

Jurado asignado:

Presidente: Dr. López Parra Marcelo.

Secretario: Dr. Dorador González Jesús Manuel.

1^{er} Vocal: Dr. Kussul Ernst Mikhailovich.

1^{er} Suplente: Dr. Santillán Gutiérrez Saúl Daniel.

2^o Suplente: Dra. Baidyk Tatiana.

Lugar donde se realizó la tesis:

Laboratorio de Micromecánica y Mecatrónica (LMM) del
Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET)
de la Universidad Nacional Autónoma de México (UNAM).

Tutores de tesis:

Dr. Ernst Kussul.

Dra. Tatiana Baidyk.

Este trabajo se desarrolló en el Laboratorio de Micromecánica y Mecatrónica del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM) bajo la tutoría de los doctores Ernst Kussul y Tatiana Baidyk. Se contó con el apoyo de una beca para estudios doctorales del CONACYT y de una beca complemento DGEP - UNAM. Además se contó con el apoyo parcial de los siguientes proyectos: DGAPA PAPIIT IN118799, PAPIIT IN108606-3, PAPIIT IN116306-3, NSF-CONACYT 39395-U, DGAPA-PAPIIT IN112102 y CONACYT 33944-U.

El autor, sin perjuicio de la legislación de la Universidad Nacional Autónoma de México, otorga el permiso para el libre uso, reproducción y distribución de esta obra siempre que sea sin fines de lucro, se den los créditos correspondientes y no sea modificada, en especial esta nota.

D.R. ©Gengis Kanhg Toledo Ramírez, México, D.F. 2007.

A Susy y Coneneti;
mi familia
♡

Vivimos en una sociedad altamente dependiente de la ciencia y de la tecnología, en la cual casi nadie sabe nada sobre ciencia y tecnología.

Carl Sagan

Se ha vuelto espantosamente obvio que nuestra tecnología ha excedido a nuestra humanidad.

Albert Einstein

Las máquinas y las computadoras deberán volverse una parte funcional en un sistema social orientado por la vida y no un cáncer que empieza por hacer estragos y acaba por matar al sistema.

Erich Fromm

Índice general

Abstract	xv
1. Sistema de visión por computadora para microensamble	1
1.1. Investigación en el Laboratorio de Micromecánica y Mecatrónica	1
1.2. Sistema automático de manejo de piezas	3
1.3. Subsistema de visión técnica (SVT)	5
2. Clasificador Neuronal de Permutación de Códigos (PCNC)	9
2.1. Estructura	9
2.1.1. Extractor de propiedades	9
2.1.2. Codificador	10
2.1.2.1. Codificación de las propiedades	11
2.1.2.2. Ejemplo de permutaciones	12
2.1.2.3. Propiedades de las permutaciones	13
2.1.2.4. Vector código \mathbf{V}	13
2.1.2.5. Adelgazador dependiente de contexto (CDT)	13
2.1.3. Clasificador neuronal	14
2.2. Proceso de entrenamiento	14
2.3. Distorsiones	15
2.4. Discusión	15
3. Experimentos y resultados	29
3.1. LIRA	29
3.1.1. Experimentos preliminares	30
3.1.2. Unicidad	30
3.1.3. Distorsiones	33
3.1.4. Ciclos de entrenamiento	34
3.1.5. Aleatoriedad de los conjuntos para entrenamiento y prueba	34
3.1.6. Mejores clasificadores PCNC	35
3.1.6.1. Base de datos A	35
3.1.6.2. Base de datos B	35
3.1.7. Prueba con conjunto de entrenamiento ampliado	36
3.2. Comparación de clasificadores	36
3.2.1. Tiempos	36
3.2.2. Estabilidad en la creación de estructuras	37
3.2.3. Parámetros	37
3.2.4. Ciclos de entrenamiento	38
3.2.5. Confiabilidad	38
3.2.6. Resultados	38
3.3. Búsqueda y reconocimiento de posición	39
Conclusiones	43

Apéndices	45
A. Software	47
A.1. Módulo Optik	48
A.2. Módulo RNA	48
A.3. Módulo Localizador	48
A.4. Interfaz	49
A.5. Implementación de LIRA	49
A.6. Software de línea de comandos	50
A.6.1. lira2007	50
A.6.2. pcnc2007	52
A.6.3. Potencial para la experimentación	53
B. Publicaciones	63
Bibliografía	65

Índice de figuras

1.1. Máquinas de la primera generación MET.	3
1.2. Sistema automático de manejo de piezas.	5
1.3. Sistema de Visión Técnica.	6
1.4. Esquema de localización de una pieza reconocida.	7
1.5. Diversas piezas utilizadas en el desarrollo del SVT.	7
2.1. Estructura del PCNC.	17
2.2. Procesos del extractor de propiedades.	18
2.3. Selección de puntos específicos.	19
2.4. Extracción de propiedades.	20
2.5. Procesos del codificador.	21
2.6. Permutación y su representación.	22
2.7. Permutaciones X.	23
2.8. Permutaciones Y.	24
2.9. Permutaciones XY.	25
2.10. Permutaciones Q.	26
2.11. Clasificador neuronal LIRA dividido.	27
2.12. Distorsiones para ampliar el conjunto de entrenamiento.	27
3.1. Imágenes reconocidas en la BD-A por el PCNC.	36
3.2. Localización y reconocimiento de piezas.	40
3.3. Localización y reconocimiento de piezas con redundancia.	40
3.4. Localización y reconocimiento de piezas que se tocan.	41
A.1. Diagrama a bloques del software OptikRNA.	57
A.2. Interfaz gráfica de usuario (IGU) de Optik.	58
A.3. Clases principales del módulo RNA y su relación de herencia.	59
A.4. Interfaz gráfica de usuario del software RNA.	60
A.5. Diagrama a bloques del software OptikRNA.	60
A.6. IGU Rna. Localizador de piezas.	61
A.7. Clase neurona y sus clases derivadas.	62

