UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLOGICO

RECONOCIMIENTO DE IMÁGENES CON REDES NEURONALES EN UNA TAREA DE MICROENSAMBLE

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

PRESENTA:

GERARDO CARRERA MENDOZA

DIRECTORA:

DRA. TETYANA BAYDYK

CIUDAD UNIVERSITARIA, MEXICO, D.F., OCTUBRE 2004

AGRADECIMIENTOS

A Dios por darme la maravillosa oportunidad de vivir, por acompañarme en cada paso que doy y enseñarme a apreciar cada momento.

A mis padres por su amor infinito.

A mi Madre por ser mi mejor amiga, por acompañarme en todos mis deseos, sueños y esperanzas. Por su comprensión, paciencia y entrega.

A mi Padre por enseñarme, entre muchas cosas, a ser humilde. Por su apoyo incondicional, por estar conmigo siempre.

A mi hermano por crecer juntos y saber que cuento siempre con el.

A mis abuelitas por ser mis inspiraciones.

A mis todos mis familiares por el cariño que me han dado.

A la Dra. Baydyk por su paciencia y por fomentar en mí el camino de la investigación.

A la Dra. Graciela, al Dr. Kussul, a Polo, Beto, Don Mario y a todos los integrantes del laboratorio de micromecánica y mecatrónica del CCADET por hacer mi estancia muy agradable.

Al apoyo que nos brindo PAPIIT para la realización del proyecto.

A todos mis profesores de la Facultad de Ingeniería por poner cada uno de ellos un grano de arena en mi formación académica.

A la UNAM.

INDICE GENERAL

RESUMEN

En esta tesis se presenta una tarea de reconocimiento de imágenes con redes neuronales en una aplicación de microensamble. La red neuronal se llama LIRA (*LImited Receptive Area Classifier*) y se desarrolló en Borland C++ Builder 6.0.

Elegimos la tecnología *MicroEquipment Technology* (MET) para hacer la aplicación. Esta tecnología consiste en el desarrollo de microequipo como secuencia de generaciones. Cada generación de equipo subsecuente es manufacturada con la ayuda de la generación que le precede. El tamaño de cada generación subsecuente es menor que el de la generación anterior. Esta tecnología tiene ventajas en comparación con otras tecnologías.

La aplicación consiste en el desarrollo de microfiltros para filtración fina de líquidos y gases, que esta relacionado con diferentes áreas de procesos tecnológicos y protección ambiental. Los filtros existentes no siempre resuelven con eficiencia sus tareas ya que los filtros de papel para líquidos no permiten la filtración de partículas menores a 0.2 μm. Con MET se propuso hacer filtros basados en un gran número de microanillos especiales, el diámetro interno de cada microanillo es de 0.8 mm, el externo es de 1.3mm y la altura de 0.6 mm. El sistema de microensamble tiene que colocar estos microanillos en los canales del microfiltro.

Para esta tarea el sistema de visión por computadora incluye una cámara para teleconferencias, cuatro lámparas, microequipo y una computadora personal. La idea principal de ésta aproximación es, con imágenes 2D obtener información sobre la posición relativa de la flecha con el anillo y el orificio. La salida del sistema de reconocimiento de imágenes son las coordenadas (*X, Y*) de la posición de la flecha. Con la información de las coordenadas (*X, Y*) podemos mover la flecha en esa dirección para colocar el anillo en el orificio.

Se realizó la simulación del sistema de reconocimiento de imágenes en la tarea de microensamble con el uso del clasificador neuronal LIRA y se obtuvieron muy buenos resultados.

Los resultados fueron obtenidos para dos bases de imágenes. La primera incluye 23 imágenes. La segunda incluye 441 imágenes. Los experimentos se hicieron de la investigación de la influencia de algunos parámetros de la red neuronal sobre la calidad del reconocimiento de las imágenes.

I. INTRODUCCIÓN

1.1. Microdispositivos y su importancia en el mundo contemporáneo

Los microsistemas son sistemas y dispositivos miniaturizados eléctricos o electro-mecánicos cuyas dimensiones críticas son en la escala de unos pocos centímetros hasta micrómetros. Ellos ofrecen la capacidad de combinar funciones de censado, procesado, cómputo y accionamiento. Las dimensiones extremadamente pequeñas inherentes a estos microsistemas ofrecen únicos beneficios y retos [1]. Consideremos tres tipos de aplicaciones de los microsistemas distinguidas por su función [2]:

- Aplicaciones orientadas al macromundo.- En estas aplicaciones el macroefecto es obtenido por la integración en una sola estructura de un vasto número de micropartes, cada uno desarrollando una función. Como ejemplos tenemos: filtros, intercambiadores de calor, monitores táctiles, etc.

- Aplicaciones orientadas al micromundo.- Las aplicaciones potenciales y contemporáneas pueden variar desde microherramientas hasta microrobots micromecánicos, por ejemplo: instrumentos de microcirugía, microsensores, aplicaciones en microelectrónica (microconectores, chips, etc) micromotores y microactuadores, biomicromanipulación (incluyendo manejo de células), etc.

- Aplicaciones que no dependen del tamaño.- Dentro de ésta categoría se encuentran las aplicaciones relacionadas al almacenamiento y procesamiento de la información. Las aplicaciones de los microsistemas dentro de ésta área es dirigida hacia la miniaturización de los dispositivos de almacenamiento de la información (sistemas de grabado en cintas magnéticas, discos ópticos, etc) y hacia el desarrollo de nuevas tecnologías y dispositivos para el procesado de información.

Los productos del futuro son flexiblemente configurados usando componentes mecatrónicos inteligentes. Los componentes tienen un alto nivel de integración de funciones y de elementos funcionales microestructurados, por ello es necesaria una tecnología de producción de alta precisión y flexibilidad. La integración de técnicas también hace necesaria la construcción de sistemas de producción en una manera modular [3].

Mientras los componentes se vuelvan cada vez más pequeños y el nivel de precisión aumente, los factores de daño y desestabilización se vuelven más grandes. La manipulación de estos componentes miniatura necesita de tecnologías adecuadas que brinden precisión y estabilidad, es por eso que es de gran utilidad la investigación de nuevas tecnologías que permitan la producción de mircocomponentes con una precisión elevada.

1.2. Tecnologías de la producción de microdispositivos.

En nuestros días, las tecnologías para la miniaturización de estructuras mecánicas han sido desarrolladas dentro de los campos que son comúnmente llamados como "*Micro Electro Mechanical System*" (MEMS) en Estados Unidos, "*MicroSystem Technology*" (MST) en Europa, "*Micromachine Technology*" (MMT) en Japón y "*MicroEquipment Technology*" (MET) en México.

Existen por lo menos dos principales técnicas de la producción de microdispositivos. La primera es la tecnología MEMS [4, 5]. Esta tecnología está basada en microelectrónica. La tecnología basada en microelectrónica permite la creación de microdispositivos que incorporan simples componentes mecánicos fabricados principalmente de silicón.

Esta tecnología se está desarrollando muy rápidamente y encuentra muchas aplicaciones, por ejemplo: en el campo de los sensores y actuadores, en el campo de las telecomunicaciones, en la industria aeroespacial y automotriz, en telecomunicaciones se usan para producir conmutadores ópticos, también se producen nuevos componentes en electrónica RF, intercambiadores de calor, biotecnología, medicina, neurofisiología, micromotores, microbombas, etc [6]. Las principales ventajas y desventajas son:

Ventajas:

- Alta precisión de los equipos;
- Fabricación de grandes lotes;

Desventajas:

- Es muy difícil y a veces no es posible generar dispositivos 3D;
- Uso de materiales compatibles con las tecnologías de silicio.

Sin embargo para realizar todas las ventajas de ésta tecnología es muy importante tener procesos de ensamblado avanzados y software desarrollado.

La segunda técnica está conectada con la producción de microequipo y microindustrias con el uso de equipo convencional. Se relaciona con el desarrollo de complejos microsistemas como herramientas, manipuladores y robots miniatura que llama al desarrollo de sofisticadas estructuras mecánicas de diversos materiales que tienen partes complejas y móviles en 3D. Esta aproximación es llamada MMT. Permite la producción de microequipo de alto costo para la producción de microdetalles y microsistemas.

Para hacer microequipo mecánico, en [2, 6, 7, 8] se propone una técnica con el uso del siguiente esquema. El equipo debe ser desarrollado como una secuencia de generaciones. Cada generación debe incluir equipo (herramientas, manipuladores, dispositivos de

ensamble, instrumentos de medición, etc) suficiente para la manufacturación de un conjunto idéntico de equipo de la siguiente generación. Cada generación de equipo subsecuente es manufacturada con la ayuda de la generación que le precede. El tamaño de cada generación subsecuente es menor que el de la generación que le precede (Figura 1). Esta tecnología basada en micromáquinas herramientas y microdispositivos de ensamble es llamada "*MicroEquipment Technology*" (MET).

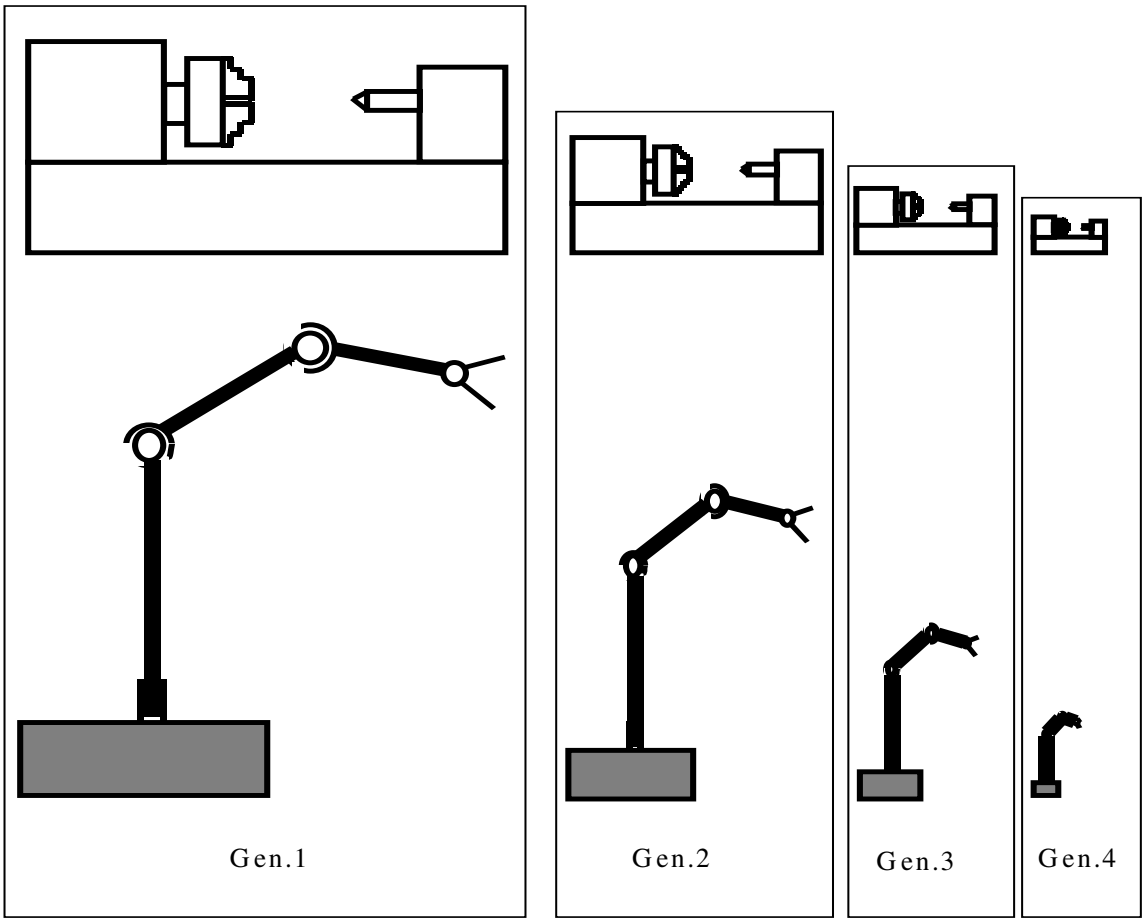


Figura 1.- Disminución del tamaño de las máquinas herramientas en distintas generaciones (tecnología MET)

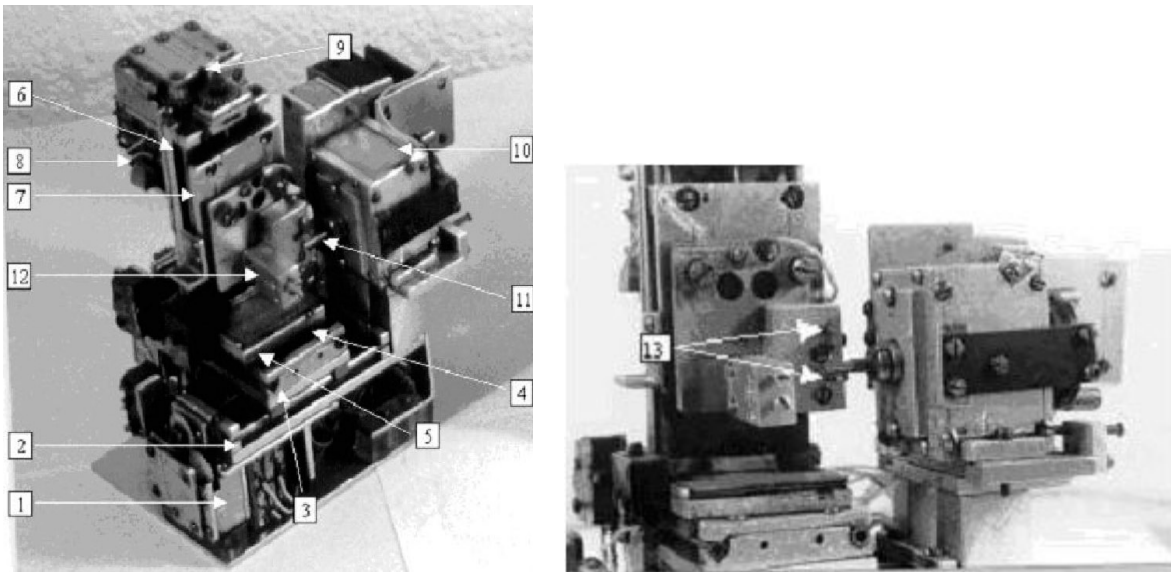
La situación que lleva a MET es la siguiente:

- La miniaturización de equipo lleva a un decremento en el área de ocupación, materiales y energía consumida, y, por lo tanto disminuye los costos asociados.
- Se reducen los costos de mantenimiento y se tiene un alto nivel de automatización.

Las ventajas de MET son las siguientes:

- Desarrollo de microdispositivos de bajo costo.
- Posibilidad de emplear diversos materiales de manufactura.
- Posibilidad de producir microcomponentes 3D.

El primer prototipo que fue desarrollado tiene las capacidades de fresado, torneado, cepillado, pulido y barrenado [8]. Para cambiar el tipo de trabajo es necesario cambiar las herramientas y los programas. En su diseño se usaron los siguientes criterios: bajo costo y posibilidad de disminuir la escala de la micromáquina herramienta. Este prototipo es mostrado en las figuras 2a y 2b. En la figura 3 se muestran algunas piezas manufacturadas por este primer prototipo [8].



(a) (b)

Figura 2a y 2b. Primer prototipo desarrollado de micromáquina herramienta [6]

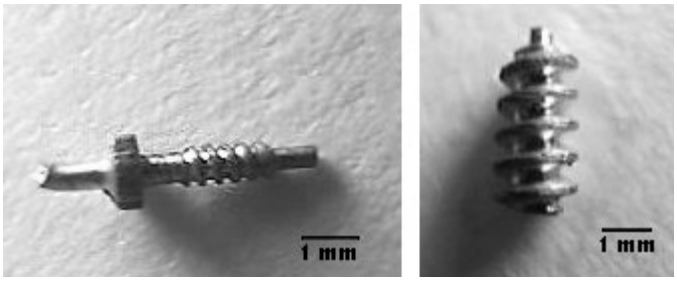


Figura 3. Piezas manufacturadas por el primer prototipo [6].

Las principales ventajas del primer prototipo son la simplicidad de su diseño y la sencilla manufactura de sus componentes, el sistema de control también es muy simple. Pero en este prototipo hubo algunos inconvenientes, los cuales se trataron de eliminar en el segundo prototipo. Era necesario incrementar la complejidad del prototipo, así como mejorar el sistema de control. Los principales inconvenientes del primer prototipo son la baja rigidez y el uso del modo dinámico para el control de los motores de pasos.

El segundo prototipo de micromáquina herramienta tiene las siguientes dimensiones 130 * 160 * 85 mm^3; la figura 4 muestra este segundo prototipo.

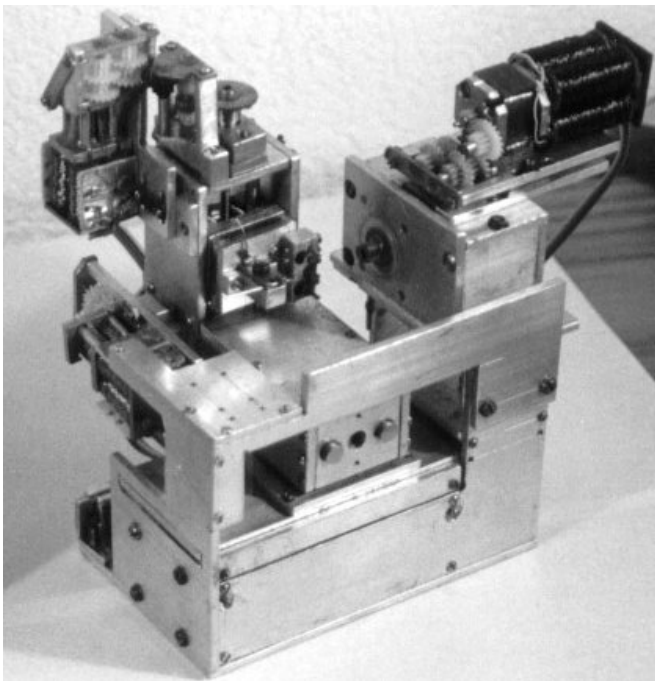


Figura 4. Segundo prototipo de micromáquina herramienta de la primera generación [6].

1.3. Descripción de la tarea de microensamble

Las tareas de ensamble pueden ser muy diferentes, dependiendo del área y de los métodos de manufactura de los microproductos. Tradicionalmente, el proceso de ensamble tiene tres etapas: 1. etapa de aproximación, 2. etapa de contacto y búsqueda, 3. etapa de inserción.

La tarea de microensamble consiste en juntar o unir microobjetos. Con el incremento de la miniaturización se vuelve más y más difícil usar robots convencionales para el ensamble de micropiezas, es necesario tener robots adecuados para manipular microcomponentes. Aparte de esto, los componentes de los microsistemas no han sido estandarizados por lo que un modulo de ensamble debe ser capaz de manejar piezas de diferente forma [9].

El microensamble difiere del ensamble convencional en algunos aspectos. Estas diferencias son causadas por las pequeñas dimensiones de las partes a ser ensambladas. Los problemas más grandes provienen de las fuerzas superficiales que se vuelven más dominantes con el decremento en las dimensiones de las partes, como [9]:

- fuerzas electrostáticas,
- tensión en las fuerzas superficiales debido a la humedad,
- fuerzas de adhesión,
- fuerzas de Van-der-Waals.

La Figura 5 muestra la relación entre las fuerzas adhesivas y el tamaño de las partes.

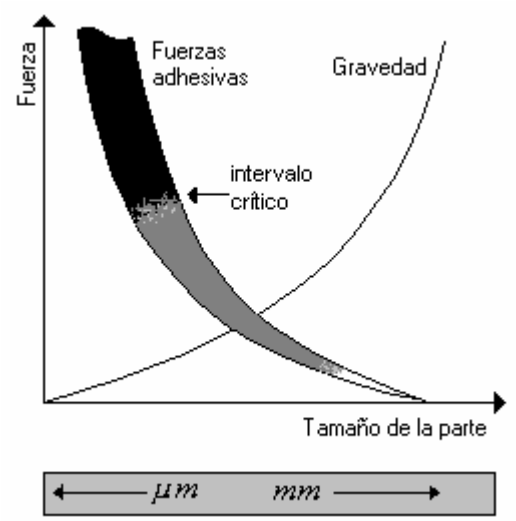


Figura 5. Relación entre fuerzas adhesivas y el tamaño de las partes

En la escala de los μm las fuerzas adhesivas entre las partes y el microsujetador hacen difícil de realizar la tarea de microensamble. Puede ser más fácil para el micromanipulador agarrar y manipular un objeto que liberarlo después. Por eso la tarea de microensamble demanda el desarrollo de métodos nuevos para resolver éstos problemas.

En [10] se reporta el desarrollo una tecnología de fabricación automática de microfiltros. El problema de los filtros existentes es que no siempre resuelven los problemas de filtración eficientemente, algunos de ellos no tienen un buen desempeño, otros no dan una filtración lo suficientemente fina o no pueden ser usados en todas las condiciones ambientales. El filtro propuesto contiene una caja que contiene una entrada, una salida y una válvula de escape, el líquido (o gas) entra y fluye por unos espacios internos de las columnas de microanillos (Figura 6). Después de esto fluye por rendijas entre los microanillos y sale del recipiente; las partículas pesadas son retenidas en los espacios internos de las columnas de los microanillos y son removidas usando la válvula de escape. En éste dispositivo la tarea de microensamble es colocar los microanillos en columnas. El diámetro de cada anillo es de 1 mm aproximadamente.

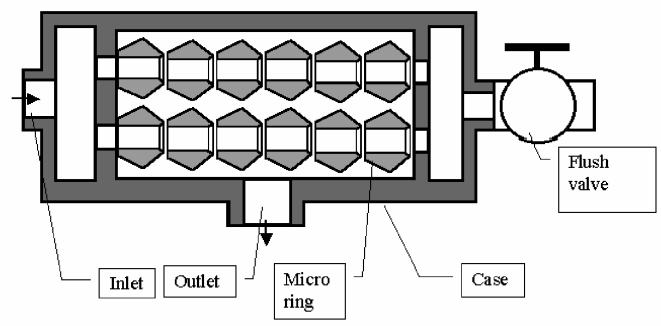


Figura 6. Microfiltro

Para eliminar los efectos de las fuerzas electrostáticas y fuerzas de Van-der-Waals fue desarrollado el siguiente esquema para el proceso de ensamble (Figura 7): El microanillo (3) es puesto en una aguja (1) y es introducido con la aguja dentro del hoyo (a, b); después de esto el microanillo es sujeto en el hoyo con el tubo (2)(c); en el siguiente paso el tubo con la aguja son movidos hacia un lado y el microanillo es detenido por el hoyo y no puede seguir al tubo (d, e) [7].

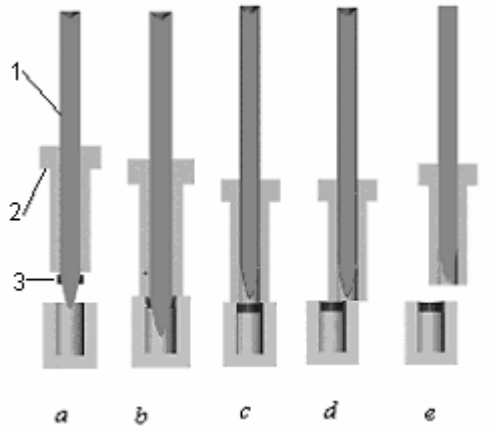


Figura 7. Ensamble de microanillos

Se menciono anteriormente que uno de los objetivos del microequipo es el bajo costo de los componentes. Los componentes de bajo costo no permiten un ensamble del dispositivo con una alta precisión. Para compensar esta desventaja es necesario desarrollar un algoritmo adaptable de microensamble usando por ejemplo, un sistema de visión por computadora de bajo costo.

1.4. Uso de visión por computadora para la tarea de microensamble.

En una tarea de microensamble hay muchos aspectos que se tienen que cuidar, entre ellos se encuentran principalmente el diseño de sensores y un buen sistema de visión. La visión por computadora es una tarea de gran importancia ya que podemos obtener una información más precisa acerca de las posiciones relativas de los microobjetos que se requieren ensamblar.

Se aplican diferentes técnicas de visión para el proceso de microensamble: cámaras CCD, microscopios, triangulación de sensores, etc [11].

Desafortunadamente la visión por computadora en el microensamble también sufre de las pequeñas dimensiones de las partes. En el caso de un microscopio óptico, los principales problemas son el pequeño campo de visión y de profundidad.

A continuación se presentan algunos trabajos acerca de la tarea de ensamble.

Esta tarea puede ser automatizada usando información de sensores o información visual. Para obtener ésta información han sido desarrollados diferentes sistemas. Por ejemplo en [14] usan múltiples rayos láser que son proyectados en el objeto, y una cámara blanco y

negro CCD, la cuál es utilizada para grabar la imagen de la escena. En ésta situación, se analizan imágenes 2-D para obtener información en 3-D.

Otros usan un microscopio estereoscópico, tres cámaras CCD, un micromanipulador y una computadora personal en la tarea de reconocimiento de micropiezas y su forma en 3D para la micromanipulación [15].

En [12] con un microscopio óptico y una cámara CCD se forma un sensor local del sistema deliberando información visual del "micromundo". Mediante este sistema se detectan la ubicación de los microobjetos manejados y las herramientas del robot, y se hace el posicionamiento. También se usa un sistema láser de medición el cual puede incrementar adicionalmente el rendimiento del sensor local. La posición del robot en la base de trabajo es supervisada por un sensor global del sistema usando otra cámara CCD.

El sistema que a continuación se describe es el que se usó en ésta tesis. En [7] se propone el uso de visión por computadora para resolver el problema del ensamble de un microfiltro [10]. El microfiltro contiene un gran número de microanillos insertados en microcanales especiales (figura 6).

Para ésta tarea el sistema de visión por computadora incluye una cámara para teleconferencias, cuatro lámparas, microequipo y una computadora personal [7].

La principal idea de ésta aproximación es remplazar el sistema de estereovisión, el cuál requiere de dos video cámaras, con un sistema basado en una cámara para teleconferencias, con el costo de 50-70 dólares, y cuatro fuentes de luz. Las sombras de las fuentes de luz permiten obtener información sobre una posición relativa en 3-D de la aguja con el microanillo y el orificio (Figura 8).

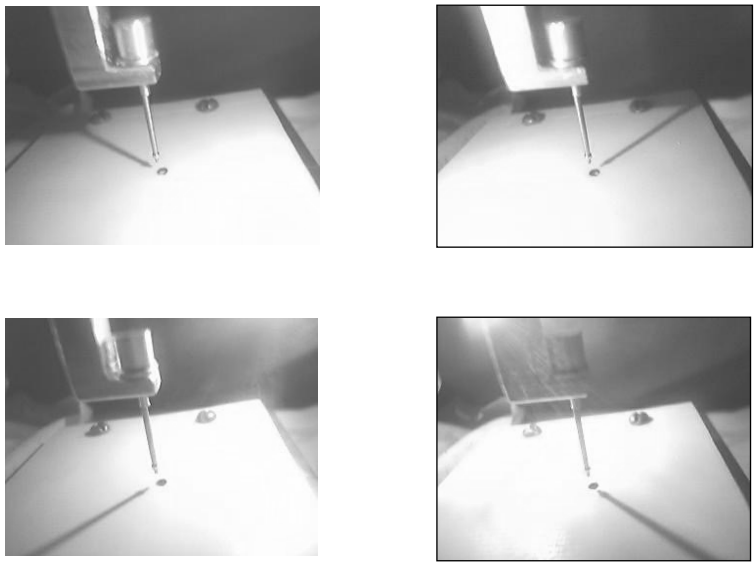


Figura 8. Ejemplos de imágenes obtenidas con un sistema de cuatro fuentes de luz

En el capítulo II vamos a analizar éste sistema más detalladamente.

1.5. Uso de redes neuronales en reconocimiento de imágenes

Existen muchas aplicaciones de reconocimiento de patrones en las que las redes neuronales trabajan muy eficientemente, por ejemplo: clasificación, extracción de características, compresión de datos, etc. Las aplicaciones de las redes neuronales para el reconocimiento de imágenes incluyen percepción remota, imágenes médicas, visión robótica, reconocimiento militar, cartografía, reconocimiento de caracteres, reconocimiento de huellas digitales, y reconocimiento facial. Para cuestiones relacionadas a esta tesis lo que nos interesa saber es la aplicación de redes neuronales en reconocimiento de imágenes.

Las redes neuronales se prefieren para tareas de reconocimiento de imágenes debido a sus capacidades de procesamiento paralelo, así como las habilidades de entrenamiento y toma de decisiones. El reconocimiento de una imagen incluye procesamiento de la imagen y reconocimiento de patrones con redes neuronales [13].

Los modelos de redes neuronales proveen de una alternativa para implementar técnicas de mejorado, como por ejemplo: existen modelos que han sido desarrollados para la restauración de la imagen, extracción de características, segmentación de texturas, manipulación de escala de grises, detección de bordes, extracción de ruido, interpolación, etc. [18]. Se usan diferentes arquitecturas de redes neuronales para trabajar con el reconocimiento de imágenes.

Debido a que la tarea del uso de visión por computadora en el microensamble es relativamente nueva, se tienen pocos trabajos acerca del uso de redes neuronales en ésta tarea. Sin embargo la tarea de microensamble requiere de un buen sistema de visión debido a la gran precisión que se necesita para maniobrar diminutos objetos, tanto para el hardware como para el software.

1.6. Organización de la tesis

En el microensamble utilizamos las redes neuronales como clasificadores de imágenes. En nuestro sistema se desarrolla un sistema de visión por computadora basado en un algoritmo de reconocimiento de imágenes con redes neuronales, el cuál lo utilizamos como clasificador y reconocedor de imágenes. Como mencioné anteriormente, este sistema se concreta en reducir las tareas de visión por computadora de 3D a tareas de 2D. El algoritmo de redes neuronales que se utiliza en este sistema se llama LIRA (*Limited Receptive Area Classifier*); los algoritmos se desarrollaron en Borland C++ Builder 6.0. En el capítulo II se describirá la tarea de microensamble. En el capítulo III se dará una descripción detallada de este clasificador neuronal.

En el Capítulo IV se presenta el software para reconocimiento de imágenes con el uso del clasificador neuronal LIRA.

En el Capítulo V se describen los resultados obtenidos y sus análisis.

Las conclusiones se presentan en el Capítulo VI.

II. TAREA DE MICROENSAMBLE Y USO DE VISION TÉCNICA

2.1. Importancia de la tarea de microensamble

La mayoría de nosotros buscamos la comodidad y la funcionalidad en las cosas, buscamos principalmente un menor peso y tamaño, además de que la funcionalidad sea mayor. A medida que avanza el tiempo, muchos de los objetos que utilizamos en nuestra vida diaria cada vez son más y más cómodos, cada día nos asombramos más del tamaño de los dispositivos electrónicos y mecánicos principalmente, además de que observamos un aumento en las capacidades de éstos. Este fenómeno se observa mejor en los dispositivos electrónicos, por ejemplo el tamaño de los teléfonos celulares de hace dos años y los teléfonos celulares de ahora tienen una gran diferencia en muchos sentidos, son más pequeños, pesan menos y realizan muchas funciones. Las computadoras se vuelven más pequeñas y más funcionales a medida que pasa el tiempo, son muchísimas las aplicaciones en donde podemos observar ésta tendencia.

