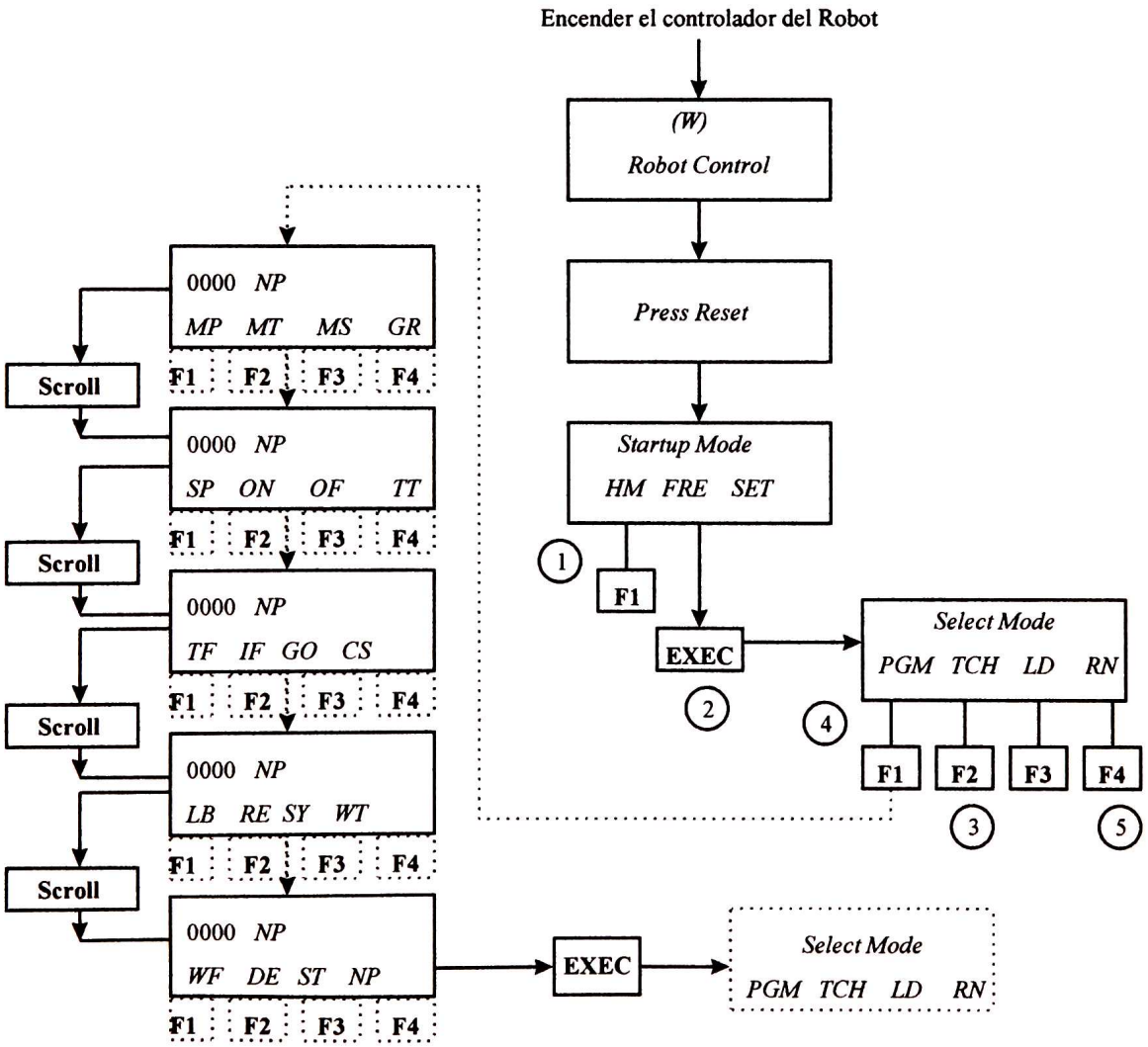


Fig.III.5 Protocolo de comunicación con el robot *Unimate S-103*.

Donde:

1. **F1/HM** : Envia a la posición de *Home* al Robot.
2. **EXEC** : Selecciona el modo de operación del Robot.
3. **F2/TCH** : Definición de los puntos donde la pinza del robot debe posicionarse
4. **F1/PGM** : Carga el programa de ensamble en la memoria del controlador del Robot.
5. **F4/RN** : Activa servomecanismos del robot para realizar el ensamble.

NOTA: Los mensajes escritos en cursiva es la información enviada por el robot en respuesta a algún comando solicitado. El significado de todos los comandos que se manejan en C/ROS se pueden ver en [5].

En el paso 3 correspondiente a la definición de los puntos sobre la mesa de ensamble que debe alcanzar el efector final es necesario realizar la secuencia de comunicación mostrada en la figura III.6.

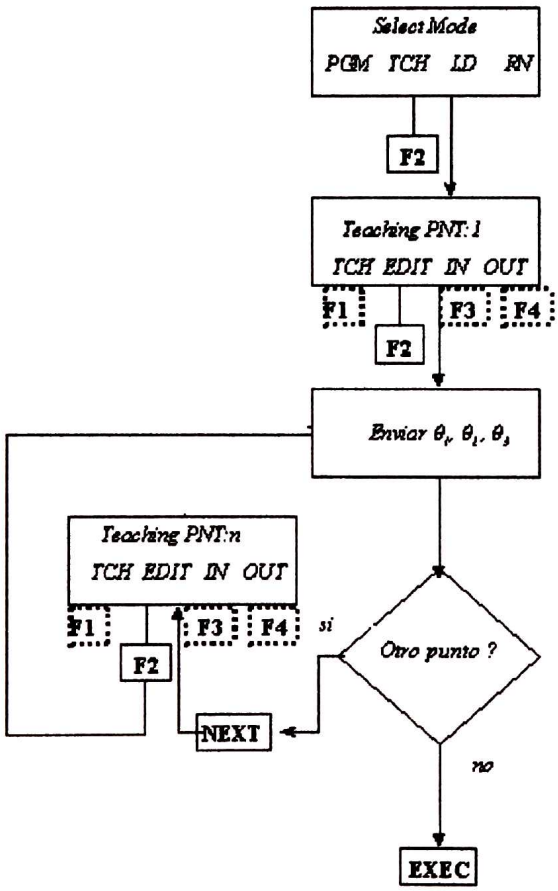


Fig. III.6 Comunicación de valores articulares al robot

El puerto serie de la PC es el medio de comunicación entre la PC y el robot, el cual permite programar este último. Esta comunicación es bidireccional y utiliza el código ASCII en un formato serie asíncrono vía la interface estándar RS 232 C. El protocolo de comunicación empleado es el mostrado por la tabla III.3 [5].

Baud rate	9600
Bits de datos	8
Paridad	Ninguna
Bits de *STOP*	1

Tabla III.3 Formato de comunicación PC- Robot

III.5 Conclusiones

En este capitulo se describieron los aspectos relevantes, de uno de los elementos que conforman el contexto de trabajo de esta tesis: el robot *Unimation S-103.*
En el primer apartado, se mencionaron los elementos básicos que conforman a dicho robot: el brazo mecánico y el sistema de comunicación-control C/ROS.
Posteriormente, se resolvió el problema de la Cinemática Directa del robot, utilizando el método de solución propuesto por Denavit-Hartenberg. Por otro lado, se mencionaron los problemas inmersos en los métodos de solución analítica (propuestos en la literatura), al problema de la Cinemática Inversa. En este sentido, el presente trabajo propone el uso de Redes Neuronales Artificiales (RNA), para la solución de dicho problema, apoyado por las propiedades que éstas exhiben: capacidad de aproximación, generalización, unicidad de solución y complejidad computacional lineal.
Considerando que el robot *Unimation S-103* es de arquitectura cerrada al usuario y debido a la necesidad de manejar información extereoceptiva (proveniente de un sistema de VA), para la realización de tareas de ensamble, en la sección III.4 se describió el formato utilizado por C/ROS. Dicho formato, representa el protocolo necesario de comunicación PC-Robot.

Capítulo IV. Sistema de Visión Artificial

IV.1 Introducción

El objetivo de un sistema de percepción para robots industriales basado en sensores de visión, es transformar la imagen del espacio de trabajo, proporcionada por la cámara, en una descripción de los elementos presentes dentro del entorno del robot. Dicha descripción (vector de atributos que dentro de un espacio de caracterización permite la diferenciación entre clases de objetos para su reconocimiento), debe contener toda la información necesaria para que el sistema de control establezca los movimientos necesarios del robot que permitan la ejecución de la tarea programada.

Dos propiedades fundamentales caracterizan a los sistemas integrados Robot-Visión [7]:

1. Capacidad de reconocimiento, localización y descripción de objetos por medio de técnicas avanzadas de procesamiento de información.

2. Capacidad de realización automática de tareas de interacción inteligente, robot-escena con base en la información obtenida en el proceso anterior.

La integración Robot-Visión no sólo es un problema de cómo utilizar la información sensorial sino que información visual debe ser extraída de las imágenes.

Considerando que en este trabajo se pretende que el robot manipule objetos cilíndricos de 5cm. de altura y color uniforme, es suficiente trabajar con imágenes binarias tales que, mediante atributos obtenidos a partir del contorno de los objetos, es posible el reconocimiento y localización de partes, dentro del espacio de trabajo del robot. En este sentido, para el caso del reconocimiento de objetos en la escena, los objetos utilizados en este trabajo se describen mediante los momentos invariantes de Hu [14] y el factor de compactación (sección § IV.6). Dichos invariantes, se seleccionaron con base en su adaptación al problema planteado en este trabajo así como por su mayor velocidad de procesamiento presentada, en comparación con otras estrategias existentes en la literatura [7].

Para alcanzar los objetivos planteados en este trabajo, nuestro sistema de visión artificial (VA) (fig. IV.1), está conformado por las siguientes etapas:

1. Captación de la imagen del entorno significativo del robot, además de su conversión analógico/digital con el propósito de ser procesada por un sistema de cómputo.
2. Filtraje de la imagen con el objetivo de corregir los efectos de iluminación ó para realzar determinados rasgos de los objetos presentes en la imagen.
3. Extracción de rasgos significativos de las piezas ú objetos.
4. Descripción y localización de la posición así como la orientación de los objetos presentes en la imagen, con objeto de establecer los servomecanismos visuales para su manipulación.
5. Obtención de las características ó atributos que modelan a cada objeto segmentado, presente en la imagen (proceso de reconocimiento).

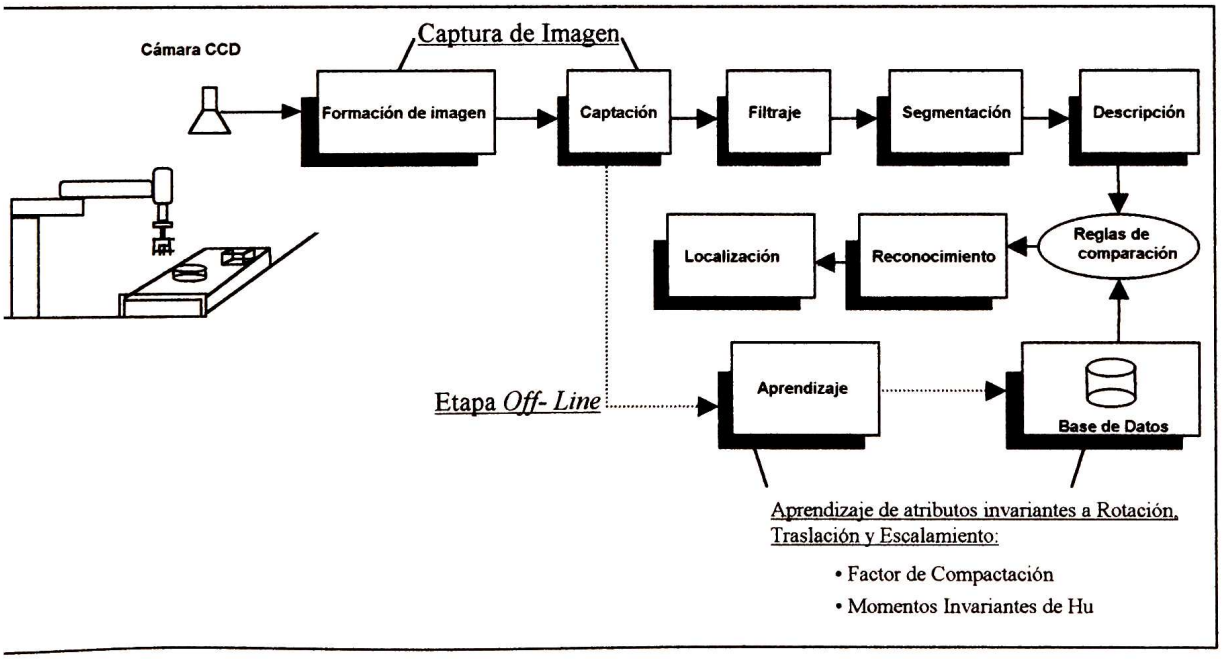


Fig. IV.1 Etapas del procesamiento de imagen del sistema de visión artificial implementado.

IV.2 Captura de imagen

En esta etapa se distinguen tres procesos: *formación*, *captación* y *digitalización* de la imagen [8]. En el proceso de *formación* de imágenes, la información visual sobre un entorno en tres dimensiones (3D), contenida en la luz reflejada por los objetos que constituyen dicho entorno, es transformada en una imagen bidimensional (2D), sobre el plano imagen del dispositivo utilizado.
En el proceso de *captación*, esta imagen óptica es transformada en una señal eléctrica continúa proporcional a la brillantez de dicha imagen, a través de un transductor que permite la obtención de la señal compuesta de vídeo.
Dicha señal analógica es transformada en señal digital mediante el proceso de *digitalización*, el cual, además, almacena esta información en la memoria de la tarjeta de adquisición de imágenes, con el propósito de poder ser procesada posteriormente por el sistema de VA.
En el sistema de visión utilizado en este trabajo, se utiliza como elemento formador y captador de imágenes, una cámara de vídeo de estado sólido acoplado por carga (CCD) [9], y una tarjeta de digitalización y adquisición de imágenes (*OCULUS -200*), para el proceso de digitalización y almacenamiento. Básicamente, el *CCD* genera una señal eléctrica con amplitud proporcional a la luminosidad de la escena. Dicha señal eléctrica, es captada, digitalizada y almacenada por la tarjeta *OCULUS -200*, cuya función es realizar una digitalización espacial completa en dos dimensiones de tal manera que exista una proyección de la imagen en una matriz de puntos. Esta imagen obtenida, es formada por una matriz de 400x400x128 pixeles que se almacena en la memoria de la tarjeta digitalizadora. Para propósitos de esta tesis, debido a la necesidad de requerir una gran velocidad de procesamiento de imagen, únicamente se considera un tamaño de matriz de 200x200x128. En la figura IV.2 se muestra el proceso de captura de la imagen.

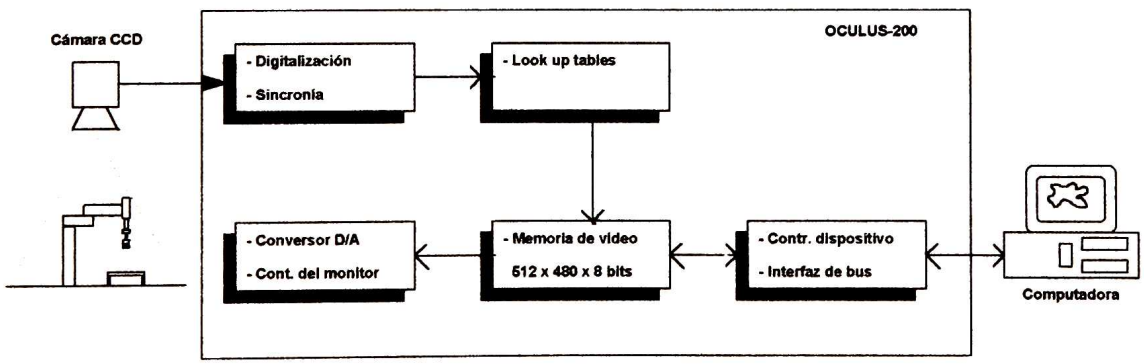


Fig. IV.2 Diagrama a bloques del proceso de captura de imagen

IV.3 Segmentación

Considerando que en este trabajo es suficiente el manejo de imágenes binarias, el siguiente proceso aplicado a nuestra imagen capturada es la *segmentación*. Esta etapa es un proceso mediante el cual una imagen es particionada en unidades significativas (segmentos), en donde cada una de las partes así obtenidas puede ser un objeto ó una región de interés particular. En los sistemas de VA, la segmentación permite agrupar los pixeles de una imagen en regiones homogéneas, de acuerdo con el valor de una cierta característica, para su posterior descripción y reconocimiento. La imagen segmentada resultante contiene los inicios de una interpretación dependiente del dominio, motivo por el cual es importante considerar los modelos de descripción de los objetos que serán utilizados en fases posteriores del sistema de VA.
En el procesamiento de imágenes monocromáticas, existen algoritmos de segmentación basados en una de las dos propiedades básicas de los valores del nivel de gris: *discontinuidad* y *similaridad*. Dentro de la primera categoría, los métodos existentes se basan en dividir una imagen considerando los cambios bruscos de nivel de gris y las principales áreas de interés de esta categoría son la detección de puntos aislados y la detección de líneas y bordes de una imagen. Por otro lado, las técnicas que caen en la segunda categoría se basan en la umbralización, crecimiento de región, división y fusión de regiones.

El método que se emplea en el contexto de esta tesis se encuentra en la segunda categoría y se conoce como *binarización* (ver algoritmo en Apéndice A), cuyo proceso base es la *umbralización*. Para la elección de este proceso, se consideró que los objetos por manipular tienen un color uniforme, por lo que existe un buen contraste entre los objetos de interés y el fondo. Además, el trabajo con imágenes binarias, permite una mayor velocidad de cómputo en el análisis de las mismas. Matemáticamente, el proceso de Segmentación se encuentra definido de la siguiente manera [11]:

Definición IV.1. *Sea:*

f(x,y): función que representa la imagen original.
g(x,y): función de binarización.
T: El valor de umbral.
Go: Representa el nivel de gris de todos los objetos presentes en escena.
G_b *: Representa el nivel de gris del fondo de la imagen.*

De tal manera que la operación de umbralización está definida por:

$$g(x,y) = \begin{cases} G_o & \text{si } f(x,y) > T \\ G_b & \text{si } f(x,y) \leq T \end{cases} \qquad \text{IV.1}$$

IV.4 Filtraje de Imágenes

Una vez que la imagen capturada por nuestro sistema de VA ha sido binarizada y considerando que, el proceso de formación y adquisición de imágenes digitales, implícitamente contiene variaciones de luminosidad (ruido), que corrompen la información original, la siguiente etapa implementada en nuestro sistema de VA es el filtraje de imagen.

El principal objetivo de las técnicas de preprocesamiento de la imagen, es procesar una imagen de tal forma que resulte más adecuada que la original. Para ello, estas técnicas corrigen las degeneraciones introducidas en las etapas de captación y formación de imágenes tales como: efectos de iluminación, aberraciones ópticas, variaciones de la reflectancia de los objetos, muestreo, cuantización, etc.

Existe una gran cantidad de técnicas de mejoramiento de imagen y de manera general se pueden clasificar en dos categorías [10]: métodos en el dominio espacial y métodos en el dominio de la frecuencia. El procesamiento en el dominio espacial se refiere al propio plano de la imagen, y las técnicas que caen en esta categoría se basan en la manipulación directa de los pixeles de la imagen. Por otro lado, las técnicas que caen en la categoría del dominio de la frecuencia se basan en la modificación de la transformada de Fourier de una imagen.

Como en este trabajo los problemas se limitan a ruido del tipo *sal* y *pimienta*, se utiliza un método de preprocesamiento simple en el dominio espacial, conocido como *erosión binaria* (ver algoritmo en Apéndice A). La erosión binaria es una operación morfológica que consiste en la eliminación de pixeles de la capa más externa de una región (considerando que esta únicamente tiene dos niveles de grises: blanco y negro), así como en correlacionar todos los pixeles de una imagen mediante el uso de un *elemento estructurante* [11]. Matemáticamente, la erosión binaria se define de la siguiente manera [11]:

Definición IV.2: *Sean X y B, región y elemento estructurante respectivamente, conjuntos en un espacio Euclidiano bidimensional. Sea* B_x *la traslación de B tal que su origen se encuentre dentro de X. Entonces la erosión de X por B se define como el conjunto de todos los puntos x tal que B se encuentra en X, es decir:*

$$\text{Erosión} : \mathbf{X} \ominus \mathbf{B} = \{x \ / \mathbf{B}_x \subset \mathbf{X}\} \qquad \text{IV.2}$$

En la figura IV.3 se observa un ejemplo del proceso de *erosión binaria*, considerando el elemento estructurante utilizado, el cual fue seleccionado con base en el tipo de objetos que se pretenden manipular en esta aplicación.

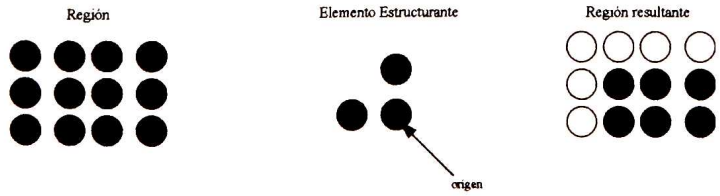


Fig. IV.3 Proceso de Erosión.

IV.5 Descripción

Una vez que la imagen se encuentra segmentada en regiones, el siguiente paso es *describir* cada uno de estas regiones en forma adecuada para su posterior reconocimiento. Para esto, es necesario definir la forma de representar cada región presente en la imagen, para lo cual se tienen dos posibilidades:

1. Considerar solamente sus características externas (el contorno del objeto).
2. Describir cada región en términos de sus características internas (total de pixeles que conforman al objeto).

La elección de un esquema ú otro, está determinada por la aplicación misma, y de manera general se elige una representación externa cuando el objetivo principal se centra en las características de forma. Por otro lado, se utiliza una representación interna cuando el interés se centra en propiedades como el color y la textura.
En esta tesis, se utiliza una representación externa considerando que el contorno es una forma sencilla de representar los objetos a manipular por el robot (ver algoritmo en Apéndice A), además de que permite una mayor velocidad de cómputo.

Considerando esta representación externa, se hace uso de un algoritmo de seguimiento de contornos para regiones 8-conexas [11], en el cual se adopta un sentido de recorrido igual a las manecillas del reloj y el que a su vez proporciona información correspondiente a los atributos de cada objeto presente en la imagen y mediante los cuales se pueden reconocer y localizar los objetos dentro del espacio de trabajo del robot.