Son muchas las aplicaciones en donde podemos ocupar componentes micromecánicos; una de ellas es la industria del almacenamiento de datos. Los discos duros cada vez aumentan su capacidad de almacenamiento impresionantemente, además de su tamaño. En [23] se describe un sistema de microensamble capaz de ensamblar micropartes magnéticas en microactuadores electromagnéticos lineales para el uso en discos duros. La industria de almacenamiento de datos ha experimentado un rápido crecimiento en la capacidad y densidad de datos en la pasada década. Hay una alta demanda para incrementar la densidad de los datos, los números de pistas por pulgada (TPI) deben ser incrementados en cada generación de discos. En la presente generación el rango es de 30 a 50 kTPI y está proyectado incrementar a 100 kTPI dentro de dos generaciones. Es por ello que se requiere el uso de microactuadores para microposicionar la cabeza.

En el presente muchas áreas de la industria tienen fuertes tendencias hacia la miniaturización de productos. Los componentes mecánicos de éstos productos como regla son manufacturados usando equipo convencional de larga escala o equipo basado en tecnología microelectrónica (MEMS). El primer método tiene algunas desventajas porque el equipo convencional de larga escala consume mucha energía, espacio y material. Es muy difícil obtener una alta precisión en la producción de micropiezas con herramientas grandes. El segundo método parece ser más avanzado pero también tiene algunas limitaciones, por ejemplo: componentes de 2 o 2.5 dimensiones y materiales compatibles con tecnología de silicón [8].

Los dispositivos micromecánicos poseen de considerables perspectivas futuras en muchas áreas de aplicación, por ejemplo en el área de la medicina, la robótica, astronomía, etc. Para manipular, fabricar u observar piezas de escalas muy pequeñas es necesario hacer sistemas con capacidades especiales que permitan el manejo de éste tipo de piezas, sistemas que tengan características muy similares a las piezas, por ejemplo en el tamaño, para esto se necesitan hacer microsistemas o microfábricas. El ensamble de un microsistema híbrido es una tarea difícil y un verdadero reto para la comunidad de investigadores en robótica y en

muchas áreas ya que debe proveer de flexibilidad, alta precisión y rápidas facilidades de microensamble.

El futuro desarrollo de la tecnología de microsistemas MET depende en gran medida de la disponibilidad de facilidades flexibles de microensamble, las cuales permiten que los componentes sean ensamblados automáticamente, reduciendo el costo de producción y simultáneamente obteniendo alta calidad. Por esta razón, el ingeniero en MET está buscando robots con micromanipulación flexible que tengan habilidades de transportación y manipulación. El ensamblado de microsistemas, incluyendo: transportación no-destructiva, manipulación y posicionamiento de objetos diminutos que tienen dimensiones en el rango de $nm\ y\ \mu m$, se ha vuelto una de las aplicaciones más prometedoras para microrobots inteligentes [12].

La miniaturización de las máquinas herramientas tiene otras razones muy importantes. Con la disminución de la escala del microequipo, se disminuyen los errores en la producción de micropiezas. Por ejemplo [5]:

- disminución en la deformación debida al calor;
- las amplitudes de las vibraciones en máquinas más pequeñas es menor que en máquinas más grandes, porque las fuerzas inerciales decrecen a la cuarta potencia del factor de escala; y las fuerzas elásticas decrecen a la segunda potencia del factor de escala.

Todo esto da la posibilidad de producir micropiezas más precisas con la disminución de la escala del microequipo.

2.2. Métodos para resolver la tarea de microensamble.

Existen diferentes métodos para realizar procesos de microensamble. Uno de ellos es mediante microfábricas, las cuáles contienen micromáquinas herramienta especialmente diseñadas para trabajar en la manufactura o ensamblado de micropartes.

En Japón se inició el primer proyecto para la creación de una microfábrica. El laboratorio de ingeniería mecánica [24] desarrolló una microfábrica de mesa (figura 9). La microfábrica consiste de máquinas herramienta como un torno, fresadora, sujetadora y maquinas ensambladoras como un brazo transportador y una mano de dos dedos. Esta microfábrica portable tiene dimensiones externas de 625*490*380 mm y pesa 34 Kg (peso del cuerpo principal 23 Kg). Usa 3 cámaras CCD miniaturas montadas en cada máquina herramienta, que despliegan la imagen de una sección de la máquina en un monitor LCD de 5.8 inch. Esta fábrica puede producir piezas miniatura y puede ensamblarlas. Solo 2 joysticks y un botón son usados para operar el equipo. Los autores reportaron algunas características de su microfábrica como son:

- Marcada reducción en el consumo de energía.
- El decremento de las fuerzas inerciales facilita el control. La velocidad y la precisión en el posicionamiento se incrementan.
- La miniaturización y la integración incrementan el grado de libertad del diseño de productos, además de que se facilita la modificación del sistema.

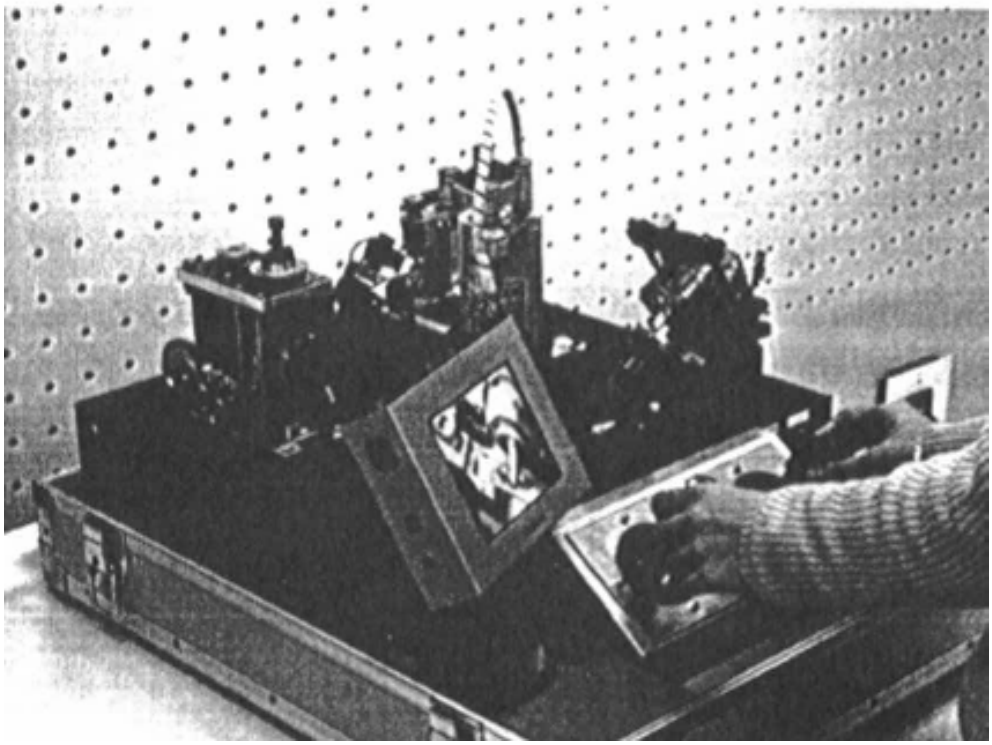


Figura 9.- Microfábrica japonesa

Con éste equipo se produjeron diferentes muestras de microdetalles y microdispositivos. Por ejemplo: delgadas agujas de 50 μm de diámetro y una longitud de 600 μm, balines con diámetro de 0.9 mm.

En [25] se desarrolla una microfábrica que puede desarrollar procesamiento y ensamblado. La unidad de ensamblado consiste de muchos módulos hechos de dispositivos micromáquinas y realiza adhesión de partes, ensamblado, inspección después del ensamblado y mucho más. Hay dos tipos de herramientas de trabajo, una es un dispositivo sujetador de partes y el otro es un dispositivo de adhesión para fijar partes. Para el dispositivo sujetador son utilizados un dispositivo electromagnético para sujetar materiales magnéticos y un dispositivo aspirador para sujetar partes. La microfábrica contiene un microbrazo el cuál maneja las herramientas de trabajo, éste microbrazo es un manipulador multiflexible. En [25] se desarrollan sujetadores de aspiración modulares para el manejo de pequeños circuitos MMC´s (Millimiterwave Monolithic Integrated Circuits) sensibles al tacto. Demuestran que el equipo modular tiene un potencial muy grande para cruzar la barrera entre el micro y el macroensamble así como la capacidad de soportar tanto métodos de manufactura convencionales como microtecnológicos. Los sujetadores son diseñados en

forma modular para permitir diferentes combinaciones de interfaces cambiables del sujetador con cabezas sujetadoras y platos sujetadores. Debido a la sencillez y a una funcionabilidad confiable, los sujetadores son muy usados en la industria, partes pequeñas como chips o micropartes mecánicas como engranes son tomados con una gran seguridad mediante aspiración. Esta es una manera muy eficiente de implementar estrategias eficientes para ensambles automatizados.

En [23] se describe un sistema de microensamble capaz de ensamblar micropartes magnéticas en microactuadores lineales electromagnéticos para el uso en discos duros de dos etapas. Los microactuadores consisten de cuatro magnetos permanentes con dimensiones de aproximadamente 100 a 200 micrometros. El ensamble es complicado por la presencia de efectos de superficie y fuerzas magnéticas. Esto implica una estrategia de planeamiento para el mecanismo de interacción de partes a tales escalas para todas las fases de contacto y así garantizar ensambles repetitivos exitosos. El dominio de las fuerzas interactivas en pequeñas escalas causa dificultades en el manejo de partes y en la micromanipulación, estas fuerzas interactivas son fuertemente dependientes del ambiente de operación y de las propiedades de los materiales de las superficies de contacto; también la presencia de fuerzas magnéticas se añade a la complejidad del manejo de partes y a la planeación del ensamble. Las estrategias de ensamble deben ser desarrolladas de tal manera que cuenten con las fuerzas interactivas durante todas las fases de contacto del ensamble. La configuración requerida de ensamble para la tarea es la siguiente: cuatro magnetos con magnetización opuesta alternada deben ser recogidos y puestos en una base. Subsecuentemente un plato sujetador es adherido a éste ensamble con una capa delgada de adhesivo.

En el ensamble tradicional y en la planeación de tareas, muchos métodos son basados en el desensamble de un producto. Sin embargo, éste no puede ser el caso del microensamble debido a la naturaleza irreversible de las operaciones. Por ello, los métodos desarrollados para el ensamble convencional a macro-escala no pueden ser utilizados directamente al microensamble. El modelado y la simulación son las herramientas fundamentales cuando se diseñan técnicas de manejo de micropartes y la planeación del ensamble. El trabajo en el modelado de fuerzas adhesivas, y la utilización de modelos en la micromanipulación han sido llevados a cabo por muchos grupos de investigación. El trabajo en éste campo se puede clasificar en 5 grupos [26]:

- Trabajo temprano en el estudio de fuerzas en el microdominio y fenómenos en la micromanipulación.
- Análisis de microoperaciones basadas en modelos analíticos de microfuerza.
- Modelos computacionales de microfuerza.
- Simulación de diferentes modelos en micromanipulación.
- Modelo dinámico para microoperaciones.

El microensamble tiene que ser hecho tomando en cuenta la geometría de las partes y herramientas, propiedades de los materiales, fuerzas en la escala de μm y condiciones ambientales. En el microensamble es requerido un modelo interactivo para micro-operaciones y debe de prestarse mucha atención a las incertidumbres. La no-homogeneidad de la superficie introduce más incertidumbre a esa escala.

Ha habido mucho progreso en el ensamble tradicional automático y la planeación de tareas. El ensamble automático y la planeación de tareas son basados en modelos. En la planeación tradicional, los modelos geométricos en muchos casos son suficientes pero ahora han sido introducidos los modelos que envuelven fuerzas de contacto. En el microensamble un modelo dinámico que incluya adhesión y fuerzas de contacto va a ser necesario para una planeación efectiva de los movimientos finos y el sujetado [26].

En [27] se desarrollan varias tecnologías elementales para ensamblar estructuras de 3 dimensiones con micropartes y también se desarrollan 2 sistemas únicos de ensamble. Para la producción a mediana escala se desarrolló un sistema de automatización tipo isla el cuál realiza una producción celular usando dispositivos baratos y de tamaño pequeño. Además la producción a escala de herramientas para tratamientos endoscopicos es pequeña debido a sus tipos variados. Se construyó una microfábrica operada por un trabajador la cual conjunta la experiencia y las habilidades del trabajador, con tecnologías de ensamblado. Se realiza un sistema de ensamble de productos microópticos. Este puede ensamblar lentes con un diámetro externo de 1mm, una cámara CCD y un marco de lente. Este trabajo es muy difícil de realizar con sistemas automatizados convencionales de ensamble. El sistema trae automatización y una mayor exactitud para insertar un microlente en una cámara CCD y un marco de lente. Este sistema de microensamble es mucho más eficiente respecto al costo que el equipo convencional para la misma tarea.

También se realiza un sistema de microsoldado (microfábrica operada por un trabajador) en la cuál se examinan tecnologías láser de soldado como elemento base del microensamble. Se tienen tres aspectos: posicionamiento, manejo y ensamble. Por lo tanto fueron desarrollados varios tipos de módulos para ensamblar micropartes. Para ensamblar estructuras de 3 dimensiones que usan micropartes, se necesitan muchos actuadores para aproximarse al área límite de trabajo, y ellos también necesitan prevenir la interferencia entre ellos mismos. Se construyó una estación de ensamblado para obtener suficiente área de trabajo usando motores de pasos para superficies. Es un tipo de motor de pasos que usa amortiguadores de aire para habilitar movimientos suaves en los ejes $X - Y$. La característica de éste sistema es que el operador tiene el control y decisión para procesos clave magnificando su vista con un microscopio y ensamblando usando un micromanipulador.

Desde mi punto de vista la presencia de un operador es una gran desventaja de este sistema, ya que la tendencia actual es que los procesos sean autónomos, y en este sistema se depende del operador para hacer el ensamble. Además de los otros errores también se pueden tener errores humanos en el proceso del ensamble.

En estos sistemas, existen muchos problemas como la exactitud en el ensamblado o la mejora de la productividad.

En el microensamble es típicamente fácil levantar objetos, pero soltarlos con exactitud es mucho más difícil debido a las fuerzas adhesivas. A partir de la escala en que las partes se adhieren fácilmente a la herramienta sujetadora, la secuencia de ensamble de partes de pequeña escala no es reversible: revirtiendo los movimientos utilizados para levantar un objeto no garantiza la liberación de este; por eso es necesario desarrollar métodos que faciliten estos procesos, por ejemplo en [7] (figura 7). En nuestro caso, el sujetador del dispositivo de ensamble es la aguja en el tubo. El proceso de ensamble (figura 7) es el siguiente: el microanillo es puesto en una aguja y es introducido con la aguja dentro del hoyo; después de esto la aguja es removida y el microanillo es sujeto en el hoyo con el tubo; en el siguiente paso el tubo con la aguja son movidos hacia un lado y el microanillo es detenido por el hoyo y no puede seguir al tubo [7].

2.3. Uso de visión por computadora aplicada para la tarea de reconocimiento de imágenes en la tarea de microensamble.

En casi todos los sistemas que analizamos en el subtema 2.2, usan visión por computadora.

La visión por computadora es una rama de la inteligencia artificial que tiene por objetivo modelar matemáticamente los procesos de percepción visual en los seres vivos y generar programas que permitan simular estas capacidades visuales por computadora.

Considerando la capacidad visual de nuestros ojos y cerebro, los sistemas artificiales correspondientes son totalmente primitivos. El intervalo de objetos que pueden manejar, la velocidad de interpretación y la susceptibilidad a problemas de iluminación y variaciones menores, en textura y reflectancia de los objetos, son ejemplos de las limitaciones de la tecnología actual. Por otra parte, la visión por computadora tiene claras ventajas en tareas repetitivas y a altas velocidades, por ejemplo en la inspección ininterrumpida en una línea de ensamble [28].

El amplio espectro de aplicaciones cubierto por la visión por computadora, se debe a que permite extraer y analizar información espectral, espacial y temporal de los distintos objetos.

La información espectral incluye frecuencia (color) e intensidad (tonos de gris). La información espacial se refiere a aspectos como forma y posición (una, dos y tres dimensiones). La información temporal comprende aspectos estacionarios (presencia y/o ausencia) y dependientes del tiempo (eventos, movimientos, procesos).

2.3.1. Definición de la tarea de reconocimiento de imágenes

El reconocimiento de imágenes es en general una serie de pasos que transforman una imagen determinada en información que la computadora pueda interpretar. El elemento mínimo de una imagen es el píxel y se puede definir como cada una de las casillas o celdas en las que se puede descomponer una imagen digital. Una imagen digital es una matriz de valores numéricos que definen el color y la intensidad de cada píxel en una imagen. Matemáticamente un pixel se define como:

$$pixel = f(x, y)$$

Donde el valor de *f* es la intensidad del color en las coordenadas *(x, y).*

Hay muchas maneras de reconocer una imagen o un objeto dentro de una imagen, por ejemplo mediante la detección de contornos. Cuando queremos reconocer el perímetro de un objeto entonces solo hay que identificar una línea cuyos pixeles tengan el valor de 1 y hay al menos un píxel con valor de 0 en su vecindad. Se mencionan 2 algoritmos para el reconocimiento de patrones, los cuales son [32]:

- Algoritmos de procesamiento de bajo nivel para extraer límites y regiones de un patrón.
- Algoritmos de alto nivel para reconocer el objeto original basado en el conocimiento del objeto y sus características morfológicas.

Tomando en cuenta estos algoritmos se mencionan 5 pasos a seguir para el reconocimiento de patrones en imágenes digitales:

- Condicionamiento: Toda imagen ingresada al sistema de reconocimiento se compone de un patrón que contiene la información del objeto que se quiere reconocer. El condicionamiento estima el patrón de información a partir de la observación de la imagen y puede suprimir ruido, así como normalizar el fondo de la imagen.

- Etiquetado: El etiquetado determina en que clase de eventos espaciales participa cada pixel. Evento espacial se considera todo aquello que posee atributos similares dentro de una imagen.

- Agrupamiento: Identifica a todos los pixeles conectados que participen en el mismo evento. El agrupamiento es una operación en la que hay un cambio de estructura lógica de datos.

- Extracción de características: La operación de extracción determina las propiedades de los grupos de pixeles, identificándolos por sus atributos como área, orientación, tonalidad de gris, etc. La extracción también mide las relaciones espaciales entre dos o más grupos de pixeles, determinando la cercanía o lejanía entre ellos.

- Comparación: La comparación determina la interpretación de los grupos de pixeles asociándolos con objetos ya conocidos.

2.3.2. Clasificación de las tareas de reconocimiento de imágenes

Según el tipo de aplicación, serán el tipo de imagen que será necesario adquirir (imágenes de rayos X, infrarrojo, etc.) y el análisis que se aplicará. La mayoría de las aplicaciones de la visión por computadora podemos clasificarlas por el tipo de tarea en inspección (medición, calibración, detección de fallas), verificación, reconocimiento, identificación y análisis de localización (posición, guía).

La **medición** o **calibración** es ajustar escalas, sensitividad, offset e intervalos dinámicos a normas o patrones de referencia estándar.

La **detección de fallas** es un análisis cualitativo que involucra la detección de defectos o artefactos no deseados, con forma desconocida en una posición desconocida. Por ejemplo, encontrar defectos en la pintura de un auto nuevo, o agujeros en hojas de papel.

La **verificación** es el chequeo cualitativo de que una operación de ensamble ha sido llevada a cabo correctamente. Por ejemplo, que no falte ninguna tecla en un teclado, o que no falten componentes en un circuito impreso.

El **reconocimiento** involucra la identificación de un objeto con base en descriptores asociados con el objeto. Por ejemplo, la clasificación de cítricos (limones, naranjas, mandarinas, etc.) por color y tamaño. Otro ejemplo de reconocimiento podría ser aplicado a células, por área y forma.

Identificación es el proceso de identificar un objeto por el uso de símbolos en el mismo. Por ejemplo, el código de barras, o códigos de perforaciones empleados para distinguir hule espuma de asientos automotrices.

El **análisis de localización** es la evaluación de la posición de un objeto. Por ejemplo, determinar la posición donde debe insertarse un circuito integrado ("chip"); otro ejemplo sería ver la localización automática del contorno de un cerebro en una imagen de tomografía, o la localización automática del contorno de la cabeza en una serie de imágenes de tomografía para su posterior reconstrucción tridimensional.

Guía significa proporcionar adaptablemente información posicional de retroalimentación para dirigir una actividad. El ejemplo típico es el uso de un sistema de visión para guiar un brazo robótico mientras solda o manipula partes. Otro ejemplo sería la navegación en vehículos autónomos

Existen muchas aplicaciones en donde la visión por computadora juega un papel muy importante, a continuación muestro algunas de las aplicaciones más recientes.

En [16] se presenta un sistema clasificador de objetos en sistemas residenciales monitoreados. Este sistema trabaja en tiempo real y puede distinguir entre humanos, mascotas y otros objetos. Un sistema de seguridad que pueda accionar una alarma cuando se presenta algún movimiento es relativamente fácil, lo difícil es un sistema que pueda filtrar las falsas alarmas y detectar cuando realmente se trata de una persona. El sistema funciona de la siguiente manera: se tiene una técnica que mantiene un historial de valores de intensidad de cada píxel en la imagen y usa un modelo estadístico no-paramétrico para estimar la probabilidad de observar valores de intensidad de un cierto píxel, dada una muestra de valores de intensidad para ese píxel. El modelo de fondo se adapta para cambiar rápidamente en la escena ya que siempre representa un modelo muy reciente de la escena. La imagen actual es comparada con la imagen de fondo y todos lo pixeles que no corresponden a la imagen de fondo son marcados como primer plano. Los pixeles de primer plano son agrupados en objetos y son rastreados usando un filtro. Ya que se obtienen las regiones resultantes de interés, son pasadas al modulo de clasificación donde se les extrae las características obteniendo gradientes. Las características extraídas son normalizadas para ser la entrada de la red neuronal con funciones radiales RBF (*radial based functions*). Me parece un sistema de visión muy interesante, el uso del gradiente solo captura la forma de la información, es insensible a la ropa o a cambios de iluminación. Este sistema tiene un 95% de eficacia.

Las redes neuronales las vamos a describir más detalladamente en el capítulo 3. Aquí únicamente constatamos que las redes neuronales se usan en una tarea real.

En [24] se han desarrollado dispositivos de reconocimiento ambiental que integran actuadores de fibra óptica para la observación estereoscópica y sensores de fibra óptica para observación microscópica. Este artículo expone un caso sobre un mecanismo de inspección visual para la microfábrica japonesa. Esta microfábrica consiste de una unidad de proceso, unidad de ensamble, unidad de transportación y una unidad de inspección.

En la unidad de inspección la observación estereoscópica es usada para monitorear el ensamble. La observación microscópica es usada para medir las micropartes procesadas, y un mecanismo de articulación es usado para asegurar las áreas de observación que son requeridas. Algo que hace más difícil la tarea es que los dispositivos de reconocimiento ambiental en la unidad de inspección deben de realizar observaciones en lugares con alta humedad, y en líquidos, especialmente en la unidad de proceso. Los dispositivos se aproximan a los objetos o hacen contacto con ellos mediante micro-servo actuadores con alta precisión.

En dicho trabajo se desarrolló una fibra de imagen-guía ultra delgada, microestereoscópica teniendo un mecanismo parallax con un actuador globo de fibra óptica, un microscopio con un sensor táctil de fibra óptica, y actuadores que pueden doblar la punta en cualquier dirección con una alta precisión. Finalmente se desarrollaron dos dispositivos de reconocimiento ambiental para la unidad de procesado con funciones de observación del campo en alejamiento y acercamiento, sensado táctil, y para la unidad de ensamblado se tienen funciones de observación estereoscópica y articulación de la punta.

En la unidad de procesado se requiere de un dispositivo de reconocimiento ambiental que haga las siguientes inspecciones: (1) medición del diámetro del punto de partida, (2) medición de la distancia entre el punto de partida y los ejes centrales del molde del microengrane, (3) observación de cada diente del molde. Es necesario para cada una de estas inspecciones obtener imágenes microscópicas de la fibra-guía de imagen.

En la unidad de ensamble el dispositivo de reconocimiento requiere de las siguientes inspecciones: (1) observación de la interfaz entre la microcaja de cambios y la tabla rotatoria cuando la micro caja de cambios es puesta en la tabla rotatoria por el microbrazo; (2) observación de la dirección de el microengranaje de la micro caja de cambios. Es necesario para las estructuras 3D obtener imágenes estereoscópicas de la fibra de imagen-guía. Irradiando el globo actuador de fibra óptica con luz diodo-laser (longitud de onda: 810 nm), el ángulo de convergencia varía de 0 a 5 grados. Observando las imágenes estereoscópicas a diferentes ángulos de convergencia, se encuentra que el binocular parallax se vuelve más ancho y la imagen estereoscópica es mejorada.

Por lo tanto en éste artículo se desarrollan dispositivos de reconocimiento ambiental que integran actuadores de fibra óptica para la observación estereoscópica y sensores de fibra óptica para observación microscópica. Este sistema parece ser muy eficiente y puede ser usado en una amplia gama de aplicaciones como en la industria o médicina.

Una posible aproximación es el uso de microrobots flexibles que pueden realizar varias manipulaciones con precisión en el posicionamiento en el rango de nanómetros (nm) ya sea bajo un microscopio o dentro de una cámara de aspiración de un microscopio de barrido de electrones (SEM *Scanning Electrón Microscope*) [30].

En [30] el sistema sensor basado en visión del FMMS (*Flexible Microrobot-based Microassembly Station*) consiste de varios módulos que se comunican por memoria compartida. Ellos comprenden métodos para calibración, reconocimiento de objetos, detección de la posición, trayectoria de un objeto, medición y estimación de profundidad.

La parte central del sistema de visión es una base de datos de objetos que contiene representaciones de las características de los objetos así como información actual sobre el estado presente. i.e. visibilidad y posición. Los mensajes para activar módulos de visión particulares son enviados al sistema de visión. Requerimientos como la locación de un objeto o imágenes procesadas son guardadas en memoria compartida y puede ser accesada ya sea por el control del sistema o por uno de los módulos de visión. Actualmente el reconocimiento de objetos en éste trabajo solo considera objetos de una y dos dimensiones identificando los objetos por su silueta. El algoritmo implementado reconoce objetos rígidos por la curvatura de su contorno. Basado en la generación de "*Curvature Zero Crossing points*"(CZC) introducido por Mokhtarian [31], un rápido y confiable reconocimiento y un método de detección de la posición han sido desarrollados. Primero, los contornos son extraídos y trazados como curvas continuas. Esta curvas son suavizadas y parametrizadas, su curvatura es calculada usando un filtro derivativo. Los segmentos CZC consistentes de dos puntos CZC adyacentes son generados y usados como características para representar los objetos. Comparando las características generadas con las de la base de

datos de objetos resultan un gran número de hipótesis. Una función de penalización evalúa cada hipótesis.

2.4. Reconocimiento de imágenes en una tarea de microensamble.

Para el ensamble de microdispositivos se propone usar la visión por computadora. Como tarea de prueba, está propuesta la tarea de colocar una flecha a un orificio. La flecha puede ser rígida o flexible [33, 34]. En nuestro trabajo usamos la flecha rígida. Proponemos reducir las tareas de tres dimensiones a tareas de dos dimensiones. Este método permite usar una cámara TV y cuatro fuentes de luz en lugar de dos cámaras TV para visión estereoscópica (Figura 10). Las sombras producidas con las fuentes de luz junto con la flecha y el orificio contienen toda la información de las imágenes 3D.

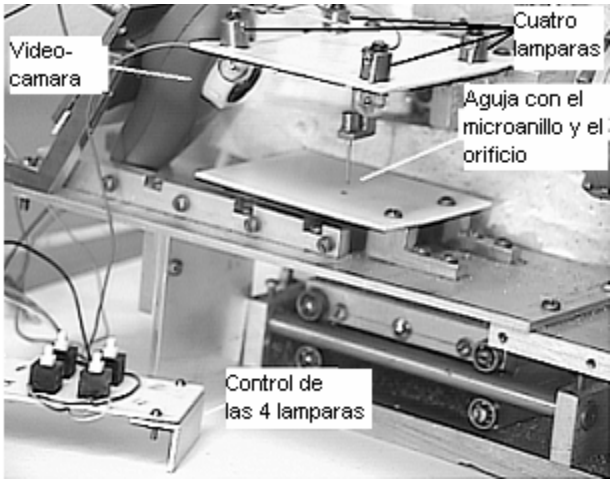


Figura 10. Prototipo del sistema de ensamble

Las imágenes que obtenemos después del tratamiento preliminar son parecidas a las imágenes de símbolos (números o letras), que nos permiten usar un clasificador neuronal elaborado para reconocimiento de cifras escritas.

Para explicar el método propuesto en el proceso de ensamble, vamos a usar la Figura 11. Se trata de colocar una flecha mecánica sobre un orificio. Con este objetivo se tiene que conocer el desplazamiento *(dx, dy, dz)* de la flecha, relativa al orificio. Para el propósito mencionado se suelen utilizar dos cámaras colocadas a diferentes ángulos para generar sistemas de visión estereo. Pero este sistema demanda 2 cámaras TV. Nuestro método trabaja con imágenes planas (2D) y demanda una cámara TV.

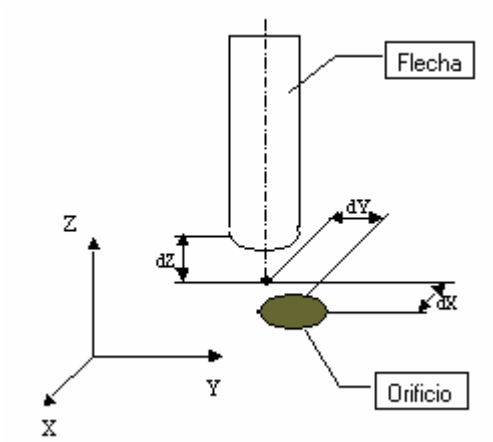


Figura 11. Localización de flecha y orificio

En la Figura 12 se muestra una imagen de la flecha con el orificio y cuatro sombras producidas por cuatro fuentes de luz.

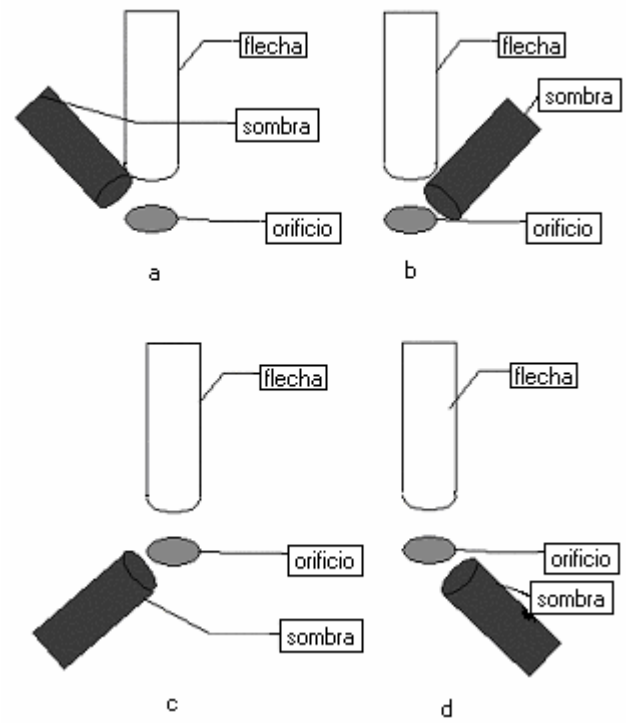


Figura 12. Sombras producidas por diferentes fuentes de luz

Con la localización mutua de las sombras y el orificio se tiene toda la información del desplazamiento de la flecha relativa con el orificio. Si se tiene el desplazamiento en el plano horizontal *(dx, dy)* es posible obtener directamente con el desplazamiento de las sombras el punto central que es conforme con el centro del orificio. El desplazamiento vertical de la flecha puede ser obtenida de la distancia entre las sombras. Para calcular el desplazamiento es necesario tener todas las sombras en una imagen. Si se encienden las

cuatro fuentes de luz simultáneamente para la extracción de contornos de las sombras, los resultados van a tener mala calidad. Por eso se capturan cuatro imágenes que corresponden a cada fuente de luz consecuente.

Después, se extraen los contornos de cada imagen y se juntan las cuatro imágenes con los contornos (Figura 12). La Figura 13 es muy parecida a los caracteres ópticos (letras y números). Para el reconocimiento de los números se ha desarrollado un clasificador neuronal. El sistema clasificador puede determinar la posición de la flecha sobre el orificio. Si la posición en que nos encontramos no es la deseada, será necesario determinar cual es la dirección o direcciones en las cuales debemos mover la flecha para que coincida con el orificio o cualquier otro punto planteado. Resulta obvio pensar que diferentes posiciones de la flecha proporcionan diferentes imágenes que deben ser procesadas y reconocidas usando algoritmos de reconocimiento de símbolos ópticos. Estos algoritmos producirán como salida un símbolo reconocido que puede ser relacionado con las coordenadas (*x, y*) de la posición de la flecha. Esto da la información sobre el movimiento necesario para corregir la posición actual y llegar a la posición deseada.

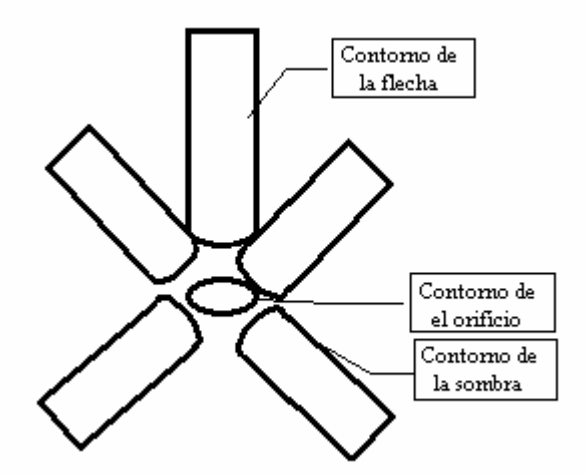


Figura 13. Cuatro sombras juntas

Como una primera aproximación, se trabajó con 23 imágenes, de las cuales 12 eran para entrenamiento y el resto para pruebas. Estas 23 imágenes corresponden a los desplazamientos de la flecha a lo largo de los ejes *X* y *Y* con un paso de 0.5 mm.

En la figura 14 se presenta la imagen de los contornos de cuatro fuentes de iluminación para una posición (imagen binaria). Estas imágenes codificadas sirven como entrada del clasificador neuronal.

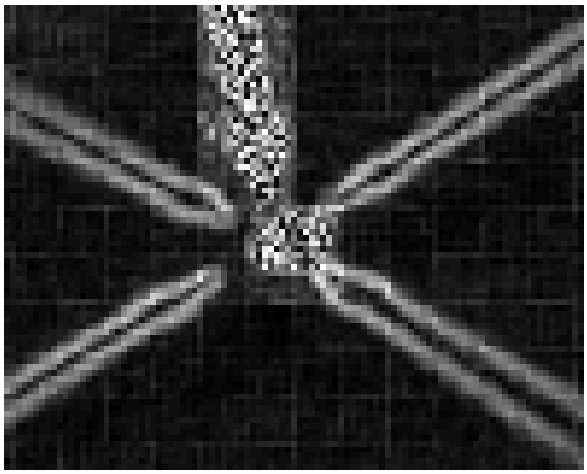


Figura 14. Imagen de contornos de cuatro fuentes de iluminación para una posición (imagen binaria)

Como resultado de este primer experimento, se observa la necesidad de incrementar la base de datos. La segunda base de datos fue hecha con 441 imágenes. También proponemos el uso de un sistema de desplazamientos de las imágenes iniciales, el cuál consiste en simular posibles imágenes resultantes de movimientos realizados. Ver Figura 15.

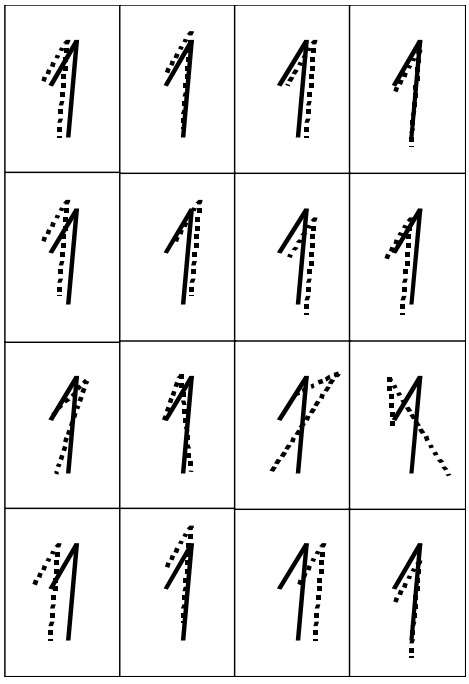


Figura 15. Ejemplos de desplazamientos de la imagen inicial ("distortions").

La segunda base de datos contiene 441 imágenes que corresponden a los desplazamientos de la flecha en *X*, *Y* con un paso de 0.1 mm. La altura de la flecha en ambos casos se mantiene constante.

La figura 16 presenta diferentes posiciones relativas entre la flecha y el orificio. En la figura 16a la flecha está relativamente elevada con respecto al orificio, la distancia entre las sombras es grande. En la figura 16*b* la flecha se suelta al orificio, uniéndose los contornos de la flecha con los del orificio. En la figura 16*c* la flecha se desplaza a la izquierda del orificio. En la figura 16*d* la flecha se centra y eleva ligeramente.

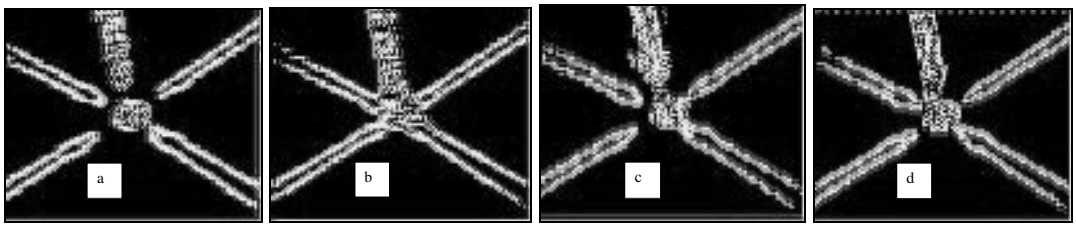


Figura 16. Ejemplos de la posición relativa de la flecha con el orificio junto con las 4 sombras.

Para la primera base de datos se crearon dos clasificadores de las posiciones de la flecha. El primero tiene tres reconocimientos de los desplazamientos en *X*: $X_{pin} < X_{hole}$; $X_{pin} = X_{hole}$; $X_{pin} > X_{hole.}$ Y el segundo tiene tres reconocimientos de los desplazamientos en *Y*: $Y_{pin} < Y_{hole}$; $Y_{pin} = Y_{hole}$; $Y_{pin} > Y_{hol}$. Se usaron 12 imágenes seleccionadas aleatoriamente de la primera base de datos para entrenar a los clasificadores y las 11 imágenes restantes para comprobarlas. Ambos clasificadores reconocen todas las imágenes correctamente *(Nerror = 0).*

Para el reconocimiento de estas imágenes se desarrolló un clasificador neuronal que demanda tiempo para entrenamiento pero trabaja en tiempo real en procesos de reconocimiento. Dicho clasificador se ha llamado: LIRA (*Limited Receptive Area Classifier*).

En el siguiente capítulo analizamos diferentes métodos y arquitecturas de redes neuronales que son posibles usar para resolver tareas de reconocimiento de imágenes. Escribimos más detalladamente sobre el clasificador LIRA que desarrollamos para resolver nuestra tarea.

III.- REDES NEURONALES

Las redes neuronales artificiales que están desarrolladas en las últimas décadas se basan en ideas que se toman de la neurofisiología. Por eso comenzamos con la descripción de una neurona biológica.

3.1. La neurona biológica

En el cerebro hay miles de millones de células llamadas neuronas las cuáles están compuestas de dendritas (entrada de la neurona), axón (salida de la neurona, al final del axón se encuentran los botones sinápticos los cuáles realizan la sinapsis con las dendritas de otra neurona) y el núcleo o soma. Estas neuronas se conectan con otras neuronas mediante un proceso llamado sinapsis en el cuál forman una "conexión" aparente, digo aparente porque nunca se unen, o sea no hay contacto físico entre neuronas (Figura 17). Una neurona consta de un cuerpo celular más o menos esférico, de 5 a 10 micras de diámetro, del que salen una rama principal, el axón y varias ramas más cortas, llamadas dendritas (Figura 17).

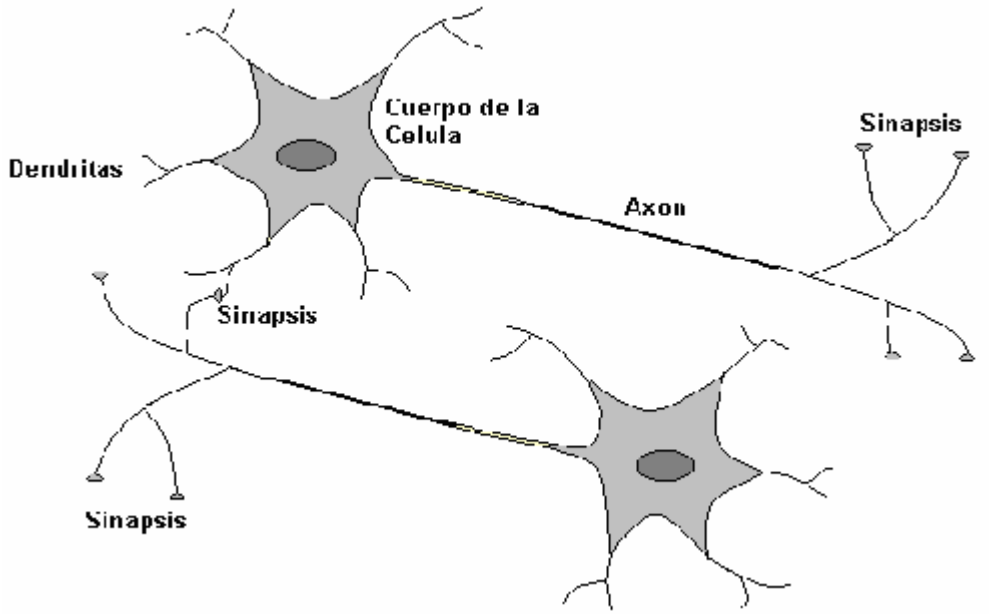


Figura 17. Composición de las neuronas

Una de las características que diferencian a las neuronas de las demás células vivas, es su capacidad de comunicarse. En términos generales, las dendritas y el cuerpo celular reciben señales de entrada; el cuerpo celular las combina e integra y emite señales de salida. El axón transporta esas señales a los terminales axónicos, que se encargan de distribuir información a un nuevo conjunto de neuronas.

Las señales que se utilizan, y a las que nos estamos refiriendo, son de dos tipos distintos de naturaleza: eléctrica y química. La señal generada por la neurona y transportada a lo largo del axón es un impulso eléctrico, mientras que la señal que se transmite entre los terminales axónicos de una neurona y las dendritas de la neuronas siguientes es de origen químico; concretamente, se realiza mediante moléculas de sustancias transmisoras

(neurotransmisores) que fluyen a través de unos contactos especiales, llamados sinapsis, que tienen la función de receptor y que están localizados entre los terminales axónicos y las dendritas de la neurona siguiente.

Para formalizar y simular el funcionamiento de la neurona biológica se fijan los siguientes aspectos: las señales que llegan a la sinapsis son las entradas a la neurona; éstas son ponderadas (atenuadas o amplificadas) a través de un parámetro denominado peso, asociado a la sinapsis correspondiente. Estas señales de entrada pueden excitar o inhibir a la neurona. El efecto es la suma de las entradas ponderadas. Mediante esta suma se activa o no la neurona, si la suma es mayor que el umbral de la neurona entonces se activa.

3.2. La neurona artificial

La neurona artificial pretende mimetizar las características más importantes de las neuronas biológicas

Generalmente, se pueden encontrar tres tipos de neuronas:

1) Aquellas que reciben estímulos externos, relacionadas con el aparato sensorial, que tomarán la información de entrada.

2) Dicha información se transmite a ciertos elementos internos que se ocupan de su procesado. Es en las sinapsis y las neuronas correspondientes a este segundo nivel donde se genera cualquier tipo de representación interna de la información.

3) Una vez que ha finalizado el periodo de procesado, la información llega a las neuronas de salida, cuya misión es dar la respuesta del sistema.

A continuación presento un esquema de neurona artificial (Figura 18). En él tenemos una neurona de interés *Yj*, *n* neuronas *Xi* que envían señales de entrada. Los pesos *Wji* representan los pesos sinápticos de las dendritas de *Yj*. La notación es la siguiente: el primer índice denota a la neurona hacia donde se dirige la información, el segundo índice denota la neurona de donde precede la información.

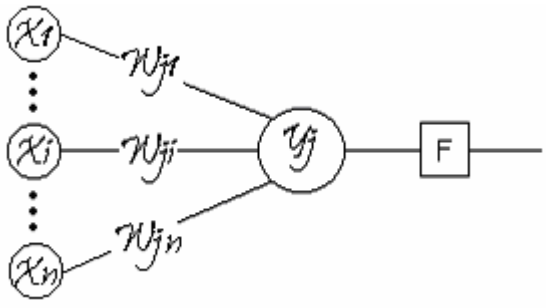


Figura 18. Modelo de una neurona artificial

Lo que hace cada peso sináptico es simplemente multiplicar a su entrada correspondiente y define la importancia relativa de cada entrada. Entonces tenemos que la entrada total de la neurona *Yj* es (Ecuación de la neurona):

$$Y_j = \sum_{i=1}^{n} w_{ji} x_i \qquad (1)$$

La neurona se activa si la entrada total supera un cierto umbral. Lo que se hace es aplicar una función de activación F (Figura 19) sobre *Yj*, que puede ser, por ejemplo, una función tipo escalón, una sigmoidal o una rampa [36].

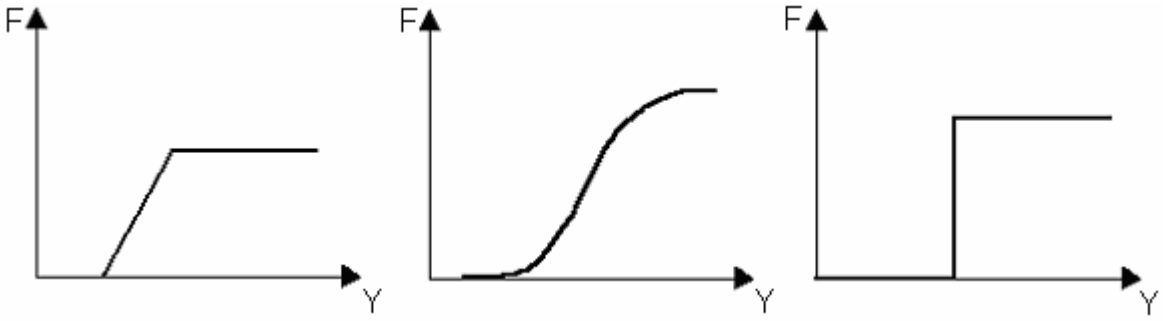


Figura 19. Funciones de activación: Lineal, Sigmoidal y Escalón

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

$$Y_j(t+1) = F(Y_j(t)) \qquad (2)$$

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

Si vamos a construir una red neuronal artificial, se pueden distinguir tres tipos de capas:

- De entrada: Es la capa que recibe directamente la información proveniente de las fuentes externas a la red

- Ocultas (intermedias): Son internas a la red y no tienen contacto directo con el entorno exterior. El número de niveles ocultos puede estar entre cero y un número elevado.

- De salida: Transfieren información de la red hacia el exterior.

La conectividad entre las neuronas de una red neuronal está relacionada con la forma en que las salidas de las neuronas están canalizadas para convertirse en entradas de otras neuronas. Cuando ninguna salida de las neuronas es entrada de neuronas del mismo nivel o de niveles precedentes, la red se describe como de propagación hacia delante; cuando las salidas pueden ser conectadas como entradas de neuronas de niveles previos o del mismo nivel, incluyéndose ellas mismas, la red es de propagación hacia atrás.

El entrenamiento es el proceso por el cuál una red neuronal modifica sus pesos en respuesta a una información de entrada. En los modelos de redes neuronales artificiales, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. Un aspecto importante es conocer cuáles son los criterios para cambiar el valor asignado a las conexiones cuando se pretende que la red entrene con nueva información. Se suelen considerar dos tipos de reglas: entrenamiento supervisado y entrenamiento no supervisado. La diferencia fundamental entre ambos tipos estriba en la existencia o no de un agente externo que controle el proceso de entrenamiento de la red.

3.3. Definición de red neuronal

A través de la historia muchos científicos he historiadores se han preguntado ¿cómo funciona el cerebro?, ¿cuál es la naturaleza de la inteligencia?, ¿como funciona nuestra percepción?, etc. Todo órgano del cuerpo humano y de muchos animales es importante pero desde mi punto de vista el cerebro es el más importante de todos, con el cerebro pensamos, hablamos, vemos, oímos, saboreamos, percibimos, sentimos, etc; todo está ligado con el cerebro.

El cerebro es tan complejo que todavía no se comprende del todo, todavía hay muchas cosas que investigar acerca de éste misterioso órgano, hacer una descripción detallada del cerebro y de sus partes no es tema de ésta tesis pero lo que si nos interesa saber, es lo que es una Red Neuronal.

Una neurona se comunica con otra mediante impulsos electroquímicos. En el núcleo o soma se suman las entradas de las dendritas, si ésta suma sobrepasa un umbral entonces se produce un impulso eléctrico que recorre el axón de la neurona produciendo que la neurona libere diminutos neurotransmisores los cuáles son recibidos por las neuronas con las que se está comunicando. En el momento en que aprendemos nuestro cerebro forma una red neuronal la cuál es débil al principio pero mientras más reforcemos este conocimiento, la red neuronal se consolida más y las conexiones entre sus neuronas se refuerzan.

La manera de crear inteligencia parecida al cerebro es construyendo sistemas exactos en suficiente detalle para expresar la esencia de cada computo que este siendo realizado y verificar su operación correcta contra medidas del sistema real. El modelo también debe de

operar con suficiente resolución para que sea comparable con el sistema real para así crear intuiciones reales sobre que información es representada en cada fase [16].

Una red neuronal es una nueva forma de computación, inspirada en modelos biológicos. Es un sistema de computación hecho por un gran numero de elementos simples, elementos de proceso muy interconectados, los cuáles procesan información por medio de su estado dinámico como respuesta a entradas externas [43].

Las redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptables) y con organización jerárquica, las cuáles intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico [44].

En las redes neuronales biológicas las interconexiones se realizan por medio de las ramas de salida (axones) que producen un número variable de conexiones (sinapsis) con otras neuronas (o quizá con otras partes, como músculos y glándulas). Las redes neuronales son sistemas de simples elementos de proceso muy interconectados.

Una peculiaridad de las redes neuronales biológicas es su tamaño: en todo el sistema nervioso central hay del orden de 10^{11} neuronas, pero el número de interconexiones es aún mayor, probablemente sobre las 10^{15}. No parece posible programar las funciones de dicho sistema teniendo en cuenta además que el tamaño y la estructura de la red están cambiando radicalmente durante y después de la niñez.

3.4. Historia de la computación neuronal

Desde hace aproximadamente 50 años se desarrolló un campo de las ciencias de la computación que integra los diferentes métodos de resolución de problemas que no pueden ser descritos fácilmente mediante un enfoque algorítmico tradicional. Estos métodos tienen su origen en la emulación del comportamiento de los sistemas biológicos y sus posibilidades de reconocimiento, la toma de decisiones, etc.

Conseguir diseñar y construir máquinas capaces de realizar procesos con cierta inteligencia ha sido uno de los principales objetivos y preocupaciones de los científicos a lo largo de la historia. En un principio los esfuerzos estuvieron dirigidos a la obtención de autómatas, en el sentido de máquinas que realizarán, alguna función típica de los seres humanos.

Se trata de una nueva forma de computación que es capaz de entrenarse para resolver problemas del mundo real. Para ello se dispone de un conjunto de metodologías, como son la lógica borrosa, las redes neuronales, los algoritmos genéticos, etc.

En éste capítulo nos vamos a enfocar en un análisis introductorio de las redes neuronales artificiales. Cuando se habla de computación neuronal, normalmente sólo se tiene en mente las funciones sensoriales y motoras, así como algún tipo de proceso interno llamado pensamiento, el objetivo es tratar de reproducir muchas de estas funciones artificialmente.

Alan Turing, en 1936, fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. El proponía un criterio (conocido como criterio de Turing) para definir si la computadora puede ser inteligente; si alguien pregunta sobre algunas cosas y obtiene respuestas con las cuales no puede decir quien generó esas respuestas (humano o computadora), entonces el sistema es inteligente. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiológico, Walter Pitts, un matemático, quienes en 1943 formalizaron y propusieron el modelo de las neuronas, ellos modelaron una red neuronal simple mediante circuitos eléctricos [35].

En 1957, Frank Rosenblatt comenzó el desarrollo del Perceptrón [36]. El Perceptrón es el modelo de red neuronal más antiguo, era capaz de generalizar; es decir, después de haber aprendido una serie de patrones era capaz de reconocer otros similares.

En 1959, Bernard Widrow y Marcial Hoff., de Stanford, desarrollaron el modelo ADALINE (*ADAptative LINear Elements*). Esta fue la primer red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) [37].

Uno de los mayores investigadores de las redes neuronales desde los años 60 hasta nuestros días es Stephen Grossberg, quién realizó en 1967 una red llamada "Avalancha", que consistía en elementos discretos con actividad que varía con el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades tales como reconocimiento continuo del habla y entrenamiento del movimiento de los brazos de un robot [38].

Desde 1969 hasta 1982, Marvin Minsky y Seymour Papert del MIT frenaron el desarrollo de las redes neuronales. Ellos publicaron un libro que además de contener un análisis matemático detallado del perceptron, consideraba la extensión a Perceptrones multicapas [39].

James Anderson desarrolló un modelo lineal llamado Asociador lineal [40]. En Japón Kunihiko Fukushima desarrolló el Neocognitron, un modelo de red neuronal para el reconocimiento de patrones visuales [41].

En 1982 John Hopfield presentó su trabajo sobre redes neuronales en la Academia Nacional de Ciencias, en el cual describe con claridad y rigor matemático una red a la que ha dado su nombre [42].

En 1985, el Instituto Americano de Física comenzó lo que ha sido la reunión anual Neural Networks for Computing. En 1987, el IEEE celebró la primera conferencia sobre redes neuronales. En el mismo año se formó la International Neural Network Society (INNS). En 1988 como resultado de la unión del IEEE y de la INNS surge la International Joint Conference on Neural Networks (IJCNN)

3.4.1. Tipos de redes neuronales más importantes.

A continuación presento de manera sintetizada (Tabla 1) las características de las redes neuronales más importantes [45].

Tabla 1. Características de los tipos de redes neuronales más importantes.

Nombre de la Red	Año	Aplicaciones más importantes	Comentarios	Limitaciones	Inventada/ Desarrollada Por
Perceptrón	1957	Reconocimiento de caracteres impresos	La red más antigua. Construida en HW	No puede reconocer caracteres complejos	Frank Rosenblat
Avalancha	1967	Reconocimiento de habla continua Control brazos robot	Ninguna red sencilla puede hacer todo esto.	No es fácil alterar la velocidad o interpolar el movimiento	Stephen Grossberg
Cerebellatron	1969	Control de movimiento de los brazos de un robot.	Semejante a Avalancha	Requiere complicadas entradas de control	David Marr, James Albus Andres Pellionez
Back Propagation	1974 -85	Síntesis de voz desde texto. Control de robots. Predicción. Reconocimiento de patrones	Red más popular Numerosas aplicaciones con éxito. Facilidad de entrenamiento	Necesita mucho tiempo para el entrenamiento y Muchos ejemplos	Paul Werbos, David Parker, David Rumelhart
Brain-Estate-In-a-Box	1977	Extracción de conocimiento de bases de datos	Posiblemente mejor realización que las redes de Hopfield	Realización y potenciales aplicaciones no estudiadas totalmente	James Anderson
Neocognitron	1978 -84	Reconocimiento de caracteres manuscritos	Insensible a la translación, rotación y escala	Requiere muchos elementos de proceso, niveles y conexiones	K. Fukushima
Self-Organizing-Map (SOM).	1980 -84	Reconocimiento de patrones, codificación de	Realiza mapas de características	Requiere mucho entrenamiento	Teuvo Kohonen

Topology-Preserving-Map(TPM)		datos, optimización	comunes de los datos aprendidos		
Hopfield	1982	Reconstrucción de patrones y optimización	Puede implementarse en VLSI. Fácil de conceptuar	Capacidad y estabilidad	John Hopfield
Memoria Asociativa Bidireccional	1985	Memoria heteroasociativa de acceso por contenido	Entrenamiento y arquitectura simple	Baja capacidad de almacenamiento. Los datos deben ser clasificados.	Bart Kosko
Maquinas de Boltzmann y Cauchy	1985-86	Reconocimiento de patrones (imágenes, sonar, y radar). Optimización.	Redes simples. capacidad de representación óptima de patrones	La máquina Boltsmann, necesita un tiempo muy largo de entrenamiento	Jeffrey Hiton, Terry Sejnowsky, Harold Szu
Counter-Propagation	1986	Compresión de imágenes	Combinación de Perceptron y TPM	Numerosas neuronas y conexiones	Robert Hecht-Nielsen
Teoría de Resonancia Adaptable (ART)	1986	Reconocimiento de patrones (radar, sonar, etc).	Sofisticada. Poco utilizada	Sensible a la translación, distorsión y escala.	Gail Carpenter, Stephen Grosberg
ADALINE/ MADALINE	1986	Filtrado de señales. Ecualizador adaptable, modems.	Rápida, fácil de implementar con circuitos VLSI.	Solo es posible clasificar espacios linealmente separados.	Bernand Wildrow

Más adelante analizamos estas redes más detalladamente.

3.4.2. Ventajas de las redes neuronales artificiales

Una red neuronal artificial (RNA) es la implementación de un algoritmo inspirado en la manera en que trabajan las redes neuronales biológicas del cerebro. Son varias las ventajas de las redes neuronales artificiales sobre otros métodos de reconocimiento de patrones por ejemplo: son adaptables, esto quiere decir que pueden aprender de ellas mismas; otra

ventaja es que pueden inferir a partir de datos aprendidos previamente. Tienen varios beneficios uno de ellos es que pueden generalizar, pueden corregir un proceso de datos que se parezcan ampliamente a los datos con los cuales fue entrenada originalmente; similarmente pueden manejar datos imperfectos o incompletos, dado una medida de tolerancia de error.

Las ventajas de los modelos de redes neuronales es que: (1) pueden ser implementados eléctricamente, ópticamente, o electro-ópticamente, o pueden ser modelados en una computadora de propósito general; (2) son robustos y tolerantes a fallas; (3) trabajan en paralelo; y (4) muchos de los paradigmas de entrenamiento o algoritmos están disponibles en la práctica. Entonces, algunas de las ventajas de las RNA son las siguientes:

- Entrenamiento adaptable.- Capacidad de aprender a realizar nuevas tareas basada en un entrenamiento o una experiencia inicial. Las redes neuronales son sistemas dinámicos adaptables. Son adaptables debido a la capacidad de autoajustarse de los elementos procesales (neuronas) que componen el sistema. Son dinámicos, pues son capaces de estar constantemente cambiando para adaptarse a las nuevas condiciones.

- Autoorganización.- Una red neuronal puede crear su propia autoorganización o representación de la información que recibe mediante una etapa de entrenamiento.
  La autoorganización consiste en la modificación de la red neuronal completa para llevar a cabo un objetivo específico.

- Tolerancia a fallos.- La destrucción parcial de una red conduce a una degradación de su estructura; si embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño. Hay dos aspectos respecto a la tolerancia a fallos: primero, las redes pueden aprender a reconocer patrones con ruido, distorsionados o incompletos, esta es una tolerancia a fallos respecto a los datos. Segundo, pueden seguir realizando su función (con cierta degradación) aunque se destruya parte de la red.

- Operación en tiempo real.- Los computadores normales son secuenciales. Algunas computadoras para la simulación de redes neuronales pueden ser realizadas para trabajar en paralelo, y se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.

- Fácil inserción dentro de la tecnología existente.- Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas.

3.4.3. Aplicaciones de las redes neuronales artificiales

Las redes neuronales se relacionan con las siguientes áreas de aplicación:

- Biología
- Empresa
- Medio ambiente
- Finanzas
- Manufacturación
- Medicina
- Militares

A continuación se describen con más detalle aquellas áreas de aplicación más importantes para los cuáles las redes neuronales, en general, se están utilizando o pueden ser utilizados:

- Reconocimiento de patrones

El término reconocimiento de patrones originalmente se refería a la detección de formas simples, tales como figuras geométricas: triángulos, cuadrados, etc. Sin embargo, en un caso más desarrollado el reconocimiento de patrones se refiere a las escenas, cifras escritas, espectros de sonido, etc.

- Control de robots

Hay dos categorías importantes de robots: los de trayectoria programada y los denominados robots inteligentes. Los robots inteligentes se supone que pueden planear sus acciones y tomar decisiones en diferentes situaciones. Las redes neuronales pueden usarse en el sistema de control del robot para interpretar la información obtenida de los sensores del robot.

- Filtrado de señales

Las redes neuronales se usan como filtros para la eliminación de ruidos y desórdenes en señales, o para la reconstrucción de patrones a partir de datos parciales.

- Segmentación, compresión y fusión de datos

La mayoría de los algoritmos de segmentación no proporcionan completamente los resultados deseables. Actualmente existen numerosas aproximaciones de redes neuronales habilitadas para segmentación de imágenes, la mayoría de los cuáles presentan una capacidad superior en comparación con alguno de los más complejos algoritmos de segmentación. Se necesita dar prioridad al desarrollo de métodos de compresión de datos. Para resolver este problema ya se han aplicado diferentes tipos de redes neuronales con prometedores resultados [45].

3.5. Análisis de diferentes tipos de redes neuronales

3.5.1. El Perceptrón

Este fue el primer modelo de red neuronal artificial desarrollado por Rosenblatt en 1958. Despertó un enorme interés en los años 60, debido a su capacidad para entrenarse y reconocer patrones sencillos. Formado por varias neuronas lineales para recibir las entradas a la red y una neurona de salida [43, 45].