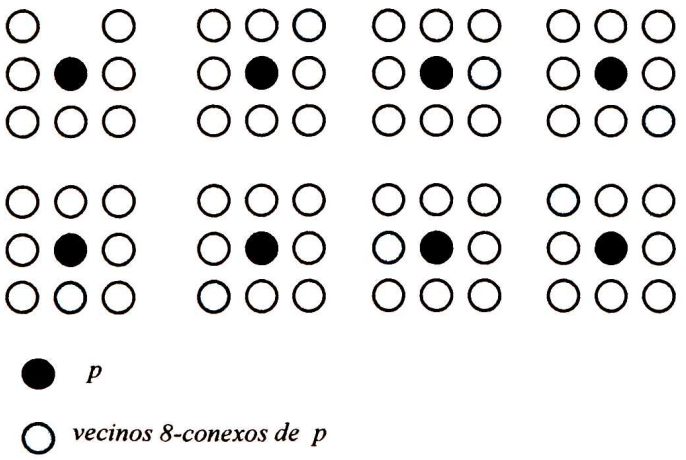


Fig.IV.4 Vecinos 8-conexos del punto *p*.

IV.6 Reconocimiento

Para el reconocimiento de objetos dentro del espacio imagen de la cámara, se requiere tener previamente establecido y almacenado en una tabla de datos (en nuestro caso, en una hoja de Excel®), un vector de atributos representativos (ecuación IV.24), de cada uno de los objetos a manipular con la finalidad de poder diferenciar cada clase ú objeto, presente en el espacio de trabajo del robot. Para esta aplicación, se seleccionaron como atributos los *momentos invariantes de Hu* [11] y el *factor de compactación*, debido a la invariabilidad que presentan a operaciones de rotación, traslación y escala en los objetos, así como también por la velocidad de los algoritmos existentes en el cálculo de estas instancias [7].
Cabe mencionar, que estos valores que describen propiamente a cada una de las clases de objetos aquí consideradas, se obtienen directamente utilizando la opción de *Aprendizaje* (sección § IV.7), cuyo proceso consiste en lo siguiente:

1. Considerar la mayor cantidad de muestras posibles de cada clase de objetos.
2. Aplicar un proceso de traslación, rotación y escalamiento por todo el campo visual de la cámara, a cada objeto perteneciente a una clase determinada.
3. Obtener el promedio y la varianza de los momentos invariantes de Hu así como del factor de compactación, para cada una de las muestras utilizadas.
4. Almacenar el vector de atributos obtenido en el paso 3, en la Base de Datos.

Una vez que el sistema se encuentra en línea, el proceso de reconocimiento consiste en:

1. Obtener los momentos *invariantes de Hu* y el *factor de compactación* de cada uno de los objetos presentes en el espacio de trabajo del robot.
2. Comparar (utilizando la distancia Euclidiana), cada uno de los vectores obtenidos en el paso 1, con los vectores de atributos almacenados en la Base de Datos.
3. Identificar el tipo de objeto presente en la escena.

En los siguientes apartados se hace una presentación de los momentos invariantes de Hu a partir de la teoría de la representación de momentos [13], haciendo énfasis en los momentos centrales, normalizados e invariantes a traslación, rotación y escala.

IV.6.1 Momentos

Teorema IV.1. *Representación de momentos [13].*

Sea $f(x,y)$ *una función continua, tal que* $f(x,y) \neq 0$ *solamente en una región finita del plano XY, entonces el conjunto infinito de momentos de dicha región* m_{pq}, $p,q = 0,1,2,...$n *existe y está determinado de manera única por* $f(x,y)$, *e inversamente los momentos* m_{pq} *determinan de manera única la función* $f(x,y)$.

Definición IV.3 [11]

Sea $f(x,y)$ una función continua, acotada y positiva, definida sobre una región R del plano XY, donde el momento de orden $(p+q)$ de $f(x,y)$ se define como:

$$m_{pq} = \iint_R x^p y^q f(x,y)dxdy \qquad \text{IV.3}$$

En nuestro caso, utilizamos la versión discreta de esta definición, la cual está dada por:

$$m_{pq} = \sum_x \sum_y x^p y^q I(x,y) \qquad \text{IV.4}$$

donde:

$I(x,y)$ representa la función definida en el dominio espacial de la imagen $M \text{ x } N (0 \le x \le M,\ 0 \le y \le N)$.

Para su aplicación a imágenes binarias $I(x,y)$ puede tomar solo dos valores (0,1), por lo que, para el caso de imágenes binarias la expresión para el cálculo de los momentos se reduce a:

$$m_{pq} = \sum_x \sum_y x^p y^q \qquad \text{IV.5}$$

en la región en la cual $I(x,y) = 1$.

IV.6.1.1 Momentos de orden cero

Sean m_{pq} los momentos de una región R, donde el área de R es de A pixeles y el centroide de R se localiza en $\{x_c, y_c\}$. Por lo que:

$$A = m_{00} \qquad \text{IV.6}$$

$$xc = \frac{m_{10}}{m_{00}} \qquad \text{IV.7}$$

$$yc = \frac{m_{01}}{m_{00}} \qquad \text{IV.8}$$

IV.6.1.2 Momentos centrales

Sean $\bar{x}, \bar{y}$ las coordenadas del centro de masa de un objeto en la imagen, tal que los momentos centrales de orden $(p+q)$ se encuentran definidos por:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - y)^q \qquad \text{IV.9}$$

Cabe mencionar que estos momentos permanecen invariantes bajo una traslación, sin embargo un cambio de escala sí los modifica.

IV.6.1.3 Momentos centrales normalizados

Los momentos centrales normalizados se encuentran definidos de la siguiente manera:

$$\eta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}} \qquad \text{IV.10}$$

donde:

$$\gamma = (p+q+2)/2$$

Dichos momentos son invariantes a traslación y escala pero la rotación sí los modifica.

IV.6.1.4 Momentos Invariantes

El concepto de momentos invariantes se basa en la teoria de invariantes [14], en donde se estudian las propiedades de ciertas clases de expresiones algebraicas que permanecen invariantes bajo transformaciones lineales, tales como: rotación, traslación y escalamiento. En este trabajo, los momentos invariantes se utilizan como atributos que caracterizan a los objetos presentes en la imagen y que se encuentran en proceso de reconocimiento independientemente de su localización, medida ú orientación.
Para obtener unos momentos invariantes, vía la teoría de invariantes algebraicos, es posible encontrar ciertos polinomios de η_{pq} que permanecen invariantes a las transformaciones lineales anteriormente mencionadas.

En 1962, Hu [11] propuso 7 invariantes, a partir de los momentos centrales normalizados [11], hasta un orden no mayor que 3, los cuales permanecen invariantes bajo traslación, rotación y cambio de escala, cuyas expresiones son las siguientes:

$$\phi_1 = \eta_{20} + \eta_{02} \qquad \text{IV.11a}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \qquad \text{IV.11b}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{02})^2 \qquad \text{IV.11c}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \qquad \text{IV.11d}$$

IV.6.2 Algoritmo de Philips modificado [7]

Para la obtención de los momentos invariantes a través del contorno de los objetos presentes en la imagen, se hace uso del algoritmo de Philips modificado, propuesto en la referencia [7].
Este algoritmo está basado en una versión discreta equivalente del teorema de Green, el cual establece lo siguiente [35]:

$$\iint_{\Omega} \frac{\partial}{\partial y} f(x,y)dx = \oint_{\partial\Omega} f(x,y)dx \qquad \text{IV.12a}$$

$$\iint_{\Omega} \frac{\partial}{\partial x} f(x,y)dy = \oint_{\partial\Omega} f(x,y)dy \qquad \text{IV.12b}$$

cuyas versiones discretas análogas son [14]:

$$\sum_{(x,y)\in\Omega} \nabla_x f(x,y) = \sum_{(x,y)\in\partial\Omega^+} f(x,y) - \sum_{(x,y)\in\partial\Omega^-} f(x,y) \qquad \text{IV.12c}$$

$$\sum_{(x,y)\in\Omega} \nabla_y f(x,y) = \sum_{(x,y)\in\partial\Omega^+} f(x,y) - \sum_{(x,y)\in\partial\Omega} f(x,y) \qquad \text{IV.12d}$$

donde:

Ω denota una región bidimensional.

$\partial\Omega$ su borde o contorno (ver figura IV.5).

$\nabla_x f(x,y) = f(x,y) - f(x-1,y)$, representa el diferencial inferior de la función $f(x,y)$ respecto a x.

$\nabla_y f(x,y) = f(x,y) - f(x,y-1)$ representa el diferencial inferior de la función $f(x,y)$ respecto a y.

Definición IV.4 [7]

Sea Ω una región discreta del plano XY, se define el borde o contorno externo de la región $\partial\Omega$, como aquellos elementos de Ω cuyos vecinos derechos ó inferiores (dirección ESTE y SUR respectivamente) no están en Ω y aquellos elementos no contenidos en Ω cuyos vecinos derechos ó inferiores si lo están (ver fig. IV.5).

Las ecuaciones siguientes expresan matemáticamente la definición anterior.

$$\partial\Omega = \partial\Omega_1^+ \cup \partial\Omega_2^+ \cup \partial\Omega_3^- \cup \partial\Omega_4^- \qquad \text{IV.13}$$

donde:

$$\partial\Omega_1^+ = \{(x,y) : (x,y) \in \Omega, (x+1,y) \notin \Omega\} \qquad \text{IV.14a}$$

$$\partial\Omega_2^+ = \{(x,y) : (x,y) \in \Omega, (x,y+1) \notin \Omega\} \qquad \text{IV.14b}$$

$$\partial\Omega_3^- = \{(x,y) : (x,y) \notin \Omega, (x+1,y) \in \Omega\} \qquad \text{IV.14c}$$

$$\partial\Omega_4^- = \{(x,y) : (x,y) \notin \Omega, (x,y+1) \in \Omega\} \qquad \text{IV.14d}$$

Para su aplicación al cálculo de los momentos discretos, convenientemente se toma:

$$f(x,y) = g(x)y^q \qquad \text{IV.15}$$

$$\nabla_x g(x) = x^p \qquad \text{IV.16}$$

y como

$$\sum_{a}^{b} \nabla_n f(n) = f(b) - f(a-1) \qquad \text{IV.17}$$

la ecuación para $g(x)$ queda de la siguiente manera:

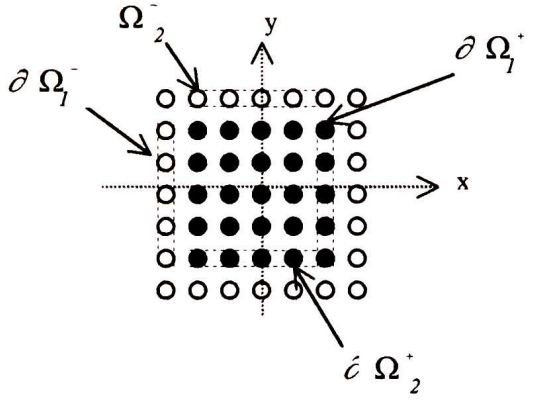
$$g(x) = \sum_{n=0}^{x} n^p \qquad \text{IV.18}$$

En [14] se demuestra que:

$$S_p(x) = \sum_{n=0}^{x} n^p = (x+1)\sum_{j=0}^{p} \frac{\left[S^{-1}\right]_{pj}}{j+1} \sum_{v=0}^{j} [S]_{jv} x^v \qquad \text{IV.19}$$

donde:

S es la matriz de Stirling de primer orden.
S^{-1} es la matriz inversa de Stirling



● $p \in \Omega$
○ $p \notin \Omega$

Fig. IV.5 El contorno externo dado por la Definición IV.4

La computación de los momentos invariantes para cualquier tipo de objeto (incluyendo aquellos que contienen n hoyos), integrando tanto respecto a X como a Y, se obtienen evaluando las siguiente ecuaciones[7]:

$$m_{pq} = \sum_{(x,y)\in\partial\Omega_x^+(cont\_ext)} S_p(x)y^q - \sum_{(x,y)\in\partial\Omega_x^-(cont\_ext)} S_p(x)y^q + \sum_{i=1}^{n}\left(\sum_{(x,y)\in\partial\Omega_x^+(cont\_int[i])} S_p(x)y^q - \sum_{(x,y)\in\partial\Omega_x^-(cont\_int[i])} S_p(x)y^q \right) \qquad \text{IV.20}$$

$$m_{pq} = \sum_{(x,y)\in\partial\Omega_y^+(cont\_ext)} S_q(y)x^p - \sum_{(x,y)\in\partial\Omega_y^-(cont\_ext)} S_q(y)x^p + \sum_{i=1}^{n}\left(\sum_{(x,y)\in\partial\Omega_y^+(cont\_int[i])} S_q(y)x^p - \sum_{(x,y)\in\partial\Omega_y^-(cont\_int[i])} S_q(y)x^p \right) \qquad \text{IV.21}$$

donde:

cont_ext se refiere al contorno externo del objeto.
Cont_int se refiere al contorno interno del objeto (ver fig. IV.6).

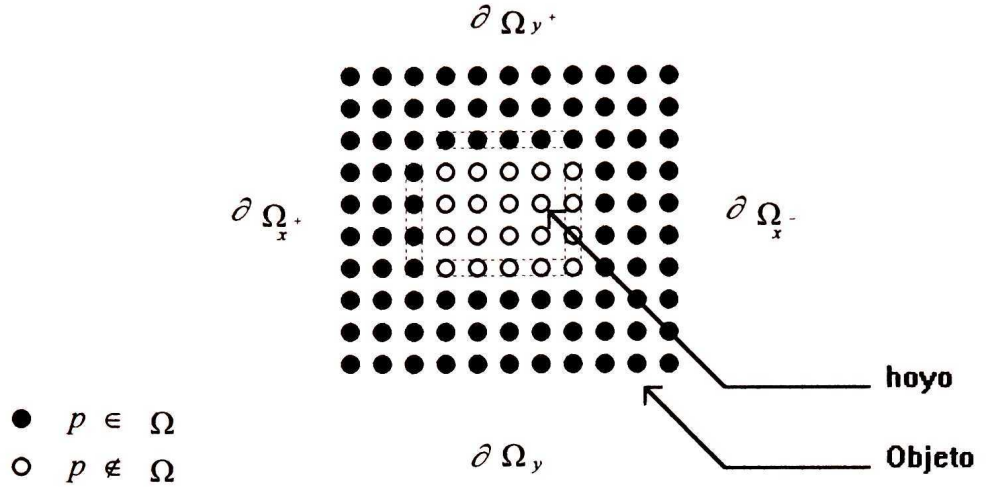


Fig. IV.6 Contorno Interior de un objeto

En resumen, siguiendo el desarrollo matemático en [7], se aplican las siguientes tablas para el cálculo de los diferentes momentos hasta un orden 3:

Momento	*Valor*
m_{00}	$x+1$
m_{01}	$(x+1)y$
m_{02}	$(x+1)y^2$
m_{03}	$(x+1)y^3$
m_{12}	$\frac{1}{2}(x+1)xy^2$
m_{10}	$(y+1)x$
m_{11}	$\frac{1}{2}(y+1)xy$
m_{20}	$(y+1)x^2$
m_{21}	$\frac{1}{2}(y+1)yx^2$
m_{30}	$(y+1)x^3$

Tabla IV.1 Momentos discretos para las secciones positivas del contorno

Momento	*Valor*
m_{00}	x
m_{01}	xy
m_{02}	xy^2
m_{03}	xy^3
m_{12}	$\frac{1}{2}(x-1)xy^2$
m_{10}	yx
m_{11}	$\frac{1}{2}(y-1)yx$
m_{20}	yx^2
m_{21}	$\frac{1}{2}(y-1)yx^2$
m_{30}	yx^3

Tabla IV.2 Momentos discretos para las secciones negativas del contorno.

Momento	*Valor*
m_{00}	1
m_{01}	y
m_{02}	y^2
m_{03}	y^3
m_{12}	xy^2
m_{10}	x
m_{11}	yx
m_{20}	x^2
m_{21}	yx^2
m_{30}	x^3

Tabla IV.3 Momentos discretos para píxels que pertenecen a las dos secciones.

Cabe mencionar que durante el seguimiento del contorno se realiza un análisis de la pertenencia de un pixel a las diferentes secciones del contorno del objeto. Para dicho análisis se tiene la siguiente definición:

Definición IV.5 [7].
Sea Ω, *una región discreta,* $\partial\Omega$ *su contorno y sea un punto* $\mathbf{p} \in \partial\Omega$, *se llama* ***dirección actual*** (a_i) *del punto* **p** *a la dirección en que es encontrado el píxel* **p+1** *perteneciente al contorno de la región, y* ***dirección anterior*** (a_{i-1}) *del punto* **p** *a la dirección en que fue encontrado dicho punto* **p** *(ver fig. IV.7).*

A partir de la Definición anterior, la pertenencia de un píxel a las diferentes secciones del contorno se determina a través del árbol de decisión mostrado en la figura IV.8 [7]. De esta manera el cálculo de los momentos discretos se resume en seguir el contorno de los objetos, determinando en cada píxel las secciones del contorno a las que pertenece y aplicando las expresiones dadas por las Tablas IV.1 a IV.3 según sea la pertenencia. Esto permite una aplicación directa a regiones representadas por su código de cadena, conocido como código de Freeman [11]. Partiendo de valores de x y y tomados al azar, lo cual es posible dado que los invariantes lo son a traslación, se sigue el contorno de la región, teniendo en cuenta variar x y y de acuerdo al valor que va tomando a_{i-1}, según se ilustra en la Tabla IV.4.

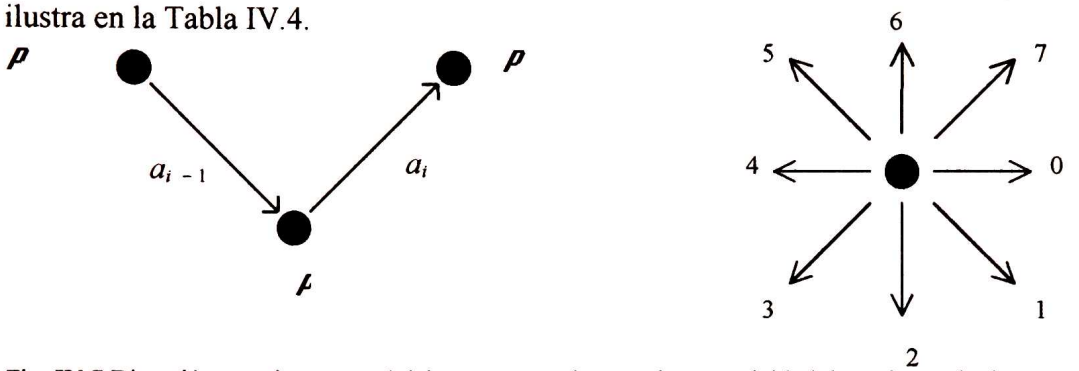


Fig. IV.7 Dirección anterior y actual del punto p con base en la conectividad de regiones de tipo 8-conexo.

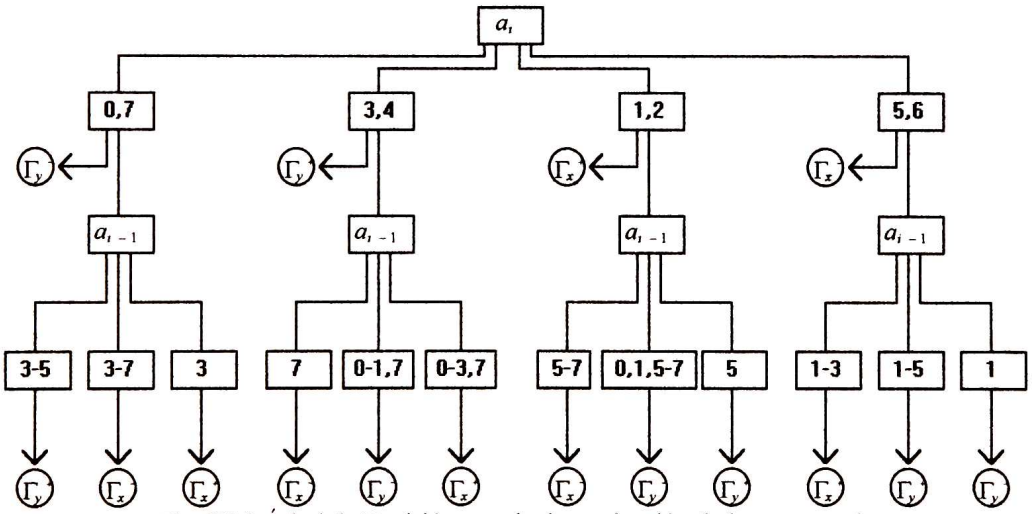


Fig. IV.8 Árbol de Decisión para la determinación de la pertenencia

a_{i-1}	x	y
0	+1	0
1	+1	+1
2	0	+1
3	−1	+1
4	−1	0
5	−1	−1
6	0	−1
7	+1	−1

Tabla IV.4 Variación de x y y en función de a_{i-1}.

Una vez computados los momentos hasta un orden 3, se obtienen los momentos invariantes ϕ_1 y ϕ_2 de Hu, que junto al factor de compactación, conforman el vector de atributos que identifica al objeto en cuestión.
El siguiente paso dentro de la fase de Reconocimiento es realizar la comparación de los objetos presentes en el espacio de trabajo del robot con los vectores de atributos almacenados en una base de datos, creada previamente durante un proceso de aprendizaje del sistema de visión, del cual se hablará en la siguiente sección.
Al identificarse la presencia de un objeto en la imagen cuyo vector de atributos corresponde con alguno de los que se encuentran almacenados en la base de datos, se detiene el proceso de análisis de la imagen, con la finalidad de determinar su centro de masa y su orientación en coordenadas imagen, de acuerdo a las siguientes ecuaciones:

Centroide:

$$xc = \frac{m_{10}}{m_{00}} \qquad \text{IV.22a}$$

$$yc = \frac{m_{01}}{m_{00}} \qquad \text{IV.22b}$$

Orientación:

$$\theta = \frac{1}{2} tg^{-1} \frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \qquad \text{IV.22c}$$

donde:

μ_{11}, μ_{20} y μ_{02} son momentos centralizados obtenidos mediante la ecuación IV.9.

IV.6.3 Factor de Compactación

El factor de compactación [11], es un atributo geométrico de los objetos muy empleado en tareas de clasificación por su invariabilidad a las transformaciones lineales anteriormente mencionadas y el cual se encuentra definido como:

$$f_{comp} = \frac{(Perímetro)^2}{Area} \qquad \text{IV.23}$$

donde:

- f_{comp} : Factor de compactación.
- *Perímetro*, *Area* : Perímetro y área del objeto.