La única neurona de salida del Perceptron realiza la suma ponderada de las entradas, resta el umbral y pasa el resultado a una función de transferencia de tipo escalón. Sin embargo, al constar sólo de una capa de entrada y otra de salida con una única neurona, tiene una capacidad de representación bastante limitada. El caso más conocido es la imposibilidad del Perceptron de representar la función XOR (función en álgebra booleana).

El algoritmo de entrenamiento del Perceptron es de tipo supervisado. Un perceptron multicapa es una red de tipo feedforward compuesta de varias capas de neuronas entre la entrada y la salida. Esta red permite establecer regiones de decisión mucho más complejas que las del perceptron original.

3.5.2. Las redes "Adaline" y "Madaline".

Las redes "Adaline" (*Adaptive Linear Element*) y "Madaline" (*Multiple Adaline*) fueron desarrolladas por Bernie Widrow y sus alumnos. Las arquitecturas de "Adaline" y "Madaline" son parecidas a las del perceptron. La red "Adaline" está limitada a una única neurona de salida, mientras que "Madaline" puede tener varias. La diferencia fundamental respecto al perceptron se refiere al mecanismo de entrenamiento. "Adaline" y "Madaline" utilizan la denominada regla Delta de Widrow-Hoff, o regla del mínimo error cuadrado medio. Esta regla de entrenamiento es un método para hallar el vector de pesos *W* deseado, el cuál deberá ser único y asociar con éxito cada vector del conjunto de vectores o patrones de entrada con su correspondiente valor de salida correcto. El entrenamiento de la red consiste en adaptar los pesos a medida que se vayan presentando los patrones de entrenamiento y salidas deseadas para cada uno de ellos. Para cada combinación entrada-salida se realizara un proceso automático de pequeños ajustes en los valores de los pesos hasta que se obtienen las salidas correctas. Se trata de eliminar, o por lo menos minimizar la diferencia entre la salida deseada y la salida real para todos los vectores de entrada.

La principal aplicación de las redes "Adaline" está en el campo del procesamiento de señales, en concreto, para la realización de filtros que eliminen el ruido en señales portadores de información. Como filtros adaptativos, se han utilizado redes tipo "Adaline" en numerosas aplicaciones [46]. Destaca su uso como filtros de ecualización adaptativos en modems de alta velocidad y canceladores adaptativos del eco para el filtrado de señales en comunicaciones telefónicas de larga distancia y comunicaciones vía satélite [47].

3.5.3. La red de retropropagación (*backpropagation)*

El algoritmo de propagación hacia atrás, o retropropagación, es una regla de entrenamiento que se puede aplicar en modelos de redes con más de dos capas de células. En forma simplificada, el funcionamiento de una red retropropagación consiste en un entrenamiento de un conjunto predefinido de pares de entradas-salidas dados como ejemplo, empleando un ciclo de dos fases: primero se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida, se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida. En la segunda fase, estos errores se transmiten hacia atrás, partiendo de la capa salida, hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la participación de la neurona intermedia en la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que reciba su aportación relativa al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida esté más cercana a la deseada; es decir, el error disminuya.

La importancia de la red retropropagación consiste en su capacidad de autoadaptar los pesos de las neuronas de las capas intermedias para aprender la relación que existe entre un conjunto de patrones dados como ejemplo y sus salidas correspondientes. Este método está basado en la generalización de la regla delta. La regla propuesta por Widrow en 1960 (regla delta) [37] ha sido extendida a redes con capas intermedias con conexiones hacia delante (feedforward) y cuyas células tienen funciones de activación continuas (lineales o sigmoidales), dando lugar al algoritmo de retropropagación.

En una red retropropagación existe una capa de entrada con *n* neuronas y una capa de salida con *m* neuronas y al menos una capa oculta de neuronas internas. Cada neurona de una capa (excepto las de entrada) recibe entradas de todas las neuronas de la capa anterior y envía su salida a todas las neuronas de la capa posterior (excepto las de salida).

Actualmente, este tipo de redes se están aplicando sistemáticamente a distintas clases de problemas, por ejemplo: Codificación de información, traducción de texto en lenguaje hablado, reconocimiento de lenguaje hablado, reconocimiento de caracteres ópticos (OCR, *Optical Caracter Recognition*); principalmente se han hecho muchas aplicaciones en cardiología, así como en compresión y descompresión de datos [48].

3.5.4. El modelo de Hopfield

Uno de los principales responsables del desarrollo que ha experimentado el campo de la computación neuronal ha sido J. Hopfield. El modelo de Hopfield es una red monocapa con *N* neuronas cuyos valores de salida son binarios: 0/1 ó –1/+1. En la versión original del modelo (DH: *Discrete Hopfield*) las funciones de activación de las neuronas eran del tipo escalón. Se trataba, por tanto, de una red discreta, con entradas y salidas binarias; sin embargo, posteriormente Hopfield desarrolló una nueva versión continua con entradas y

salidas analógicas, utilizando neuronas con funciones de activación tipo sigmoidal (*CH: Continuos Hopfield*). Ver figura 19.

Cada neurona de la red se encuentra conectada a todas las demás (conexiones laterales), pero no consigo misma. Además, los pesos asociados a las conexiones entre pares de neuronas son simétricos.

Una de las características del modelo de Hopfield, es que se trata de una red autoasociativa. Así, varias informaciones diferentes pueden ser almacenadas en la red durante la etapa de entrenamiento. Posteriormente, si se presenta a la entrada alguna de las informaciones almacenadas, la red evoluciona hasta estabilizarse, ofreciendo entonces en la salida la información almacenada, que es asociada con la presentada en la entrada. Si, por el contrario, la información de entrada no coincide con ninguna de las almacenadas, por estar distorsionada o incompleta, la red evoluciona generando como salida la más parecida.

En la etapa de entrenamiento se fijan los valores de los pesos en función de las informaciones que se pretende memorice o almacene en la red. Esta red utiliza un entrenamiento no supervisado de tipo Hebbiano.

Existen varios problemas asociados a la red Hopfield, pero principalmente se refieren a la cantidad limitada de datos que se pueden almacenar y la necesidad de que estos datos sean ortogonales entre sí. Si se almacenan demasiadas informaciones, durante su funcionamiento la red puede converger a valores de salida diferentes de los aprendidos.

Una segunda limitación del modelo es que no siempre se puede garantizar que la red realice una asociación correcta entre una entrada y una de las informaciones almacenadas. Si estas últimas no son suficientemente diferentes entre sí, es decir, si no son ortogonales, puede ocurrir que cada una de ellas no represente un mínimo de la función de energía, con la probabilidad de que se generen salidas diferentes a todas ellas. También puede ocurrir que ante una entrada que coincida con una de las informaciones aprendidas, la red converja hacia una salida correspondiente a otra de las informaciones almacenadas que fuese muy parecida.

En cuanto a las aplicaciones más conocidas del modelo destacan las relacionadas con el reconocimiento de imágenes y de voz, el control de motores y, sobre todo, la resolución de problemas de optimización [49].

3.5.5. Modelo de Resonancia Adaptable (ART)

Una de las características de la memoria humana consiste en su habilidad para prender nuevos conceptos sin necesidad para ello olvidar los aprendidos en el pasado.

Lo que se ha intentado mostrar en esta descripción es lo que S. Grossberg denomina el dilema de la estabilidad y plasticidad del entrenamiento [50]. Este dilema plantea los siguientes interrogantes:

- ¿Como una red podría aprender nuevos patrones (plasticidad del entrenamiento)?
- ¿Como una red podría retener los patrones con los cuales previamente fue entrenado (estabilidad del entrenamiento)?

En respuesta a este dilema Grossberg, Carpenter y otros colaboradores desarrollaron la denominada teoría de la resonancia adaptativa (*Adaptive Resonance Theory*). Esta teoría se aplica a sistemas competitivos en los cuales cuando se presenta cierta información de entrada solo una de las neuronas de salida de la red (o una por cierto grupo de neuronas) se activa alcanzando su valor de respuesta máximo después de competir con las otras.

Para solucionar el dilema de la plasticidad y estabilidad, el modelo ART propone añadir a las redes un mecanismo de realimentación entre las neuronas competitivas de la capa de salida de la red y la capa de entrada. Este mecanismo facilita el entrenamiento de nueva información sin destruir la ya almacenada.

La teoría de la resonancia adaptativa se basa en la idea de hacer resonar la información de entrada con los representantes o prototipos de las categorías que reconoce la red. Si entra en resonancia con alguno, que es suficientemente similar, la red considera que pertenece a dicha categoría y únicamente realiza la pequeña adaptación del prototipo almacenado representante de la categoría para que incorpore algunas características del dato presentado. Cuando no resuena con ninguno, recordados por la red hasta este momento, la red se encarga de crear una nueva categoría con el dato de entrada como prototipo de la misma.

Los autores mencionados presentaron dos redes neuronales. Estas redes suelen denominarse ART1 (o ART) y ART2. ART1 trabaja con vectores de entrada binarios, mientras que ART2 es capaz de procesar informaciones continuas o analógicas.

La utilización de la red ART suele estar relacionada con tareas de reconocimiento de patrones. También se ha utilizado para el diseño de sistemas de control y diagnostico adaptativo. Esta res ha sido también utilizada para modelas procesos biológicos y así poder estudiar y predecir su comportamiento. [45]

3.5.6. El modelo de Kohonen

Existen evidencias que demuestran que en el cerebro las neuronas se organizan en muchas zonas, de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en formas de mapas de zonas y capas de neuronas. Esto sugiere, por lo tanto, que el cerebro podría poseer la capacidad inherente de formar mapas topológicos de las informaciones recibidas del exterior.

A partir de estas ideas, T. Kohonen presentó en 1982 un sistema neuronal con un comportamiento semejante [51]. Se trataba de un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro. El objetivo de Kohonen era demostrar que un estímulo externo (información de entrada) por sí

solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas.

Este modelo tiene dos variantes, denominadas LVQ (*Learning Vector Quantization*) y TPM (*Topology Preserving Map*) o SOM (*Self-Organizing Map*). Difieren en las dimensiones de éstos, siendo de una sola dimensión en el caso de LVQ, y bidimensional, e incluso tridimensional, en la red TPM.

La arquitectura de la versión original (LVQ) del modelo de Kohonen es parecida a la de la red ART, aunque en este caso no existen conexiones feedback. Se trata de una red de dos capas con *N* neuronas de entrada y *M* de salida. Cada una de las *N* neuronas de entrada se conecta a las *M* de salida a través de conexiones hacia delante (*feedforward*).

Entre las neuronas de la capa de salida, existen conexiones laterales de inhibición (peso negativo). La influencia que una neurona ejerce sobre las demás es función de la distancia entre ellas.

Por otra parte, la versión del modelo denominada TPM trata de establecer una correspondencia entre los datos de entrada y un espacio bidimensional de salida, creando mapas topológicos de dos dimensiones.

Lo que hace la red de Kohonen, en definitiva, es realizar una tarea de clasificación, ya que la neurona de salida activada ante una entrada representa la clase a la que pertenece dicha información de entrada. Además, como ante otra entrada parecida se activa la misma neurona de salida, u otra cercana a la anterior, debido a la semejanza entre las clases, se garantiza que las neuronas topológicamente próximas sean sensibles a entradas físicamente similares. Por esta causa, la red es especialmente útil para establecer relaciones, desconocidas previamente, entre conjuntos de datos.

El entrenamiento en el modelo de Kohonen es de tipo OFF LINE, por lo que se distingue una etapa de entrenamiento y otra de funcionamiento. Esta red utiliza un entrenamiento no supervisado de tipo competitivo. En este modelo, el entrenamiento no concluye después de presentarle una vez todos los patrones de entrada, sino que habrá que repetir el proceso varias veces para refinar el mapa topológico de salida, de tal forma que cuantas más veces se presenten los datos, tanto más se reducirán las zonas de neuronas que se deben activar ante entradas parecidas, consiguiendo que la red pueda realizar una clasificación más selectiva.

Como aplicaciones, destacan las relacionadas con el reconocimiento de patrones (voz, texto, imágenes, señales, etc), codificación de datos, compresión de imágenes y resolución de problemas de optimización. También se ha utilizado este modelo en robótica, comprobándose su utilidad en el diseño de sistemas para controlar el movimiento de un brazo mecánico en un espacio tridimensional. En dichos sistemas, se utiliza una red de Kohonen para aprender las magnitudes tensoriales necesarias para moverse en un entorno real, considerando los efectos del desgaste que pueden alterar la dinámica del brazo con el transcurso del tiempo [52].

3.6. Descripción de la red neuronal "LIRA"

Este clasificador neuronal está basado en el modelo perceptron de Rosenblatt, se hicieron algunas modificaciones al algoritmo de entrenamiento, al procedimiento aleatorio de conexiones entre la capa de entrada y la capa intermedia y a la regla de selección del ganador en la capa de salida. A continuación hago una breve descripción del perceptrón de Rosenblatt, para poder explicar la red neuronal LIRA [7, 13].

El perceptrón de Rosenblatt contiene tres capas de neuronas. La primera capa *S* corresponde a la entrada. La segunda capa *A* llamada capa asociativa corresponde al subsistema de extracción de características. La tercera capa *R* corresponde a la salida de todo el sistema. Las conexiones entre la capa *S* y *A* son establecidas usando un procedimiento aleatorio y no puede ser cambiado por el entrenamiento del perceptron. Las conexiones entre las capas *A* y *R* son establecidas por el principio de que cada neurona de la capa *A* es conectada con todas las neuronas de la capa *R*. Inicialmente los pesos son puestos a cero y son cambiados durante el entrenamiento.

Muchas investigaciones fueron dedicadas a perceptrones con solo una neurona en la capa de salida *R*. Por lo tanto este perceptrón puede reconocer solo 2 clases, si la salida de la neurona *R* es más alta que un cierto umbral entonces pertenece a la clase 1, y si es más baja entonces pertenece a la clase 2. La capa *S* contiene elementos de dos estados {1,-1}. El elemento es puesto a 1 si pertenece al objeto imagen, y es puesto a –1 si pertenece al fondo.

La capa asociativa *A* contiene neuronas con 2 estados de salida {1,0}. Las entradas de estas neuronas son conectadas con las salidas de las neuronas de la capa *S* con conexiones no modificables. Cada conexión debe tener peso 1(conexión positiva) o –1(conexión negativa). Dejemos que el umbral de esta neurona sea igualado al número de las conexiones de su entrada, esta neurona es activada solo en caso de que todas las conexiones positivas correspondan al objeto y las conexiones negativas correspondan al fondo.

La neurona *R* es conectada con todas las neuronas de la capa *A*. Los pesos de estas conexiones son cambiados durante el proceso de entrenamiento. La regla más popular de entrenamiento es incrementar los pesos entre las neuronas de la capa *A* y la neurona de la capa *R* si el objeto pertenece a la clase 1. Si el objeto pertenece a la clase 2, los pesos se decrementan. Es conocido que éste perceptron tiene rápida convergencia y puede formar superficies discriminantes no lineales. La complejidad de la superficie discriminante depende del número de neuronas de la capa *A*.

3.6.1. Modificaciones al Perceptron de Rosenblatt

La primera modificación es que se incluyeron *N* neuronas a la capa *R*. En este caso es necesario introducir la regla de la selección del ganador. En la primera serie de experimentos se uso la regla simple para la selección del ganador. La neurona de la capa *R* con la más alta excitación determina la clase bajo reconocimiento.

La segunda modificación fue hecha en el proceso de entrenamiento. Dejemos que la neurona ganadora tenga excitación E_w, y su competidor más cercano tiene excitación E_c. Si:

$$\frac{(E_w - E_c)}{E_w} < T_E \tag{3}$$

el competidor es considerado como ganador, donde T_E es la excitación superflua de la neurona ganadora.

La tercera modificación es concerniente a las conexiones. Las conexiones entre la capa *A* y la capa *R* del perceptron de Rosenblatt pueden ser negativas o positivas. Nosotros usamos solo conexiones positivas. En este caso el proceso de entrenamiento es el siguiente: durante el proceso de reconocimiento obtenemos las excitaciones de las neuronas de la capa *R*. La excitación de la neurona R_j (correspondiente a la clase correcta) es decrementada por el factor (1 - T_E). Después de esto la neurona que tenga la excitación máxima R_k es seleccionada como ganadora.

Si j=k, no hay nada que hacer; (4)

Si j!=k, $w_{ij}(t+1) = w_{ij}(t) + a_i$; (5)

donde $w_{ij}(t)$ es el peso de la conexión entre la neurona *i* de la capa A y la neurona *j* de la capa *R* antes del refuerzo, $w_{ij}(t+1)$ es el peso después del entrenamiento, a_i es la señal de salida (0 ó 1) de la neurona *i* de la capa *A*.

$$w_{ik}(t+1) = w_{ik}(t) - a_i, \qquad Si\,(w_{ik}(t) > 0) \tag{6}$$

$$w_{ik}(t+1) = 0, \qquad Si\,(w_{ik}(t) = 0) \tag{7}$$

donde $w_{ik}(t)$ es el peso de la conexión entre la neurona *i* de la capa *A* y la neurona *k* de la capa *R* antes del entrenamiento, $w_{ik}(t+1)$ es el peso después del entrenamiento.

Cada neurona de la capa *A* tiene conexiones aleatorias con la capa *S*. Para instalar estas conexiones es necesario enumerar todos los elementos de la capa *S*. Dejemos que el número de estos elementos sea igual a N_S . Para determinar la conexión de la neurona de la capa *A*, seleccionamos un número aleatorio uniformemente distribuido en el intervalo de [1, N_S]. Este número determina la neurona de la capa *S* que va a ser conectada con la neurona mencionada de la capa *A*. La misma regla es usada para determinar todas las conexiones entra las neuronas de la capa *A* y las neuronas de la capa *S*. Frank Rosenblatt propuso ésta regla. Nuestra experiencia indica que es posible mejorar el desempeño del perceptrón mediante la modificación de ésta regla.

La cuarta modificación es la siguiente. Conectamos la neurona de la capa *A* con las neuronas de la capa S las cuales no son seleccionadas aleatoriamente de toda la capa *S*, sino del rectángulo (*h***w*).

Las distancias *dx* y *dy* son números aleatoriamente seleccionados de los rangos: para *dx* de [0, $W_S - w$] y para *dy* de [0, $H_S - h$], donde H_S es la altura y W_S es el ancho de la capa *S*.

El perceptrón con estos cambios es llamado *Llimited Receptive Area Classifier* (LIRA) (Figura 20).

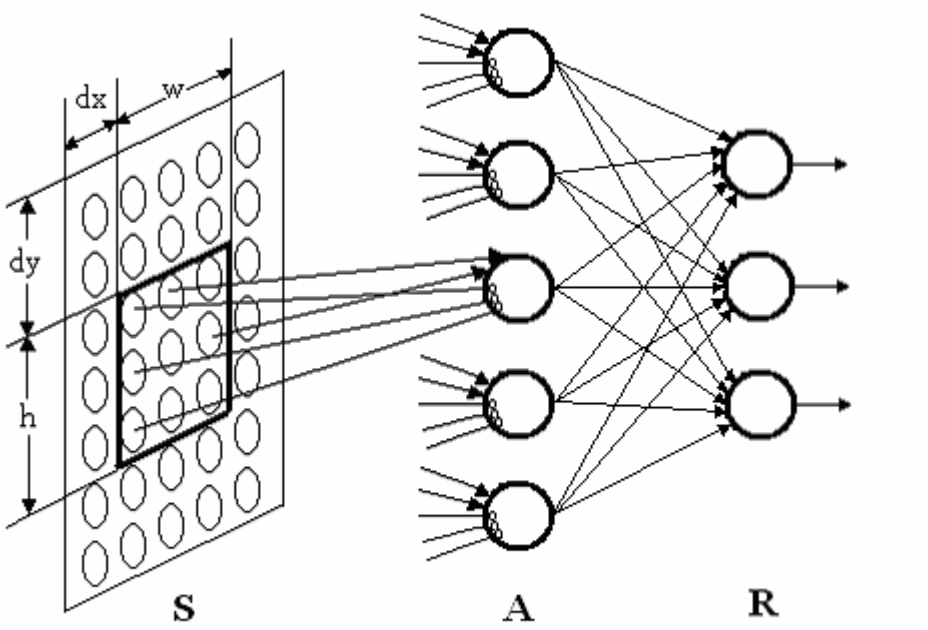


Figura 20. LIRA- clasificador neuronal (Limited Receptive Area Classifier)

3.6.2. Diseños de las mascaras.

La mascara de la neurona asociativa es una estructura de conexiones positivas y negativas de la neurona de la capa *A* con la capa *S*. El procedimiento de la selección aleatoria de las conexiones es usado para diseñar la máscara. Este procedimiento comienza escogiendo la esquina superior izquierda del rectángulo en donde están localizadas todas las conexiones positivas y negativas de la neurona asociativa. Se usan las siguientes fórmulas:

$$\begin{aligned} dx_i &= random_i(W_S - w), \\ dy_i &= random_i(H_S - h), \end{aligned} \tag{8}$$

donde:

i - posición de una neurona en la capa asociativa *A;*

$random_i(z)$ - número aleatorio el cuál está uniformemente distribuido en el intervalo de [0, *z*].

Después de esto cada posición de la conexión positiva y negativa dentro del rectángulo es definida por un par de números:

$$\begin{aligned} x_{ij} &= random_{ij}(w), \\ y_{ij} &= random_{ij}(h), \end{aligned} \tag{9}$$

donde:

j – número de la conexión de la neurona *i* con la capa *S*.

Las coordenadas absolutas de la conexión en la capa *S* son definidas por un par de números:

$$\begin{aligned} X_{ij} &= x_{ij} + dx_i, \\ Y_{ij} &= y_{ij} + dy_i. \end{aligned} \tag{10}$$

3.6.3. Codificación de la imagen.

Cualquier imagen de entrada define las actividades de las neuronas de la capa *S* en correspondencia uno-a-uno. El vector binario que corresponde a la actividad de las neuronas asociativas $A = a_1,...,a_n$, (donde *n* es el número de neuronas en la capa *A*). El procedimiento, el cuál transforma la imagen de entrada a un vector binario *A*, lo llamamos la "codificación de la imagen".

En nuestro sistema la neurona *i* de la capa *A* se activa solo si todas las conexiones positivas con la capa *S* corresponden al objeto, y todas las conexiones negativas corresponden al fondo de la imagen. En éste caso $a_i = 1$, en el caso opuesto $a_i = 0$. De la experiencia de trabajar con dichos sistemas, es conocido que el número de las neuronas activas *m* en la capa *A* debe ser mucho menor que todo el número de neuronas *n* de ésta capa. En nuestro trabajo usamos la expresión: $m = c\sqrt{n}$, donde *c* es una constante que pertenece al rango de 1 a 5. Esta relación corresponde a factores neurofisiológicos. El número de neuronas activas en la corteza cerebral es cientos de veces menor que el número total de neuronas.

Tomando en cuenta el número pequeño de neuronas activas es conveniente representar al vector binario *A* no explícitamente, sino como una lista de números de neuronas activas. Por ejemplo, si el vector *A* es:

$$A = 00010000100000010000,$$

la correspondiente lista de números de neuronas activas sería 4, 9 y 16. Esta lista es usada para salvar los códigos de la imagen en forma compacta y para el cálculo rápido de la actividad de las neuronas en la capa de salida. Entonces después de la ejecución del código, todas las imágenes tienen su correspondiente lista de números de neuronas activas.

3.6.4. Procedimiento de entrenamiento.

Antes del entrenamiento todos los pesos de las conexiones entre neuronas de la capa *A* y la capa *R* son puestos a cero.

1. El procedimiento de entrenamiento comienza con la presentación de la primera imagen a la red neuronal. La imagen es codificada y la excitación E_i de la neurona de la capa *R* es computada. E_i es definida como

$$E_i = \sum_{j=1}^{n} a_j * w_{ji}\,, \qquad (11)$$

donde:

E_i - excitación de la neurona *i* de la capa *R,*

a_j - excitación de la neurona *j* de la capa*,*

w_{ji} - peso de la conexión ente la neurona *j* de la capa *A* y la neurona *i* de la capa *R*.

2. Se requiere que el reconocimiento sea robusto. Después de el cálculo de las excitaciones de todas las neuronas de la capa *R*, se elige a la neurona de la capa *R* con la excitación máxima y su competidor más cercano. La excitación *E* de la neurona con la excitación máxima es recalculada acorde a la fórmula.

$$E_k^* = E_k * (1 - TDS)\,, \qquad (12)$$

donde: *TDS* - parámetro de seguridad para encontrar a la neurona con excitación máxima.

Después de esto, si la neurona-ganadora todavía tiene la excitación más grande que la competidora cercana decimos que es respuesta de la red. Si la excitación de la competidora es mayor, la elegimos como respuesta de la red.

3. Denotamos el numero de la neurona ganadora como i_w, y el numero de la neurona, el cual realmente corresponde a la imagen de entrada, como i_c. Si $i_w = i_c$ no hay nada que hacer. Si $i_w \neq i_c$:

$$\begin{aligned} &(\forall j)\left(w_{ji_c}(t+1) = w_{ji_c}(t) + a_j\right) \\ &(\forall j)\left(w_{ji_w}(t+1) = w_{ji_w}(t) - a_j\right) \\ &if\,(w_{ji_w}(t+1) < 0) \quad w_{ji_w}(t+1) = 0 \end{aligned} \tag{13}$$

donde $w_{ij}(t)$ es el peso de la conexión entre la neurona *j* de la capa *A* y la neurona *i* de la capa *R* antes del entrenamiento, $w_{ij}(t+1)$ es el peso después del entrenamiento.

El proceso de entrenamiento es llevado a cabo iterativamente. Después de la representación de todas las imágenes del subconjunto de entrenamiento, es calculado el número total de errores de entrenamiento. El proceso de entrenamiento se detiene cuando el número de ciclos es mayor que el valor antes preescrito. En experimentos previos este valor era de 10 ciclos, y en los finales de 30 ciclos.

Es obvio que en cualquier ciclo nuevo de entrenamiento el procedimiento de codificación de la imagen es repetido y da los mismos resultados que en ciclos previos. Por lo tanto en los experimentos finales mejoramos el proceso de codificación de imágenes solo una vez y grabamos en el disco duro las listas de los números de neuronas activas para cada imagen. Después, para todos los ciclos no usamos las imágenes sino las listas correspondientes de las neuronas activas.

Es conocido que el porcentaje de reconocimiento correcto puede incrementar esencialmente si durante el ciclo de entrenamiento se representan a las imágenes no solo en su estado inicial sino también con distorsiones [53]. En los experimentos finales usamos aparte de las imágenes iniciales, hasta 13 variantes de cada imagen con distorsiones (Figura 15, Tabla 2).

Los modelos de distorsión pueden ser usados para incrementar el tamaño efectivo de un conjunto de datos sin requerir colectar más datos.

Tabla 2. Distorsiones en las imágenes de entrada

X	0	-1	0	1	0	-1	-1	1	1	-2	0	2	0
Y	0	0	-1	0	1	-1	1	-1	1	0	-2	0	2

Los resultados de la investigación del clasificador neuronal LIRA las presentaremos en el Capítulo V, después de la descripción del software que desarrollamos para la simulación del sistema en el Capítulo IV.

IV.- DESARROLLO DEL SOFTWARE PARA RECONOCIMIENTO DE IMÁGENES EN BORLAND C++.

Como se mencionó anteriormente, nuestra tarea consiste en el reconocimiento de imágenes de la flecha y el orificio para obtener las coordenadas de la posición de la flecha. Esta información va a usarse para colocar la flecha al orificio en la tarea de microensamble (Figura 10). Proponemos reducir las tareas de tres dimensiones a tareas de dos dimensiones con uso de una cámara TV y cuatro fuentes de luz en lugar de dos cámaras TV para visión estereoscópica. Las sombras producidas con las fuentes de luz junto con la flecha y el orificio contienen toda la información de las imágenes 3D. Como ejemplo de las imágenes originales con cuatro fuentes de iluminación esta la Figura 8.

En la primera etapa de la investigación formamos una base de imágenes de la flecha con el orificio para tener la posibilidad de trabajar con imágenes fuera de línea ("*offline*"). La primera base de datos contenía solo (vea §2.4) 23 imágenes; con ésta base de datos desarrollamos el software para la simulación del clasificador neuronal LIRA y para reconocimiento de imágenes. Los resultados obtenidos mostraron buenas perspectivas para el uso del clasificador.

Entonces formamos la segunda base de imágenes. Esta base de datos tiene 441 imágenes que se refieren a posiciones en el plano *X-Y* alrededor del orificio. Las posiciones de estas imágenes fueron acomodadas como una matriz de 21 × 21, la distancia entre posiciones vecinas fue de 0.1 mm (Figura 21). Esta distancia corresponde a aproximadamente 1.8 pixeles en la coordenada *X* y 1 pixel en la coordenada *Y*. Cada coordenada *X-Y* dentro de esta matriz representa una clase.

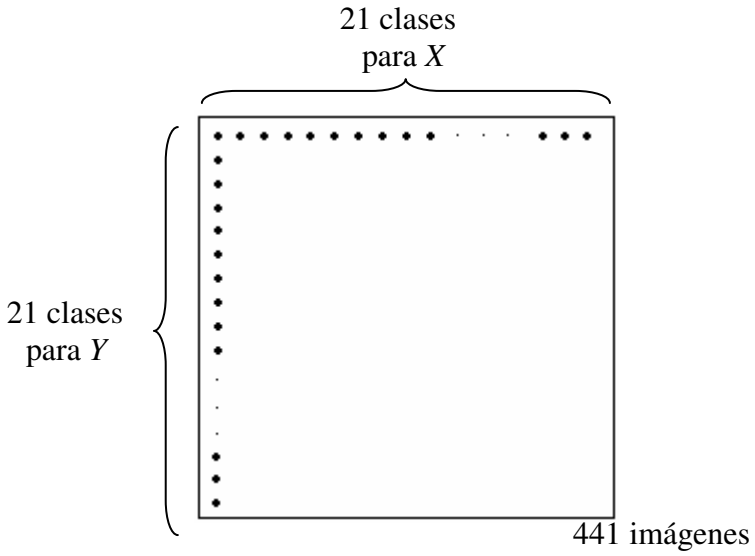


Figura 21. Esquema de la obtención de imágenes.

En la Figura 22 se puede observar el proceso que se sigue para el reconocimiento de las imágenes.

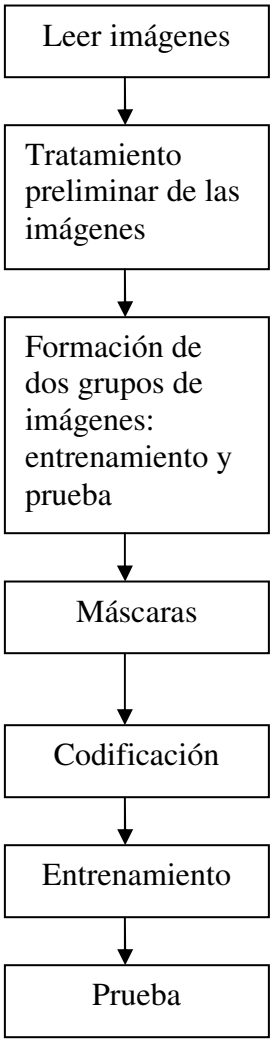


Figura 22 .Diagrama del proceso de reconocimiento de imágenes con redes neuronales

4.1. Estructura del programa, bloques y módulos.

Para el software que se desarrolló se uso el lenguaje de programación C++. Se utilizo Borland C++ Builder 6.0 debido a que el ambiente es muy cómodo para trabajar con el lenguaje C++, tiene muchas librerías útiles para manejar imágenes, se pueden hacer aplicaciones en tiempo real, etc. Además se maneja una librería llamada VCL (Visual Component Library) la cuál es la fuente de muchos componentes usados para crear aplicaciones. En la Figura 23 observamos una ventana llamada “Form 1”, la cuál es la interfaz de usuario de la aplicación, ésta forma esta compuesta de componentes visuales que hacen más amigable la interfaz. En la Figura 23 se pueden observar los módulos

desarrollados para hacer diferentes etapas de la investigación de procesos de reconocimiento de imágenes con el uso del clasificador neuronal LIRA y con los cuales puede interactuar el usuario. Arriba de esta forma existen los menús: "Train&TestSET", "Masks", "Coding", "Train" y "Test". Cada módulo cuenta con diferentes procedimientos o funciones, muchos de estos procedimientos son propios del lenguaje, otros son definidos por Borland y otros fueron creados para nuestra tarea.

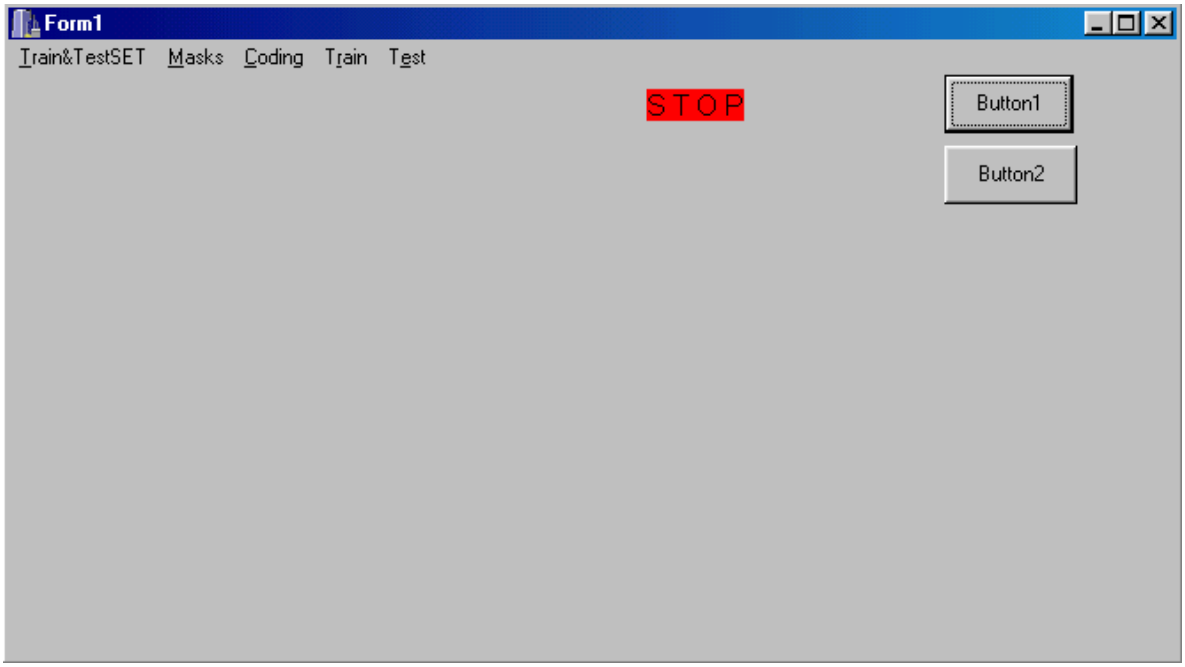


Figura 23.- Interfaz de usuario para la realización del reconocimiento de imágenes.

La estructura de nuestro programa, al igual que la mayoría, cuenta con variables globales las cuales son definidas al principio del programa, procedimientos o funciones donde también se definen variables locales, además de los componentes visuales.

A continuación se hará una descripción de cada módulo. El texto del programa es presentado en el Apéndice 1 al final de la tesis.

4.2. Funcionamiento de los módulos

4.2.1. Modulo "*Train&TestSET*"

Este modulo tiene la función de dividir las imágenes que son de prueba y las que son de entrenamiento. Esto lo hace de la siguiente manera: primero llama a la función "first()", ésta función se realiza una sola vez.

Probamos dos algoritmos para formar un conjunto de entrenamiento y otro de prueba. El primer algoritmo elige de las 441 imágenes, un número aleatorio de imágenes y las coloca

en el subconjunto de entrenamiento. Cuando tenemos 220 números diferentes de imágenes terminamos éste proceso. El número restante de imágenes sirven para probar el sistema.

El segundo algoritmo es el siguiente: la base fue considerada como un juego de ajedrez de 21×21, las celdas negras fueron consideradas imágenes para el entrenamiento y las celdas blancas como imágenes de prueba. (Figura 24)

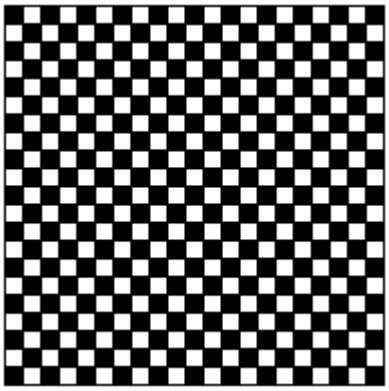


Figura 24.- Representación de la base de imágenes como cuadro de ajedrez.

Al dividir la base de esta manera obtenemos imágenes cercanas presentadas en dos juegos, esto ayuda a tener imágenes representativas de cada juego (prueba y entrenamiento). La distancia entre posiciones vecinas es de 0.1 mm. Esta distancia corresponde a aproximadamente 1.8 pixeles en la coordenada *X* y a 1 pixel en la coordenada *Y*.

Para realizar éstos métodos de preparación de los subconjuntos desarrollamos la función "first()".

Primero abre el archivo "LargeSet.dat". Este archivo contiene todas las 441 imágenes después del tratamiento premilitar. El tratamiento preliminar incluye la extracción de los contornos de la flecha, el orificio y la sombra de cada imagen inicial. Después juntamos o reunimos cuatro imágenes para cada posición de la flecha en una sola imagen y la escribimos al archivo "LargeSet.dat" junto con las coordenadas (*x, y*) de la posición de la flecha.

A continuación se describe el algoritmo de extracción de contornos.

El contorno es definido como un vector que está perpendicularmente al gradiente de brillo. Este gradiente lo calculamos por 4 pixeles vecinos. Este algoritmo es muy sencillo de realizar. En la figura 25 están presentados los 4 pixeles.

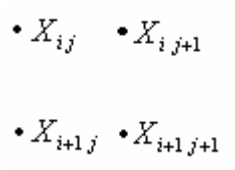


Figura 25. Cuatro pixeles

Calculamos los valores absolutos

$$Y_1 = \left| X_{i\,j} - X_{i+1\,j+1} \right|$$
$$Y_2 = \left| X_{i+1\,j} - X_{i\,j+1} \right| \qquad (14)$$

y elegimos

$$Z = \max(Y_1, Y_2) \qquad (15)$$

Si Z > C_1 (donde C_1 es una constante experimental) entonces el contorno existe, en otro caso no existe.

Este algoritmo es muy fácil de programar. En el módulo "*Train & TestSET*" ya trabajamos con las imágenes después del tratamiento preliminar, con imágenes presentadas como contornos de los objetos (Capítulo II, figura 14).

Carga cada imagen como el objeto gráfico "Bitmap4" y pueden presentarse en la pantalla. Ya que analiza cada imagen, se escribe la imagen como una matriz de [150×150] pixeles, y también sabe las coordenadas de cada imagen del archivo "LargeSet.dat". (Apéndice lineas: 93-147)

Ya que termino de ejecutarse la función "first()", se crea un arreglo de 441 elementos en donde se separan las imágenes que son de prueba y las que son de entrenamiento, en éste arreglo se alternan ceros y unos uniformemente. Los "0" representan las imágenes de entrenamiento y los "1" las imágenes de prueba. Después este arreglo se guarda en el archivo "KeySet.dat" (Apéndice líneas: 159-174). Aquí presentamos el arreglo del archivo "KeySet.dat" para el segundo algoritmo, donde dividimos en dos subconjuntos de imágenes como una tabla de ajedrez.

$$\underbrace{01010101010101...010101}_{441 \text{ elementos}}$$

En la pantalla se muestra una de las 441 imágenes (Figura 26) en el trabajo del módulo "*Train&TestSET*".

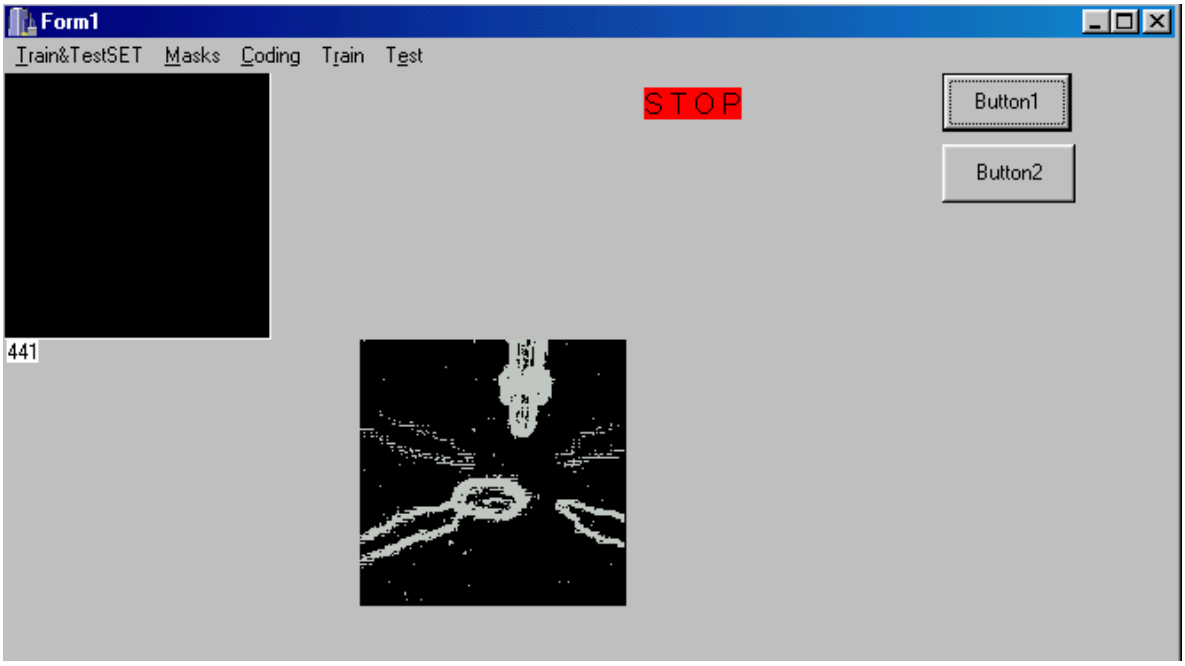


Figura 26. Ejecución del módulo "Train&TestSet"

4.2.2. Modulo "*Masks*"

Este modulo tiene por objetivo crear una estructura de conexiones entre la capa de entrada (*S*) y la capa intermedia (*A*) de nuestra red neuronal. Esta estructura de conexiones se llama máscara. La capa de entrada es del tamaño del número de pixeles que tiene la imagen, las imágenes que utilizamos tienen dimensiones de 150×150, por lo que la capa de entrada es de tamaño 22500.

Este modulo utiliza dos funciones principales que son "MaskCreation()" y "MaskTest()". La función MaskCreation() crea la mascara de la siguiente manera: primero se elige una coordenada aleatoria en la imagen y sus coordenadas, sobre esas coordenadas se asignan cinco puntos aleatoriamente sobre una ventana de 20×20 (Figura 27). Estos puntos elegidos corresponden a las neuronas de la capa *S*. Los puntos con signo "+" corresponden a las "On"-neuronas, y los puntos con signo "-" corresponden a las "Off"-neuronas. Esto es haciendo analogía con la neurofisiología. En la salida de las "On"-neuronas hay una señal cuando el punto está en la imagen. En la salida de las "Off"-neuronas hay una señal cuando el punto esta en el fondo de la imagen. Los puntos (en nuestro caso 5: 3 con signo "+" y 2 con signo "-") de la ventana de 20×20 están conectados con una neurona de la capa *A*, que contiene en total *N* neuronas (N=64000, 128000, 256000 y 512000). Cada ventana corresponde a una característica de la imágen. Ya que se tienen todos los puntos, se guardan en el archivo "maska.dat" (Apéndice líneas: 190-213).

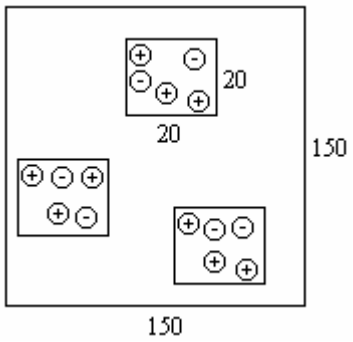


Figura 27. Imagen con las ventanas para extraer las características

Cuando se llama a la función "MaskTest()", se lee el archivo "maska.dat" y las mascaras son dibujadas en la pantalla (Figura 27) (Apéndice líneas 223-260). Esto lo hacemos para tener un control visual y saber si las máscaras están generándose correctamente. Si en el algoritmo de generación de las máscaras hay un error o una falla, lo podemos detectar con la imagen de la distribución de las máscaras. Cada punto en el cuadro (Figura 28) presenta el centro de la ventana de 20×20 generada aleatoriamente. Estas ventanas las generamos *N* veces (*N* = 64000, 128000, 256000, 512000). Si la densidad de los puntos difiere en el área del cuadro, necesitamos buscar el error en el algoritmo. Si todos los puntos en el cuadro están distribuidos uniformemente, las máscaras son generadas correctamente.

Cabe destacar que el tamaño de la capa intermedia *A* puede variar, todo depende de la precisión que queramos, si tenemos 512000 neuronas, quiere decir que vamos a tener 512000 ventanas para analizar en nuestra imagen. Si en la capa *A* tuviéramos solo 32000 entonces el análisis de la imagen sería más pobre.

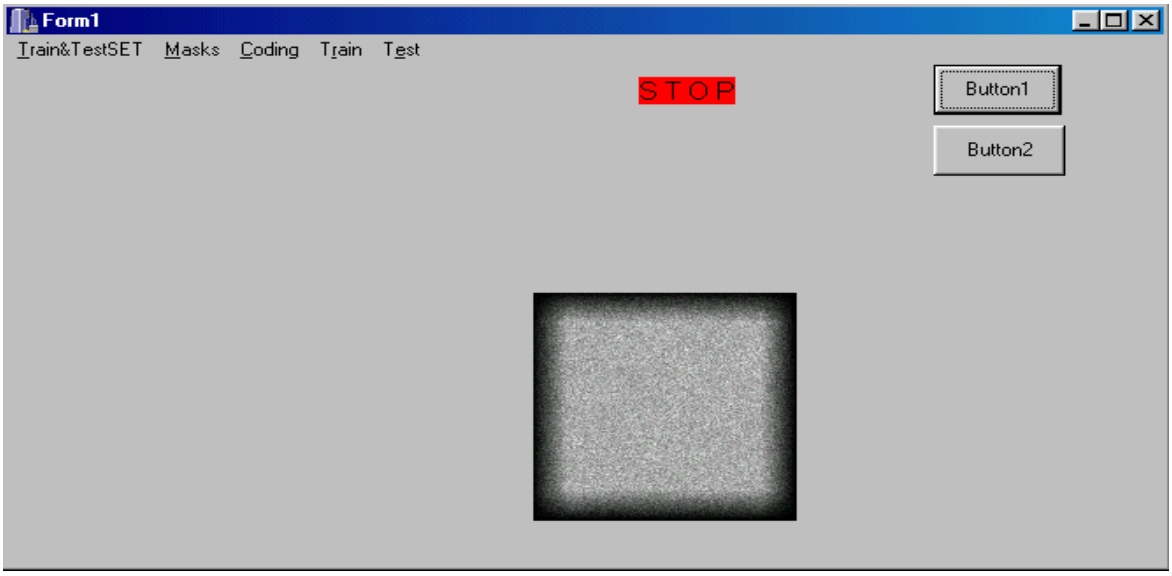


Figura 28.Ejemplo de un conjunto de mascaras.

4.2.3. Módulo "*Coding*"

Este módulo tiene la función de codificar las imágenes y presentarlas en el formato de la red neuronal, es decir, en base a las máscaras previamente creadas. La codificación corresponde a las conexiones que existen entre la capa *S* y la capa *A*. Este módulo tiene la particularidad de detener la codificación y en algún otro momento volverla a comenzar, por eso se tienen las funciónes "Coding_beg" (comienza la codificación) y "Coding_cont" (continúa la codificación). Estas dos funciónes principales básicamente hacen lo mismo solo que una es para comenzar la codificación y la otra es para continuarlo en dado caso de que se hubiera suspendido. (Figura 29)

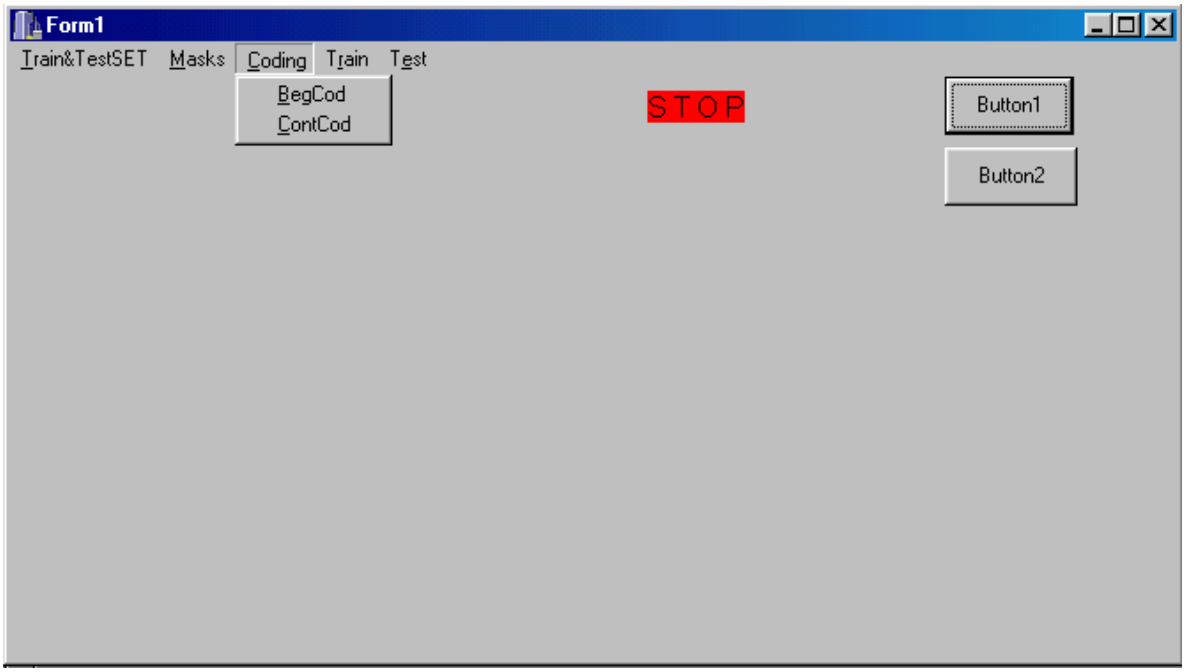


Figura 29. Submenú "Coding"

Cuando se comienza el codificado "Coding_beg" primero se revisa que existan los archivos que se van a utilizar en todo el proceso de codificación, uno de ellos es "actn.dat" y el otro es "sumN.dat". Una vez hecho esto, se lee el archivo "maska.dat" y se obtiene la máscara, la cuál es guardada en un vector de dimensión: (PositivPoint + NegativPoint)*NetSize, o sea, (3+2)*512000. Después se llama a la función "CodeGeneration()", en esta función se lee el archivo "LargeSet.dat", el cuál contiene las imágenes y las coordenadas de cada imagen. Se lee cada imagen y a esta imagen se le hacen distorsiones, se tienen en total 13 distorsiones (Figura 15). Estas distorsiones ayudan a tener un rango más amplio de imágenes de entrenamiento. Entonces primero se lee la imagen y se dibuja, después se le hace la primer distorsión y se dibuja (Figura 30), ya que se hizo esto, viene el proceso de codificación el cual consiste en aplicar la mascara a cada imagen, cuando los "puntos positivos" corresponden a un "1" y los puntos negativos corresponden a un "0" entonces se tiene una neurona activa las cuales son guardadas en el archivo "actn.dat", la suma de estas neuronas es guardada en "sumN.dat". Una vez que se hace esto para cada imagen y sus distorsiones, se completa entonces el proceso de codificación y se sigue con el proceso de entrenamiento. (Apéndice líneas: 267-299)

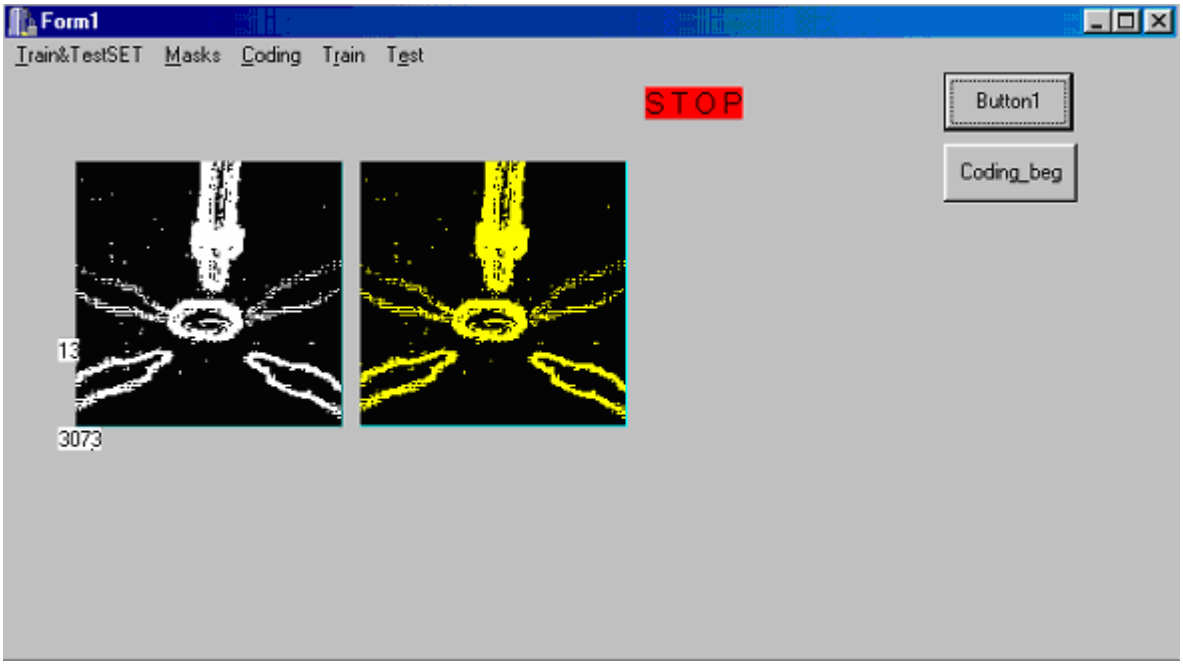


Figura 30. Imagen y distorsiónes.

4.2.4. Modulo "*Train*"

Como su nombre lo dice, este modulo esta encargado de entrenar a la red. El algoritmo de entrenamiento del clasificador neuronal LIRA fue escrito detalladamente en el §3.6.1, fórmulas (4) – (7) y en el §3.6.4, fórmulas (11) – (13). Con éste algoritmo programado cambiamos los pesos de las conexiones entre la capa intermedia *A* y la capa de salida *R*. En la capa de salida tenemos 21 neuronas para *X* y 21 neuronas para *Y*. De las 441 imágenes en la base de datos, se utilizan 220 para entrenamiento.

Este modulo también tiene la particularidad de comenzar el entrenamiento desde el principio (train_beg), o continuar el entrenamiento (train_cont). Básicamente estas dos funciones realizan lo mismo. El procedimiento se realiza de la siguiente manera:

Primero se obtienen las sumas de las neuronas activas de cada imagen, las cuales fueron guardadas previamente en el proceso de codificado de la imagen en un archivo llamado "sumN.dat", a su vez también se obtienen del archivo "KeySet.dat" las imágenes que son de entrenamiento, recordemos que este archivo fue hecho en el modulo de "*Train&TestSET*".

Como se menciono anteriormente, este modulo va a cambiar los pesos de las conexiones entre la capa intermedia *A* y la capa de salida *R*. Se van a guardar estos pesos en dos archivos, que son "matrX.dat" para la coordenada *X*, y "matrY.dat" para la coordenada *Y* . También se cuenta con un archivo llamado "errors.dat" que va a contabilizar el numero de errores durante este proceso de entrenamiento. (Apéndice líneas: 787-912)

Después de esto, se hace un ciclo para entrenar el clasificador neuronal a las 220 imágenes, donde se llama a función "train( )" (Apéndice líneas:566-627). Lo primero que hace esta función es obtener del archivo "actn.dat" las neuronas activas de la imagen con la que se este trabajando, también se obtienen las coordenadas de esta imagen del archivo "LargeSet.dat". Para no trabajar con coordenadas, una manera de facilitar el trabajo es clasificando todas las imágenes en clases, 21 clases para *X* y 21 clases para *Y* (Figura 24), que se relacionan con las coordenadas de la flecha para cada imagen.

Ya que se tienen las coordenadas de la imagen, se llama a dos funciones "ClNameX ( )" y "ClNameY ( )", las cuales van a asignar una clase con base a la coordenada *X* y otra clase con base a la coordenada *Y* respectivamente.

Durante el ciclo de entrenamiento, llamamos a dos funciones que van a ser las encargadas de reconocer la imagen. Estas funciones son: "recognition_tr ( )" y "recognition_trY ( )". Esto funciona de la siguiente manera: se presenta una imagen a la entrada de la red, esta imagen una vez que pasa por la mascara y se obtienen las neuronas activas de esa imagen; se calculan las excitaciones de las neuronas de la capa *R* y se elige la neurona con la excitación máxima. Se reconoce mediante estas funciones mencionadas anteriormente, en estas funciones se cuenta con un parámetro de seguridad llamado "*TDS*" (fórmula 12) con el cual podemos obtener una neurona de respuesta más segura, que este lo suficientemente lejos de neuronas cercanas. Ya que se tiene una neurona de respuesta, se obtiene la clase de esta imagen.

Ya que se obtuvo la clase de reconocimiento de esta imagen, se compara con la clase verdadera de la imagen. Si son diferentes se aumenta en número de errores y se llama a la función de entrenamiento ("training ( )"). Esto se hace tanto para *X* como para *Y*.

En la función "training ( )" se modifican los pesos de las conexiones de la siguiente manera (fórmula 13):

- a las conexiones entre las neuronas activas de la capa intermedia *A* y la clase reconocida por el programa (clase incorrecta), se les disminuye el peso en 1;

- a las conexiones entre las neuronas activas de la capa intermedia *A* y la clase correcta, se les aumenta el peso en 1.

Una vez que se completó el ciclo donde se analizaron las 220 imágenes de entrenamiento, se guardan los pesos de las conexiones en los archivos "matrX.dat" para la coordenada *X* y "matrY.dat" para la coordenada *Y*, y se aumenta el número de paso.

Existen varios métodos para terminar el entrenamiento de una red. Uno de ellos es que el proceso de entrenamiento termine hasta que se tenga un determinado porcentaje de error. En otro se maneja un número determinado de pasos. Se pueden combinar estos dos métodos y terminar con la condición que se cumpla primero. En nuestro programa usamos el número de pasos porque se tiene un tiempo fijo de entrenamiento. En nuestro programa se utilizan 30 pasos para completar el entrenamiento. Una vez que se completaron los 30

pasos la red hacemos el ciclo de prueba del clasificador neuronal. Para éste propósito hacemos el módulo "Test"

4.2.5. Módulo "*Test*"

En éste módulo se va a probar la red con imágenes diferentes a las que fue entrenada. Tenemos en la base de datos 221 imágenes que no han sido utilizadas y que ahora se van a utilizar para probar la red.

Este módulo comienza con la función "recognize ( )" la cuál primero lee de la memoria las matrices de los pesos de las conexiones entre la capa *A* y la capa *R*. Después obtiene la máscara, las coordenadas de cada imagen, obtiene la clase de las imágenes que son de prueba. Estas imágenes las codifica y después llama a dos funciones que reconocen la imagen "recognitionX ( )" y "recognitionY ( )". Estas funciones son casi iguales a "recognition_tr ( )" y "recognition_trY ( )", solo que el parámetro *TDS* es igual a cero. Después de esto se calcula el número de errores al igual que se hizo en el entrenamiento, y se despliega en la pantalla. (Apéndice líneas 968-1081)

En la figura 31 se muestra la pantalla final después que acabó el proceso de prueba.

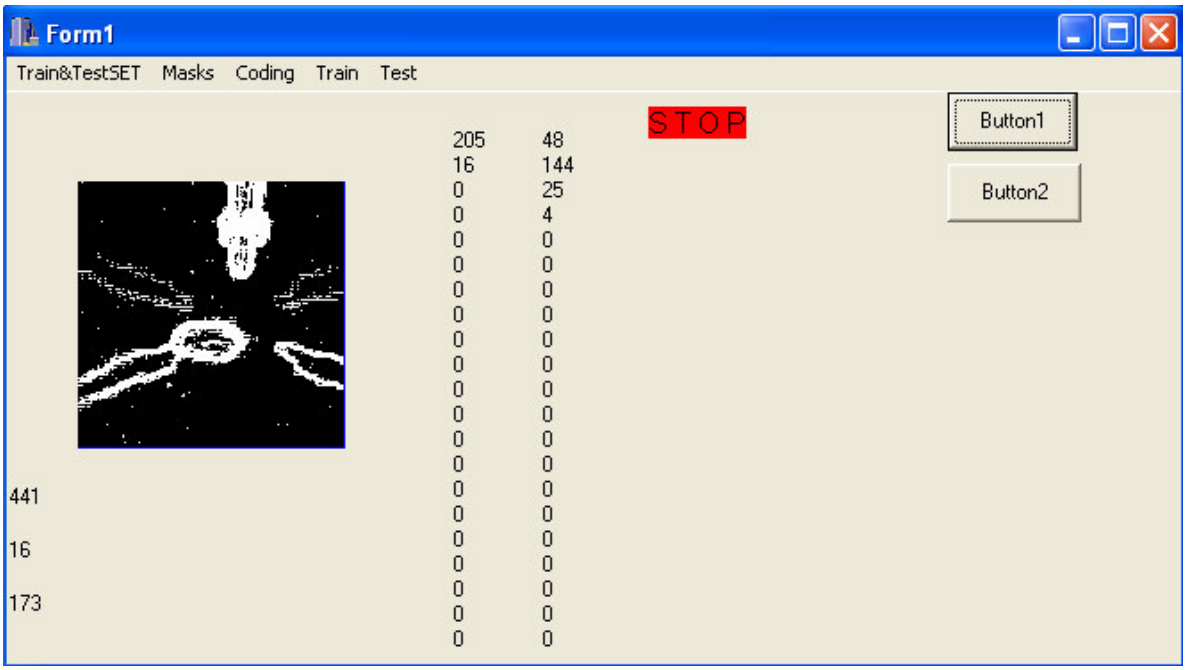


Figura 31.- Pantalla final con los resultados.