Para la obtención del perímetro del objeto, se tiene en cuenta la dirección en que son encontrados los píxeles vecinos en el contorno del objeto (fig. IV.4), así como sus dimensiones, para lo cual se aplica la siguiente tabla con el objeto de conocer los incrementos que deben darse a un acumulador iniciado en cero y que al término del recorrido del contorno contendrá el valor del perímetro.

Dirección	Valor agregado
Este	1.44
Sureste	1.04
Sur	0.88
Suroeste	1.04
Oeste	1.44
Noroeste	1.04
Norte	0.88
Noreste	1.04

Tabla IV.5 Incrementos al perímetro.

Por otro lado, el área del objeto el cual corresponde al momento de orden cero, es obtenido a través del cálculo de los momentos invariantes, dado por la ecuación IV.6.

IV.7 Aprendizaje del Sistema de Visión

Para llevar a cabo la identificación y reconocimiento de objetos presentes en la escena, se hace necesario que el sistema de VA tenga un conocimiento previo de los posibles objetos que se pueden presentar. Es éste el objetivo de la etapa de aprendizaje.

En esencia el entrenamiento consiste en asociar a cada clase $\mathbf{C}_i$ de objetos un vector de atributos $\mathbf{x}_i$, resultando un modelo ($\mathbf{C}_i$, $\mathbf{x}_i$) correspondiente a cada clase u objeto. En el sistema creado, cada clase $\mathbf{C}_i$ es representada por el nombre del objeto y el vector de atributos $\mathbf{x}_i$ (f_{comp}, ϕ_1, ϕ_2).

$$(\mathbf{C}_i, \mathbf{x}_i) = \begin{bmatrix} Nombre \\ f_{comp} \\ \phi_1 \\ \phi_2 \end{bmatrix} \qquad \text{IV.24}$$

Dadas las imperfecciones de todo sistema de procesamiento de imágenes, la medición del vector de atributos de una misma clase sufre variaciones de una imagen a otra, lo que motiva la imposibilidad de establecer valores constantes a sus componentes, por lo que se hace necesario tomar el mayor número de muestras posibles, trasladando, rotando y escalando el objeto bajo aprendizaje por todo el campo visual de la cámara.
Una vez que el objeto que se desea enseñar al sistema de visión ha sido mostrado un número de veces considerable, se obtienen la media y la varianza de cada componente

del vector de atributos, y éstos junto con el nombre del objeto es almacenado en una base de datos, accesada también, durante la etapa de reconocimiento.

IV.8 Modelo Cinemático Inverso del Sistema de Visión

En aplicaciones de Robótica, en particular, en tareas de ensamble, el objetivo es determinar la posición y orientación de las piezas existentes en la mesa de trabajo con respecto a la base del robot (coordenadas cartesianas) y una vez obtenida esta información se debe aplicar el modelo Cinemático Inverso del robot. Con ayuda de un sistema de VA ya sea fijo ó móvil (para propósitos de esta tesis es fijo), se puede determinar la posición y orientación de esas piezas con respecto a la cámara, por lo que, para determinar las coordenadas cartesianas de dichos objetos, es necesario encontrar un modelo matemático preciso que transforme coordenadas de cámara en coordenadas cartesianas. A dicho modelo matemático se le conoce como el modelo *Cinemático Inverso del Sistema de VA.*

Proposición IV.1
Sea un objeto (ver fig. IV.9) cuya ubicación en el espacio cartesiano (mesa de ensamble), está dada por el vector:

$$\mathbf{p^b} = \begin{bmatrix} \mathbf{x}_p \\ \alpha \end{bmatrix}, \quad \mathbf{x}_p \in \Re^3, \alpha \in \Re \qquad \text{IV.25}$$

en donde $\mathbf{x}_p$ y α son respectivamente su posición y su orientación cartesianas.

La posición y la orientación de este objeto en un sistema de VA de 2D, cuyo plano imagen es paralelo a la mesa de trabajo, está dada por:

$$\mathbf{p^i} = \begin{bmatrix} \mathbf{x}^i_p \\ \theta \end{bmatrix}, \mathbf{x}^i_p \in \Re^2, \theta \in \Re \qquad \text{IV.26}$$

en donde $\mathbf{x}^i_p$ es el vector de coordenadas imagen (2D) del objeto considerado y θ es el momento de primer orden. Además, la posición y la orientación de este objeto con respecto a la cámara (cuyos ejes de referencia se encuentran en el plano imagen), está dada por:

$$\mathbf{p^c} = \begin{bmatrix} \mathbf{x}^c_p \\ \theta \end{bmatrix}, \mathbf{x}^c_p \in \Re^3, \theta \in \Re \qquad \text{IV.27}$$

donde $\mathbf{x}^c_p$ y θ representan al vector de coordenadas de cámara y a la orientación del objeto, respectivamente.

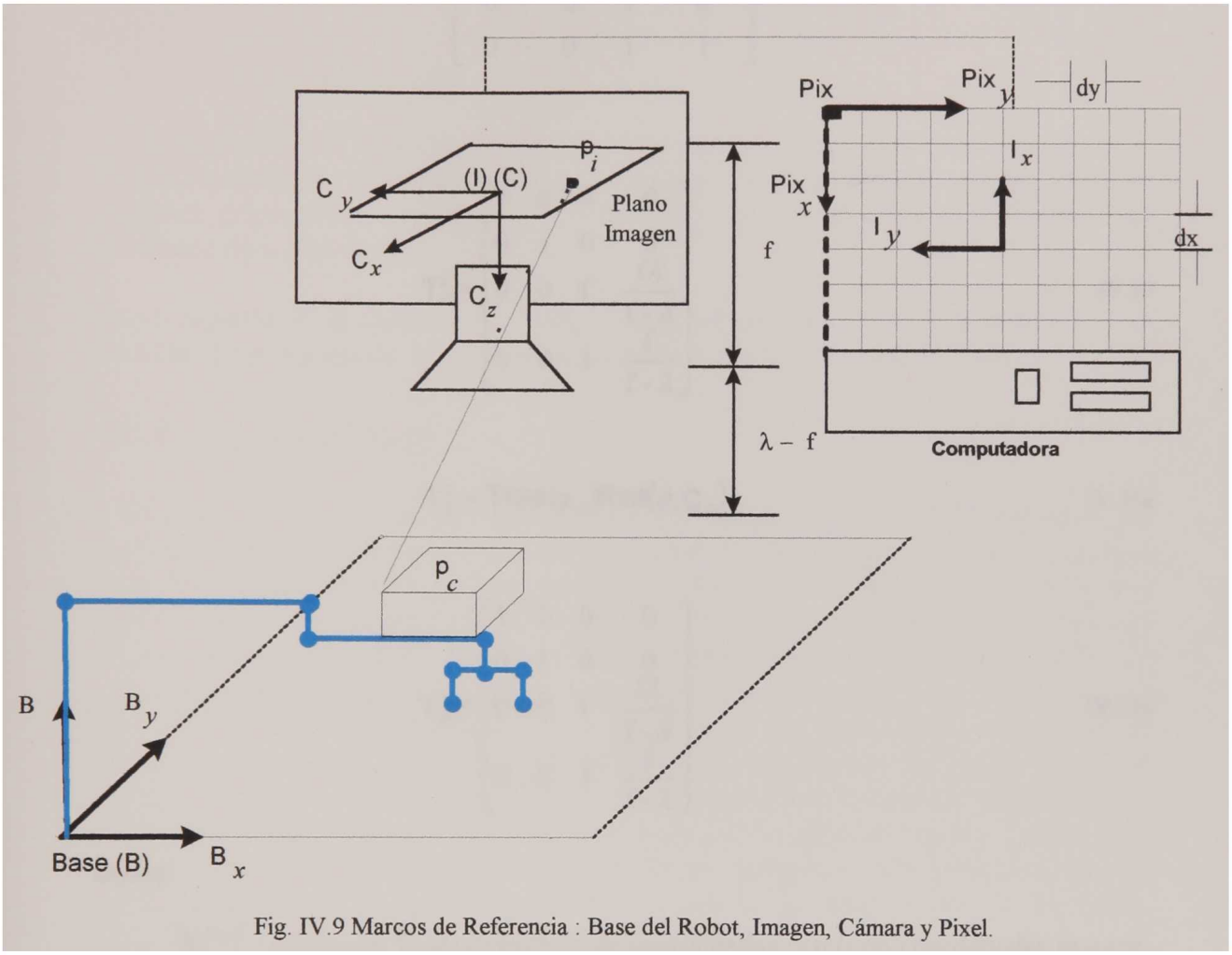


Fig. IV.9 Marcos de Referencia : Base del Robot, Imagen, Cámara y Pixel.

De modo que la relación entre las coordenadas cartesianas (B), y las coordenadas cámara (C), de un objeto (visto como un objeto *puntual* y considerando que las coordenadas imagen no se obtienen directamente de la cámara sino a través del uso de una computadora que procesa la imagen), está dada por la siguiente transformación:

$$\mathbf{p}^{\mathbf{b}} = \mathbf{T}_b^c \mathbf{T}_c^i \mathbf{T}_i^{pixel} \mathbf{p}^{pixel} \qquad \text{IV.28}$$

$$\mathbf{T}_i^{pixel} = \mathbf{S}(dx, dy, 1, 1)\mathbf{Tras}\left(\frac{m+n}{2}\right)\mathbf{Rot}(\pi, pixel_z) \qquad \text{IV.29a}$$

$$\mathbf{T}_i^{pixel} = \begin{bmatrix} -dx & 0 & 0 & \frac{m*dx}{2} \\ 0 & -dy & 0 & \frac{n*dy}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad \text{IV.29b}$$

$$\mathbf{T}_c^i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f & \frac{f\lambda}{f-\lambda} \\ 0 & 0 & 1 & \frac{f}{f-\lambda} \end{bmatrix} \qquad \text{IV.30}$$

$$\mathbf{T}_b^c = \mathbf{Tras}(\mathbf{x}_o)\mathbf{Rot}(\phi, \mathbf{C}_z) \qquad \text{IV.31a}$$

$$\mathbf{T}_b^c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f & \frac{f\lambda}{f-\lambda} \\ 0 & 0 & 1 & \frac{f}{f-\lambda} \end{bmatrix} \qquad \text{IV.31b}$$

donde:

$\mathbf{T}_i^{pixel}$: Matriz de Transformación de coordenadas *pixels* en coordenadas imagen.
S : Matriz de transformación de escala.
Tras : Matriz de traslación.
Rot : Matriz de rotación de un eje referencial con respecto al otro.
$M*N$: Tamaño de la imagen en pixeles .
$\mathbf{T}^i_c$: Transformación de coordenadas imagen en coordenadas de cámara. (Matriz de transformación de *perspectiva inversa*).
$\mathbf{T}^c_b$: Transformación de coordenadas de cámara en coordenadas cartesianas.
$\mathbf{X}_o$: Vector de traslación del eje referencial de cámara a eje referencial (B).
f : Distancia focal.
λ : Distancia entre el plano imagen y el plano de ensamble de objetos.

Cabe mencionar que al proceso de obtener de forma experimental la matriz de transformación $\mathbf{T}^c_b$, se le conoce con el nombre de *calibración de cámara*.

Considerando los valores necesarios en la ecuación IV.28, se puede observar que el problema de encontrar de forma analítica la transformación de coordenadas pixeles en coordenadas cartesianas de una manera precisa, radica en la *imposibilidad* de medir con exactitud varios de los parámetros que están involucrados en esta transformación, entre los que destacan la posición-orientación de la cámara con respecto a la base del robot y el factor de escala en la transformación de coordenadas pixeles en coordenadas imagen.

En este sentido, el presente trabajo propone resolver dicho problema mediante la utilización de Redes Neuronales Artificiales (RNA), con la justificación de que éstas, no requieren del conocimiento de ningún parámetro ya que se basan en el aprendizaje de relaciones de entrada-salida.

En el capítulo V se describe el diseño e implantación de la RNA propuesta, en la solución del problema de la Cinemática Inversa de nuestro Sistema de Visión.

IV.9 Conclusiones

A lo largo de este capítulo, se mencionaron las diferentes etapas que conforman un Sistema de Visión Artificial (VA): captura, filtraje de ruido, segmentación, descripción y reconocimiento. Además, se mencionó que dentro del contexto de este trabajo, éste último proceso (reconocimiento), tiene como propósito identificar la posición (dado en coordenadas imagen), de los objetos a manipular por el robot así como la clase a la cual pertenece cada objeto presente en el espacio de trabajo del robot.
Por otro lado, en el apartado IV.8 se presentaron las matrices necesarias en la transformación conocida como la Cinemática Inversa del Sistema de Visión. Como se mencionó en dicho apartado, el problema de encontrar una solución analítica a dicha transformación y en particular al problema de la *calibración de cámara*, radica en la *imposibilidad* de medir con exactitud varios de los parámetros que están involucrados en este proceso, razón por la cual, en esta tesis se propone el uso de las Redes Neuronales Artificiales (RNA), como solución a este problema.

Capítulo V.
Sistema de Control Visuomotor basado en RNA.

V.1 Introducción

Las Redes Neuronales Artificiales (RNA), son sistemas de cómputo conformadas por unidades simples de procesamiento de información conocidas como *nodos* ó *neuronas artificiales* (Fig. V.1), cuya salida varía en respuesta a entradas externas a la red. Estos elementos básicos de información se encuentran interconectados a través de *pesos sinápticos* que representan el conocimiento de la RNA en forma numérica. Cabe mencionar que el valor numérico de los pesos sinápticos se actualiza durante el proceso de aprendizaje ó durante la operación misma de la RNA, a través de un algoritmo heurístico ó de optimización.

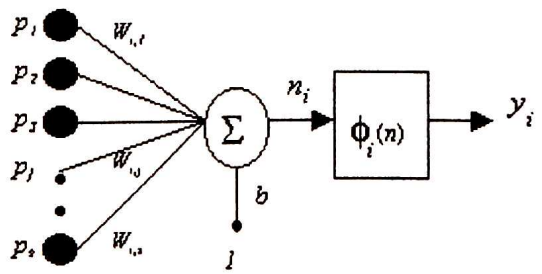


Fig. V.1 Arquitectura básica de una neurona artificial

La ecuación básica de una neurona artificial es:

$$y_i = \phi_i\left(\sum_{j=1}^{n}\left(w_{ij}^T p_j + b_i\right)\right) \qquad \text{V.1}$$

donde:

y_i Valor de salida de la i-ésima neurona.

ϕ_i Función de activación de la neurona.

w_{ij} Ponderación de la conexión entre la j-ésima entrada y la i-ésima neurona .

p_j j-ésimo componente del vector de entrada.

b_i Entrada de polarización de la i-ésima neurona.

En la actualidad existen diversos modelos de RNA [3], los cuales, de manera formal están definidos mediante la siguiente 3-tupla [17]:

$$\text{RNA}=(\mathbf{S},\mathbf{A},\mathbf{T}) \qquad \text{V.2}$$

donde:

S es el conjunto de datos de entrada-salida, y está definido por:

$$\mathbf{S} = \{\mathbf{P},\mathbf{Y}\} \qquad \text{V.3}$$

$\mathbf{P}= \{\mathbf{p}_i \mid \mathbf{p}_i \text{ es el i-ésimo vector de entrada}\}= \{\mathbf{p}_1, \mathbf{p}_2, .., \mathbf{p}_i,.., \mathbf{p}_n\}$

$\mathbf{Y}= \{\mathbf{q}_i \mid \mathbf{q}_i \text{ es el i-ésimo vector de salida}\}=\{\mathbf{q}_1, \mathbf{q}_2,.., \mathbf{q}_i,..., \mathbf{q}_n\}$

A representa al conjunto de parámetros constantes de la RNA (razón de aprendizaje, momentum, etc) .

$$\mathbf{A} = \{a_1, a_2, a_3,., a_i,.., a_m\} \qquad \text{V.4}$$

T representa la topología de la RNA y está dada por:

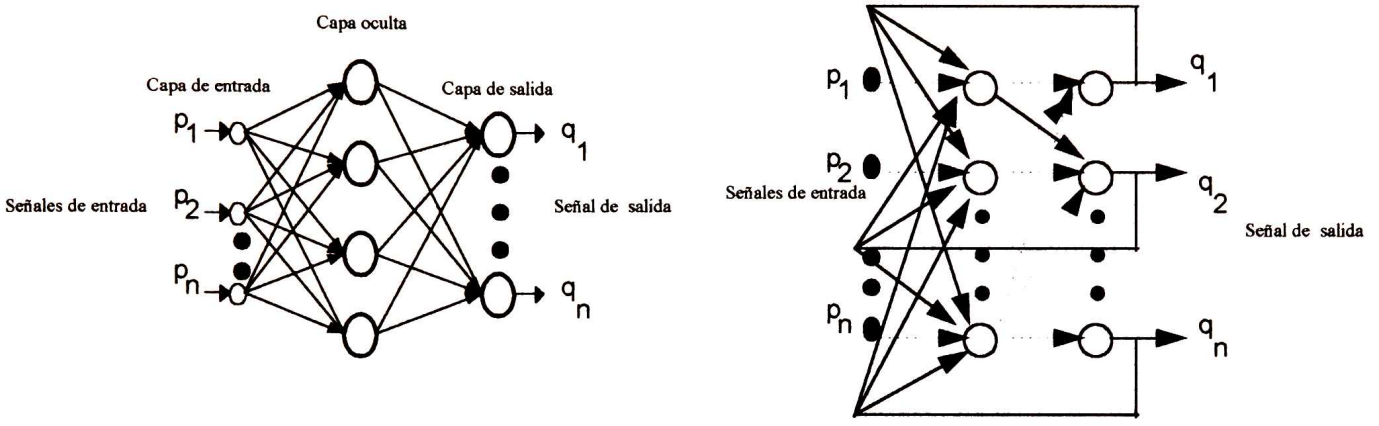
$$\mathbf{T} = \{\mathbf{F},\mathbf{L}\} \qquad \text{V.5}$$

$\mathbf{F}= \{\mathbf{c}_i \mid \mathbf{c}_i \text{ es la i-ésima capa }\}= \{\mathbf{c}_1, \mathbf{c}_2,.., \mathbf{c}_i,..., \mathbf{c}_n\}$

$\mathbf{L}= \{w_{i,j-->k,l} \mid w \textit{ valor} \text{ de la conexión del nodo } j \text{ – capa } i \text{ y el nodo } k \text{ – capa } l\}$

Cabe mencionar que los valores finales de $w_{i,j->k,l}$, se ajustan mediante la utilización de un algoritmo de *aprendizaje* (gradiente descendente, recocido simulado, algoritmos genéticos, L-M, entre otros), cuyo objetivo es definir los valores óptimos de $w_{i,j->k,l}$. (En esta formalización de las RNA, se consideran a los valores de umbralización b_i como valores w con la excepción de que estos son multiplicados por un valor de entrada unitario).

Existen diversas funciones de activación utilizadas en las RNA entre las que destacan: umbral, lineal, tangente hiperbólica, logística, exponencial, gaussiana, sigmoidal, entre otras [20].



V.2a.- *perceptrón multicapas*

V.2b.- RNA recurrente

Fig. V.2 Topologías típicas de RNA.

Algoritmo de aprendizaje

En contraste a la forma de programar los sistemas de cómputo convencionales para que estos desempeñen una función especifica, las RNA requieren una fase de *aprendizaje*. En dicho proceso, los valores de los enlaces entre las neuronas se ajustan a través de un algoritmo Heuristico ó de Optimización, de tal manera que el error de los resultados obtenidos por la RNA, sea mínimo.

Existe una gran diversidad de algoritmos aplicados al aprendizaje de las RNA, y la aplicación de uno ú otro depende del contexto en que se desee aplicar a las RNA. Entre los algoritmos más importantes se encuentran: regla de Hebb [18], regla *Delta Generalizada* ó *Backpropagation* [19], *Algoritmo L-M* [20], regla de Kohonen [21], entre otros.

V.2 RNA y Teoría de aproximación de funciones

Una de las aplicaciones de las RNA, es la aproximación de funciones [20]. En este sentido, las RNA son consideradas como la aproximación uniforme a una función continua no lineal dada por [16]:

$$\mathbf{q} = \left\{ f(\mathbf{x}) \ : \mathbf{D} \subset \Re^n \rightarrow \Re^m \right\} \qquad \text{V.6}$$

dado un conjunto de vectores de entrada - salida $\{\mathbf{x},\mathbf{q}\}$, una precisión arbitraria $f(\mathbf{x},\mathbf{W})$ en $\mathbf{D}$, donde:

- $\mathbf{D}$: Subconjunto compacto en $\Re^n$.
- $\mathbf{W}$: Matriz de valores de los enlaces entre las neuronas (pesos sinápticos), a ajustar mediante algún método heurístico ó de optimización de tal manera que el error de aproximación de los datos de entrada-salida, se minimice.

Cabe mencionar que en el contexto de la teoría de aproximación polinomial, una RNA es considerada como una representación de un conjunto $\mathbf{B}$ de funciones paramétricas $f(\mathbf{p},\mathbf{W})$ y unas reglas de aprendizaje que actualizan la matriz de pesos $\mathbf{W}$ y además evalúan la mejor función paramétrica de $\mathbf{B}$ tal que exista una aproximación con un error menor o igual a ε.
Se han llevado a cabo demostraciones matemáticas rigurosas [22][23][24] para demostrar que las RNA tienen la capacidad para aproximar cualquier función dada por la ecuación V.6, por lo que, las RNA son consideradas *aproximadores universales de funciones* [3].
El teorema siguiente, establece la correspondencia existente entre las RNA y la teoría de aproximación de funciones.

Teorema V.1 [22]
Sea φ una función continua y $G(\mathbf{x},\mathbf{W},\mathbf{b}) : [0,1]^n \rightarrow \Re$ una función parametrizada dada por:

$$G := \varphi_M\left(\mathbf{W}_M \cdots \mathbf{W}_d \varphi_2\left(\mathbf{W}_2\left(\varphi_1\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_d\right) \cdots + \mathbf{b}_M\right) \qquad \text{V.7}$$

donde:

- $\mathbf{W_d}$: Vector de pesos sinápticos en la capa d.
- $\mathbf{X}$: Vector de entrada.
- $\mathbf{b_d}$: Vector de umbrales de las neuronas en la capa d.
- M : No. de capas de la RNA.

Dada una función continua f en $[0,1]^n$ (ó en algún subconjunto compacto de $\Re^n$) y $\varepsilon > 0$, existe un conjunto de valores w_{ij} y b_{ij} tal que:

$$\left[G(\mathbf{x},\mathbf{W},\mathbf{b}) - f(\mathbf{x})\right] < \varepsilon \quad \forall x \in [0,1]^n \qquad \blacklozenge \qquad \text{V.8}$$

V.3 Control Visuomotor conexionista

En él capitulo **II** se mencionaron los diferentes esquemas de control visuomotor y se hizo énfasis en un problema que se presenta en cualquiera de estos esquemas: la calibración de cámara.