En la columna 1 están presentados los resultados del reconocimiento correcto de las 221 imágenes de prueba para la coordenada *X*.

En la columna 2 están presentados los resultados del reconocimiento correcto de las 221 imágenes de prueba para la coordenada *Y*.

Las filas indican lo siguiente:

- primera: reconocimiento exacto;
- segunda: con tolerancia de 0.1 mm;
- tercera: con tolerancia de 0.2 mm;
- cuarta: con tolerancia de 0.3 mm.

Cada fila siguiente tiene la tolerancia aumentada por 0.1 mm.
Por ejemplo, para la coordenada *X* fueron reconocidas 205 imágenes exactas y 16 imágenes con tolerancia de 0.1 mm. Los resultados y sus análisis los vamos a dar en el Capítulo V.

V.- RESULTADOS OBTENIDOS EN LA INVESTIGACIÓN DEL SISTEMA DE RECONOCIMIENTO DE IMÁGENES CON REDES NEURONALES EN LA TAREA DE MICROENSAMBLE

5.1. Descripción de los experimentos.

Nuestra tarea fue relacionada con la investigación de la influencia que tienen ciertos parámetros de la red neuronal sobre los resultados del reconocimiento de las imágenes.

Para la investigación del sistema de reconocimiento fue desarrollado el software con el uso de C++ Builder de la compañía Borland (Capítulo IV).

Como una primera aproximación, se trabajó con 23 imágenes, de las cuales 12 eran para entrenamiento y el resto para pruebas. Estas 23 imágenes que corresponden a los desplazamientos de la flecha a lo largo de los ejes X y Y con un paso de 0.5 mm.

Para la primera base de datos se crearon dos clasificadores de las posiciones de la flecha. El primero tiene tres reconocimientos de los desplazamientos en X: $X_{pin} < X_{hole}$; $X_{pin} = X_{hole}$; $X_{pin} > X_{hole.}$ Y el segundo tiene tres reconocimientos de los desplazamientos en Y: $Y_{pin} < Y_{hole}$; $Y_{pin} = Y_{hole}$; $Y_{pin} > Y_{hol}$. Se usaron 12 imágenes seleccionadas aleatoriamente de la primera base de datos para entrenar a los clasificadores y las 11 imágenes restantes para comprobarlas. Ambos clasificadores reconocen todas las imágenes correctamente *(Nerror = 0).* Con ésta primera base de datos fueron aprobados desplazamientos (*distortions*) de imágenes iniciales para aumentar el número de imágenes en la etapa de entrenamiento del clasificador neuronal LIRA.

Como resultado de este primer experimento, se observa la necesidad de incrementar la base de datos. La segunda base de datos fue hecha con 441 imágenes.

Para la investigación, dividimos esta segunda base en dos partes: la mitad de imágenes la usamos para entrenamiento de la red neuronal (220) y la otra mitad (221) la usamos para probar nuestro sistema.

Nuestra red neuronal LIRA tiene 22500 neuronas en la capa de entrada correspondientes a el tamaño en píxeles de la imagen (150×150). El número de neuronas *NetSize* de la capa intermedia *A* es uno de los parámetros sobre investigación. En la primera etapa elegimos *NetSize=64000*. Los experimentos se realizaron siguiendo la siguiente secuencia de pasos:

- Hacemos un tratamiento preliminar de todas las imágenes, extraemos los contornos.

- Dividimos las imágenes en dos subconjuntos: para entrenamiento y para prueba.

- Formamos las máscaras y codificamos la imagen.

- Elegimos la primera imagen del juego de imágenes de entrenamiento.

- Entrenamos nuestra red neuronal a ésta imagen. Durante el entrenamiento cambiamos los pesos de las conexiones entre la capa *A* y la capa *R,* estos pesos los guardamos en 2 matrices que corresponden a la coordenada *X* y *Y*. Todo el entrenamiento consiste de 30 ciclos.

- Repetimos éste procedimiento para todas las imágenes del grupo de entrenamiento.

- Después trabajamos con el segundo grupo de imágenes para probar el sistema

5.2. Investigación de la influencia de diferentes parámetros de la red a los resultados obtenidos

Se realizaron 5 experimentos, cada uno de estos experimentos está relacionado con una máscara (Capítulo V) de conexiones entre la capa *S* y la capa *A*. Hacemos 5 máscaras diferentes para tener resultados de la investigación estadísticamente estables. En la Tabla 3 estos experimentos se llaman "corridas". Cada corrida da resultados sobre el número de imágenes reconocidas correctamente *N* de las 221 imágenes de prueba. También se obtuvo el promedio de éstas 5 corridas para poder obtener un porcentaje de reconocimiento. Para cada corrida tenemos un número de fila *F* que representa a *N* con una tolerancia que se incrementa en 0.1mm por cada fila. Los experimentos se realizaron en una computadora personal Pentium 4 a 2.2 GHz y 512 Mb en RAM. El tiempo que demandó esta computadora para el entrenamiento, con 64000 neuronas en la capa *A*, fue de 40 seg, y para probar el sistema demandó 10 seg.

Tabla 3.- Resultados de la investigación del sistema con 64000 neuronas en la capa intermedia A

F	Corridas												% promedio		% reconocimiento con tolerancia	
	1		2		3		4		5		Promedio					
	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
1	173	63	165	73	176	67	180	76	160	78	170.8	71.4	77.64	32.45	77.64	32.45
2	46	108	54	108	43	118	41	116	61	112	49	112.4	22.27	51.09	99.91	83.55
3	2	23	1	21	2	21	0	23	0	19	1	21.4	0.45	9.73	100	93.27
4	0	23	0	11	0	10	0	4	0	10	0	11.6	0.00	5.27	100	98.55
5	0	3	0	3	0	3	0	2	0	2	0	2.6	0.00	1.18	100	99.73
6	0	1	1	5	0	2	0	0	0	0	0.2	1.6	0.09	0.73	100	100

Es necesario explicar que las últimas dos columnas en la Tabla 3 "% de reconocimiento con tolerancia" son calculadas de la siguiente manera. Por ejemplo, para la coordenada *X*. El promedio de el número de imágenes correctas en la fila 1 es de 170.8, que corresponde a 77.64%. En la fila 2 (con tolerancia de 0.1mm) tenemos 49 imágenes correctas que corresponden a 22.27%. Para saber cuantas imágenes reconoce el clasificador neuronal correctamente con una tolerancia de 0.1mm, tenemos que sumar el número de imágenes reconocidas exactamente, i.e. 77.64% con el número de imágenes

reconocidas con tolerancia de 0.1mm, i.e. 22.27%; en total vamos a obtener 99.91% de reconocimiento correcto con tolerancia de 0.1mm. Para saber que porcentaje de las imágenes correctas vamos a obtener con tolerancia de 0.2mm tenemos que sumar 99.91% con 0.9% (corresponde a la fila 3) y obtenemos el 100%. En las tablas siguientes vamos a presentar las columnas "porcentaje de reconocimiento con tolerancia".

Después se realizaron experimentos con *NetSize= 128000, 256000 y 512000* neuronas, en las tablas 4, 5 y 6 se muestran los resultados.

Tabla 4.- Resultados de la investigación del sistema con: 128000 neuronas en la capa intermedia A.

F	Corridas										Promedio		% reconocimiento con tolerancia	
	1		2		3		4		5					
	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
1	166	68	170	78	176	77	186	71	162	71	172	73	78.18	33.18
2	54	120	50	117	44	109	33	126	55	122	47.2	118.8	99.64	87.18
3	0	23	0	19	0	20	1	17	3	19	0.8	19.6	100	96.09
4	0	9	0	5	0	9	0	5	0	6	0	6.8	100	99.18
5	0	0	0	1	0	1	0	0	0	2	0	0.8	100	99.55
6	0	0	0	0	0	3	0	1	0	0	0	0.8	100	99.91
7	0	0	0	0	0	1	0	0	0	0	0	0.2	100	100

Tabla 5.- Resultados de la investigación del sistema con: 256000 neuronas en la capa intermedia A.

F	Corridas										Promedio		% reconocimiento con tolerancia	
	1		2		3		4		5					
	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
1	168	73	167	71	170	71	172	72	163	66	168	70.6	76.36	32.09
2	52	127	54	117	50	115	47	116	57	119	52	118.8	100	86.09
3	1	13	0	19	1	21	2	20	1	24	1	19.4	100	94.91
4	0	6	0	10	0	11	0	12	0	9	0	9.6	100	99.27
5	0	1	0	3	0	2	0	1	0	3	0	2	100	100
6	0	1	0	1	0	1	0	0	0	0	0	0.6	100	100

Tabla 6.- Resultados de la investigación del sistema con: 512000 neuronas en la capa intermedia A.

F	Corridas										Promedio		% reconocimiento con tolerancia	
	1		2		3		4		5					
	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
1	165	76	175	70	170	72	171	75	166	72	169.4	73	77.00	33.18
2	56	113	46	114	51	112	50	109	55	118	51.6	113.2	100	84.64
3	0	23	0	21	0	24	0	26	0	21	0	23	100	95.09
4	0	9	0	15	0	13	0	10	0	8	0	11	100	100
5	0	0	0	1	0	0	0	1	0	21	0	4.6	100	100

Se puede observar que en esta primera etapa de experimentos, el aumento de los números de neuronas en la capa *A* de 64000 hasta 512000 no indica necesariamente un incremento en el porcentaje de reconocimiento. Pero con el aumento del número de neuronas obtenemos el 100% de reconocimiento correcto con menos tolerancia. Para la coordenada *Y* en la Tabla 3 obtenemos el 100% con una tolerancia de 0.4mm. En la tabla 6 obtenemos el 100% con una tolerancia de 0.3mm. Esto significa que podemos reconocer la posición de la flecha con más precisión si aumentamos el número de neuronas en la capa *A*.

Es conocido que el porcentaje de reconocimiento puede ser incrementado si durante el ciclo de entrenamiento, se usan las imágenes iniciales con distorsiones [43]. En la segunda etapa de experimentos también utilizamos este parámetro para la investigación del sistema. Se realizaron experimentos con el número de distorsiones "*DistNum"* igual a: 5, 9 y 13. Cabe destacar que el parámetro *DistNum* incluye a la imagen inicial, por lo que si *DistNum=5* entonces tenemos 4 distorsiones. Las tablas 7, 8 y 9 muestran los resultados obtenidos en esta segunda etapa de experimentos con *NetSize=64000* y *DistNum=5, 9 y 13.*

Tabla 7.- Resultados de la investigación del sistema con neuronas en la capa A=64000 y número de distorsiones=5.

Net	Dist	Corridas												% reconocimiento con tolerancia	
Size	Num	1		2		3		4		5		Promedio			
64000	5	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
		200	74	204	79	206	72	203	82	200	77	202.6	76.8	92.09	34.91
		21	141	17	136	15	146	18	134	21	138	18.4	139	100	98.09
		0	5	0	4	0	3	0	5	0	5	0	4.4	100	100
		0	1	0	2	0	0	0	0	0	1	0	0.8	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100

Tabla 8.- Resultados de la investigación del sistema con neuronas en la capa A=64000 y número de distorsiones=9.

Net	Dist	Corridas												% reconocimiento con tolerancia	
Size	Num	1		2		3		4		5		Promedio			
64000	9	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
		200	58	196	46	198	58	197	48	202	68	198.6	55.6	90.27	25.27
		20	142	24	130	22	133	23	128	18	127	21.4	132	100	85.27
		0	16	0	30	0	19	0	29	0	20	0	22.8	100	95.64
		0	4	0	14	0	10	0	15	0	5	0	9.6	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100

Tabla 9.- Resultados de la investigación del sistema con neuronas en la capa A=64000 y número de distorsiones=13.

Net Size	Dist Num	Corridas												% reconocimiento con tolerancia	
		1		2		3		4		5		Promedio			
64000	13	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
		192	50	186	41	193	42	187	42	185	42	188.6	43.4	85.73	19.73
		29	107	35	119	28	110	34	111	36	102	32.4	109.8	100	69.64
		0	37	0	32	0	42	0	42	0	37	0	38	100	86.91
		0	27	0	23	0	24	0	24	0	31	0	25.8	100	98.64
		0	0	0	5	0	3	0	1	0	3	0	2.4	100	99.73
		0	0	0	0	0	0	0	0	0	0	0	0	100	99.73

Se hicieron experimentos con: *NetSize= 128000, 256000 y 512000 y DistNum= 1, 5, 9 y 13*. En las tablas 10-12 se muestran estos resultados. Para tener una mejor apreciación de estos resultados se muestran unos gráficos (Figuras 32 y 33) en donde podemos observar resumidamente el número de imágenes reconocidas con tolerancia de 0.1mm.

Tabla 10. Resultados de la investigación del sistema con neuronas en la capa A= 128000 y número de distorsiones=5, 9 y 13.

Net Size	Dist Num	Corridas												% reconocimiento con tolerancia	
		1		2		3		4		5		Promedio			
		X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
128000	5	201	74	205	82	200	75	202	79	204	77	202.4	77.4	92.00	35.18
		19	141	15	133	20	140	18	136	16	140	17.6	138	100	97.91
		0	4	0	5	0	5	0	4	0	3	0	4.2	100	99.82
		0	1	0	0	0	0	0	1	0	0	0	0.4	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
	9	201	53	205	50	210	56	204	45	201	48	204.2	50.4	92.82	22.91
		19	133	15	140	10	135	16	137	19	137	15.8	136	100	84.91
		0	26	0	24	0	22	0	25	0	26	0	24.6	100	96.09
		0	8	0	6	0	7	0	12	0	8	0	8.2	100	99.82
		0	0	0	0	0	0	0	0	0	1	0	0.2	100	99.91
	13	197	46	195	44	194	43	200	35	198	42	196.8	42	89.45	19.09
		24	120	26	99	27	102	21	126	23	113	24.2	112	100	70.00
		0	38	0	40	0	38	0	40	0	38	0	38.8	100	87.64
		0	16	0	32	0	30	0	19	0	20	0	23.4	100	98.27
		0	1	0	6	0	6	0	0	0	7	0	4	100	100
		0	0	0	0	0	1	0	0	0	0	0	0.2	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	0	0	1	0	0	0	0	0	0.2	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	0	0	0	0	1	0	1	0	0.4	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100

Tabla 11. Resultados de la investigación del sistema con neuronas en la capa A= 256000 y número de distorsiones=5, 9 y 13.

Net Size	Dist Num	Corridas												% reconocimiento con tolerancia	
		1		2		3		4		5		Promedio			
		X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
256000	5	206	85	204	81	206	70	203	77	208	81	205.4	78.8	93.36	35.82
		15	132	17	136	15	146	18	140	13	133	15.6	137.4	100	98.27
		0	2	0	4	0	5	0	3	0	5	0	3.8	100	100
		0	2	0	0	0	0	0	1	0	2	0	1	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
	9	200	48	204	60	206	47	201	45	200	53	202.2	50.6	91.91	23.00
		20	136	16	128	14	141	14	131	20	137	16.8	134.6	99.55	84.18
		0	24	0	26	0	21	0	26	0	23	0	24	99.55	95.09
		0	11	0	6	0	10	0	17	0	7	0	10.2	99.55	99.73
		0	1	0	0	0	1	0	1	0	0	0	0.6	99.55	100
	13	199	35	189	38	198	36	205	35	202	39	198.6	36.6	90.27	16.64
		22	122	32	111	23	111	16	110	19	109	22.4	112.6	100	67.82
		0	36	0	38	0	37	0	41	0	39	0	38.2	100	85.18
		0	27	0	30	0	32	0	31	0	28	0	29.6	100	98.64
		0	1	0	1	0	3	0	2	0	5	0	2.4	100	99.73
		0	0	0	0	0	2	0	1	0	1	0	0.8	100	100
	13	199	35	189	38	198	36	205	35	202	39	198.6	36.6	90.27	16.64
		22	122	32	111	23	111	16	110	19	109	22.4	112.6	100	67.82
		0	36	0	38	0	37	0	41	0	39	0	38.2	100	85.18
		0	27	0	30	0	32	0	31	0	28	0	29.6	100	98.64
		0	1	0	1	0	3	0	2	0	5	0	2.4	100	99.73
		0	0	0	0	0	2	0	1	0	1	0	0.8	100	100

Tabla 12. Resultados de la investigación del sistema con neuronas en la capa A = 512000 y numero de distorsiones=5, 9 y 13.

Net Size	Dist Num	Corridas 1		2		3		4		5		Promedio		% reconoci-miento con tolerancia	
		X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
	5	203	80	202	70	205	79	206	70	203	76	203.8	75	92.64	34.09
		17	133	18	147	15	138	14	149	17	138	16.2	141	100	98.18
		0	7	0	2	0	3	0	1	0	3	0	3.2	100	99.64
		0	0	0	1	0	0	0	0	0	0	0	0.2	100	99.73
		0	0	0	0	0	0	0	0	0	0	0	0	100	99.73
	9	205	51	207	52	208	49	206	48	206	48	206.4	49.6	93.82	22.55
		16	143	14	145	13	141	15	140	15	138	14.6	141.4	100	86.82
		0	22	0	18	0	23	0	25	0	25	0	22.6	100	97.09
		6	5	0	6	0	8	0	8	0	10	1.2	7.4	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
512000	13	197	33	197	39	199	37	201	32	204	36	199.6	35.4	90.73	16.09
		23	108	23	126	21	105	20	118	17	104	20.8	112.2	100	67.09
		0	34	0	35	0	45	0	36	0	44	0	38.8	100	84.73
		0	35	0	14	0	32	0	26	0	31	0	27.6	100	97.27
		0	2	0	1	0	0	0	7	0	4	0	2.8	100	98.55
		0	0	0	0	0	0	0	2	0	1	0	0.6	100	98.82
		0	0	0	1	0	1	0	0	0	0	0	0.4	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	1	0	0	0	0	0	0	0	0.2	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	1	0	0	0	0	0	0	0	0.2	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	1	0	0	0	0	0	0	0	0.2	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	0	0	0	0	0	0	0	0	0	100	100
		0	0	0	1	0	0	0	0	0	0	0	0.2	100	100

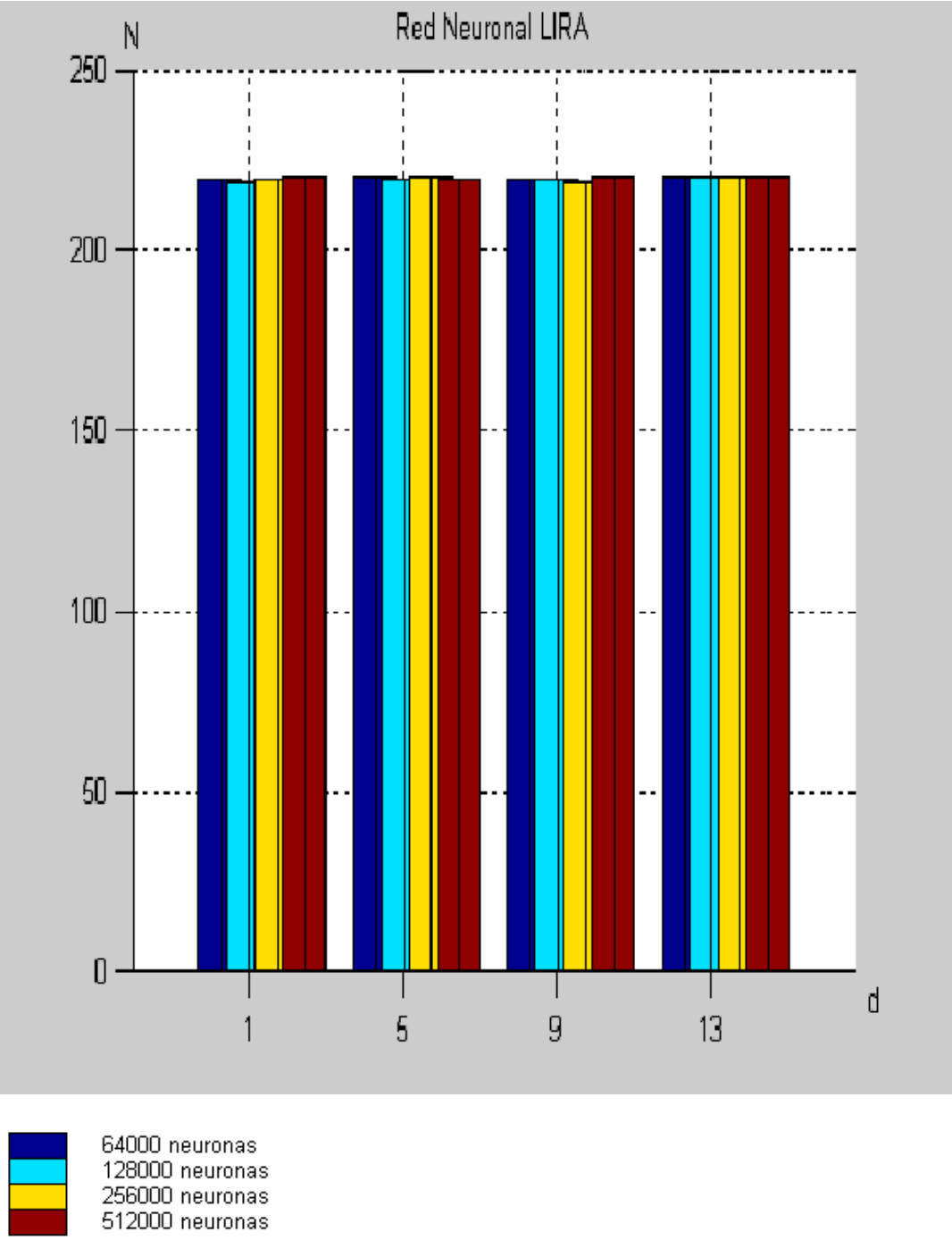


Figura 32. Resultados del reconocimiento para la coordenada X, donde N- Imágenes reconocidas en la coordenada X con tolerancia de 0.1 mm; d- número de distorsiones.

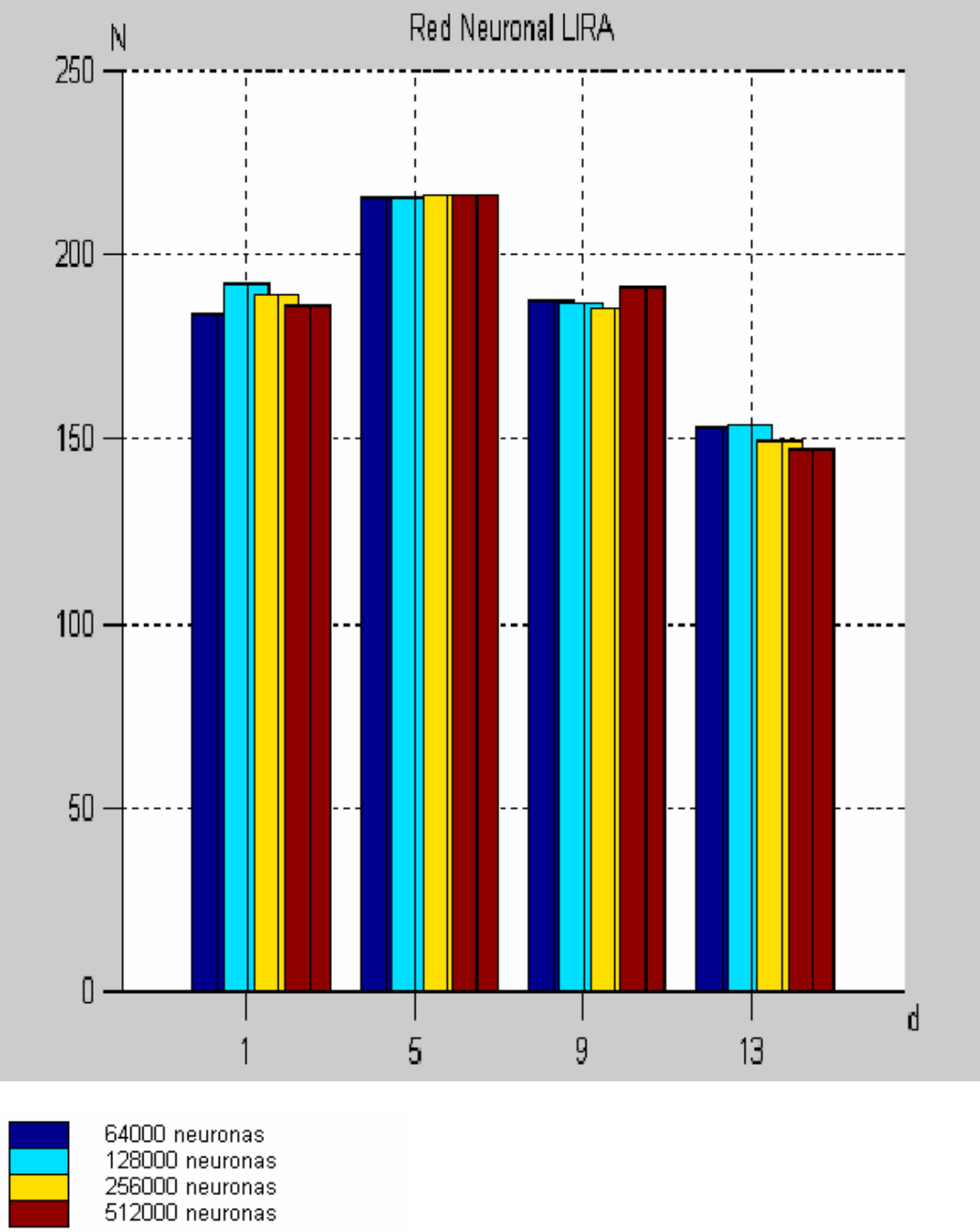


Figura 33. Resultados del sistema para la coordenada Y, donde N- Imágenes reconocidas en la coordenada X con tolerancia de 0.1 mm; d- número de distorsiones.

Se puede observar que en nuestra tarea los desplazamientos de las imágenes no mejoran la calidad del reconocimiento.

El análisis de los resultados lo hacemos en el siguiente párrafo.

5.3 Análisis de los resultados obtenidos

Se realizó la simulación de un sistema de reconocimiento de imágenes en la tarea de microensamble con el uso del clasificador neuronal LIRA. Los resultados obtenidos del reconocimiento de las posiciones de la flecha son suficientemente buenos. Por ejemplo, para la coordenada *X* con una tolerancia de 0.1mm tenemos el 100% de reconocimiento correcto. Para la coordenada *Y* con una tolerancia de 0.1mm tenemos el 87.78% de reconocimiento correcto. Estas tolerancias son aceptables en trabajos de micromecánica en primera etapa. Pero con la disminución de los tamaños del microequipo, va a aumentar la precisión necesaria. Por eso, el reconocimiento exacto que obtenemos en este trabajo (78.18% para *X* y 33.18% para *Y*) es insuficiente para trabajos futuros y es necesario mejorar estos resultados.

La introducción de las distorsiones de las imágenes para aumentar la base de datos no mejoraba el reconocimiento en nuestra tarea. La explicación de éste fenómeno esta relacionada con la resolución de nuestras imágenes. Tenemos distorsiones por 1 y 2 pixeles. El desplazamiento de la imagen por la coordenada *X* a 0.1 mm corresponde a 1.8 pixeles, por la coordenada *Y* corresponde a 1 pixel. Por eso cuando hacemos distorsiones por un píxel la imagen distorsionada cae en otra clase y no en la que estamos reconociendo.

Para la primera base de imágenes, donde la diferencia entre clases fue de 0.5 mm, el desplazamiento por 1 pixel no cambio la clase a reconocer. Para la segunda base de imágenes el desplazamiento por 1 pixel mejora los resultados de reconocimiento, pero el desplazamiento por 2 pixeles especialmente para la coordenada *Y* empeora los resultados del reconocimiento.

Podemos resolver este problema si aumentamos la resolución de nuestras imágenes iniciales o usar un sistema sin distorsiones.

Para trabajos futuros es necesario tener una base de imágenes más grande. Es conocido que para el entrenamiento de las redes neuronales demandan muchas imágenes. Como solución de este problema podemos proponer crear un sistema de microensamble que pueda trabajar en tiempo real. En este caso podemos tomar una mayor cantidad de imágenes.

VI. CONCLUSIONES

- Es de gran importancia el desarrollo de tecnologías que fabriquen o manipulen microdispositivos con una gran precisión y exactitud ya que el uso de estos es cada vez mayor. La mayoría de las veces no nos damos cuenta de esto pero indirectamente utilizamos muchos microdispositivos a lo largo del día, por ejemplo: relojes, computadoras, etc. Existen dos principales tecnologías para la fabricación de microdispositivos: MEMS y MET.

- En esta tesis se realizo un sistema de reconocimiento y clasificación de imágenes con redes neuronales que se aplica directamente sobre una tarea de microensamble desarrollada para una aplicación con la tecnología MET. Esta aplicación consiste en el desarrollo de microfiltros para filtración fina de líquidos y gases, que esta relacionado con diferentes áreas de procesos tecnológicos y protección ambiental. La filtración fina del aire es necesaria para tener lugares limpios usados en microelectrónica, medicina y otras aplicaciones, y la filtración de gases es necesaria para la proteger el ambiente de contaminación producida por industrias, vehículos y otros dispositivos que utilicen combustible fósil. Los filtros existentes no siempre resuelven con eficiencia sus tareas ya que los filtros de papel para líquidos no permiten la filtración de partículas menores a 0.2 μm. Con MET se propuso hacer filtros basados en un gran número de microanillos especiales, el diámetro interno de cada microanillo es de 0.8 mm, el externo es de 1.3mm y la altura de 0.6 mm. El sistema de microensamble tiene que colocar estos microanillos en los canales del microfiltro.

- El microensamble es una parte muy importante del trabajo con microdispositivos. Es necesario contar con un sistema de microensamble adecuado a las dimensiones con las que se trabaja ya que como se mencionó anteriormente, al tener piezas más pequeñas las fuerzas superficiales se vuelven más dominantes. Por eso fue desarrollado un esquema para la colocación de los microanillos en el tubo (canal) y evitar efectos negativos de las fuerzas superficiales.

- En el futuro el microensamble debe ser una tarea completamente automatizada. Si vamos a construir las siguientes generaciones de microequipo, un operador no puede controlar todos los procesos. Si no hacemos los procesos de producción y ensamble totalmente automatizados para resolver la tarea, contaríamos con errores humanos. Por eso se requiere de sensores adecuados, además de sistemas de visión que nos permitan tener un mejor control de la tarea. El sistema de visión es de gran importancia ya que son partes extremadamente pequeñas y por lo tanto difíciles de observar, además podemos obtener información más precisa acerca de las posiciones relativas de los microobjetos que se requieren ensamblar.

- Existen muchas técnicas para el reconocimiento de imágenes pero en esta tesis me enfoco al área de redes neuronales. Las redes neuronales nos proporcionan muchas ventajas dentro de las cuáles se encuentran: el carácter paralelo del

tratamiento de la información, y la capacidad de ser entrenadas y responder con gran velocidad (en tiempo real) a situaciones para las que fueron entrenadas.

- Se propone adaptar tecnología OCR (*Optical Character Recognition*) para controlar el proceso de microensamble. Para éste propósito se desarrollo un método para obtener información de imágenes 3D con las imágenes 2D. Para este propósito usamos una cámara WEB y cuatro fuentes de luz. Con las sombras de la flecha con el anillo y el orificio podemos obtener la información sobre la posición de la flecha.

- Se hicieron experimentos con imágenes reales de la microaguja con el microanillo y el orificio. Estos experimentos primero se hicieron con una base de datos de 23 imágenes y se obtuvieron resultados positivos. El clasificador neuronal reconoció correctamente la posición relativa de la aguja al orificio; con esta base obtuvimos resultados cualitativos. No se hicieron investigaciones más profundas. Todos estos experimentos se hicieron fuera de línea (*offline*).

- En la siguiente serie de experimentos se trabajo con una base de 441 imágenes, y se obtuvieron resultados cuantitativos. Por ejemplo, para el reconocimiento de la posición de la flecha obtuvimos los siguientes resultados: para la coordenada X con tolerancia de ± 0.1mm tenemos el 100% de reconocimiento correcto; para la coordenada Y con tolerancia de ± 0.1mm obtuvimos un 87.18% de reconocimiento correcto. Pero si queremos reconocer la posición exacta tenemos resultados no tan buenos. Para mejorar estos resultados podemos proponer trabajos futuros para el desarrollo de nuestro sistema.

- Se hicieron diferentes experimentos para investigar la influencia de algunos parámetros de la red neuronal a los resultados del reconocimiento. Por ejemplo, el aumento del número de neuronas de la capa A no aumenta significativamente el reconocimiento correcto de las imágenes. Pero con el aumento del número de neuronas podemos reconocer las imágenes con mejor tolerancia.

- Nos dimos cuenta que al trabajar con distorsiones de las imágenes iniciales no mejoraba el resultado del reconocimiento. Las distorsiones que hacemos con las imágenes son desplazamientos de 1 o 2 pixeles en las coordenadas X, Y. La distancia de 0.1mm en la posición de la flecha corresponde a 1.8 pixeles en la coordenada X y a 1 pixel en la coordenada Y en la imagen. Por eso, si hacemos distorsiones de 1 pixel que equivale a 0.1mm en la coordenada Y, obtenemos una imagen de otra clase. La introducción de las distorsiones funcionaba muy bien para la primera base de datos, porque el paso en las posiciones de la flecha fue de 0.5mm. En este caso el desplazamiento de la imagen tanto en la coordenada X como en la coordenada Y pertenecen todavía a la misma clase de la imagen inicial. Por eso, la introducción de las distorsiones en la primera base de imágenes tuvo éxito. En la segunda base de imágenes tenemos una tendencia clara de retroceso en el mejoramiento de los resultados del reconocimiento de las imágenes con el aumento del número de distorsiones.

6.1. Trabajos futuros

- Todos estos resultados son muy importantes para la planeación de los trabajos futuros con el sistema desarrollado. Lo primero que necesitamos haces es desarrollar el sistema de microensamble para que funcione en tiempo real. Con esto podemos aumentar significativamente el número de imágenes para entrenar nuestro clasificador neuronal LIRA.

- Necesitamos mejorar la calidad del reconocimiento de imágenes. Introducir las modificaciones al clasificador neuronal para aumentar el porcentaje del reconocimiento correcto y disminuir la tolerancia en las posiciones reconocidas.

- En los trabajos futuros es necesario obtener las imágenes con una mejor resolución. Esto va a aumentar el porcentaje de reconocimiento correcto.

APENDICE

```
1:      //---------------------------------------------------------------------------
2:      #include <vcl.h>
3:      #include <io.h>
4:      #include <stdio.h>
5:      #include <stdlib.h>
6:      #include <string.h>
7:      #include <conio.h>
8:      //#include <exception.h>
9:      #include <iostream.h>
10: #pragma hdrstop
11: //#include "locallib.h"
12: #include "Assembl.h"
13: //---------------------------------------------------------------------------
14: #pragma package(smart_init)
15: #pragma resource "*.dfm"

16: void first(void);
17: void MaskCreation(void);
18: void MaskTest(void);
19: void CodeGeneration(void);
20: void ImRead(long int iNum);
21: long int Coding(void);
22: void Distortion(void);
23: void Ima(void);
24: void training(void);
25: void trainingY(void);
26: void train(void);
27: unsigned short recognitionX(void);
28: unsigned short recognitionY(void);
29: unsigned short recognition_tr(void);
30: unsigned short recognition_trY(void);
31: unsigned short ClNameX(void);
32: unsigned short ClNameY(void);

33: unsigned int ImNum=441; //Número de imagenes en la base de datos
34: unsigned int NumTr=221; //Número de imagenes para el entrenamiento
35: unsigned int imWidth=150; //Ancho de la imagen en pixeles
36: unsigned int imHeight=150;//Altura de la imagen en pixeles
37: long int WindowWidth=20;
38: long int WindowHeight=20;
39: int ClassSize=21; //tamaño de la clase de salida
40: int INCLIN=2;  //inclinacion de la distorsion
41: long int delta=1; //parametro de distorsion
42: unsigned int PassN=30; //Número de pasos para finalizar el entrenamiento
43: long int NetSize=512000; //Número de neuronas en la capa A
44: long int ActivSize=NetSize/4;
45: long int PositivPoint=2;
46: long int NegativPoint=3;
47: unsigned DISTNUM=9; //Numero de distorsiones (Numero maximo=12)
48: float TDS=0.15;
49: long int MatSizeX=ClassSize*NetSize;
50: long int MatSizeY=ClassSize*NetSize;
51: long unsigned distortion,dist;
52: long MaskSize=(PositivPoint+NegativPoint)*NetSize;
53: unsigned short* mtrX;  //[MatSize] = ClassSize*NetSize;
54: unsigned short* mtrY;  //[MatSize] = ClassSize*NetSize;
55: long unsigned* ActivNeuronNumber=new long unsigned[ActivSize];
56: long unsigned* SUMNumber=new long unsigned[DISTNUM*ImNum];
```

```
unsigned int im[640][480],im2[150][150];
unsigned char imbin[150][150];
int p1,p2,p3,p4,d1,d2,grad;
float x,y,z;
unsigned char* BinaryImage;
unsigned char* BinaryImage2;
unsigned int iNum, Dist;
unsigned short true1,rec;
unsigned short true2,rec2;
long unsigned offset, offset1;
long int ErrorNumber,ErrorNumber2;
unsigned int pass;
long unsigned serv[3];
long unsigned Errors[1000],Errors2[1000];
long* ImMask;
int c1=0;

char* SetFormation;
FILE *fp;
FILE *strips;
FILE *ActiveNeuron;
FILE *matrX;
FILE *matrY;
FILE *errors;
FILE *mark;
FILE *remark;
FILE *nsum;
TForm1 *Form1;
Graphics::TBitmap* Bitmap1 = new Graphics::TBitmap();
Graphics::TBitmap* Bitmap2 = new Graphics::TBitmap;
Graphics::TBitmap* Bitmap3 = new Graphics::TBitmap;
Graphics::TBitmap* Bitmap4 = new Graphics::TBitmap;

__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}

//---------------------------------------------------------------------------
void first(void)
{
int m;
unsigned int i,j;
unsigned int kk[1];
float z;
float coor[3];
    char ch[20];
    z=2.5;
    fp=fopen("LargeSet.dat","wb");
    if(fp==NULL)
    { Form1->Button1->Caption="Cannot open LargeSet";
    terminate();}
    fclose(fp);

    x=-1.0;y=-1.0; m=1;
    met:

    Form1->Canvas->TextOutA(1,150,AnsiString(m));
```

```
110:    if((9>=m)&&(m>=1)){sprintf(ch,"00%d.bmp",m);};
111:    if((99>=m)&&(m>=10)){sprintf(ch,"0%2d.bmp",m);};
112:    if((999>=m)&&(m>=100)){sprintf(ch,"%3d.bmp",m);};

113:    Bitmap4->LoadFromFile(ch);

114:    Form1->Canvas->Brush->Bitmap = Bitmap4;
115:    Form1->Canvas->FillRect(Rect(0,0,imWidth,imHeight));

116:    for (i=0; i<imHeight; i++)
117:    for (j=0; j<imWidth;j++)
118:    {
119:    imbin[j][i]=(char) Bitmap4->Canvas->Pixels[j][i];
120:    int wrk=(int)imbin[j][i];wrk=wrk*200;
121:    Form1->Canvas->Pixels[200+j][i+150]=TColor( wrk + (wrk<<8)+(wrk<<16));
122:    }

123:    coor[0]=x;
124:    coor[1]=y;
125:    coor[2]=z;

126:    fp=fopen("LargeSet.dat","ab");
127:    if(fp==NULL)
128:    { Form1->Button1->Caption="Cannot open LargeSet";
129:    terminate();}

130:    for(i=0; i<imHeight; i++)
131:    for(j=0; j<imWidth;j++)
132:    {
133:    kk[0]=Bitmap4->Canvas->Pixels[j][i];
134:    if(fwrite(kk,sizeof(unsigned char),1,fp)==NULL)
135:    { Form1->Button1->Caption="Cannot write to LargeSet";
136:    terminate();}
137:    }

138:    if(fwrite(coor,sizeof(float),3,fp)==NULL)
139:    { Form1->Button1->Caption="Cannot write to LargeSet";
140:    terminate();}
141:    fclose(fp);

142:    m=m++;
143:    if (m<=ImNum)
144:    {
145:    if(x<=0.95){x=x+0.1; goto met;}
146:    if(x>0.95){if(y<=0.95){y=y+0.1; x=-1; goto met;} {goto final;}}
147:    }

148:    final: ;
149:    }
150:    //---------------------------------------------------------------------------


151:    void __fastcall TForm1::TrainTest1Click(TObject *Sender)
152:    {
153:    int i,ii;
154:    int ch;

155:    SetFormation=new char[ImNum+1];
156:    FILE* trset;
157:    randomize();
```

```
158:    first();

159:    for(i=0;i<=ImNum; i++){SetFormation[i]=0;}

160:    ii=-1;
161:    for (i=0;i<=ImNum;i++)
162:    {
163:    ii*=-1;
164:    SetFormation[i]=(ii+1)/2;
165:    }

166:    //KeySet- Archivo con el codigo para las imagenes de entrenamiento y prueba
167:    trset=fopen("KeySet.dat","wb");
168:    if(trset==NULL)
169:    { Form1->Button1->Caption="Cannot open KeySet";
170:    terminate();}
171:    fwrite(SetFormation,sizeof(unsigned char),ImNum+1,trset);
172:    fclose(trset);

173:    delete SetFormation;
174:    }
175:    //---------------------------------------------------------------------------

176:    //---------------------------------------------------------------------------

177:    void __fastcall TForm1::MaskGeneration(TObject *Sender)
178:    {
179:    randomize();
180:    MaskCreation();
181:    MaskTest();
182:    }
183:    //---------------------------------------------------------------------------

184:    void MaskCreation(void)
185:    {
186:    long c[1];
187:    long int xx,yy,uu,vv;
188:    long int ii,jj,q;
189:    ImMask=new long [MaskSize];

190:    for(jj=0;jj<NetSize;jj++)
191:    {
192:    uu=random(imWidth-WindowWidth);
193:    vv=random(imHeight-WindowHeight);
194:    for(ii=0;ii<(PositivPoint+NegativPoint);ii++)
195:    {
196:    xx=random(WindowWidth)+uu;
197:    yy=random(WindowHeight)+vv;
198:    q=yy*imWidth + xx;
199:    ImMask[jj*(PositivPoint+NegativPoint)+ii]=q;
200:    }
201:    }
202:    fp=fopen("maska.dat","wb");
203:    if(fp==NULL)
204:    { Form1->Button1->Caption="Cannot open file";
205:    terminate();}

206:    for(ii=0;ii<MaskSize;ii++)
207:    {
208:    c[0]=ImMask[ii];
```

```
209:    fwrite(c,sizeof(long),1,fp);
210:    }

211:    fclose(fp);
212:    delete ImMask;
213:    }
214:    //---------------------------------------------------------------------

215:    void MaskTest(void)
216:    {
217:    unsigned m,s;
218:    long k[1];
219:    unsigned* check=new unsigned[imWidth*imHeight];
220:    long int ii,jj;
221:    ImMask=new long [MaskSize];

222:    s=0;
223:    for(ii=0;ii<imWidth*imHeight;ii++)
224:    check[ii]=0;

225:    fp=fopen("maska.dat","rb");
226:    if(fp==NULL)
227:    { Form1->Button1->Caption="Cannot open file maska.dat";
228:    terminate();}

229:    for(ii=0;ii<MaskSize;ii++)
230:    {
231:    fread(k,sizeof(long),1,fp);
232:    ImMask[ii]=k[0];
233:    }
234:    fclose(fp);

235:    for(ii=0;ii<MaskSize;ii++)
236:    {
237:    m=ImMask[ii];
238:    check[m]++;
239:    }
240:    check[0]=0;
241:    //Normalización de la imagen
242:    for(ii=0;ii<imHeight*imWidth;ii++)
243:    {
244:    if (check[ii]>s)
245:    s=check[ii];
246:    }

247:    for(ii=0;ii<imHeight*imWidth;ii++)
248:    {
249:    m=(check[ii]*255)/s;
250:    check[ii]=m;
251:    }

252:    for(ii=0;ii<imHeight;ii++)
253:    for(jj=0;jj<imWidth;jj++)
254:    {
255:    m=check[ii*imWidth+jj]+(check[ii*imWidth+jj]<<8)+(check[ii*imWidth+jj]<<16);
256:    Form1->Canvas->Pixels[jj+300][ii+150]=(TColor)m;
257:    }
258:    delete check;
259:    delete ImMask;
260:    }
```

```
261:    //---------------------------------------------------------------------------

262:    void __fastcall TForm1::empty(TObject *Sender)
263:    {
264:    int i=1;
265:    }
266:    //---------------------------------------------------------------------------

267:    void __fastcall TForm1::Coding_beg(TObject *Sender)
268:    {
269:    long int iii;
270:    unsigned k[1];
271:    Form1->Button2->Caption="Coding_beg";
272:    iNum=1;
273:    Dist=0;
274:    ImMask=new long [MaskSize];
275:    BinaryImage=new unsigned char[imWidth*imHeight];
276:    BinaryImage2=new unsigned char[imWidth*imHeight];

277:    Form1->Timer1->Enabled=true;
278:    ActiveNeuron=fopen("actn.dat","wb");
279:    if(ActiveNeuron==NULL)
280:    { Form1->Button1->Caption="Cannot open file of active neurons";
281:    Form1->Timer1->Enabled=false;}
282:    fclose(ActiveNeuron);

283:    nsum=fopen("sumN.dat","wb");
284:    if(nsum==NULL)
285:    { Form1->Button1->Caption="Cannot open file of sums";
286:    Form1->Timer1->Enabled=false;}
287:    fclose(nsum);

288:    fp=fopen("maska.dat","rb");
289:    if(fp==NULL)
290:    { Form1->Caption="Can't open maska.dat";
291:    Form1->Timer1->Enabled=false;}

292:    for(iii=0;iii<MaskSize;iii++)
293:    {
294:    fread(k,sizeof(long),1,fp);
295:    ImMask[iii]=k[0];
296:    }
297:    fclose(fp);

298:    CodeGeneration();
299:    }
300:    //---------------------------------------------------------------------------

301:    void __fastcall TForm1::Coding_cont(TObject *Sender)
302:    {
303:    long int iii;
304:    unsigned k[1];
305:    ImMask=new long [MaskSize];
306:    BinaryImage=new unsigned char[imWidth*imHeight];
307:    BinaryImage2=new unsigned char[imWidth*imHeight];
308:    FILE* servis;

309:    Form1->Button2->Caption="Coding_cont";
310:    Form1->Timer1->Enabled=true;
```

```
311:    servis=fopen("serv.dat","rb");
312:    fread(serv,sizeof(long),2,servis);
313:    fclose(servis);

314:    fp=fopen("maska.dat","rb");
315:    if(fp==NULL)
316:    { Form1->Caption="Can't open maska.dat";
317:    Form1->Timer1->Enabled=false;}

318:    for(iii=0;iii<MaskSize;iii++)
319:    {
320:    fread(k,sizeof(long),1,fp);
321:    ImMask[iii]=k[0];
322:    }
323:    fclose(fp);
324:    iNum=serv[0];
325:    Dist=0;

326:    Form1->Timer1->Enabled=true;

327:    CodeGeneration();
328:    }
329:    //---------------------------------------------------------------------------

330:    void CodeGeneration(void)
331:    {
332:    unsigned k[1];
333:    long int jjj,sum;

334:    strips=fopen("LargeSet.dat","rb");
335:    if(strips==NULL)
336:    { Form1->Button1->Caption="Cannot open file LargeSet";
337:    terminate();}

338:    ImRead(iNum);
339:    Form1->Canvas->TextOutA(30,150,AnsiString(iNum));

340:    for(dist=Dist;dist<DISTNUM;dist++)
341:    {
342:    Distortion();
343:    sum=Coding();

344:    Form1->Canvas->TextOutA(30,200,AnsiString(sum));

345:    nsum=fopen("sumN.dat","ab");
346:    if(nsum==NULL)
347:    { Form1->Button1->Caption="Cannot open file of sums";
348:    Form1->Timer1->Enabled=false;}
349:    {
350:    k[0]=sum;
351:    if(fwrite(k,sizeof(long),1,nsum)==NULL)
352:    {Form1->Button1->Caption="Cannot write to file sums";
353:    Form1->Timer1->Enabled=false;
354:    }
355:    }
356:    fclose(nsum);

357:    ActiveNeuron=fopen("actn.dat","ab");
358:    if(ActiveNeuron==NULL)
```

```
359:    { Form1->Button1->Caption="Cannot open file of active neurons";
360:    Form1->Timer1->Enabled=false;}

361:    for(jjj=0; jjj<sum; jjj++)
362:    {
363:    k[0]=ActivNeuronNumber[jjj];
364:    if(fwrite(k,sizeof(long),1,ActiveNeuron)==NULL)
365:    { Form1->Button1->Caption="Cannot write to file of active neurons";
366:    Form1->Timer1->Enabled=false;}
367:    }
368:    fclose(ActiveNeuron);

369:    }
370:    fclose(strips);

371:    iNum++;
372:    if(iNum>ImNum)
373:    {
374:    delete ImMask;
375:    delete BinaryImage;
376:    delete BinaryImage2;
377:    Form1->Timer1->Enabled=false;
378:    }

379:    }
380:    //---------------------------------------------------------------------------

381:    void ImRead(long int iNum)
382:    {
383:    long int i,j,block;
384:    unsigned char* wind=new unsigned char [imHeight*imWidth];
385:    int color;
386:    unsigned char kk[1];

387:    block=imHeight*imWidth;
388:    offset=(iNum-1)*(block + 3*sizeof(float));
389:    fseek(strips,offset,SEEK_SET);

390:    for(i=0; i<imHeight; i++)
391:    for(j=0; j<imWidth; j++)
392:    {
393:    fread(kk,sizeof(unsigned char),1,fp);
394:    wind[i*imWidth+j]=kk[0];
395:    }
396:    for(i=0;i<imHeight;i++)
397:    for(j=0;j<imWidth;j++)
398:    {
399:    color=wind[i*imWidth+j];
400:    color=color+(color<<8)+(color<<16);
401:    Form1->Canvas->Pixels[j+40][i+50]=(TColor)color*255;
402:    }

403:    for(i=0;i<imHeight*imWidth;i++)
404:    {
405:    BinaryImage[i]=wind[i];
406:    }
407:    delete wind;
408:    }

409:    //---------------------------------------------------------------------------
```

```
410:    long int Coding(void)
411:    {
412:    long int i;
413:    long int ii,jj,kk,q,p;
414:    long int sum1;

415:    q=PositivPoint+NegativPoint;
416:    i=0; sum1=1;

417:    for(ii=0;ii<MaskSize;ii=ii+q)
418:    {
419:    jj=ii;
420:    for(kk=0;kk<PositivPoint;kk++)
421:    {
422:    p=ImMask[jj++];
423:    if(BinaryImage2[p]==0) goto m1;
424:    }
425:    for(kk=0;kk<NegativPoint;kk++)
426:    {
427:    p=ImMask[jj++];
428:    if(BinaryImage2[p]!=0) goto m1;
429:    }

430:    ActivNeuronNumber[sum1++]=i;
431:    if(sum1>ActivSize-2) goto m2;
432:    m1: i++;
433:    }
434:    m2: ActivNeuronNumber[0]=sum1; //Número de neuronas activas de la imagen

435:    return(sum1);
436:    }

437:    //---------------------------------------------------------------------------

438:    void Distortion(void)
439:    {
440:    long int i,j;
441:    unsigned char* work1=new unsigned char [imWidth*imHeight];

442:    switch(dist)
443:    {
444:    case 0:
445:    {
446:    for(i=0;i<imHeight;i++)
447:    for(j=0;j<imWidth;j++)
448:    BinaryImage2[i*imWidth+j]=BinaryImage[i*imWidth+j];
449:    Ima();
450:    break;
451:    }
452:    case 1:
453:    {
454:    for(i=0;i<imHeight-delta;i++)
455:    for(j=0;j<imWidth;j++)
456:    BinaryImage2[i*imWidth+j]=BinaryImage[i*imWidth+j+delta];
457:    Ima();
458:    break;
459:    }
460:    case 2:
461:    {
```

```
462:    for(i=0;i<imHeight;i++)
463:    for(j=delta;j<imWidth;j++)
464:    BinaryImage2[i*imWidth+j]=BinaryImage[i*imWidth+j-delta];
465:    Ima();
466:    break;
467:    }
468:    case 3:
469:    {
470:    for(i=0;i<imHeight-delta;i++)
471:    for(j=0;j<imWidth;j++)
472:    BinaryImage2[i*imWidth+j]=BinaryImage[(i+delta)*imWidth+j];
473:    Ima();
474:    break;
475:    }
476:    case 4:
477:    {
478:    for(i=delta;i<imHeight;i++)
479:    for(j=0;j<imWidth;j++)
480:    BinaryImage2[i*imWidth+j]=BinaryImage[(i-delta)*imWidth+j];
481:    Ima();
482:    break;
483:    }
484:    case 5:
485:    {
486:    for(i=delta;i<imHeight;i++)
487:    for(j=0;j<imWidth-1;j++)
488:    BinaryImage2[i*imWidth+j]=BinaryImage[(i-delta)*imWidth+j-delta];
489:    Ima();
490:    break;
491:    }
492:    case 6:
493:    {
494:    for(i=delta;i<imHeight;i++)
495:    for(j=0;j<imWidth-delta;j++)
496:    BinaryImage2[i*imWidth+j]=BinaryImage[(i-delta)*imWidth+j+delta];
497:    Ima();
498:    break;
499:    }
500:    case 7:
501:    {
502:    for(i=0;i<imHeight-delta;i++)
503:    for(j=delta;j<imWidth;j++)
504:    BinaryImage2[i*imWidth+j]=BinaryImage[(i+delta)*imWidth+j-delta];
505:    Ima();
506:    break;
507:    }
508:    case 8:
509:    {
510:    for(i=0;i<imHeight-delta;i++)
511:    for(j=0;j<imWidth-delta;j++)
512:    BinaryImage2[i*imWidth+j]=BinaryImage[(i+delta)*imWidth+j+delta];
513:    Ima();
514:    break;
515:    }
516:    case 9:
517:    {
518:    for(i=0;i<imHeight-2*delta;i++)
519:    for(j=0;j<imWidth;j++)
520:    BinaryImage2[i*imWidth+j]=BinaryImage[i*imWidth+j+2*delta];
521:    Ima();
```

```
522:    break;
523:    }
524:    case 10:
525:    {
526:    for(i=0;i<imHeight;i++)
527:    for(j=2;j<imWidth;j++)
528:    BinaryImage2[i*imWidth+j]=BinaryImage[i*imWidth+j-2*delta];
529:    Ima();
530:    break;
531:    }
532:    case 11:
533:    {
534:    for(i=0;i<imHeight-2*delta;i++)
535:    for(j=0;j<imWidth;j++)
536:    BinaryImage2[i*imWidth+j]=BinaryImage[(i+2*delta)*imWidth+j];
537:    Ima();
538:    break;
539:    }
540:    case 12:
541:    {
542:    for(i=2*delta;i<imHeight;i++)
543:    for(j=0;j<imWidth;j++)
544:    BinaryImage2[i*imWidth+j]=BinaryImage[(i-2*delta)*imWidth+j];
545:    Ima();
546:    break;
547:    }

548:    }
549:    delete work1;
550:    }

551:    //---------------------------------------------------------------------

552:    void Ima(void)
553:    {
554:    int i,j;
555:    long unsigned color;

556:    for(i=0;i<imHeight;i++)
557:    for(j=0;j<imWidth;j++)
558:    {
559:    color=BinaryImage[i*imWidth+j];
560:    color=(color<<8)+BinaryImage2[i*imWidth+j];
561:    color=color*255;
562:    Form1->Canvas->Pixels[j+200][i+50]=(TColor)color;
563:    }
564:    }

565:    //---------------------------------------------------------------------------

566:    void train(void)
567:    {
568:    long int jj,sum0,sum1,im,j;
569:    unsigned k[1];
570:    float q[3];

571:    for(dist=Dist;dist<DISTNUM;dist++)
572:    {
573:    sum1=0;
574:    j=(iNum-1)*DISTNUM+dist;
```

```
575:    if(j==0)j=1;
576:    sum0=SUMNumber[j];
577:    for(jj=1;jj<=j;jj++)
578:    {
579:    sum1=sum1+SUMNumber[jj];
580:    }
581:    ActiveNeuron=fopen("actn.dat","rb");
582:    if(ActiveNeuron==NULL)
583:    { Form1->Button1->Caption="Cannot open file of active neurons";
584:    Form1->Timer2->Enabled=false;}

585:    offset1=sum1*sizeof(long);
586:    fseek(ActiveNeuron,offset1,SEEK_SET);

587:    for(jj=1; jj<sum0; jj++)
588:    {
589:    if(fread(k,sizeof(long),1,ActiveNeuron)==NULL)
590:    { Form1->Button1->Caption="Cannot read file of active neurons";
591:    Form1->Timer2->Enabled=false;}
592:    ActivNeuronNumber[jj]=k[0];
593:    }
594:    fclose(ActiveNeuron);

595:    mark=fopen("LargeSet.dat","rb");
596:    if(mark==NULL)
597:    { Form1->Button1->Caption="Cannot open LargeSet with marks";
598:    Form1->Timer2->Enabled=false;}

599:    im=((iNum-1)*(imHeight*imWidth+3*sizeof(float)))+(imHeight*imWidth);
600:    fseek(mark,im,SEEK_SET);
601:    fread(q, sizeof(float),3,mark);
602:    x=q[0];
603:    y=q[1];
604:    z=q[2];
605:    fclose(mark);

606:    //Funciones para definicion de clases, usando coordenadas
607:    ClNameX();
608:    ClNameY();
609:    //Funciones para el reconocimiento durante el entrenamiento
610:    rec=recognition_tr();
611:    rec2=recognition_trY();
612:    //Comparacion entre la clase reconocida y la verdadera
613:    if(rec!=true1)
614:    {
615:    ErrorNumber++;
616:    training();  //Entrenamiento
617:    }
618:    if(rec2!=true2)
619:    {
620:    ErrorNumber2++;
621:    trainingY(); //Entrenamiento
622:    }
623:    }
624:    Form1->Canvas->TextOut(1,180,AnsiString(ErrorNumber)+"      ");
625:    Form1->Canvas->TextOut(1,200,AnsiString(ErrorNumber2)+"      ");
626:    Form1->Canvas->TextOut(1,220,AnsiString(iNum)+"      ");

627:    }
```

```
628:    //-----------------------------------------------

629:    unsigned short recognition_tr(void)
630:    {
631:    long int j,k,Active;
632:    unsigned short i,rec;
633:    long int jj,max;
634:    long int* sum=new long int[ClassSize];
635:    float f1,f2;

636:    j=(iNum-1)*DISTNUM+dist;
637:    if(j==0)j=1;
638:    Active=SUMNumber[j];

639:    for(i=0;i<ClassSize;i++)  sum[i]=0;
640:    for(j=1;j<Active;j++)
641:    {
642:    jj=(long)ActivNeuronNumber[j]*ClassSize;
643:    for(k=0;k<ClassSize;k++)
644:    sum[k]+=mtrX[jj++];
645:    }
646:    f1=(float)(sum[true1]);
647:    f2=f1*(1-TDS);
648:    sum[true1]=(long int)f2;
649:    max=0;
650:    rec=0;
651:    for(i=0;i<ClassSize;i++)
652:    if(sum[i]>max)
653:    {
654:    max=sum[i];
655:    rec=i;
656:    }

657:    delete sum;
658:    return(rec);
659:    }
660:    //-----------------------------------------------

661:    unsigned short recognition_trY(void)
662:    {
663:    long int j,k,Active;
664:    unsigned short i,rec2;
665:    long int jj,max;
666:    long int* sum=new long int[ClassSize];
667:    float f1,f2;

668:    j=(iNum-1)*DISTNUM+dist;
669:    if(j==0)j=1;
670:    Active=SUMNumber[j];

671:    for(i=0;i<ClassSize;i++)  sum[i]=0;

672:    for(j=1;j<Active;j++)
673:    {
674:    jj=(long)ActivNeuronNumber[j]*ClassSize;
675:    for(k=0;k<ClassSize;k++)
676:    sum[k]+=mtrY[jj++];
677:    }
678:    f1=(float)(sum[true2]);
679:    f2=f1*(1-TDS);
```

```
680:    sum[true2]=(long int)f2;
681:    max=0;
682:    rec2=0;
683:    for(i=0;i<ClassSize;i++)
684:    if(sum[i]>max)
685:    {
686:    max=sum[i];
687:    rec2=i;
688:    }

689:    delete sum;
690:    return(rec2);
691:    }
692:    //-----------------------------------------

693:    unsigned short recognitionX(void)
694:    {
695:    long int j,k,Active;
696:    unsigned short i,rec;
697:    long int jj,max;
698:    long int* sum=new long int[ClassSize];

699:    Active=ActivNeuronNumber[0];//Suma total de neuronas activas de la imagen

700:    for(i=0;i<ClassSize;i++)  sum[i]=0;

701:    for(j=1;j<Active;j++)
702:    {
703:    jj=(long)ActivNeuronNumber[j]*ClassSize;
704:    for(k=0;k<ClassSize;k++)
705:    sum[k]+=mtrX[jj++];
706:    }
707:    max=0;
708:    rec=0;
709:    for(i=0;i<ClassSize;i++)
710:    if(sum[i]>max)
711:    {
712:    max=sum[i];
713:    rec=i;
714:    }

715:    delete sum;
716:    return(rec);
717:    }
718:    //-----------------------------------------

719:    unsigned short recognitionY(void)
720:    {
721:    long int j,k,Active;
722:    unsigned short i,rec2;
723:    long int jj,max;
724:    long int* sum=new long int[ClassSize];

725:    Active=ActivNeuronNumber[0];//Suma total de neuronas activas de la imagen
726:    for(i=0;i<ClassSize;i++)  sum[i]=0;

727:    for(j=1;j<Active;j++)
728:    {
729:    jj=(long)ActivNeuronNumber[j]*ClassSize;
730:    for(k=0;k<ClassSize;k++)
```

```
731:    sum[k]+=mtrY[jj++];
732:    }
733:    max=0;
734:    rec2=0;
735:    for(i=0;i<ClassSize;i++)
736:    if(sum[i]>max)
737:    {
738:    max=sum[i];
739:    rec2=i;
740:    }

741:    delete sum;
742:    return(rec2);
743:    }
744:    //-------------------------------------------

745:    void training(void)
746:    {
747:    long int j,ii,jj, sum;
748:    unsigned short c1;

749:    j=(iNum-1)*DISTNUM+dist;
750:    if(j==0)j=1;
751:    sum=SUMNumber[j];
752:    for(j=1;j<sum;j++)
753:    {
754:    ii=ActivNeuronNumber[j]*ClassSize;
755:    jj=ii+true1;
756:    c1=mtrX[jj];
757:    if(c1<64000) c1++;
758:    mtrX[jj]=c1;
759:    jj=ii+rec;
760:    c1=mtrX[jj];
761:    if(c1>0) c1--;
762:    mtrX[jj]=c1;
763:    }
764:    }

765:    //-----------------------------------------------------------------------

766:    void trainingY(void)
767:    {
768:    long int j,ii,jj, sum;
769:    unsigned short c1;

770:    j=(iNum-1)*DISTNUM+dist;
771:    if(j==0)j=1;
772:    sum=SUMNumber[j];
773:    for(j=1;j<sum;j++)
774:    {
775:    ii=ActivNeuronNumber[j]*ClassSize;
776:    jj=ii+true2;
777:    c1=mtrY[jj];
778:    if(c1<64000) c1++;
779:    mtrY[jj]=c1;
780:    jj=ii+rec2;
781:    c1=mtrY[jj];
782:    if(c1>0) c1--;
783:    mtrY[jj]=c1;
784:    }
```

```
785:    }

786:    //---------------------------------------------------------------------------

787:    void __fastcall TForm1::Train_beg(TObject *Sender)
788:    {
789:    unsigned int cc,xx;
790:    long int ii,jj;
791:    unsigned k[1];
792:    unsigned short kkk[1];
793:    SetFormation=new char[ImNum+1];
794:    mtrX=new unsigned short [MatSizeX];
795:    mtrY=new unsigned short [MatSizeY];

796:    Form1->Button2->Caption="train_beg";
797:    Form1->Timer2->Enabled=true;
798:    Form1->Timer1->Enabled=false;

799:    cc=1;
800:    xx=1;
801:    pass=0;
802:    ErrorNumber=0;
803:    ErrorNumber2=0;
804:    Dist=0;
805:    iNum=1;
806:    offset=0;
807:    offset1=0;

808:    nsum=fopen("sumN.dat","rb"); //active neuron sum for all images
809:    if(nsum==NULL)
810:    { Form1->Button1->Caption="Cannot open file of sums";
811:    Form1->Timer1->Enabled=false;}
812:    {
813:    for(ii=1; ii<ImNum*DISTNUM; ii++)
814:    {
815:    if(fread(k,sizeof(long),1,nsum)==NULL)
816:    {Form1->Button1->Caption="Cannot write to file sums";
817:    Form1->Timer1->Enabled=false;}
818:    SUMNumber[ii]=k[0];
819:    }
820:    }
821:    fclose(nsum);

822:    fp=fopen("KeySet.dat","rb");
823:    if(fp==NULL)
824:    { Form1->Button1->Caption="Cannot open KeySet with marks";
825:    Form1->Timer2->Enabled=false;}

826:    fread(SetFormation,sizeof(unsigned char),ImNum+1,fp);
827:    fclose(fp);

828:    matrX=fopen("matrX.dat","wb");
829:    if(matrX==NULL)
830:    {Form1->Button1->Caption="Cannot open matrX";
831:    Form1->Timer2->Enabled=false;}
832:    fclose(matrX);

833:    matrY=fopen("matrY.dat","wb");
834:    if(matrY==NULL)
835:    {Form1->Button1->Caption="Cannot open matrY";
```

```
836:    Form1->Timer2->Enabled=false;}
837:    fclose(matrY);

838:    errors=fopen("errors.dat","wt");
839:    if(errors==NULL)
840:    {Form1->Button1->Caption="Cannot open errors.dat";
841:    Form1->Timer2->Enabled=false;}
842:    fclose(errors);

843:    for(ii=0;ii<MatSizeX;ii++)
844:    mtrX[ii]=0;  //limpiar

845:    for(ii=0;ii<MatSizeY;ii++)
846:    mtrY[ii]=0;  //limpiar

847:    m:     if(SetFormation[cc]==1)
848:    {
849:    iNum=cc;
850:    train();
851:    xx++;
852:    }
853:    if (xx>NumTr) goto m2;
854:    cc++;
855:    Form1->Canvas->TextOutA(100,100,"cc=    "+AnsiString(cc)+"   ");
856:    if (cc<=ImNum)  goto m;

857:    m2:    iNum=1;
858:    xx=1;
859:    offset=0;
860:    ErrorNumber=0;
861:    ErrorNumber2=0;

862:    Errors[pass]=ErrorNumber;
863:    Errors2[pass]=ErrorNumber2;

864:    Form1->Canvas->TextOutA(10,20,AnsiString(ErrorNumber)+"       ");
865:    Form1->Canvas->TextOutA(10,50,AnsiString(pass));

866:    matrX=fopen("matrX.dat","wb");
867:    if(matrX==NULL)
868:    {Form1->Button1->Caption="Cannot open matrX.dat";
869:    Form1->Timer2->Enabled=false;}

870:    for(jj=0; jj<MatSizeX; jj++)
871:    {
872:    kkk[0]=mtrX[jj];
873:    if(fwrite(kkk,sizeof(unsigned short),1,matrX)==NULL)
874:    { Form1->Button1->Caption="Cannot open matrX.dat";
875:    Form1->Timer2->Enabled=false;}
876:    }
877:    fclose(matrX);

878:    matrY=fopen("matrY.dat","wb");
879:    if(matrY==NULL){Form1->Button1->Caption="Cannot open matrY.dat";
880:    Form1->Timer2->Enabled=false;}

881:    for(jj=0; jj<MatSizeY; jj++)
882:    {
883:    kkk[0]=mtrY[jj];
884:    if(fwrite(kkk,sizeof(unsigned short),1,matrY)==NULL)
```

```
885:    { Form1->Button1->Caption="Cannot open matrY.dat";
886:    Form1->Timer2->Enabled=false;}
887:    }
888:    fclose(matrY);

889:    pass++;
890:    if(pass>=PassN)
891:    {
892:    errors=fopen("errors.dat","ab");
893:    if(errors==NULL)
894:    {Form1->Button1->Caption="Cannot open errors.dat";
895:    Form1->Timer2->Enabled=false;}
896:    for(jj=0;jj<pass;jj++)
897:    {
898:    k[0]=Errors[jj];
899:    fwrite(k,sizeof(unsigned long),1,errors);
900:    }
901:    for(jj=0;jj<pass;jj++)
902:    {
903:    k[0]=Errors2[jj];
904:    fwrite(k,sizeof(unsigned long),1,errors);
905:    }
906:    fclose(errors);

907:    Form1->Timer2->Enabled=false;
908:    delete mtrX;delete mtrY;
909:    delete SetFormation;
910:    }
911:    else {cc=1; xx=1; goto m;};

912:    }
913:    //---------------------------------------------------------------------------

914:    void __fastcall TForm1::Train_cont(TObject *Sender)
915:    {
916:    long int jj;
917:    long int cc;
918:    unsigned short kkk[1];
919:    SetFormation=new char[ImNum];
920:    mtrX=new unsigned short [MatSizeX];
921:    mtrY=new unsigned short [MatSizeY];

922:    Dist=0;

923:    FILE* servis;

924:    Form1->Button2->Caption="train_cont";
925:    Form1->Timer2->Enabled=true;
926:    Form1->Timer1->Enabled=false;

927:    matrX=fopen("matrX.dat","rb");
928:    for(jj=0; jj<MatSizeX; jj++)
929:    {
930:    if(fread(kkk,sizeof(unsigned short),1,matrX)==NULL)
931:    { Form1->Button1->Caption="Cannot open matrX.dat";
932:    Form1->Timer2->Enabled=false;}
933:    mtrX[jj]= kkk[0];
934:    }
935:    fclose(matrX);
```

```
936:    matrY=fopen("matrY.dat","rb");
937:    for(jj=0; jj<MatSizeY; jj++)
938:    {
939:    if(fread(kkk,sizeof(unsigned short),1,matrY)==NULL)
940:    { Form1->Button1->Caption="Cannot open matrY.dat";
941:    Form1->Timer2->Enabled=false;}
942:    mtrY[jj]= kkk[0];
943:    }
944:    fclose(matrY);

945:    fp=fopen("KeySet","rb");
946:    if(fp==NULL)
947:    { Form1->Button1->Caption="Cannot open KeySet with marks";
948:    Form1->Timer2->Enabled=false;}

949:    fread(SetFormation,sizeof(unsigned char),ImNum,fp);
950:    fclose(fp);

951:    servis=fopen("serv.dat","rb");
952:    fread(serv,sizeof(long),3,servis);
953:    fclose(servis);
954:    iNum=serv[0]; // iNum con "1" en KeySet
955:    cc=iNum;
956:    pass=serv[1];
957:    offset=serv[2];

958:    m1:   if(SetFormation[cc]==1)
959:    {
960:    iNum=cc;
961:    train();
962:    }
963:    cc++;
964:    if (cc<ImNum)goto m1;
965:    delete SetFormation;
966:    }
967:    //---------------------------------------------------------------------------

968:    void __fastcall TForm1::recognize(TObject *Sender)
969:    {
970:    long int i,ii,jj,sum,im;
971:    int ErrX[1000];
972:    int ErrY[1000];
973:    int dxx,dyy;
974:    unsigned k[1];
975:    unsigned short kkk[1];
976:    mtrX=new unsigned short [MatSizeX];
977:    mtrY=new unsigned short [MatSizeY];
978:    float q[3];
979:    SetFormation=new char[ImNum];
980:    BinaryImage=new unsigned char[imWidth*imHeight];
981:    BinaryImage2=new unsigned char[imWidth*imHeight];
982:    ImMask=new long [MaskSize];

983:    iNum=1;
984:    ErrorNumber=0;
985:    ErrorNumber2=0;

986:    for(i=0;i<ClassSize;i++)
987:    {
988:    ErrX[i]=0;
```

```
989:    ErrY[i]=0;
990:    }

991:    matrX=fopen("matrX.dat","rb");
992:    for(jj=0; jj<MatSizeX; jj++)
993:    {
994:    if(fread(kkk,sizeof(unsigned short),1,matrX)==NULL)
995:    { Form1->Button1->Caption="Cannot open matrX.dat";
996:    exit(1);}
997:    mtrX[jj]=kkk[0];
998:    }
999:    fclose(matrX);

1000:   matrY=fopen("matrY.dat","rb");
1001:   for(jj=0; jj<MatSizeY; jj++)
1002:   {
1003:   if(fread(kkk,sizeof(unsigned short),1,matrY)==NULL)
1004:   { Form1->Button1->Caption="Cannot open matrY.dat";
1005:   exit(1);}
1006:   mtrY[jj]=kkk[0];
1007:   }
1008:   fclose(matrY);

1009:   fp=fopen("maska.dat","rb");
1010:   if(fp==NULL)
1011:   { Form1->Button1->Caption="Cannot open file maska.dat";
1012:   terminate();}

1013:   for(ii=0;ii<MaskSize;ii++)
1014:   {
1015:   fread(k,sizeof(long),1,fp);
1016:   ImMask[ii]=k[0];
1017:   }
1018:   fclose(fp);

1019:   remark=fopen("KeySet.dat","rb");
1020:   if(remark==NULL)
1021:   { Form1->Button1->Caption="Cannot open KeySet";
1022:   exit(1);}
1023:   fread(SetFormation,sizeof(char),ImNum,remark);
1024:   fclose(remark);

1025:   strips=fopen("LargeSet.dat","rb");
1026:   if(strips==NULL)
1027:   { Form1->Button1->Caption="Cannot open LargeSet.dat";
1028:   exit(1);}

1029:   cc1:if(SetFormation[iNum]==0)
1030:   {
1031:   ImRead(iNum);
1032:   for(ii=0;ii<imHeight;ii++)
1033:   for(jj=0;jj<imWidth;jj++)
1034:   BinaryImage2[ii*imWidth+jj]=BinaryImage[ii*imWidth+jj];

1035:   im=((iNum-1)*(imHeight*imWidth+3*sizeof(float)))+(imHeight*imWidth);
1036:   fseek(strips,im,SEEK_SET);
1037:   fread(q, sizeof(float),3,strips);
1038:   x=q[0];
1039:   y=q[1];
1040:   z=q[2];
```

```
1041:   ClNameX();
1042:   ClNameY();

1043:   sum=Coding();

1044:   rec=recognitionX();
1045:   rec2=recognitionY();

1046:   dxx=(int)rec-(int)true1;
1047:   dyy=(int)rec2-(int)true2;
1048:   if(dxx<0)dxx=-dxx;
1049:   if(dyy<0)dyy=-dyy;
1050:   ErrX[dxx]+=1;
1051:   ErrY[dyy]+=1;

1052:   if(rec!=true1)
1053:   {
1054:   ErrorNumber++;
1055:   }

1056:   if(rec2!=true2)
1057:   {
1058:   ErrorNumber2++;
1059:   }

1060:   Canvas->TextOutA(1,220,AnsiString(iNum));
1061:   Canvas->TextOutA(1,250,AnsiString(ErrorNumber));
1062:   Canvas->TextOutA(1,280,AnsiString(ErrorNumber2));
1063:   }
1064:   iNum++;
1065:   if(iNum>ImNum)goto end;
1066:   goto cc1;

1067:   end:  Form1->Canvas->TextOutA(1,10,AnsiString(ErrorNumber));
1068:   Form1->Canvas->TextOutA(1,30,AnsiString(ErrorNumber2));

1069:   for(i=0;i<ClassSize;i++)
1070:   {
1071:   Form1->Canvas->TextOutA(250,20+14*i,AnsiString(ErrX[i]));
1072:   Form1->Canvas->TextOutA(300,20+14*i,AnsiString(ErrY[i]));
1073:   }

1074:   fclose(strips);
1075:   delete BinaryImage;
1076:   delete BinaryImage2;
1077:   delete ImMask;
1078:   delete mtrX;
1079:   delete mtrY;
1080:   delete SetFormation;
1081:   }
1082:   //---------------------------------------------------------------------------

1083:   unsigned short ClNameX(void)
1084:   {
1085:   unsigned short cl;

1086:   cl=0;
1087:   if((x>=-1.05)&&(x<=-0.95))cl=0; if((x>-0.95)&&(x<=-0.85))cl=1;
1088:   if((x>-0.85)&&(x<=-0.75))cl=2;if((x>-0.75)&&(x<=-0.65))cl=3;
```

```
1089:   if((x>-0.65)&&(x<=-0.55))cl=4;if((x>-0.55)&&(x<=-0.45))cl=5;
1090:   if((x>-0.45)&&(x<=-0.35))cl=6;if((x>-0.35)&&(x<=-0.25))cl=7;
1091:   if((x>-0.25)&&(x<=-0.15))cl=8;if((x>-0.15)&&(x<=-0.05))cl=9;
1092:   if((x>-0.05)&&(x<=0.05))cl=10;if((x>0.05)&&(x<=0.15))cl=11;
1093:   if((x>0.15)&&(x<=0.25))cl=12;if((x>0.25)&&(x<=0.35))cl=13;
1094:   if((x>0.35)&&(x<=0.45))cl=14;if((x>0.45)&&(x<=0.55))cl=15;
1095:   if((x>0.55)&&(x<=0.65))cl=16;if((x>0.65)&&(x<=0.75))cl=17;
1096:   if((x>0.75)&&(x<=0.85))cl=18;if((x>0.85)&&(x<=0.95))cl=19;
1097:   if((x>0.95)&&(x<=1.05))cl=20;

1098:   true1=cl;
1099:   return(true1);
1100:   }
1101:   //---------------------------------------------------------------------------
1102:   unsigned short ClNameY(void)
1103:   {
1104:   unsigned short cl;

1105:   cl=0;
1106:   if((y>=-1.05)&&(y<=-0.95))cl=0; if((y>-0.95)&&(y<=-0.85))cl=1;
1107:   if((y>-0.85)&&(y<=-0.75))cl=2;if((y>-0.75)&&(y<=-0.65))cl=3;
1108:   if((y>-0.65)&&(y<=-0.55))cl=4;if((y>-0.55)&&(y<=-0.45))cl=5;
1109:   if((y>-0.45)&&(y<=-0.35))cl=6;if((y>-0.35)&&(y<=-0.25))cl=7;
1110:   if((y>-0.25)&&(y<=-0.15))cl=8;if((y>-0.15)&&(y<=-0.05))cl=9;
1111:   if((y>-0.05)&&(y<=0.05))cl=10;if((y>0.05)&&(y<=0.15))cl=11;
1112:   if((y>0.15)&&(y<=0.25))cl=12;if((y>0.25)&&(y<=0.35))cl=13;
1113:   if((y>0.35)&&(y<=0.45))cl=14;if((y>0.45)&&(y<=0.55))cl=15;
1114:   if((y>0.55)&&(y<=0.65))cl=16;if((y>0.65)&&(y<=0.75))cl=17;
1115:   if((y>0.75)&&(y<=0.85))cl=18;if((y>0.85)&&(y<=0.95))cl=19;
1116:   if((y>0.95)&&(y<=1.05))cl=20;

1117:   true2=cl;
1118:   return(true2);
1119:   }

1120:   //---------------------------------------------------------------------------
1121:   void __fastcall TForm1::stop(TObject *Sender)
1122:   {
1123:   FILE* servis;
1124:   long jj;
1125:   unsigned short kkk[1];

1126:   // coding begin y continue

1127:   if((Form1->Button2->Caption=="Coding_beg")||(Form1->Button2->Caption=="Coding_cont"))
1128:   {
1129:   servis=fopen("serv.dat","wb");
1130:   serv[0]=iNum;
1131:   fwrite(serv,sizeof(long),2,servis);
1132:   fclose(servis);
1133:   Form1->Timer1->Enabled=false;

1134:   delete ImMask;
1135:   delete BinaryImage;
1136:   delete BinaryImage2;
1137:   }

1138:   // training begin y continue

1139:   if((Form1->Button2->Caption=="train_beg")||(Form1->Button2->Caption=="train_cont"))
```

```
1140:   {
1141:   servis=fopen("serv.dat","wb");
1142:   serv[0]=iNum;
1143:   serv[1]=pass;
1144:   serv[2]=offset;
1145:   fwrite(serv,sizeof(long),3,servis);
1146:   fclose(servis);

1147:   matrX=fopen("matrX.dat","wb");
1148:   if(matrX==NULL)
1149:   {Form1->Button1->Caption="Cannot open matrX.dat";
1150:   Form1->Timer2->Enabled=false;}

1151:   for(jj=0; jj<MatSizeX; jj++)
1152:   {
1153:   kkk[0]=mtrX[jj];
1154:   if(fwrite(kkk,sizeof(unsigned short),1,matrX)==NULL)
1155:   { Form1->Button1->Caption="Cannot open matrX.dat";
1156:   Form1->Timer2->Enabled=false;}
1157:   }
1158:   fclose(matrX);

1159:   matrY=fopen("matrY.dat","wb");
1160:   if(matrY==NULL){Form1->Button1->Caption="Cannot open matrY.dat";
1161:   Form1->Timer2->Enabled=false;}

1162:   for(jj=0; jj<MatSizeY; jj++)
1163:   {
1164:   kkk[0]=mtrY[jj];
1165:   if(fwrite(kkk,sizeof(unsigned short),1,matrY)==NULL)
1166:   { Form1->Button1->Caption="Cannot open matrY.dat";
1167:   Form1->Timer2->Enabled=false;}
1168:   }
1169:   fclose(matrY);

1170:   Form1->Timer2->Enabled=false;
1171:   }

1172:   }
1173:   //---------------------------------------------------------------------------

1174:   void __fastcall TForm1::Ttt(TObject *Sender)
1175:   {
1176:   CodeGeneration();
1177:   }
1178:   //---------------------------------------------------------------------------

1179:   void __fastcall TForm1::Ttt2(TObject *Sender)
1180:   {
1181:   train();
1182:   }
1183:   //---------------------------------------------------------------------------
```