Esta situación se presenta debido a que se tiene la necesidad de calcular los valores de las variables articulares en función de los puntos presentes en el espacio de las imágenes. Si bien es posible tener formas analíticas para llevar a cabo las transformaciones necesarias [16], algunos parámetros deben ser calculados ó medidos mediante procedimientos que, generalmente, introducen algunos errores de modelado y de cómputo.
Una de las principales propiedades que exhiben las RNA es que no requieren de ningún modelo matemático ni de parámetros para modelar procesos. Por tal motivo, dentro del esquema de control visuomotor propuesto (figura N°. V.3), la RNA tiene por objetivo aproximar el modelo cinemático inverso del sistema robot-visión (calibración de cámara y cinemática inversa del robot *Unimate S-103*). Es decir, la tarea de la RNA será calcular las coordenadas articulares del robot correspondientes a una lectura en el sistema de visión artificial y con base a esa información, llevar a cabo el posicionamiento relativo pinza-objeto.

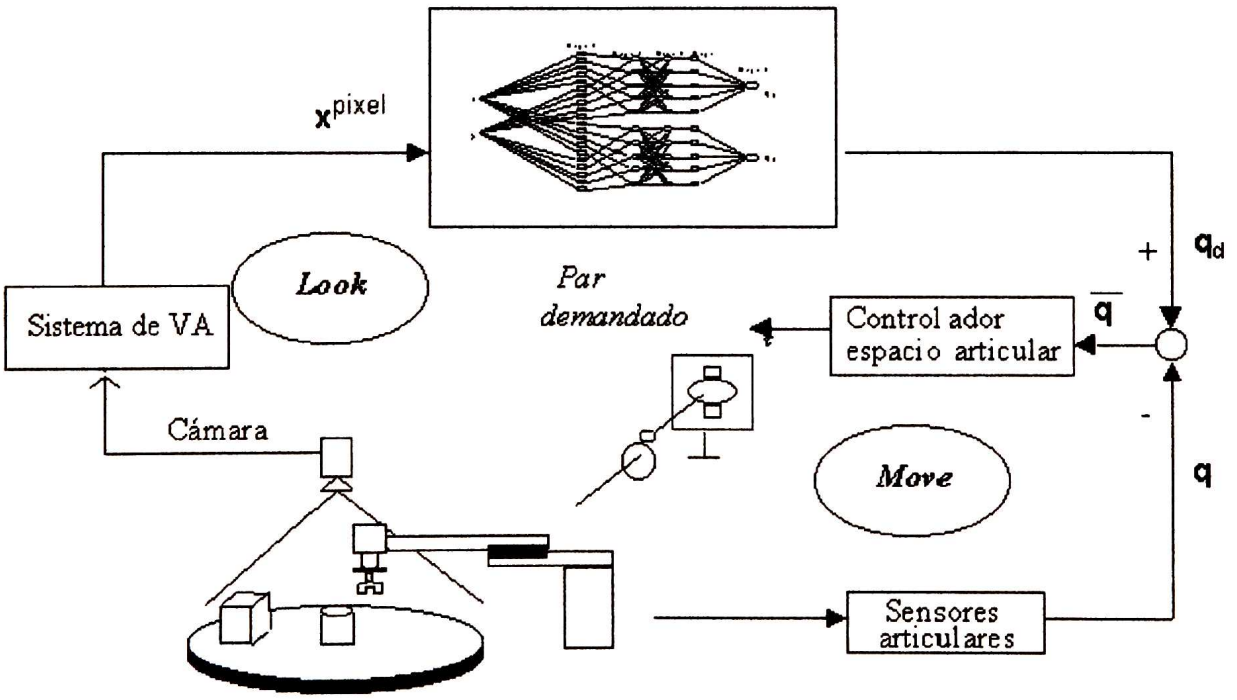


Fig. V.3 Esquema visuomotor propuesto.

V.3.1 Planteamiento del Problema

Sea un objeto estático (ver Fig. IV.9 y Fig. V.4), sobre la mesa de ensamble de un robot tipo SCARA cuya ubicación en el espacio cartesiano, está dada por el vector:

$$\mathbf{p}^{\mathbf{b}} = \begin{bmatrix} \mathbf{x}_p \\ \alpha \end{bmatrix}, \mathbf{x}_p \in \Re^3, \alpha \in \Re \qquad \text{V.9}$$

en donde $\mathbf{x}_p$ y α son respectivamente su posición y su orientación cartesianas.

La posición y la orientación de este objeto en un sistema de VA de 2D, cuyo plano imagen es paralelo a la mesa de trabajo, está dada por:

$$\mathbf{p}^{\mathbf{i}} = \begin{bmatrix} \mathbf{x}_p^i \\ \theta \end{bmatrix}, \mathbf{x}_p^i \in \Re^2, \theta \in \Re \qquad \text{V.10}$$

en donde $\mathbf{x}_p^i$ es el vector de coordenadas imagen (2D) del objeto considerado y θ es el momento de primer orden. Además, la posición y la orientación de este objeto con respecto a la cámara está dada por:

$$\mathbf{p}^{\mathbf{c}} = \begin{bmatrix} \mathbf{x}_p^c \\ \theta \end{bmatrix}, \mathbf{x}_p^c \in \Re^3, \theta \in \Re \qquad \text{V.11}$$

donde $\mathbf{x}_p^c$ y θ representan las coordenadas de cámara y la orientación del objeto.

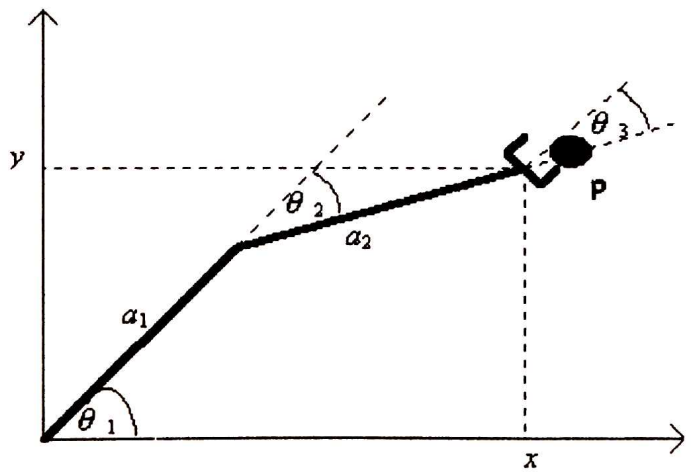


Fig. V.4 Vista superior del plano de ensamble.

De modo que la relación entre las coordenadas imagen y las coordenadas en la mesa de trabajo de un punto está dada por la siguiente transformación:

$$\mathbf{p}_{\mathrm{i}}^{\mathrm{b}} = \frac{f}{z-f} Rot(\phi, z)\left[\mathbf{x}_{\mathrm{p}} + \mathbf{x}_{\mathrm{o}}\right] \qquad \text{V.12}$$

en donde f es la distancia focal de la cámara, $Rot(\phi, z)$ es la matriz de rotación del referencial robot con respecto al referencial imagen, $\mathbf{x}_o$ es el vector de coordenadas de la intersección del plano de trabajo del robot con el eje de la cámara y ϕ es el ángulo que hacen los ejes x de los referenciales imagen y de base del robot. Por su parte, la relación entre la orientación del objeto en los sistemas considerados está dada por:

$$\gamma = \alpha + \phi \tag{V.13}$$

Las ecuaciones (V.12) y (V.13) representan el modelo cinemático directo del sistema de visión, el cual permite calcular las coordenadas imagen de un punto en la mesa del robot a partir de sus coordenadas cartesianas medidas en dicha mesa, conociendo los parámetros cinemáticos del sistema de VA: f, $\mathbf{x}_o$ y ϕ. Mientras que la siguiente ecuación representa el modelo cinemático inverso del sistema de VA.

$$\begin{aligned} \mathbf{x}_b^i &= \frac{z-f}{f} Rot(-\phi, z)\mathbf{x}_b^i - \mathbf{x}_o \\ \alpha &= \theta - \phi \end{aligned} \tag{V.14}$$

Considerando la cinemática del robot, la transformación de coordenadas articulares en coordenadas imagen (cinemática directa del sistema robot-visión) queda como sigue:

$$\begin{aligned} \mathbf{x}_i^q &= \frac{\lambda}{z-\lambda} Rot(\phi, z)[f(\mathbf{q}) + \mathbf{x}_o] \\ \theta &= q_3 + \phi \end{aligned} \tag{V.15}$$

donde:

$$\mathbf{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \tag{V.16}$$

En este sentido, la transformación inversa (cinemática inversa del sistema robot-visión) queda como sigue:

$$\begin{aligned} \mathbf{q} &= f^{-1}\left[\frac{z-\lambda}{\lambda} Rot(-\phi, z)\mathbf{x}_q^i - \mathbf{x}_o\right] \\ \theta_3 &= \gamma - \phi \end{aligned} \tag{V.17}$$

Para obtener las coordenadas articulares correspondientes a las coordenadas imagen dadas por el sistema de VA es necesario obtener los parámetros cinemáticos f, $\mathbf{x}_o$ y ϕ, mediante algún algoritmo de calibración y aplicar la ecuación (V.17). Generalmente, este procedimiento es complicado y se enfrenta a problemas de singularidades.
En este trabajo de tesis proponemos evitar dicho problema utilizando una RNA, la cual lleva a cabo el aprendizaje de la transformación dada por la ecuación (V.17), así, la RNA reconstruirá el espacio articular del robot *Unimate S-103* a partir de información visual 2D de su espacio de trabajo.

V.3.2 Aproximador Neuronal

La RNA propuesta en el marco de esta tesis, tiene como objetivo aproximar la función multivariable correspondiente a la cinemática inversa del sistema de Robot-Visión considerado.

Proposición V.1

Sea $\mathbf{q_d} = \mathbf{f}(\mathbf{x}^i)$, la función que describe la transformación no lineal de coordenadas imagen en coordenadas articulares, donde $\mathbf{q_d}$ es el vector articular de salida y $\mathbf{x}^i$ es el vector imagen de entrada.

Dado un conjunto de n vectores (considerando que las coordenadas $\mathbf{x}^i$ no se pueden medir directamente sino a través de las coordenadas $\mathbf{x}^{pixel}$), $\mathbf{x}^{pixel} \in \Re^2 \ (i = 1,2,...,n)$, y un conjunto de vectores $\mathbf{q_d} \in \Re^2 \ (i = 1,2,...,n)$, por el teorema V.1, existe al menos una RNA con una función $\mathbf{F}$ tal que:

$$\left\| \mathbf{F}(\mathbf{W},\mathbf{b}) - \mathbf{f}(\mathbf{x}^{pixel}) \right\| < \varepsilon \quad \forall \mathbf{x}^{pixel}$$

V.18

donde:

$\mathbf{W}$: Matriz de pesos sinápticos de la RNA.
$\mathbf{b}$: Matriz de umbrales de cada neurona en la RNA.
$\mathbf{x}^{pixel}$: Vector de coordenadas pixel (x,y).
$\mathbf{q_d}$: Vector de coordenadas articulares (q_1, q_2).

Para encontrar una RNA que satisface la ecuación V.18 se llevaron a cabo las siguientes etapas:

- Selección de la topología de la RNA.
- Mecanismo de aprendizaje (Algoritmo L-M).
- Fase de Entrenamiento.
- Fase de Verificación (capacidad de generalización).
- Definición de las ponderaciones numéricas entre neuronas una vez entrenada la RNA.

V.3.2.1 Selección de la Topología del aproximador neuronal

Considerando la capacidad de aproximador universal de funciones [24] (§ Teorema V.1), la topología seleccionada en este trabajo fue un *perceptrón multicapas con una capa oculta* (fig. V.5). Como ya establecimos en V.17, $\mathbf{q}$ contiene las 2 primeras variables articulares del robot, ya que el valor de la tercera variable articular se obtiene a través del sistema de VA implementado (ecuación IV.22c y ecuación V.17). Con esta

información, podemos deducir que la RNA tiene dos neuronas de entrada $\mathbf{x}^i=(x, y)$, y dos neuronas de salida $\mathbf{q}=(\theta_1, \theta_2)$.

Por otra parte, la selección del número de neuronas en la capa oculta, se determinó experimentalmente considerando los datos de entrenamiento y verificación **p-q** (§ Sección V.3.2.3 y V.3.2.4), con la finalidad de obtener la menor cantidad de neuronas que permitiera tener un error de aproximación aceptable.

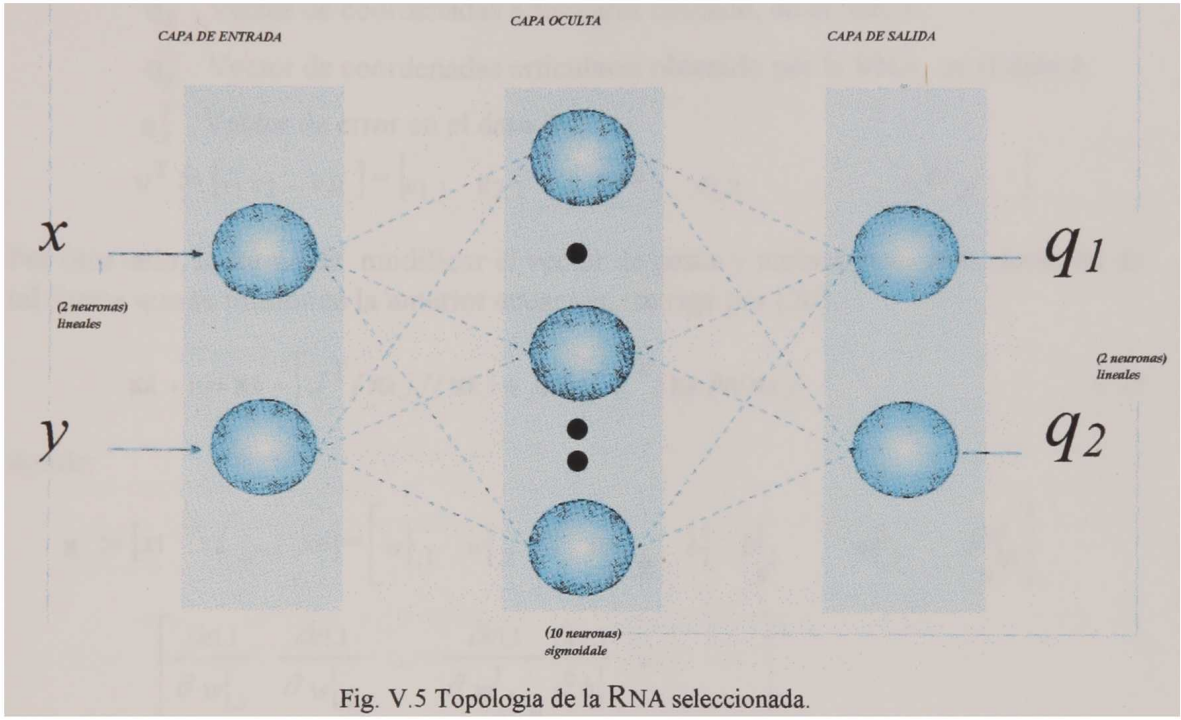


Fig. V.5 Topologia de la RNA seleccionada.

V.3.2.2 Algoritmo de aprendizaje (Algoritmo L-M)

Como método de aprendizaje, se seleccionó un algoritmo iterativo de optimización conocido como el algoritmo de Levenberg-Marquardt (L-M) [20][26]. La elección de este algoritmo está justificada por las siguientes razones:

1. Velocidad de convergencia (convergencia cuadrática, § sección V.3.2.2.1).
2. Garantía de encontrar una solución si esta existe.
3. Algoritmo implementado en las librerias de Matlab®.

El objetivo de este algoritmo es encontrar los valores de los pesos asociados entre neuronas y los valores de los umbrales de cada neurona tales que minimicen una función de error dada (ecuación V.19). Cabe destacar que este algoritmo es una extensión al método de Newton [25], cuyos fundamentos se encuentran en las series de Taylor de segundo orden.

En el algoritmo L-M, la función de error está definida por [20]:

$$\mathbf{F}(\mathbf{x}) = \sum_{k=1}^{n} \left(\mathbf{q}_k^d - \mathbf{q}_k^o\right)^T \left(\mathbf{q}_k^d - \mathbf{q}_k^o\right) = \sum_{k=1}^{n} \mathbf{e}_k^T \mathbf{e}_k = \sum_{k=1}^{n} \sum_{l=1}^{s} \left(e_{l,k}\right)^2 = \sum_{i=1}^{N} \left(v_i\right)^2 \qquad \text{V.19}$$

donde:

$\mathbf{q}_k^d$: Vector de coordenadas articulares deseado, en el dato *k*.

$\mathbf{q}_k^o$: Vector de coordenadas articulares obtenido por la RNA, en el dato *k*.

$\mathbf{e}_k^T$: Vector de error en el dato *k*.

$$\mathbf{v}^T := [v_1\, v_2 \cdots v_N] = \left[e_{1,1} \quad e_{2,1} \quad .. \quad e_{sM,1} \quad e_{1,2} \quad .. \quad .. \quad e_{sM,n}\right]$$

Por otro lado, la forma de modificar el vector de pesos y umbrales en cada iteración de tal forma que se minimice la anterior ecuación, se rige por [20]:

$$\mathbf{x}k+1 = \mathbf{x}k - \left[J^T(\mathbf{x}k)J(\mathbf{x}k) + \mu k I\right]^{-1} J^T(\mathbf{x}k)\mathbf{v}(\mathbf{x}k) \qquad \text{V.20}$$

donde:

$$\mathbf{x} := [x1 \quad x2 \quad .. \quad xn] = \left[w_{1,1}^1 \quad w_{1,2}^1 \quad .. \quad w_{s^1,R}^1 \quad b_1^1 \quad b_{s^1}^1 \quad w_{1,1}^2 \quad . \quad b_{s^M}^M\right]$$

$$\mathbf{J}(\mathbf{x}) := \begin{bmatrix} \frac{\partial e1,1}{\partial w_{1,1}^1} & \frac{\partial e1,1}{\partial w_{1,2}^1} & \cdots & \frac{\partial e1,1}{\partial w_{s^1,R}^1} & \frac{\partial e1,1}{\partial b_1^1} & \cdots \\ \frac{\partial e2,1}{\partial w_{1,1}^1} & \frac{\partial e2,1}{\partial w_{1,2}^1} & \cdots & \frac{\partial e2,1}{\partial w_{s^1,R}^1} & \frac{\partial e2,1}{\partial b_1^1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \\ \frac{\partial e_{SM},1}{\partial w_{1,1}^1} & \frac{\partial e_{SM},1}{\partial w_{1,2}^1} & \cdots & \frac{\partial e_{SM},1}{\partial w_{s^1,R}^1} & \frac{\partial e_{SM},1}{\partial b_1^1} & \cdots \\ \frac{\partial e1,2}{\partial w_{1,1}^1} & \frac{\partial e1,2}{\partial w_{1,2}^1} & \cdots & \frac{\partial e1,2}{\partial w_{s^1,R}^1} & \frac{\partial e1,2}{\partial b_1^1} & \cdots \end{bmatrix} \qquad \text{V.21}$$

μ_k : Razón de aprendizaje de la RNA.

Algoritmo L-M.

1. Inicializar los pesos de la red con valores aleatorios pequeños.
2. Presentar a la RNA todos los patrones de entrada y llevar a cabo el cálculo de la salida correspondiente a cada patrón utilizando las siguientes ecuaciones:

$$\mathbf{q}^0=\mathbf{p} \qquad \text{V.22a}$$

$$\mathbf{q}^{m+1}=f^{m+1}(\boldsymbol{W}^{m+1}\mathbf{q}^m+\mathbf{b}^{m+1}) \text{ para m=0,1,..,M-1 capas} \qquad \text{V.22b}$$

3. Llevar a cabo el cálculo de error para cada patrón de entrada.

$$\mathbf{E}_n=\mathbf{t}_n-\mathbf{q}^M{}_n \qquad \text{V.23}$$

4. Calcular la suma de los errores cuadráticos medios considerando todos los patrones de entrada a la RNA utilizando la ecuación V.19.
5. Calcular cada uno de los elementos de la matriz Jacobiana (ecuación V.21).
6. Resolver la ecuación V.20.
7. Volver a calcular la suma de los errores cuadráticos usando $\mathbf{x}_k + \Delta\,\mathbf{x}_k$. Si esta nueva suma es menor que la calculada en el paso cuatro, entonces se procede a dividir μ entre un valor dado ϑ, se vuelve a utilizar la ecuación V.20 y se regresa al paso número dos. En caso de que la suma no sea más pequeña, entonces se procede a multiplicar μ por ϑ y se regresa al paso 6.

Este algoritmo converge a la solución en el momento que la suma de los errores cuadráticos es igual ó menor a un error aceptable (en nuestro caso $\varepsilon \leq 0.1$).

V.3.2.2.1 Prueba de Convergencia

Teorema V.2 [28]

Sea:

$V : \Re^n \rightarrow \Re^m$, una función continua diferenciable en un conjunto convexo-abierto $D \subset \Re^n$

$f(\mathbf{x}) = \frac{1}{2} v(x)^T v(x)$, donde $f(\mathbf{x})$ cumple las propiedades de v

$J(\mathbf{x}) \in \mathrm{Lip}_\gamma(D)$, donde $\|J(\mathbf{x})\|_2 \leq \alpha \;\; \forall\, x \in D.$

$\sigma, \lambda \geq 0$ tal que $J(x_*)^T v(x_*) = 0$. (λ es el eigenvalor más pequeño de $J(x_*)^T J(x_*)$).

$\|(J(x)-J(x))^T v(x_*)\|_2 \leq \sigma \|x - x_*\|_2 \quad \forall\, x \in D.$

$\{\mu_k\}$: Secuencia de Números reales no negativos acotados por $b > 0$. (k=1,2,..,m).

$\sigma < \lambda$ dado un $c \in (1, (\lambda + b)/(\sigma + b))$.