REFERENCIAS

1.- M.E. Zaghloul. Mems, Microsystems and Nanosystems. Proceedings of the 2002 7th IEEE International Workshop on Cellular Neural Networks and Their Applications, Washington USA, 22-24 jul 2002, pp. 512-514.

2.- E.Kussul, D.A.Rachkovskij, T.N.Baidyk and S.A.Talayev. Micromechanical Engineering: A Basis For the Low-Cost Manufacturing of Mechanical Microdevices Using Microequipment, J. Micromech, Microeng. 6 (1996), pp. 410-425.

3.- E.Westkamper. The Miniaturization of Components and Parts in Mechanical Engineering. Second International Workshop on Microfactories, Fribourg Switzerland, 9-10 Oct 2000, pp. 20-22.

4.- J.Angell, S.Terry, and P.Barth. Silicon Micromechanical Devices. Micromechanics and MEMS; Classic and Seminal Papers to 1990, Edited by William Trimmer, IEEE (Institute of Electrical and Electronics Engineers, Inc.) Press, 1997, ISBN 0-7803-1085-3, pp 38-49.

5.- K.Gabriel, J.Jarvis, W.Trimmer. Small Machines, Large Oportunities: A Report of the Emerging Field of Microdynamics. Micromechanics and MEMS; Classic and Seminal Papers to 1990, Edited by William Trimmer, IEEE (Institute of Electrical and Electronics Engineers, Inc.) Press, 1997, ISBN 0-7803-1085-3, pp. 117-144.

6.- E.Kussul, T.Baidyk, L.Ruiz, A.Caballero, G.Velasco. Development of Low-Cost Microequipment. Proceedings of Micomechatronics and Humand Science, Nagoya, Japan, 2002, pp. 125-134.

7.- T.Baidyk, E.Kussul., O.Makeyev, A.Caballero, L.Ruiz, G.Carrera, G.Velasco. Flat Image Recognition in the Process of Microdevice Assembly. Pattern Recognition Letters. Vol 25, 2004, pp. 107-118.

8.- E.Kussul, T.Baidyk, L.Ruiz, A.Caballero and L.Kasatkina. Development of Micromachine Tool Prototypes for Microfactories.J. Micromech Microeng. 12, 2002, pp. 795-812.

9.- S.Fatikow, A.Faizullin and J.Seyfried. Planning of a Microassembly Task in a Flexible Microrobot Cell. Proceedings of the 2000 IEEE, International conference on Robotics and Automation, San Francisco CA., USA, April 2000, pp. 1121-1126.

10.- E.Kussul, L.Ruiz, A.Caballero, L.Kasatkina, T.Baydyk. The Perspectives of Micromechanical Filtres Application for Fine Filtration of Liquids and Gases. Proc. of First Int. Conf. On Mechatronics and Robotics, Saint-Petesburg, May 29- Jun 2, 2000, Vol1, pp. 103-108.

11.- S. Fahlbusch, A. Shirinov and S. Fatikow. AFM-based Micro Force Sensor and Haptic Interface for a Nanohandling Robot. Proceedings of the IEEE/RSJ, Intl. Conference on Intelligent Robots and Systems, Lausane, Switzerland, oct 2002, pp. 1772-1777.

12.- S. Fatikow, J. Seyfried, St. Fahubusch, A. Buerkle, F. Schmoeckel. A Flexible Microrobot-Based Microassembly Station. Proceedings, ETFA '99. 1999 7th IEEE International Conference on Emerging Technologies and Factory Automation, Vol 1, University of Karlsruhe, oct 1999, pp. 397-406.

13.- T. Baidyk. Application of Flat Image Recognition Technique for Automation of Microdevice Production. Proceedings of the International Conference on Advance Intelligent Mechatronics. Italy 2002, pp. 488-494.

14.- J. Wu, R. Lee, C. W. De Silva. Intelligent 3-D Sensing in Automated Manufacturing Processes. Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Italy 2001, pp. 366-370.

15.- Lee, S.J., K. Kim, D. Kim, J.Park, G. Park. Recognizing and Tracking of 3-D Shaped Micro Parts Using Multiple Visions for Micromanipulation. IEEE International Symposium on Micromechatronics and Human Science, Seoul, Korea, 2001, pp. 203-210.

16.- David B. Fogel and Charles j. Robinson. Computational Intelligence. Ed Wiley-IEEE Press, ISBN: 0-471-27454-2, 2003, pp. 46

17.- F. Izaurieta y C. Saavedra. Redes Neuronales Artificiales. Departamento de Física, Universidad de Concepción, Concepción, Chile. http://www.uta.cl/revistas/charlas/volumen16/Indice/Ch-csaavedra.pdf.

18.- A. Kulkarni. Artificial Neural Networks for Image Understanding. John Wiley & Sons, Inc, New York, USA, ISBN:0442009216.

19.- G. Burel, F. Bernard, W. Venema. Vision Feedback for SMD Placement Using Neural Networks, IEEE Internat. Conf. on Robotics and Automation, 1995, pp. 1491-1496.

20.- E. Kussul, T. Baidyk. Neural Random Threshold Classifier in OCR Application. Proceedings of the second all-Ukrainian International Conference, Kiev, Ukraine, 1994, pp. 154-157.

21.- E. Kussul, T. Baidyk, L. Kasatkina, V. Lukovich. Rosenblat Perceptrons for Handwritten Digit Recognition. Proceedings of International Joint Conference on Neural Networks. Washington, USA, 2001, pp. 1516- 1520.

22.- E. Kussul, T. Baidyk. Improve Method of Handwritten Digit Recognition Tested on MNIST Database. 15 th International Conference on Vision Interface, Calgary Canada, 2002, pp. 192- 197.

23.- J. Nelson, B. Vikramaditya, G. Yang, T. Enikov. Microassembly of Electromagnetic Microactuators Through Self-Assembly. Second International Workshop on Microfactories, Fribourg, Switzerland, 2000, pp. 67-70.

24.- O. Tohyama, S. Maeda, K. Abe and M. Murayama. Visual Inspection Mechanism for Microfactory. Second International Workshop on Microfactories, Fribourg, Switzerland, 2000, pp. 23-25.

25.- M. Nienhaus, W. Fhifeld, Y. Ansel, U. Berg, F. Schmitz, M. Begeman. Strategies and New Developments for Automated Microassembly. Second International Workshop on Microfactories, Fribourg, Switzerland, 2000, pp. 59-62.

26.-Q. Zhou, P. Kallio y N. Koivo. Model-Based Handling and Planning in Micro Assembly. Second International Workshop on Microfactories, Fribourg, Switzerland, 2000, pp. 71- 74.

27.- H. Ogawa. Indispensable Technologies for Microassembly. Second International Workshop on Microfactories, Fribourg, Switzerland, 2000, pp. 103- 106.

28.-http://www.lania.mx/spanish/actividades/newsletters/1999-primavera-verano/aplicaciones.html.

29.- K. Fukushima. Use of Top-Down Signals for Restoring Partly Occluded Patterns. International Joint Conference on Neural Networks, Japan, 2003, pp. 17 – 22.

30.- S. Fatikow, J. Seyfried, St. Fahlbusch, A. Buerkle y F. Schmoeckel. Flexible Microrobots for a Microfactory. Second International Workshop on Microfactories, Fribourg, Switzerland, 2000, pp. 31-34.

31.- F. Mokhtarian: Silhouette-Based Object Recognition with Occlusion through Curvature Scale-Space. Proceedings of the 4th European Conference on Computer Vision, Lecture Notes in Computer Science, Vol 1064, 1996, pp. 566-578.

32.- WWW.PUE.UDLAP.MX/~TESIS/LEM/MENDIETA_D_D/CAPITULO2.PDF

33.- Kim, J.Y., H.S.Cho. A Vision Based Error-Corrective Algorithm for Flexible Parts Assembly. Proceedings of the IEEE International Symposium on Assembly and Task Planning, Portugal, 1999, pp. 205-210.

34.- Kim, J.Y., H.S.Cho. Visual Sensor-Based Measurement for Deformable Peg-in-Hole Tasks. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999. pp 567-572.

35.- W.S.McCulloch, W.a.Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, Vol 5, pp. 115-133.

36.- F.Rosenblatt. “The Perceptrón: A Probabilistic Model for Information Storage and Organization in the Brain”. Psychological Review. 1958. pp 386-408. Reimpreso en el texto “Neurocomputing” (J.Anderson). 1988. MIT press . pp 92-114.

37.- B.Widrow, M.Hoff. Adaptive Switching Circuits. *IRE Western Electric Show and Convention Record, Part 4*. August 23, 1960. pp 96-104.

38.- S.Grossberg. Studies of Mind and Brain: Neural Principles of Learning, perception, Development, Cognition and Motion Control. Ed Reidel press. Amsterdan 1982.

39.- M.Minsky, S.Papert. Perceptrons: An Introduction to Computational Geometry. 1969. MIT Press.

40.- J.Anderson, J.Silverstein, S.Ritz, R.Jones. Distinctive Features, Categorical Perception and Probabilistic Learning: some applications on a neural model. Psychological Review, 1977, pp. 413-451.

41.- K.Fukushima. Neocognitron: A Self Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics, 1980, pp. 193-202.

42.- J.Hopfield. Neural Networks and Physical Systems with Emergent Colective Computacional Abilities. Proceedings of the National Academy of Sciences, 1982, pp. 2554-2558.

43.- R.Hecht-Nielsen. Neurocomputing: Picking the Human Brain. IEEE Spectrum. March 1988, pp. 36-41.

44.- T.Kohonen. An Introduction to Neural Computing and Neural Networks, Vol 1, 1988, pp. 3-16.

45.- R.Kilera, V.Martinez. Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones. Madrid, 1995. Ed. Ra-Ma.

46.- B.Widrow, R.Winter. Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition. IEEE Computer, Marzo 1988. pp 25-39.

47.- B.Widrow, S.D.Stearns. Adaptive Signal Processing. Ed Prentice Hall, 1985.

48.- T.Sejnowski y C.Rosenberg. Parallel Networks that Let to Pronounce English Text. Complex Systems, 1987, pp. 145-168.

49.- C.Koch, J.Marroquin, A.Yuille. Analog Neural Networks for Early Vision. Proceedings of the National Academy of Sciences, 1986, pp. 4263-4267.

50.- S.Grossberg. How Does the Brain Build a Cognitive Code?. Psychological Review. 1980. pp 1-51. Reimpreso en el texto “Neurocomputing” (J.Anderson y E.Rosenfeld). MIT Press, 1988.

51.- T.Kohonen. Self-Organized Formation of Topological Correct Feature Maps. Biological Cybernetics. Vol 43. pp 59-69. Reimpreso en el texto “Neurocomputing” (J.Anderson y E.Rosenfeld). MIT Press, 1988.

52.- R.Hecht-Nielsen. Neurocomputing. Ed Addison-Wesley. 1990.

53.- L.Bottou, C.Cortes, J.Denker, H.Drucker, L.Guyon, L.Jackel, J.LeCun, U.Muller, E.Sackinger, P.Simard, V.Vapnik. Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition, Vol 2, 1994, pp. 77-82.