Existe un $\varepsilon > 0$ tal que $\forall\, x_o \in N(x_*, \varepsilon)$, la secuencia generada por el método L-M,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[J^{\mathrm{T}}(\mathbf{x}_k) J(\mathbf{x}_k) + \mu_k I \right]^{-1} J^{\mathrm{T}}(\mathbf{x}_k) \mathbf{v}(\mathbf{x}_k)$$

satisface las ecuaciones :

$$\|x_{k+1} - x_*\|_2 \leq \frac{c(\sigma + b)}{\lambda + b}\|x_k - x_*\|_2 + \frac{c\alpha\gamma}{2(\lambda + b)}\|x_k - x_*\|_2^2 \qquad \text{V.24}$$

$$\|x_{k+1} - x_*\|_2 \leq \frac{c(\sigma + b) + (\lambda + b)}{2(\lambda + b)}\|x_k - x_*\|_2 < \|x_k - x_*\|_2 \qquad \text{V.25}$$

$$V(x_*) = 0, \mu_k = O(\|J(x_k)^T V(x_k)\|_2)$$

y la secuencia $\{x_k\}$, converge de manera *q-cuadrática a* x_* ♦

En la figura V.6 se muestra la gráfica de convergencia cuadrática obtenida por nuestra RNA en la solución del problema planteado en la sección V.3.1, considerando datos de entrenamiento. Con base en esta figura podemos observar que el algoritmo L-M requirió de 9 iteraciones para minimizar la función de error dada en la ecuación V.19, a un valor de aproximación $\varepsilon \leq 0.1$.

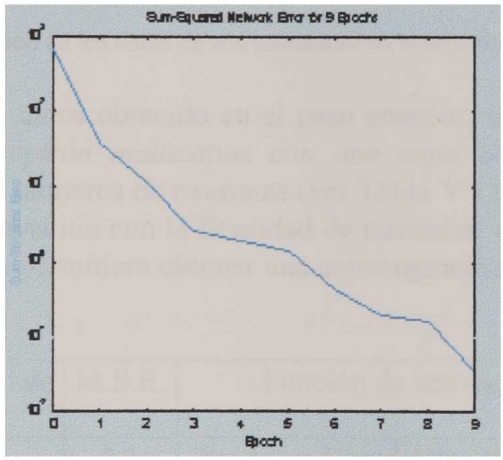


Fig. V.6 Gráfica de Convergencia del algoritmo de Levenberg-Marquardt

V.3.2.3 Fase de Entrenamiento

Considerando que la RNA propuesta en este trabajo utiliza un algoritmo de aprendizaje supervisado, en esta fase, se utilizó el modelo cinemático del robot (§ Sección III.2 y Sección III.3), con el propósito de generar el conjunto de datos de entrenamiento entrada-salida, para la RNA. Para ello, se llevó a cabo el siguiente proceso:

1. Se utilizó el modelo cinemático directo del robot (ecuación III.9 y ecuación III.10), para generar de manera indirecta, un conjunto de 500 pares de entrada-salida, **p-q**, correspondientes al modelo cinemático inverso del robot (ver fig. V.7). Cabe mencionar que este conjunto de datos de entrenamiento se seleccionó de manera aleatoria considerando que todos ellos, tienen la misma probabilidad de ocurrir.

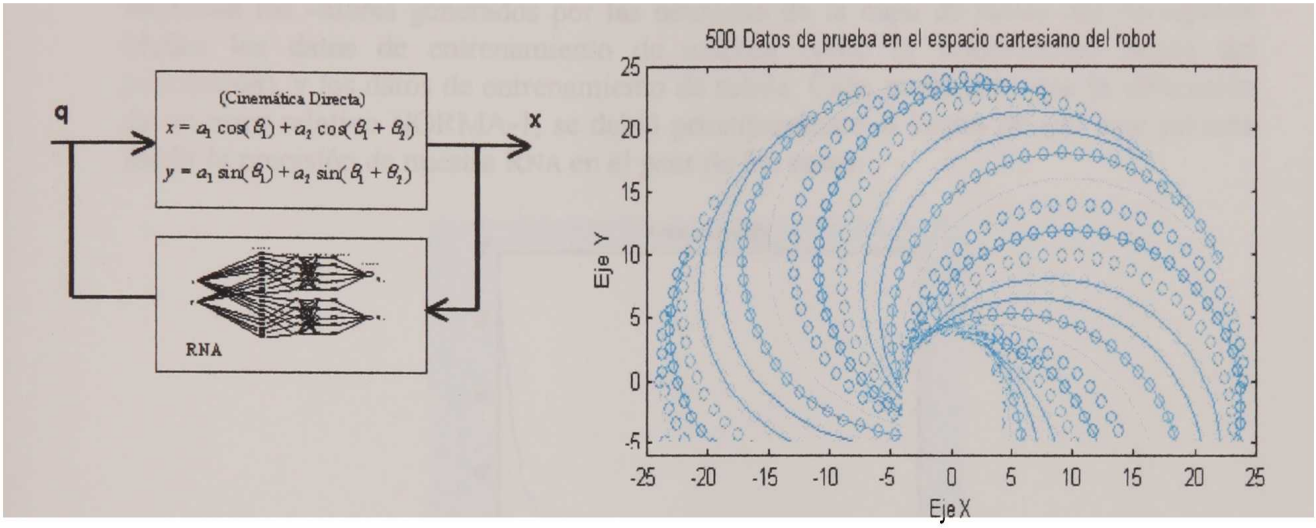


Fig. V.7 Generación de los datos de entrenamiento en el espacio cartesiano del robot

2. Con el conjunto de datos obtenido en el paso anterior, se procedió a entrenar una RNA del tipo *perceptrón multicapas con una capa oculta*. En dicha capa, se probaron diferentes números de neuronas (ver Tabla V.1), así como diferentes tipos de funciones de activación con la finalidad de encontrar una configuración mínima de neuronas tal que permitiera obtener una convergencia a una solución aceptable.

Neuronas	Razón de aprendizaje	M.S.E.	Función de activación	Convergencia
5	0.01	0.1	Tangente Hiperbólico Sigmoidal	NO converge
5	0.01	0.1	Lineal	NO converge
6	0.01	0.1	Tangente Hiperbólico Sigmoidal	NO converge
6	0.01	0.1	Lineal	NO converge
7	0.01	0.1	Tangente Hiperbólico Sigmoidal	NO converge
7	0.01	0.1	Lineal	NO converge
8	0.01	0.1	Tangente Hiperbólico Sigmoidal	NO converge
8	0.01	0.1	Lineal	NO converge
9	0.01	0.1	Tangente Hiperbólico Sigmoidal	NO converge
9	0.01	0.1	Lineal	NO converge
10	0.01	0.1	Tangente Hiperbólico Sigmoidal	CONVERGE

Tabla V.1 Configuraciones probadas en las neuronas de la capa oculta del *perceptrón multicapas.*

En la figura V.8, se muestra la gráfica de convergencia obtenida por el *perceptrón multicapas* al minimizar la función de error dada por la ecuación V.19, utilizando 10 neuronas sigmoidales en la capa oculta. En esta gráfica se puede observar la convergencia cuadrática, característica en los algoritmos basados en el método de optimización de Newton [20], tal como lo es el algoritmo L-M. Por otro lado, en la figura V.9 se muestra la gráfica de error (considerando una norma-1), obtenida mediante los valores generados por las neuronas de la capa de salida del *perceptrón* (dados los datos de entrenamiento de entrada como la información fuente del *perceptrón*), y los datos de entrenamiento de salida. Cabe mencionar que la utilización de un error relativo NORMA-1, se debió principalmente al hecho de que éste permite medir la precisión de nuestra RNA en el peor de los casos.

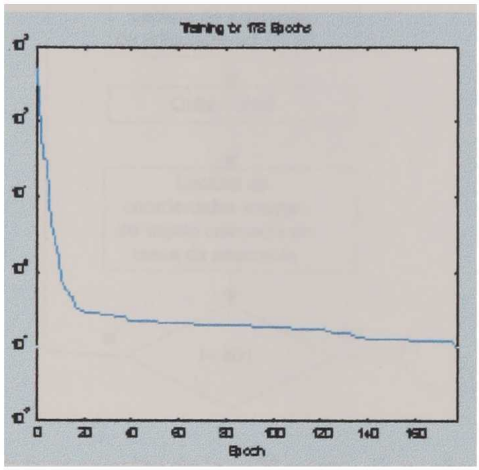


Fig. V.8 Gráfica de convergencia del *perceptrón multicapas*.

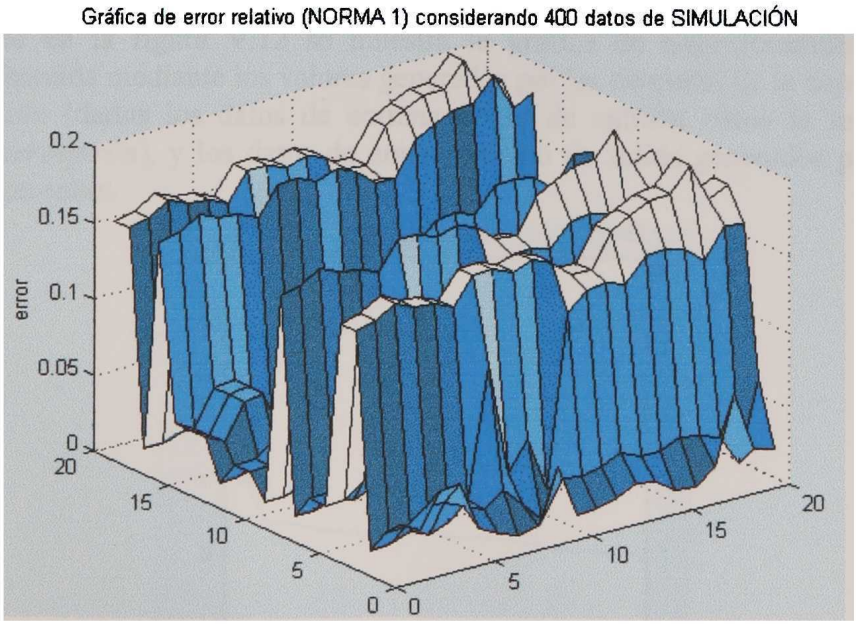


Fig. V.9 Gráfica de error NORMA-1, en datos de entrenamiento.

3. Utilizando el *perceptrón multicapas* obtenido en el paso anterior, se procedió a entrenar dicha RNA con 400 datos reales (obtenidos del sistema de visión artificial y de los sensores del robot), correspondientes al modelo cinemático inverso del sistema de Robot-Visión, mediante el algoritmo mostrado en la figura V.10.

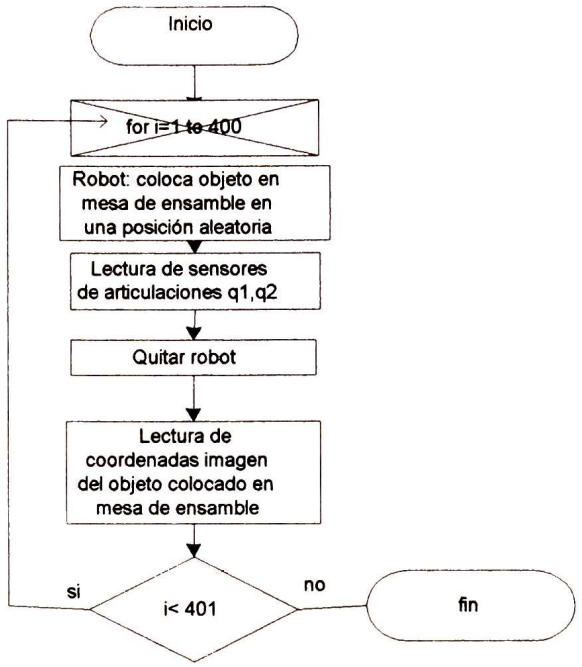


Fig. V.10 Algoritmo utilizado en la generación de los 400 puntos de prueba del modelo cinemático inverso del sistema de Robot-Visión.

En la figura V.11, se muestra la gráfica de convergencia obtenida por el *perceptrón multicapas* considerando los datos generados por nuestro sistema de visión-robot, mientras que en la figura V.12 se muestra la gráfica de error (considerando una norma-1), obtenida mediante los valores generados por las neuronas de la capa de salida del *perceptrón* (dados los datos de entrenamiento de entrada como la información fuente del *perceptrón*), y los datos de entrenamiento de salida generados por nuestro sistema visión-robot.

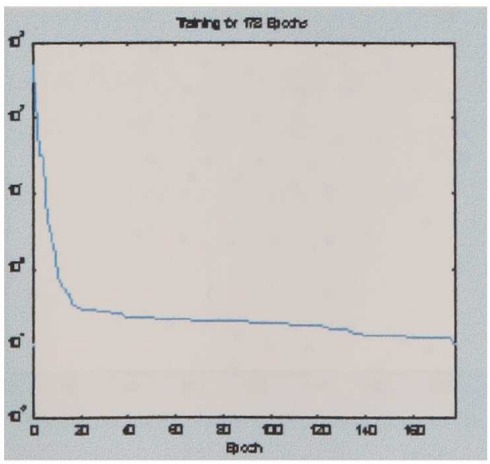


Fig. V.11 Gráfica de convergencia de la RNA utilizando datos reales Robot-Visión.

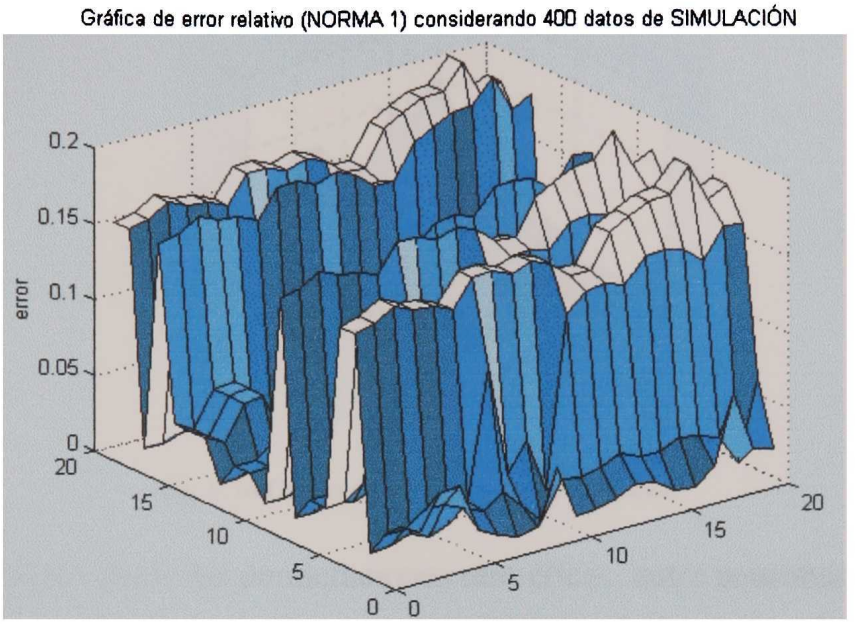


Fig. V.12 Gráfica de error en datos de entrenamiento generados por nuestro sistema de visión-robot.

V.3.2.4 Fase de Validación (Generalización)

Para comprobar la capacidad de *generalización* de nuestra RNA implementada, se seleccionaron un conjunto de 100 pares aleatorios ($\mathbf{x}^i$, $\mathbf{q}$), cuyos valores cubren la parte del espacio de trabajo del robot así como el espacio del sistema de VA (figura V.13). De esta forma, en la figura V.14 se muestra el error relativo NORMA-1, proporcionado por nuestro *perceptrón multicapas* (considerando como entrada estos 100 datos $\mathbf{x}^i$), y los 100 datos $\mathbf{q}$. En esta gráfica, se puede observar que la precisión obtenida por nuestra RNA corresponde a un dígito decimal en la mayoría de los casos a excepción de algunos puntos los cuales corresponden a configuraciones articulares cercanas al punto *home* del robot.

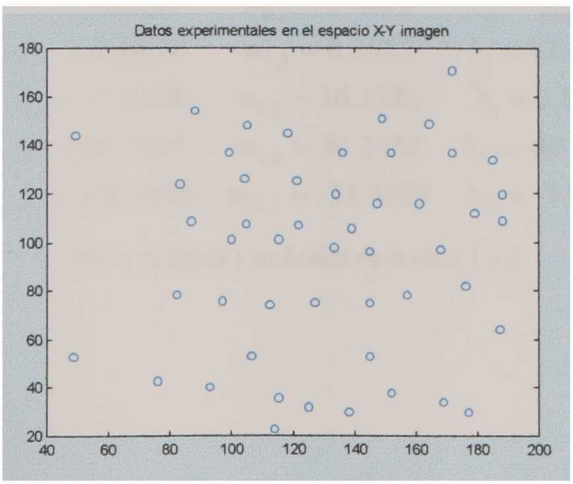


Fig. V.13 Conjunto de datos de validación en el espacio imagen.

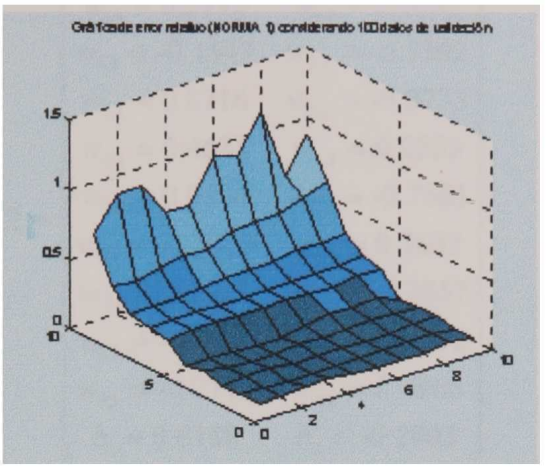


Fig. V.14 Gráfica de error en datos de validación.

V.3.2.5 Definición de las ponderaciones numéricas entre neuronas

Como se mencionó anteriormente (§ Sección V.3.2.1), la RNA obtenida, tiene *dos* neuronas en la capa de entrada, *diez* neuronas en la capa oculta y *dos* neuronas en la capa de salida. Con base en la sección V.3.2.2.1, donde obtuvimos un error de aproximación aceptable, la matriz de pesos y umbrales de la RNA que minimiza la ecuación V.19, quedó definida como:

$$\mathbf{W1} := \begin{bmatrix} w_{1,1} = -0.0724 & w_{1,2} = -0.0635 & b_1 = 0.0637 \\ w_{2,1} = 0.2227 & w_{2,2} = 0.0612 & b_2 = 0.0125 \\ w_{3,1} = -0.0016 & w_{3,2} = 0.1789 & b_3 = -0.1075 \\ w_{4,1} = -0.0916 & w_{4,2} = -0.1333 & b_4 = 0.1195 \\ w_{5,1} = -0.6897 & w_{5,2} = 0.7021 & b_5 = -0.4117 \\ w_{6,1} = -0.8533 & w_{6,2} = 0.8202 & b_6 = 0.0516 \\ w_{7,1} = 0.4669 & w_{7,2} = 0.3582 & b_7 = 17.6925 \\ w_{8,1} = -2.1324 & w_{8,2} = 36.1388 & b_8 = 3.8573 \\ w_{9,1} = 18.7404 & w_{9,2} = 50.2452 & b_9 = -40.7554 \\ w_{10,1} = -36.4956 & w_{10,2} = -21.4598 & b_{10} = -5.0027 \end{bmatrix} \quad \text{V.26a}$$

Matriz de pesos y umbrales de la capa 1 y 2.

$$\mathbf{W2} := \begin{bmatrix} w_{1,1} = 0.3713 & w_{1,2} = 0.1633 \\ w_{2,1} = -0.1547 & w_{2,2} = -0.1362 \\ w_{3,1} = 0.8716 & w_{3,2} = -0.0733 \\ w_{4,1} = 0.4638 & w_{4,2} = 0.0533 \\ w_{5,1} = 0.0906 & w_{5,2} = -0.7361 \\ w_{6,1} = -0.1909 & w_{6,2} = 0.3652 \\ w_{7,1} = 0.1607 & w_{7,2} = -0.2857 \\ w_{8,1} = -0.0726 & w_{8,2} = -0.2192 \\ w_{9,1} = -0.2110 & w_{9,2} = 0.4966 \\ b_1 = 0.6188 & b_2 = -0.2903 \end{bmatrix}^T \quad \text{V.26b}$$

Matriz de pesos y umbrales de la capa 2 y 3.

V.4 Ambiente de simulación Neuronal

En esta sección, se describe de manera general el ambiente de simulación desarrollado, con el propósito de modelar, algunas de las topologías de RNA más utilizadas y en particular, el modelo de RNA propuesto en la solución del problema planteado en la § sección V.3.1. En este sentido, en los siguientes párrafos se muestra dicho simulador (Figura V.15), así como los diferentes componentes que lo constituyen. Cabe mencionar, que este simulador está basado en las librerías propias de Matlab Versión 5.2 ® [26].

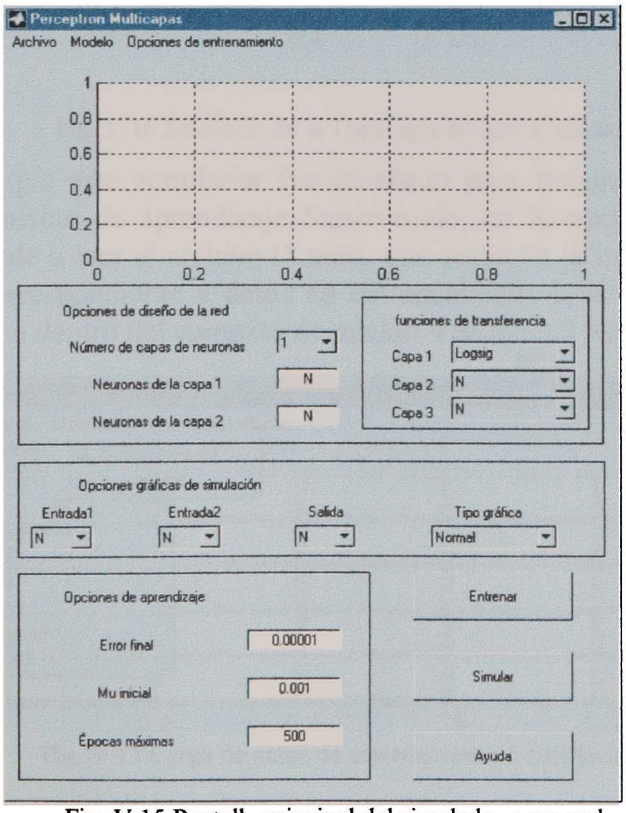


Fig. V.15 Pantalla principal del simulador neuronal.

Para utilizar el simulador neuronal, es necesario llevar a cabo el siguiente procedimiento:

1. Seleccionar el modelo ó topología de RNA a utilizar. – En esta opción, el simulador permite seleccionar una de las ocho topologias de RNA mas utilizadas en diversas áreas científicas y tecnológicas [3]: lineal, Perceptrón Multicapas, Base Radial, Competitiva, SOM, LVQ, Elman ó Hopfield .

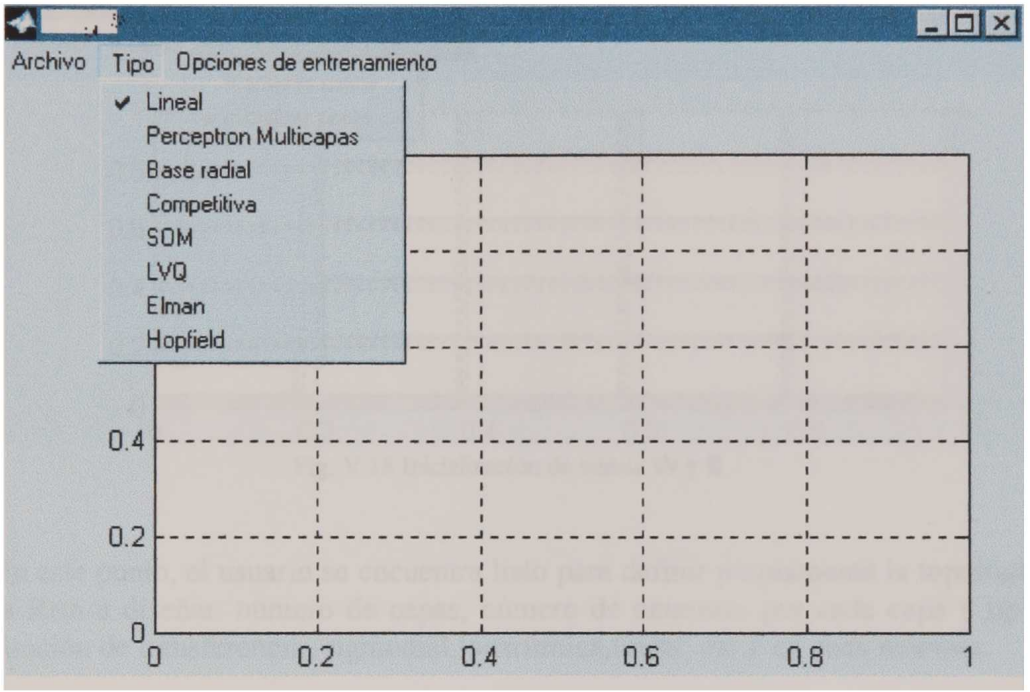


Fig. V.16 Selección de la Topología de RNA a utilizar.

2. Considerando que este simulador fue diseñado para trabajar con algoritmos que utilizan un mecanismo de aprendizaje *Supervisado*, en la opción *Archivo* del menú principal se procede a leer el archivo (*.mat), que contiene la información de datos de entrada-salida, correspondiente a datos de entrenamiento ó verificación. (Nota: estos archivos se generan dentro del contexto de matlab Versión 5.2 ® [26]).

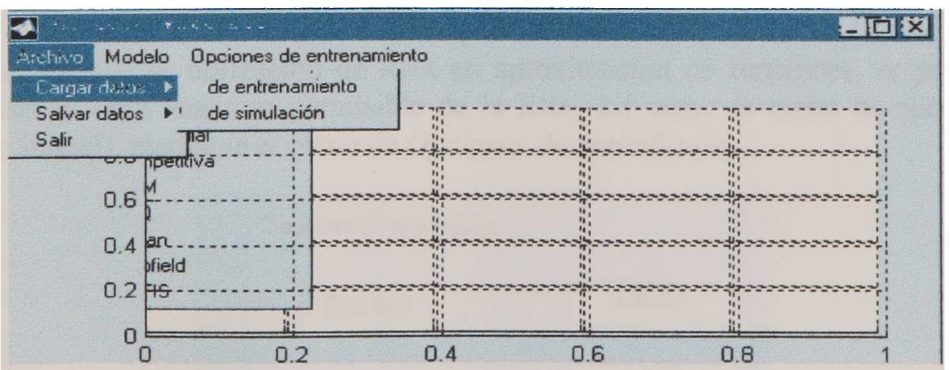


Fig. V.17 Carga de datos de entrenamiento ó simulación.

2. Una vez que se han cargado los datos de entrada-salida, se procede a inicializar la matriz de pesos y umbrales (**W** y **B**), mediante el menú de *opciones de entrenamiento*. En este menú existen dos opciones: *Mantener pesos* e *Inicializar pesos*. En la opción *Mantener pesos* se considera la matriz de pesos y umbrales (**W** y **B**), definida por el usuario, mientras que en la opción *Inicializar pesos* se utliza una función aleatoria que inicializa la matriz de pesos y umbrales (**W** y **B**).

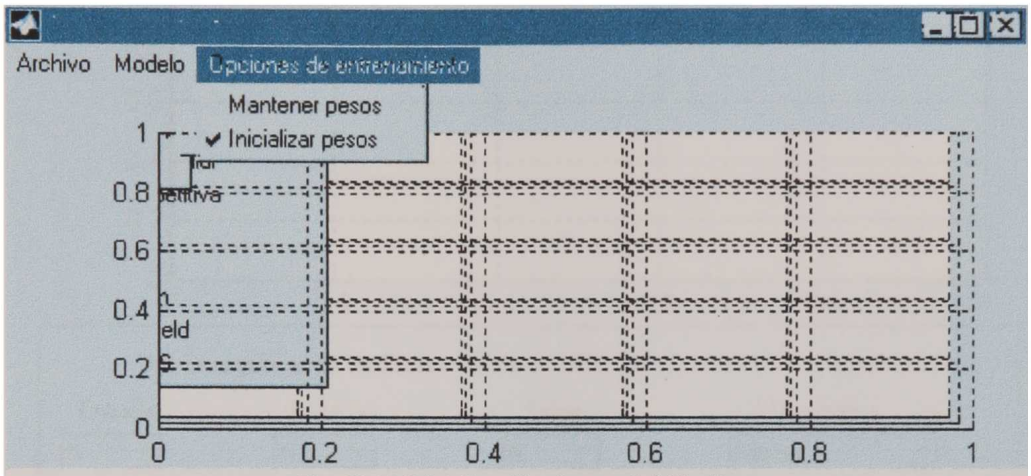


Fig. V.18 Inicialización de matriz W y B.

3. En este punto, el usuario se encuentra listo para definir propiamente la topología de la RNA a diseñar: número de capas, número de neuronas por cada capa y tipo de función de transferencia (sigmodial,logaritmica,lineal, etc.), en cada neurona.

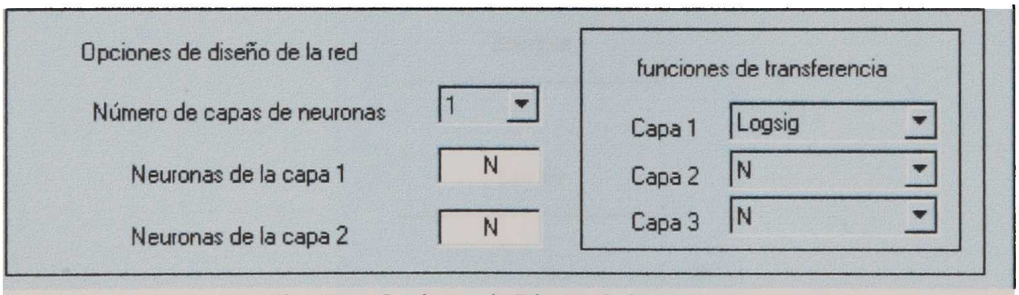


Fig. V.19 Opciones de Diseño de la RNA .

4. Considerando la utilización de RNA en aproximación de funciones, se procede a definir el error máximo permisible de la RNA, así como la razón de aprendizaje (Mu Inicial), mediante el menú de *Opciones de aprendizaje.*

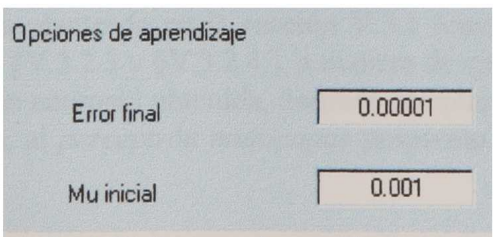


Fig. V.20 Opciones de aprendizaje.

6. Considerando la necesidad de verificar la capacidad de aprendizaje de la RNA en cuestión, se define el tipo de gráfica que se desea analizar: *normal, comparativa ó error* así como la fuente de datos a analizar (Entrada 1, Entrada 2 ó Salida).

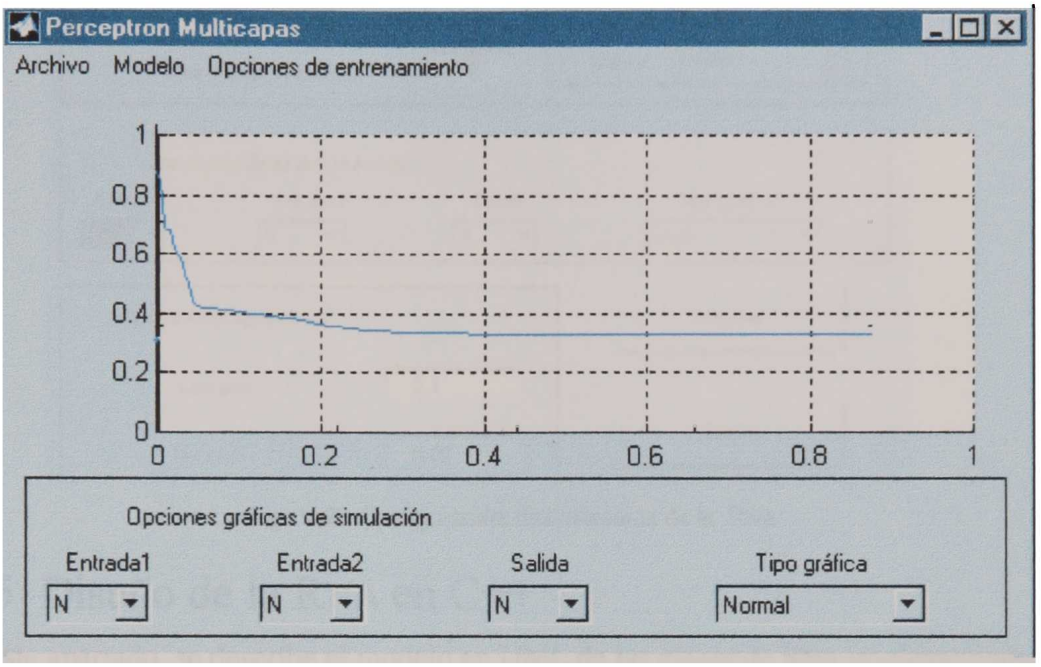


Fig. V.21 Tipos de Gráficas de simulación.

7. Finalmente se procede a entrenar la RNA y validar el aprendizaje de la RNA, a través de los controles de *Entrenar* y *Simular*, respectivamente.

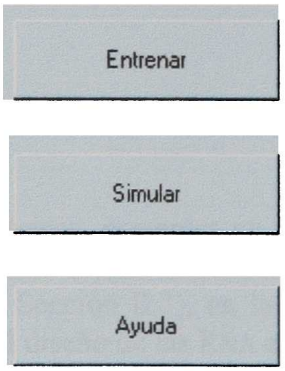


Fig. V.22 Controles de Entrenamiento y simulación.

Considerando el problema planteado en la sección V.3.1 (cuya solución neuronal fué mostrada en las secciones §V.3.2.3 y §V.3.2.4.), a manera de ejemplo, en la figura V.23 se muestra la configuración neuronal obtenida, después de aplicar la fase de aprendizaje y validación de resultados, al *perceptrón multicapas* propuesto en la solución de dicho problema.

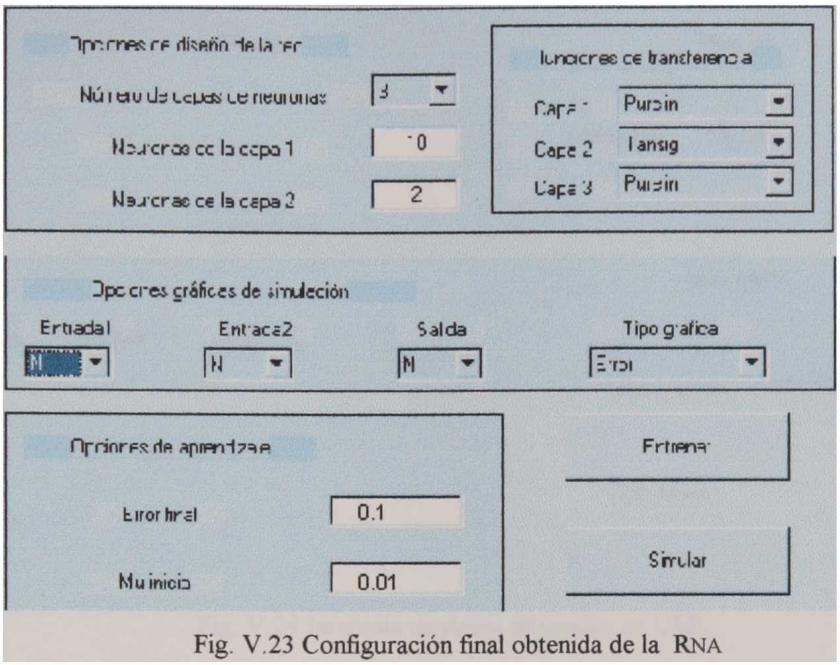


Fig. V.23 Configuración final obtenida de la RNA

V.5 Diseño de la RNA en C++

En este apartado, se describe el modelo en UML de las clases de RNA implementadas en C++. Cabe mencionar, que este modelo neuronal está basado en una formalización de RNA propuesta en [17], por lo que dicho modelo no es especifico de ninguna topología; por el contrario, dicho modelo tiene como objetivo, modelar cualquier topología de RNA. Con base en lo anterior, en los siguientes párrafos se presentan los aspectos más relevantes, considerados en el diseño e implementación de las RNA en C++

V.5.1 Jerarquía de clases

Considerando que los elementos básicos, presentes en toda red neuronal artificial son los nodos (neuronas), así como los enlaces existentes entre ellos, con base en las ecuaciones II.2, II.3, II.4 y II.5 (§ Sección II.3), en la siguiente figura se muestra la jerarquía de clases propuesta para el diseño de las RNA en C++.

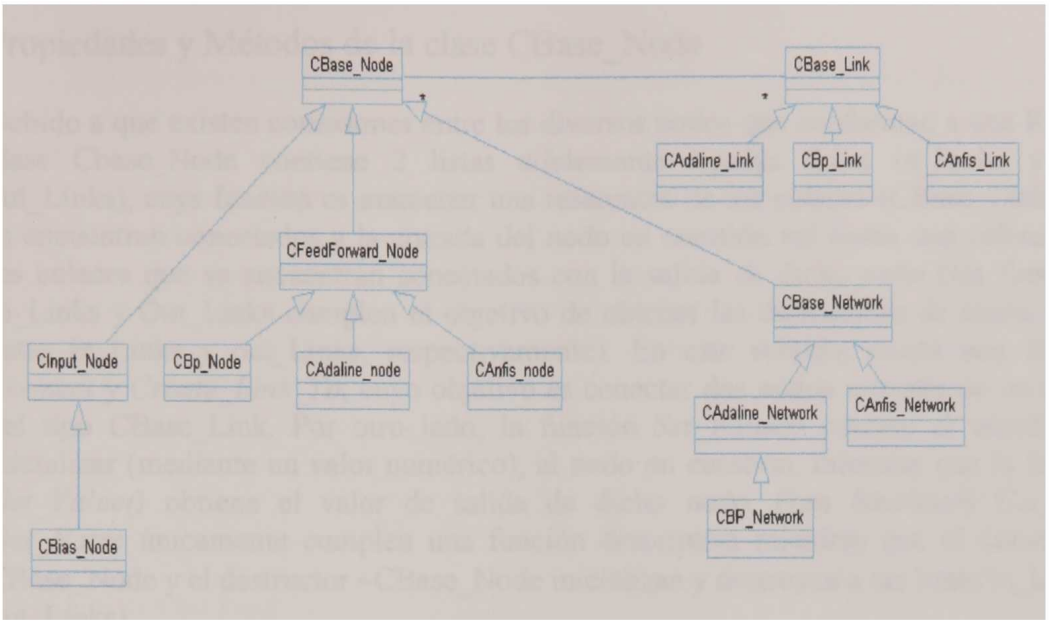


Fig. V.24 Jerarquía de clases propuesta en UML.

V.5.1.1 Clase Base Cbase_Node

La clase base Cbase_Node, es una entidad abstracta que representa a los elementos más importantes de toda RNA: los nodos (neuronas). Por definición (§Sección V.I), los nodos en las RNA tienen la función de recibir información de otros nodos ó del exterior, procesar dicha información y, posteriormente, producir un valor de salida. Con base en lo anterior, la clase base propuesta Cbase_Node, cumple el objetivo de recibir y distribuir información, mientras que las clases heredadas cumplen el propósito de procesar información.
En este sentido, considerando que en una RNA, cada nodo cumple una función determinada, en este modelo se proponen las siguientes clases que heredan los métodos y propiedades básicas de Cbase_Node:

- Cinput_Node .- La clase Cinput_Node representa a cada neurona, presente en la capa de entrada de toda RNA. (Un caso particular de esta clase es el Bias de cada neurona, el cual se encuentra representado mediante una neurona que proporciona un valor de entrada unitario y que en este modelo, está definido por la clase Cbias_Node).

- CFeedforward_Node.- Esta clase representa la clase padre de todas las clases de neuronas que pertenecen a una red de tipo *feedforward* (conexiones hacia adelante). Algunos posibles nodos pertenecientes a la categoria de red de tipo *feedforward* son: CBp_node, CAdaline_node, CAnfis_node, entre otros.

- Cbase_Network.- Cbase_Network es una abstracción de un sistema que transforma un valor de entrada en un valor de salida, de manera similar a una neurona; es decir, una RNA, sin importar el tipo de topología, cumple el mismo objetivo que una neurona (además de extender las capacidades de esta).

Propiedades y Métodos de la clase CBase_Node

Debido a que existen conexiones entre los diversos nodos que conforman a una RNA, la clase Cbase_Node contiene 2 listas doblemente ligadas (Llist in_Links y Llist out_Links), cuya función es mantener una referencia de los enlaces (CBase_Link), que se encuentran conectados a la entrada del nodo en cuestión así como una referencia a los enlaces que se encuentran conectados con la salida de dicho nodo (las funciones In_Links y Out_Links cumplen el objetivo de obtener las direcciones de inicio de las listas in_Links y out_Links, respectivamente). En este sentido, existe una función *Connect* y *Create_Link_To*, cuyo objetivo es conectar dos nodos a través de un enlace del tipo CBase_Link. Por otro lado, la función *Set_Value()* cumple el objetivo de inicializar (mediante un valor numérico), al nodo en cuestión, mientras que la función *Get_Value()* obtiene el valor de salida de dicho nodo. (Las funciones *Get_ID* y *Get_Name* únicamente cumplen una función descriptiva mientras que el constructor CBase_Node y el destructor ~CBase_Node inicializan y destruyen a las listas in_Links y out_Links).

Propiedades y Métodos de la clase CFeedforward_Node

Transfer_Function()
Esta función es una función virtual debido a que en las clases heredades de esta clase, se encuentra implementado el tipo de función de transferencia que realiza cada nodo particular (CBp_node, CAdaline_node, CAnfis_node, entre otros).

Cfeedforward_Node()
Esta función es el constructor de esta clase y únicamente tiene el objetivo de inicializar mediante un valor numérico a este nodo.

Propiedades y Métodos de la clase CBase_Network

Create_Network()
La función virtual Create_Network() proporciona una estructura general de implementación de cada tipo particular de RNA, por lo que en cada clase heredada de CBase_Network, se especifica la topología propia de la RNA en cuestión.

Load_Inputs()
Esta función cumple el objetivo de recuperar las matrices W y B.

Num_Nodes
Esta variable contiene el número total de nodos que conforman a la RNA en cuestión.

**node
node es un doble apuntador a un arreglo bidimensional de apuntadores a variables del tipo CBase_Node.

Num_links
Num_links es una variable que contiene el número total de enlaces (links), en la RNA.

**link
link es un doble apuntador a un arreglo bidimensional de apuntadores a variables del tipo CBase_Link.

En las figuras V.25 y V.26, se muestran el conjunto de propiedades y métodos que contemplan las clases anteriormente expuestas (la codificación completa de estas clases en C++, se encuentra en el Apéndice B).

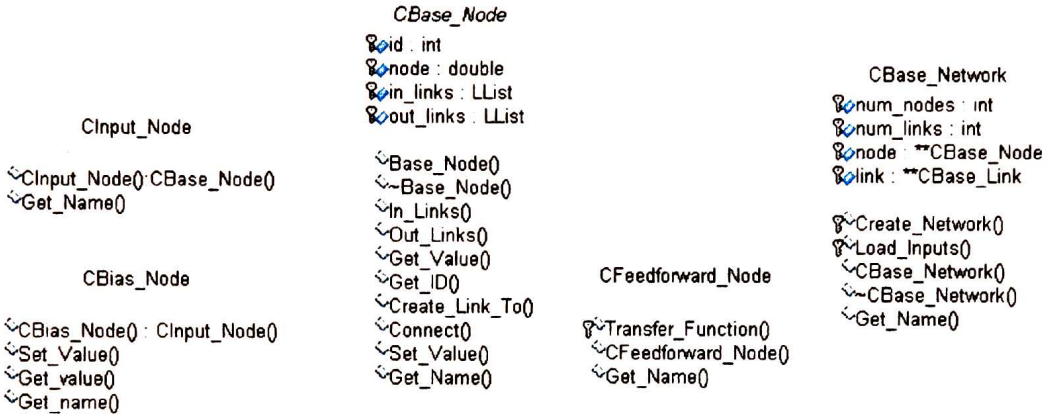


Fig. V.25 Prototipos de las clases Cbase_Node,Cinput_Node,Cbias_Node y Cbase_Network.

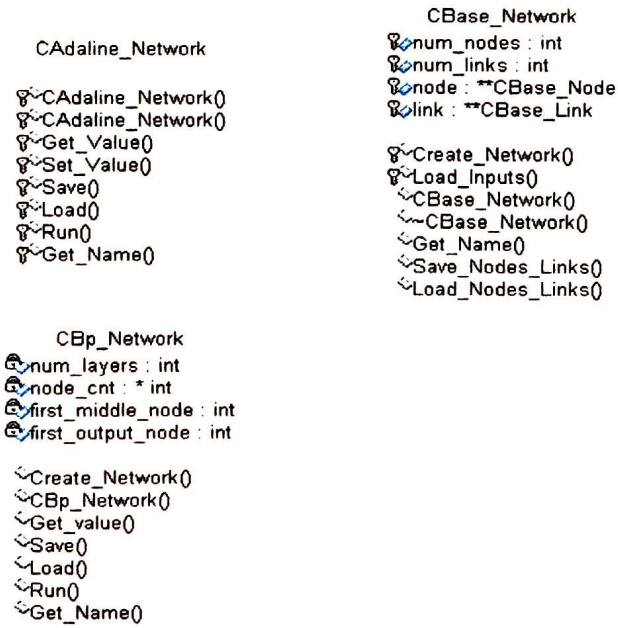


Fig. V.26 Prototipos de las clases Cadaline_Network y CBp_Network (red Backpropagation).

V.5.1.2 Clase Base Cbase_Link

La clase Cbase_Link tiene como objetivo mantener una relación de las conexiones existentes entre los diversos nodos que conforman a una RNA. Al respecto, está clase contiene dos apuntadores a Cbase_Node: in_node y out_node, así como también funciones similares a las encontradas en la clase CBase_Node como *Get_Value()*, *Set_Value()*, *In_Node()*, *Set_In_Node()*, *Out_Node()*, *Set_Out_Node()*, *Weighted()* y *Get_Name()*, cuyo propósito es leer información de los nodos así como realizar operaciones con este valor y la ponderación de la conexión. En los siguientes párrafos, se describe la utilidad de cada uno de estos elementos que conforman a la clase CBase_Link.

Propiedades y Métodos de la clase CBase_Link

in_node
Esta variable es un apuntador al nodo de entrada.

out_node
Esta variable es un apuntador al nodo de salida.

Get_Value()
Esta función obtiene el valor de la conexión en cuestión.

Set_Value()
Esta función define un valor para una conexión dada.

In_Node()
La función de In_Node(), es obtener el valor del nodo de entrada.

Set_In_Node()
Esta función cumple el objetivo de definir un valor en el nodo de entrada.

Out_Node()
Out _Node() obtiene el valor del nodo de salida.

Set_Out_Node()
Esta función es equivalente Set_In_Node() con la diferencia de que en esta, se define un valor en el nodo de salida.

Weighted()
El objetivo de esta función es realizar el producto entre el valor del nodo de entrada al objeto CBase_Link y el valor del mismo.

En la figura V.27 se muestran los prototipos de las funciones de esta clase así como de las clases derivadas de ésta. (El código completo de esta clase en C++, se encuentra en el Apéndice B).

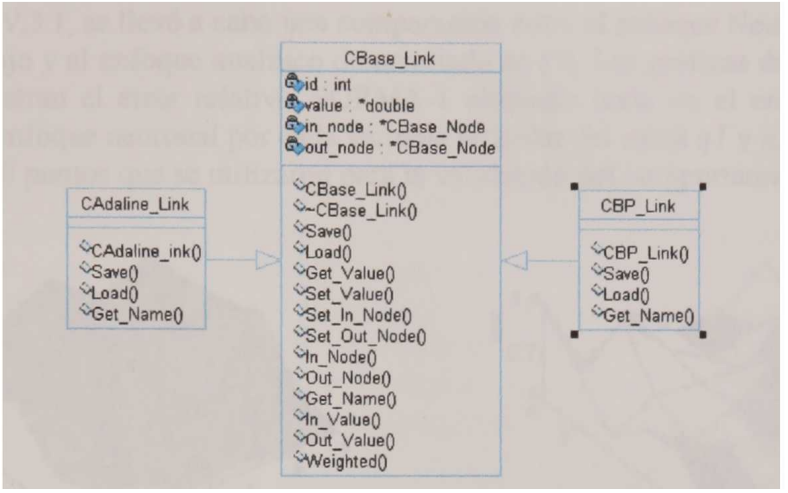


Fig. V.27 Prototipos de las clases Cbase_Link, Cadaline_Link y CBp_Link.

A manera de ejemplo, en la figura V.28 se muestra el diseño de un *Perceptrón Multicapas* basado en los objetos propuestos en la sección anterior, cuya topología está definida por dos neuronas en la capa de entrada, dos neuronas en la capa oculta y dos neuronas en la capa de salida. (En el apéndice B se muestra el código en C++, utilizado en la generación de la topología de RNA obtenida en la sección §V.3.2.1).

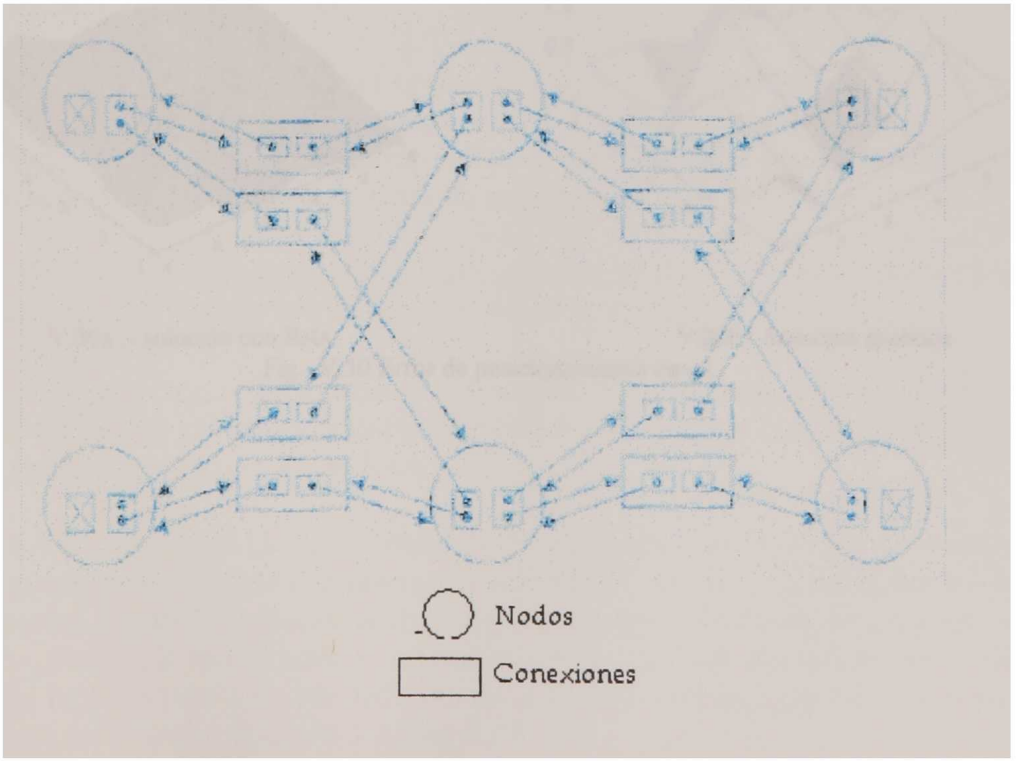
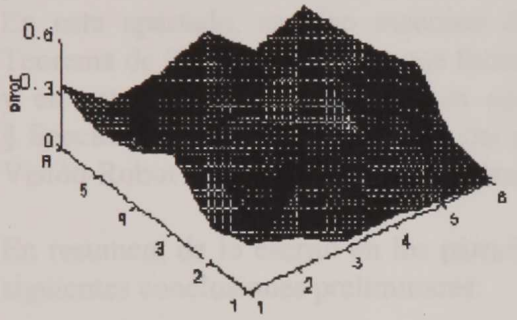


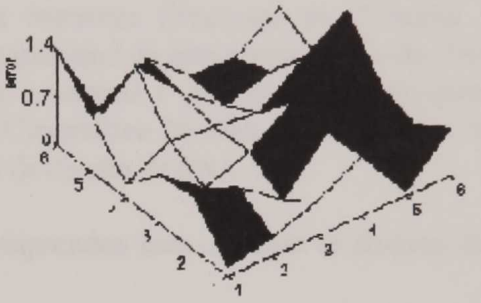
Fig. V.28 Implantación de un *Perceptrón Multicapas*

V.6 Comparación Enfoque Neuronal-Enfoque Analítico

Para validar los resultados obtenidos por nuestra RNA en la solución del problema planteado en V.3.1, se llevó a cabo una comparación entre el enfoque Neuronal propuesto en este trabajo y el enfoque analítico desarrollado en [7]. Las gráficas de la figura V.29 y V.30 muestran el error relativo NORMA-1 obtenido tanto en el enfoque analítico como en el enfoque neuronal por cada variable articular del robot *q1* y *q2*, considerando 36 de los 100 puntos que se utilizaron para la validación del comportamiento de la RNA.

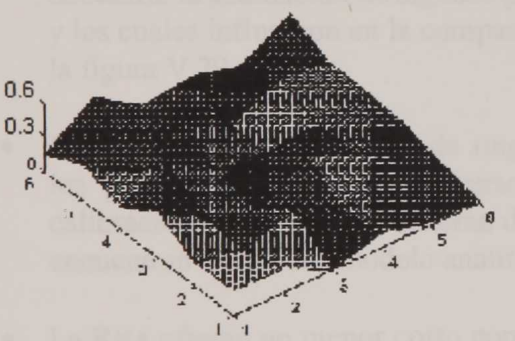


V.29a .- Solución con RNA

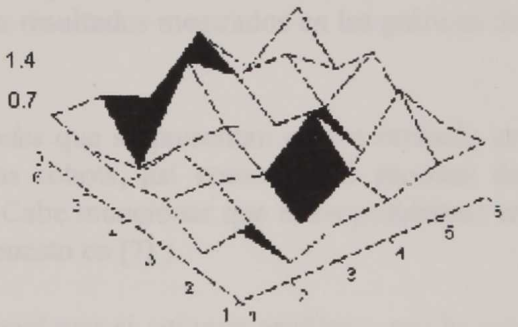


V.29b.- Solución Analítica

Fig. V.29 Error de posicionamiento de *q1*.



V.30a .- solución con RNA



V.30b.- Solución analítica

Fig. V.30 Error de posicionamiento de *q2*.

V.7 Conclusiones preliminares

En este capitulo se abordó el tema de las *Redes Neuronales Artificiales* (RNA) así como su integración dentro de nuestro Esquema de Control Visuomotor. En primera instancia se describieron las características generales de las RNA: ecuación básica, topología, conectividad, funciones de transferencia y algoritmos de aprendizaje. Posteriormente se presentó la conexión existente entre las RNA y la teoría de aproximación de funciones. En este apartado, se hizo mención de algunos teoremas (Teorema de Cybenko y Teorema de Stone-Weirstrass) que formalizan la capacidad de aproximación de las RNA y en este sentido, se propuso un aproximador neuronal (*perceptrón multicapas*, § Sección V.3.2), para la solución del problema Cinemático Inverso de un Sistema de Visión-Robot así como también, una metodología de diseño de RNA.

En resumen, de lo escrito en los párrafos que comprenden este capítulo se derivan las siguientes conclusiones preliminares:

- La capacidad de aproximación de la RNA fue mayor que en el enfoque analítico, debido principalmente a que en la RNA no se requirió de ninguna *estimación* de ningún parámetro, mientras que, por el contrario, en el enfoque analítico fue necesaria la estimación de algunos parámetros que no pueden medirse directamente y los cuales influyeron en la comparación de resultados mostrados en las gráficas de la figura V.29 y V.30.

- La RNA evitó los problemas de *singularidades* que se presentan muy a menudo en los problemas cinemáticos inversos de los robots, así como en el proceso de calibración de cámara del sistema de VA. (Cabe mencionar que estos problemas se encuentran dentro del modelo analítico propuesto en [7]).

- La RNA ofreció un menor costo computacional que el enfoque analítico, por lo que resulta factible utilizar la RNA en *línea* con el robot y el sistema de VA.

- El desarrollo de una plataforma de simulación neuronal basada en matlab ® [26], permite reducir el tiempo de experimentación mediante una simple reconfiguración de parámetros propios de una RNA.

- El modelo de clases en UML propuesto en este trabajo de tesis cumple el objetivo de generalizar el diseño e implantación (orientado a objetos), a cualquier topología neuronal existente. Además, la implantación de este modelo en un lenguaje como C++, permite la reutilización de código así como una fácil generación (utilizando la clase base propuesta en este trabajo), de nuevas topologías neuronales mediante el uso de primitivas propias del lenguaje C++.

Capítulo VI.
Conclusiones y Perspectivas

VI.1 Conclusiones

En la actualidad se ha generalizado el uso de las redes neuronales artificiales (RNA) en diferentes tareas de clasificación, modelado de sistemas, etc., [3], y se ha establecido como un método emergente-alternativo de procesamiento e interpretación de información, debido a las características intrínsecas que presentan: adaptabilidad, paralelismo, comportamiento no lineal, facilidad de implantación y sobre todo, capacidad de generalización.

En este trabajo, se mostraron las ventajas que ofrecen las RNA en comparación con un método analítico propuesto en [7], en la solución de la cinemática inversa de un sistema de Robot-Visión. Como se observó en él capitulo V, la RNA utilizada, evitó la *estimación* de los parámetros necesarios en la calibración de cámara así como los problemas de *singularidades*, ambos, presentes en la solución analítica de la cinemática inversa del robot. Todo lo anterior debido a que la RNA utilizada únicamente necesitó de datos de entrada-salida (correspondientes a coordenadas *pixels* y coordenadas articulares respectivamente), como datos para el aprendizaje del problema inverso mencionado, además de considerar otros parámetros importantes como: razón de aprendizaje, máximo error permitido, etc.

En este sentido, la aportación principal del presente trabajo es la *identificación del sistema* Cinemático Inverso Robot-Visión, a través de la optimización de una función de error (error cuadrático medio), mediante el uso del Algoritmo de Levenberg-Marquardt [28], aplicado en el aprendizaje de la RNA, cuyo objetivo fue, definir la matriz de pesos W y umbrales B, tales que, minimizaran el valor de dicha función cuadrática. Mención especial tiene la forma en como se generaron los datos de entrenamiento y los datos de validación cuya información resultó de suma importancia en la definición final de la topología del *perceptrón multicapas*.

Por otro lado, es de relevancia mencionar el ambiente de simulación neuronal desarrollado así como el modelo de clases en UML propuesto, cuyas aportaciones principales son:

- Reducir los tiempos de programación de algunas topologias de RNA.

- Permitir la reutilización de código así como una fácil generación (utilizando la clase base propuesta en este trabajo), de nuevas topologías neuronales mediante el uso de primitivas propias del lenguaje C++, respectivamente.

Es importante destacar que el carácter de este trabajo es eminentemente práctico y que se apoya en algunos aspectos teóricos de control inteligente, reportados en otros trabajos [29][16]. En este sentido, el sistema de control interno del robot fue transparente en este trabajo, por lo que no se necesitó del conocimiento del funcionamiento interno del mismo.

En resumen, los principales logros alcanzados en este trabajo son:

- La utilización del robot en tareas de manipulación y ensamble de objetos sin necesidad de conocer a priori sus posiciones, usando las capacidades que nos ofrecen las RNA.
- Posibilidad de emplear información exteroceptiva (Visión Artificial), en labores de ensamble de piezas.
- La identificación de un sistema inverso mediante RNA.
- Solución al problema cinemático inverso de un sistema de Robot-Visión, sin necesidad de conocer las complejidades intrínsecas en el esquema de control interno del Robot.
- El desarrollo de una plataforma de simulación neuronal basada en matlab ® [26], que permite reducir el tiempo de experimentación mediante una simple reconfiguración de parámetros propios de una RNA.
- Propuesta de un modelo de clases en UML que cumple el objetivo de generalizar el diseño e implantación (orientado a objetos), de cualquier topología neuronal existente.

VI.2 Perspectivas

Como se mencionó en el apartado anterior, las áreas de aplicación e investigación de las RNA son muy diversas y se encuentran en un estado incipiente. En este sentido, existe un gran interés en la comunidad científica por un desarrollo de un marco de estandarización y formalización de las mismas, que permita, entre otras cosas:

- Definir a priori, el número de capas ocultas así como el número de neuronas por cada capa oculta, en sistemas donde se requiere considerar múltiples entradas y múltiples salidas.
- Definir el conjunto exacto de datos de entrenamiento y validación tales que produzcan un valor óptimo de aprendizaje.
- Desarrollar nuevos algoritmos de aprendizaje (supervisados, no supervisados, etc.), con la finalidad de reducir tiempos de convergencia.
- Descubrir factores intrínsecos de las RNA que ayuden a mejorar el desempeño de las mismas.
- Medir, mediante técnicas formales, la capacidad de generalización y aprendizaje de las RNA.
- En el área de Control así como en el área de Robótica, es importante validar la estabilidad de las RNA dentro de un sistema completo de control.
- Analizar los resultados que ofrecen las RNA en combinación con otras tecnologías emergentes como la lógica borrosa, algoritmos genéticos, entre otras.

Y en particular, posibles tareas a desarrollar relacionadas con este trabajo de tesis serían:

- Utilizar un mecanismo de aprendizaje *no supervisado* (e.g. Mapas de Kohonen), en la identificación de la Cinemática Inversa del sistema de Robot-Visión considerado.
- Modificar el esquema de integración Visión-Robot tipo *Fixed Eye* por un esquema *eye in hand.*
- Integrar las RNA dentro del sistema de control del Robot Unimate.
- Generar el conjunto exacto de datos de entrenamiento y validación.
- Considerar un sistema de ensamble de objetos *virtual*, utilizando Internet (en curso).

Apéndice A
Código del sistema de VA en C

A.1 Código del Sistema de Visión Artificial en C

En el presente Apéndice, se describe la implantación en lenguaje C, de los algoritmos utilizados en las etapas más importantes de un sistema de Visión Artificial, descritas en el capitulo IV del presente trabajo. En este sentido, en los siguientes apartados se muestra la codificación de las etapas:

- Filtraje de ruido.
- Segmentación.
- Descripción.
- Reconocimiento.

A.2 Algoritmo del Filtraje de ruido (Proceso de Erosión)

```
//-------------------------------------------------------------------------------------------------
// Erosion (Image *IMAGE, int MASK[][], Image *FILTER)

// En este programa consideramos que la imagen es binaria (fondo color negro y objetos
//  color blanco), y además se encuentra almacenada en un objeto del tipo Image, cuyas
// dimensiones son Rows x Cols.
//   Por otro lado MASK es un arreglo bidimensional que representa al elemento
//  estructurante y FILTER es la imagen obtenida después de haber aplicado el filtraje.
//-------------------------------------------------------------------------------------------------


Erosion (Image *IMAGE, int MASK[][], Image *FILTER)

{
  int X,Y,I,J,smin;

 for (Y=N/2; Y<IMAGE->Rows-N/2;Y++)

  {
    for (X=N/2; X<IMAGE->Cols-N/2;X++)
       {
          smin=255;
           for(J=-N/2;J<=N/2;J++)
              {
               for(I=-N/2;I<=N/2;I++)
                 {
                    if (MASK[I+N/2][ J+N/2]==1)
                          {
                          if (* (IMAGE->Data+X+I+(long)(Y+J)*IMAGE->Cols)<smin)
                             {
                              smin=*(IMAGE->Data+X+I+(long)(Y+J)*IMAGE->Cols)
                             }
                          }
                 }
              }
          *(FILTER->Data+X+(long)Y*IMAGE->Cols)=smin;
           FILTER->Rows= IMAGE ->Rows;
           FILTER->Cols= IMAGE ->Cols;
       }
    }
  }
```

A.3 Segmentación (Proceso de umbralización)

```
//-------------------------------------------------------------------------------------------------
// Threshold (Image *IMAGE, Image *IMAGE1, int THRES)

// En este programa consideramos que la imagen IMAGE es multinivel (diferentes
// niveles de gris), y el resultado del proceso de Segmentación se almacena en IMAGE1
// THRES es el valor de umbral.
//-------------------------------------------------------------------------------------------------

Threshold (Image *IMAGE, Image *IMAGE1, int THRES)

{
  int X,Y,G0,GB;

  G0=255;
  GB=0;

  for (Y=0; Y<IMAGE->Rows;Y++)
   {
       for (X=0; X<IMAGE->Cols;Y++)
         {
           if (* (IMAGE->Data+X+(long)Y*IMAGE->Cols)> THRES)
                 *(IMAGE1->Data+X+(long)Y*IMAGE->Cols)=G0);
           else
                 *(IMAGE1->Data+X+(long)Y*IMAGE->Cols)=GB);
         }
    }
  }
```

A.4 Descripción

```
//-------------------------------------------------------------------------------------------------
// region_8 (Image *x, int value, int *error_code)

// En esta función consideramos que la imagen IMAGE ya se encuentra binarizada.
//  El objetivo del proceso de Descripción es localizar una región de fondo y marcarla
// con un valor que representa a la conectividad  8-conexo
//-------------------------------------------------------------------------------------------------

void region_8 (Image *x, int value, int *error_code)
{
   int i,j,ii,jj;

 *error_code = 0;
   ii= -1; jj = -1;
   for (i=0; i<x->info->nr; i++)
     {
        for (j=0; j<x->info->nc; j++)
           if (x->data[i][j] == 0)
               {
                 ii=i; jj=j;
                 break;
               }
            if (ii >= 0) break;
     }
  if (ii < 0)
     {
       *error_code = NO_REGION;
        return;
     }
mark8 (x, value, ii,jj);
}
```

```
//-----------------------------------------------------------------------------------------------
// mark8 (struct image *x, int value, int iseed, int jseed)
// En estas función se comienza a marcar una región 8-conexa en el punto iseed, jseed
//-----------------------------------------------------------------------------------------------
void mark8 (struct image *x, int value, int iseed, int jseed)
{
     int i,j,n,m, again;

     if (x->data[iseed][jseed] != 0)
         return;
         x->data[iseed][jseed] = value;
      do
        {
         again = 0;
         for (i=0; i<x->info->nr; i++)
         for (j=0; j<x->info->nc; j++)
          if (x->data[i][j] == value)
             for (n=i-1; n<=i+1; n++)
            for (m=j-1; m<=j+1; m++)
             {
              if (range(x, n, m) == 0) continue;
                if (x->data[n][m] == 0)
                    {
                      x->data[n][m] = value;
                      again =  1;
                    }
             }
             for (i=x->info->nr-1; i>=0; i--)
             for (j=x->info->nc-1; j>=0; j--)
             if (x->data[i][j] == value)
                for (n=i-1; n<=i+1; n++)
                for (m=j-1; m<=j+1; m++)
                 {
                   if (range(x, n, m) == 0) continue;
                     if (x->data[n][m] == 0)
                       {
                          x->data[n][m] = value;
                          again = 1;
                        }
                    }
          } while (again);
}
```

A.5 Reconocimiento

```
//-------------------------------------------------------------------------------------------------
// Mom_cent (char i)
// Esta función tiene como objetivo encontrar los atributos propios de cada objeto
// presente en la imagen. Estos atributos corresponden a los momentos invariantes de
// Hu (momentos centralizados), el factor de compactación, etc.
// Todos estos atributos tienen la particularidad de que son invariantes a
// transformaciones lineales como la rotación, traslación y escalamiento.
//-------------------------------------------------------------------------------------------------

void Mom_cent(int i)
{
  mx = m10 / m00;
  my = m01 / m00;
  mc20 = LARGO*m20 - mx * m10*LARGO;
  mc02 = ANCHO*m02 - my * m01*ANCHO;
  mc11 = ANCHO*m11 - my * m10*ANCHO;
  mcn20 = mc20 / (m00*m00*LARGO*LARGO);
  mcn02 = mc02 / (m00*m00*LARGO*LARGO);
  mcn11 = mc11 / (m00*m00*LARGO*LARGO);
   fi1 = (mcn20 + mcn02);
   fi2 = log10((mcn20 - mcn02)*(mcn20-mcn02)+4*mcn11*mcn11);
   alfa = -(0.5*atan2(2*mc11,mc20-mc02)*180.0/3.1416);
   fcomp = per[i]*per[i]/(m00*(LARGO));
}

//-------------------------------------------------------------------------------------------------
// momento(char *p, char *pi, char num)
// función auxiliar en el cálculo de los momentos normalizados
//-------------------------------------------------------------------------------------------------

void momento(char *p, char *pi, char num)
{
    char *pm,ii,flag=0;
    register char inicio = 0, ia = 0;
   char busqueda(char *, char *, char, char);
   char *p_nuevo(char *, char);
   pm = p;
   do
    {
      y = (unsigned)(pm - pi) / 200;  /* Coordenadas del punto anterior. */
      x = (unsigned)(pm - pi) % 200;
      inicio += 5;
      inicio = (inicio >= 8) ? inicio - 8 : inicio;
      inicio = busqueda(pm, pi, inicio, num);
```

```
 pm = p_nuevo(pm, inicio);
 if(flag==0)
 {
    flag=1;
    ii=ia =inicio;
    y = (unsigned)(pm - pi) / 200;  /* Coordenadas del punto anterior. */
    x = (unsigned)(pm - pi) % 200;
    inicio += 5;
    inicio = (inicio >= 8) ? inicio - 8 : inicio;
    inicio = busqueda(pm,pi,inicio,num);
    pm = p_nuevo(pm, inicio);
  }
*pm = num;
  switch (inicio)
     {
       case 0:
        if(ia>=4&&ia<=7)
          {
           m00 -= x;
           m01 -= (x*y);
           m10 -= ((x-1)*x/2);
           m02 -= (x*y*y);
           m03 -= (x*y*y*y);
           m11 -= ((x-1)*x*y)/2;
           m12 -= ((x-1)*x*y*y)/2;
          }
        if(ia>=4&&ia<=5)
          {
           m20 += (x*x);
           m21 += (x*x*y);
           m30 += (x*x*x);
          }
         else
          if(ia>=0&&ia<=1||ia>=6&&ia<=7)
          {
            m20 -= (y*x*x);
            m21 -= ((y-1)*y*x*x/2);
            m30 -= (y*x*x*x);
          }
          per[num] += LARGO;
          break;
       case 1:
         if(ia>=0&&ia<=3)
           {
             m00 += (x+1);
             m01 += ((x+1)*y);
             m10 += (x*(x+1)/2);
```

```
            m02 += ((x+1)*y*y);
            m03 += ((x+1)*y*y*y);
            m11 += ((x+1)*x*y/2);
            m12 += ((x+1)*x*y*y/2);
           }
        else
           if(ia>=5&&ia<=7)
            {
              m00 += 1;
              m01 += y;
              m10 += x;
              m02 += (y*y);
               m03 += (y*y*y);
               m11 += (x*y);
               m12 += (x*y*y);
             }
           if(ia==5)
              {
               m20 += (x*x);
               m21 += (y*x*x);
               m30 += (x*x*x);
              }
            else
             if(ia>=0&&ia<=1||ia>=6&&ia<=7)
              {
               m20 -= (y*x*x);
                m21 -= ((y-1)*y*x*x/2);
                m30 -= (y*x*x*x);
              }
             per[num] += DIAG;
    break;
case 2:
    if(ia>=0&&ia<=3)
     {
      m00 += (x+1);
      m01 += ((x+1)*y);
      m10 += (x*(x+1)/2);
      m02 += ((x+1)*y*y);
      m03 += ((x+1)*y*y*y);
      m11 += ((x+1)*x*y/2);
      m12 += ((x+1)*x*y*y/2);
     }
    else
     if(ia>=6&&ia<=7)
     {
      m00 += 1;
      m01 += y;
```

```
            m10 += x;
            m02 += (y*y);
            m03 += (y*y*y);
            m11 += (x*y);
            m12 += (x*y*y);
          }
       if(ia>=0&&ia<=1||ia>=6&&ia<=7)
          {
            m20 -= (y*x*x);
            m21 -= ((y-1)*y*x*x/2);
            m30 -= (y*x*x*x);
          }
          per[num] += ANCHO;
          break;
     case 3:
        if(ia>=0&&ia<=3)
         {
          m00 += (x+1);
          m01 += ((x+1)*y);
          m10 += (x*(x+1)/2);
          m02 += ((x+1)*y*y);
          m03 += ((x+1)*y*y*y);
          m11 += ((x+1)*x*y/2);
          m12 += ((x+1)*x*y*y/2);
         }
        else
         if(ia==7)
         {
          m00 += 1;
          m01 += y;
          m10 += x;
          m02 += (y*y);
          m03 += (y*y*y);
          m11 += (x*y);
          m12 += (x*y*y);
         }
        if(ia>=2&&ia<=5)
          {
          m20 += ((y+1)*x*x);
          m21 += ((y+1)*y*x*x/2);
          m30 += ((y+1)*x*x*x);
          }
       break;
     }
  while(pm != p);
```

```
if(ii==0)
    {
      m00 -= x;
      m01 -= x*y;
      m10 -= (x-1)*x/2;
      m02 -= x*y*y;
      m03 -= x*y*y*y;
      m11 -= (x-1)*x*y/2;
      m12 -= (x-1)*x*y*y/2;
      if(inicio>=6&&inicio<=7)
       {
         m20 -= y*x*x;
         m21 -= (y-1)*y*x*x/2;
         m30 -= y*x*x*x;
       }
      else
       {
        m20 += x*x;
        m21 += x*x*y;
        m30 += x*x*x;
       }
      per[num] += LARGO;
     }
   else
    {
    m00 += 1;
    m01 += y;
    m10 += x;
    m02 += y*y;
    m03 += y*y*y;
    m11 += x*y;
    m12 += x*y*y;
    if(inicio>=6&&inicio<=7)
       {
         m20 -= y*x*x;
         m21 -= (y-1)*y*x*x/2;
         m30 -= y*x*x*x;
       }
    else
      {
        m20 += x*x;
        m21 += x*x*y;
        m30 += x*x*x;
       }
    per[num] += DIAG;
  }
}
```

Apéndice B
Código de la RNA en C++

B.1 Descripción general

En este Apéndice, se presenta el código de las clases de RNA implementadas en C++ así como la implantación de éstas dentro de SERNA. Cabe mencionar, que este modelo neuronal está basado en una formalización de RNA propuesta en [17], por lo que dicho modelo no es especifico de ninguna topología; por el contrario, dicho modelo tiene como objetivo, modelar cualquier topología de RNA.

Con base en lo anterior, en los siguientes apartados se presenta el código fuente de las clases CBase_node, CBase_Link, CBase los aspectos más relevantes, considerados en el diseño e implementación de las RNA en C++.

B.2 CBase_node

```
//-------------------------------------------------------------------
------
class CBase_Node            // Clase Base CBase_Node
      {
      private:
           static int ticket;
      protected:
           int id;              // Id
           double value;       // Valor almacenado por este nodo
(neurona)
           LList in_links;      // Lista de valores de entrada al nodo
en cuestión
           LList out_links;     // // Lista de valores de salida al
nodo en cuestión
      public:
           CBase_Node( void );      // Constructor
           ~CBase_Node( void  );                          // Destructor
           LList *In_Links( void );
           LList *Out_Links( void );
           virtual void Run( int mode=0 );
           virtual void Load( ifstream &infile );
           virtual void Save( ofstream &outfile);
           inline virtual double Get_Value( int id=NODE_VALUE );
           inline virtual void Set_Value( double new_val, int
id=NODE_VALUE );
           inline virtual void Set_Error( double new_val, int
id=NODE_ERROR );
           inline int Get_ID( void );
           inline virtual char *Get_Name( void );
           void Create_Link_To( CBase_Node &to_node, CBase_Link *link
);
           virtual void Print( ofstream &out );
           friend void Connect( CBase_Node &from_node, CBase_Node
&to_node,
                                CBase_Link *link );
           friend int Disconnect( CBase_Node *from_node, CBase_Node
*to_node);
};
//-------------------------------------------------------------------
------
CBase_Node::CBase_Node( )      // Constructor
      {
      id=++ticket;
      value=0.0;
      };
//-------------------------------------------------------------------
------
CBase_Node::~CBase_Node( void  ) {};                    // Destructor
//-------------------------------------------------------------------
------
LList *CBase_Node::In_Links( void ) { return &in_links; };
```

```
//-----------------------------------------------------------------------
------
LList *CBase_Node::Out_Links( void ) { return &out_links; };
//-----------------------------------------------------------------------
------
void CBase_Node::Run( int mode ) {};
//-----------------------------------------------------------------------
------
void CBase_Node::Load( ifstream &infile )
     {
     infile >> id;
     infile >> value;
     };
//-----------------------------------------------------------------------
------
void CBase_Node::Save( ofstream &outfile)
     {
     outfile << setw(4) << id << endl;
     outfile << " " << setprecision(18) << value;
     outfile << endl;
};



//-----------------------------------------------------------------------
------
double CBase_Node::Get_Value( int id ) { return value; };
//-----------------------------------------------------------------------
------
void CBase_Node::Set_Value( double new_val ) { value=new_val; };
//-----------------------------------------------------------------------
------
int CBase_Node::Get_ID( void ) { return id; };
//-----------------------------------------------------------------------
------
char *CBase_Node::Get_Name( void )
     {
     static char name[]="CBASE_NODE";
     return name;
     };
//-----------------------------------------------------------------------
------
void CBase_Node::Create_Link_To( CBase_Node &to_node, CBase_Link *link
)
      {
      out_links.Add_To_Tail(link);
      to_node.In_Links()->Add_To_Tail(link);
      link->Set_In_Node(this, id);
      link->Set_Out_Node(&to_node, to_node.Get_ID());
      };
//-----------------------------------------------------------------------
------
void Connect( CBase_Node &from_node, CBase_Node &to_node, CBase_Link
*link )
      {
      from_node.Create_Link_To(to_node,link);
      };
```

```
//-----------------------------------------------------------------
------
void Connect( CBase_Node &from_node, CBase_Node &to_node, CBase_Link
&link )
      {
      from_node.Create_Link_To(to_node,&link);
      };
//-----------------------------------------------------------------
------
void Connect( CBase_Node *from_node, CBase_Node *to_node, CBase_Link
*link )
      {
      from_node->Create_Link_To(*to_node,link);
      };
//-----------------------------------------------------------------
------
int Disconnect( CBase_Node *from_node, CBase_Node *to_node )  //
Remove link
     {
     LList *out_links=from_node->Out_Links();
     int flag=0;
     out_links->Reset_To_Head();
     for (int i=0; i<out_links->Count(); i++)  // for each output link
          {
          if (out_links->Curr()->Out_Node()==to_node)
               {
               flag=1;
               break;
               }
          out_links->Next();
          }

     if (flag==1)          // link exists, delete it from both nodes
          {
    out_links->Curr()->Out_Node()->In_Links()->Del(out_links->Curr());
          out_links->Del_Node();
          return 1;
          }
     else
          return 0;        // link not found
     };
//-----------------------------------------------------------------
------
int CBase_Node::ticket=-1;   // This static variable is shared by all
                             // links derived from the base link class.
Its
                             // purpose is to give each link created
from
                             // the CBase_Link class a unique
identification
                             // number.
//-----------------------------------------------------------------
------
```

B.3 CBase_Link

```
//-----------------------------------------------------------------------
------
// Clase CCBase_Link
//-----------------------------------------------------------------------
------

class CBase_Link   // Clase Base CBase_Link
     {
     private:

          static int ticket;

     protected:
          int id;                      // ID de la conexión
          double value;               // Valor de la conexión
          Base_Node *in_node;          // Nodo de entrada a la conexión
          Base_Node *out_node;         // Nodo de salida de la conexión

     public:
          CBase_Link(void);       // Constructor
          ~CBase_Link( void );            // Destructor de CBase_Link
          virtual void Save( ofstream &outfile );
          virtual void Load( ifstream &infile );
          inline virtual double Get_Value( void );
          inline virtual void Set_Value( double new_val);
          inline virtual void Set_In_Node( Base_Node *node, int id );
          inline virtual void Set_Out_Node( Base_Node *node, int id );
          inline virtual Base_Node *In_Node( void );
          inline virtual Base_Node *Out_Node( void );
          inline virtual char *Get_Name( void );
          inline int Get_ID( void );
          inline virtual double In_Value(void );
          inline virtual double Out_Value( void );
          inline virtual double Weighted_In_Value(void );
          inline virtual double Weighted_Out_Value( void );
};
CBase_Link::Get_Set_Size( void )
     {
     return value_size;
     }
//-----------------------------------------------------------------------
------
CBase_Link::CBase_Link( int size )          // Constructor
     {
     id=++ticket;
     value_size=size;
     if (value_size<=0) value=NULL;
     else value=new double[value_size];
     for (int i=0; i<value_size; i++)     // initialize value set to
zero
          value=0.0;
     in_node=out_node=NULL;
     };
//-----------------------------------------------------------------------
------
```

```
CBase_Link::~CBase_Link( void ) {};              // Destructor for Base
Links

//-------------------------------------------------------------------
------
void CBase_Link::Save( ofstream &outfile )
     {
     outfile << id << endl;
     outfile << value_size;                 // Store value set
     if (value) delete []value;
     value=new double[value_size];
     for (int i=0; i<value_size; i++)
          outfile << " " << setprecision(18) << value[i];
     outfile << endl;
     };
//-------------------------------------------------------------------
------
void CBase_Link::Load( ifstream &infile )
     {
     infile >> id;
     infile >> value_size;
     if (value) delete []value;
     value=new double[value_size];        // Load value set
     for (int i=0; i<value_size; i++)
          infile >> value[i];
     };
//-------------------------------------------------------------------
------
double CBase_Link::Get_Value( int id ) { return value[id]; };
//-------------------------------------------------------------------
------
void CBase_Link::Set_Value( double new_val, int id) {
value[id]=new_val; };
//-------------------------------------------------------------------
------
void CBase_Link::Set_In_Node( Base_Node *node, int id ) {
in_node=node; };
//-------------------------------------------------------------------
------
void CBase_Link::Set_Out_Node( Base_Node *node, int id ) {
out_node=node; };
//-------------------------------------------------------------------
------
Base_Node *CBase_Link::In_Node( void ) { return in_node; };
//-------------------------------------------------------------------
------
Base_Node *CBase_Link::Out_Node( void ) { return out_node; };
//-------------------------------------------------------------------
------
char *CBase_Link::Get_Name( void )
     {
     static char name[]="CBASE_LINK";
     return name;
     };
int CBase_Link::Get_ID( void ) { return id; };
//-------------------------------------------------------------------
------
//-------------------------------------------------------------------
------
```

```
int CBase_Link::ticket=-1;   // This static variable is shared by all
                             // links derived from the base link class.
Its
                             // purpose is to give each link created
from
                             // the CBase_Link class a unique
identification
                             // number.
//-------------------------------------------------------------------
------
```

B.4 Lista doblemente ligada (Llist)

```
//  Implantación de la lista doblemente ligada
//-------------------------------------------------------------------
----------
class LList
     {
     private:

          struct NODE
               {
               NODE *next, *prev;
               CBase_Link *element;
               };

          NODE *head,*tail,*curr;
          int count;

     public:
          LList( void );
          ~LList( void );
          int Add_To_Tail( CBase_Link *element );
          int Add_Node( CBase_Link *element );
          int Del_Node( void );
          int Del( CBase_Link *element );
          int Find( CBase_Link *element );
          inline void Clear( void );
          inline int Count( void );
          inline void Reset_To_Head( void );
          inline void Reset_To_Tail( void );
          inline CBase_Link *Curr( void );
          inline void Next( void );
          inline void Prev( void );
     };
//-------------------------------------------------------------------
------
```

Referencias

[1] Dodd G. G., Rossol L.. ***Computer Vision and Sensor-Based Robots.*** Plenum Press, NY, USA 1979.

[2] Martinetz T.M., Ritter H.J. ***Three/dimensional neural net for learning visuomotor coordination of a robot arm.*** *IEEE Transactions on Neural Networks*. pp 131-136, March 1990.

[3] Arbib Michael A., ***The Handbook of Brain Theory and Neural Networks.*** The MIT Press, 1998.

[4] Wijesona S.W., Wolfe D.F.H., Richards R.J.. ***Eye-to-hand coordination for vision-guided robot control applications. The International Journal of Robotics Reserch***. Vol. 12, No. 1, pp 65-78. February, 1993.

[5] Malo Tamayo A.J., ***Manual de usuario del robot UNIMATE S-130***. Departamento de publicaciones técnicas. Cinvestav-IPN, 1993.

[6] Fu K.S., González R.C., ***ROBOTICS: Control, Sensing, Vision and Intelligence***. **McGraw-Hill/Interamericana .** 1988.

[7] Mazaira Israel, ***Manipulación de objetos en movimiento con el robot Unimate S-103, asistida por visión***. Tesis de Maestria, Cinvestav-IPN, Noviembre1994.

[8] Gonzalez R.C., Woods R.E., ***Digital Image Processing***. Reading, MA: Addison-Wesley, 1992.

[9] Parker J.R., ***Practical Computer Vision using C*** . John Wiley & Sons,Inc., 1994.

[10] Watt Alan, Policarpo Fabio, ***The Computer Image***. Addison Wesley, 1999.

[11] Jain A.K., ***Fundamentals of digital image processing***. Prentice Hall, 1986.

[12] ***Tesis de Licenciatura*** . 1992.

[13] Papoulis, ***Fundamentals of mathematics*** . 1965.

[14] Philips, ***Algoritmo de Philips en el cálculo de momentos*** . 1965.

[15] Schilling, Rober J., ***Fundamentals of robotics: analysis and control***. Prentice Hall, 1990.

[16] Harris C.J., Moore C.G., Brown M., ***INTELLIGENT CONTROL: Aspects of Fuzzy Logic and Neural Nets***. World Scientific, 1993.

[17] Rogers, Samuel Joe, Satyadas Antony, ***Applying a Neural Network Formalism***. Department of Computer Science, The University of Alabama, Alabama, 1994.

[18] Hebb D.O., ***The organization of Behavior***. Wiley, New York, 1949.

[19] Rumelhart D.E., McClelland J.L., ***Parallel Distributed Processing: Explorations in Microstructure of Cognition***. Vol. 1, Cambridge,MA: MIT Press, 1986.

[20] Hagan M.T., Demuth B. Howard, Beale Mark. ***Neural Network Design***. *PWS Publishing Company*, 1996.

[21] Kohonen Teuvo. ***Self organization and Associative Memory***. Springer Verlag, 1987.

[22] Cybenko, G. ***Approximation by superpositions of a sigmoidal function***. *Mathematics of Control, Signals and Systems* Vol. 2, pp 303-314, 1989.

[23] Funahashi, K. ***On the approximate realization of continous mapping by neural networks***. *IEEE Transactions on Neural Networks*. Vol. 2, pp 183-192, 1989.

[24] Hornik, K. ***Multilayer feedforward networks are universal approximators***. *IEEE Transactions on Neural Networks*. Vol. 2, pp 359-366, 1989.

[25] Haykin, S. ***Neural Networks, A comprenhensive foundation***. Prentice Hall, 1999.

[26] Demuth, Howard, Beale M., ***Neural Network TOOLBOX User's Guide***, The MATH WORKS INC, 1998.

[27] Vidyasagar M., ***A Theory of Learning and Generalization***, Springer , 1997.

[28] Moré J.J., ***The Levenberg-Marquardt algorithm: Implementation and theory***, Lecture Notes in Math. 630, Springer-Verlag, Berlin.

[29] Bassi Danilo, Ibarra J.M, ***Sensorial Robot Control based on Connectionist Models for Assembly Tasks***, International Workshop on Neural Networks applied to Control and Image Processing, November 7-10,1994, México D.F.

[30] J. M. Ibarra Zannatha, I. Mazaira, C. A. Campos., ***Desarrollo de un Sistema de Manipulación Robotizada de Objetos***, Memorias del VI CIECE, 6° Congreso Interuniversitario de Electrónica, Computación y Eléctrica. Morelia Michoacán. 11-15 de marzo de 1996.

[31] J. M. Ibarra Zannatha, C. A. Campos. ***Sistema Visuomotor Directo del Robot Unimate S-103 Basado en Redes Neuronales***. Memorias del CIE'96, 2ª Conferencia de Ingeniería Eléctrica. CINVESTAV-IPN. pp. 187-190. México, D.F. 11-13 de septiembre de 1996.

[32] J. M. Ibarra Zannatha, C. A. Campos. ***SERNA: Sistema de Ensamble de piezas basado en Redes Neuronales Artificiales***. Memorias, 2ª Conferencia Instituto Tecnológico de Toluca, Julio 2000.

[33] J. M. Ibarra Zannatha, C. A. Campos. ***Solución al problema de la Cinemática Inversa de un sistema integrado Robot-Visión basado en un enfoque Neuro-Fuzzy***. II Jornadas Iberoamericanas de Inteligencia Artificial aplicada a Sistemas Eléctricos. Santa Cruz de la Sierra, Bolivia. 27-31 octubre, 1997.

[34] Powell Douglass Bruce. Real-Time UML: Developing efficient objects for embedded systems. Addison –Wesley, 1998.

[35] Larson E. Roland, Hostetler P. Robert, Edwards H. Bruce. ***CALCULUS WITH Analytic Geometric*** Mc Graw Hill, 1999.

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará el **Sr.Carlos Alberto Campos Bracho**, declaramos que hemos revisado la tesis titulada:

"Coordinación Visuomotora de Robots basada en Redes Neuronales".

Y consideramos que cumple con los requisitos para obtener el Grado de Maestro en Ciencias en la especialidad en Ingeniería Eléctrica opción Computación.

Atentamente,

Dr. Sergio V. Chapa Vergara

Dr. Alejandro Justo Malo Tamayo

Dr. Juan Manuel Ibarra Zannatha

Dra. Josefina Barrera Cortés