Índice de tablas

1.1. Generaciones de microequipo de acuerdo a su escala de reducción.	2
3.1. Experimento preliminar para la sintonización de parámetros LIRA.	30
3.2. Experimento de unicidad con LIRA.	31
3.3. Experimento con el parámetro p	31
3.4. Experimento con el parámetro n	31
3.5. Experimento con los parámetro p y n	32
3.6. Mejor conjunto de parámetros LIRA.	32
3.7. Experimento con el parámetro D_c	32
3.8. Experimento con el parámetro q	33
3.9. Mejor conjunto de parámetros PCNC.	33
3.10. Experimento con distorsiones con el PCNC.	34
3.11. Experimento con diversos ciclos de entrenamiento.	34
3.12. Experimento con conjuntos aleatorios PCNC.	35
3.13. Resultado con los mejores clasificadores PCNC.	35
3.14. Tiempos empleados por los clasificadores.	37
3.15. Comparación de unicidad entre los clasificadores.	37
3.16. Confiabilidad de los clasificadores.	38
3.17. Resultados de los clasificadores sobre las bases de datos.	38

Research and development of a computer vision system for micro machiney and micro assembly applications

Doctoral Thesis

Abstract

Gengis Kanhg Toledo Ramírez

Faculty of Engineering - CCADET

National Autonomous University of Mexico

The aim of this thesis is the development of a computer vision system for work pieces recognition and position location. The objective of the system is to recognize and to locate one certain work piece randomly located in a work area, so this work piece can be manipulated. The system contributes to the automatization of the machining and the assembly processes within a micro factory. Although the system had born as an automatization requeriment for a microfactory, the system can be used in any work scale.

The developed system was born from the project that consists in the creation of fully automated and shelf-reproducing microfactories. This project started at the Micromechanics and Mechatronics Laboratory of the National Autonomous University of Mexico. One fundamental requirement of this project is to achieve performance and automation for its different processes. One way to reach such requirements is to use computer vision methods. Computer vision methods allow high flexibility in the systems. For micro manipulation and micro assembly tasks, it is necessary to have a recognition and location system for work pieces. Therefore, the detailed researches in this thesis have the objective of to develop such system allowing that these tasks will be done.

The main goal of the researches that were done was to develop one system able to accurately recognize and to locate with acceptable precision one work piece of certain type randomly located in a work area for manipulation. In order to achieve this goal two technical aspects require to be develop, one of them is the recognition of work pieces and the other one is the location of the position of such work pieces.

The main reach of the investigations has been to develop an off-line working system in experimental stage.

In order to manage the system develop the particular problematic was studied with special approach in the critical tasks. These tasks are the classification and the localization of work pieces. For these tasks it is fundamental to use technologies and algorithms from artificial intelligence. Two algorithms were selected, adapted and applied. These algorithms are based in artificial neural networks and code permutation paradigms. Both algorithms allowed very good performance in similar tasks. The algorithms are called neural classifier LIRA and Permutative Code Neural Classifier (PCNC). For the analysis and proves of the system, three different databases of images from work pieces were created. The used work pieces were screws of several types, nuts, washers and other common used work pieces found in an assembly line. The images were taken from source images that contain several ramdomly located work pieces. The data bases were made with six or seven types of work pieces. The images of the databases have real characteristics actually found in the industrial enviroment such as no special illumination condition, shines, shades and not uniform background. Two databases, which are called A and B, were made with images that have only one type of work piece at a time whereas the third one, which is called D, was made with images that contains all the types of work pieces turned around and heaped up.

For the selected algorithms several experiments for validation and testing were done as well as concrete experiments with practical characteristics.

The recognition rates reached for data bases A, B and D were, for the neuronal classifier LIRA 93.75 %, 94.14 % and 90.47 % respectively, whereas for the PCNC were 96.87 %, 97.80 % and 91.43 % respectively. The PCNC required double amount of time for image recognition with respect to the neuronal classifier LIRA.

For both used classifiers the system obtained good performance in the recognition task. The best result of almost 98 % gives us good perspective for the developed system. The performance for the data base D, with the existence of heaped up, turned round and partially occluded work pieces is encouraging.

About the work pieces location task the system is able to provide the coordinates of the required work piece. Nevertheless to improve its precision it is necessary to use larger processing time which reduces the efficiency of the proposed system. Thus, the searching and location method must be improved.

This work marks a novel research line on the computer vision problem for automation of microassembly and micromachinery. The automation of such processes means a fundamental stage in order to achieve the fully automatic microfactory.

Capítulo 1

Sistema de visión por computadora para microensamble

Este capítulo describe el sistema de visión por computadora para microensamble propuesto. Dicho sistema forma parte importante de la microfábrica en desarrollo en el laboratorio de investigación del autor. En el capítulo anterior se expuso la necesidad de que dicho sistema de visión sea desarrollado como parte de las investigaciones sobre microfábricas. Se explicó por qué un sistema de este tipo debe de estar basado en visión por computadora. El presente trabajo tiene como objetivo fundamental desarrollar el sistema automático de visión por computadora de reconocimiento de piezas para una microfábrica. Para explicar lo anterior, la siguiente Sección (1.1) introduce el laboratorio donde se ha desarrollado este trabajo dando cuenta de sus objetivos y líneas de investigación. En la Sección 1.2 se explica a detalle el sistema de manejo de piezas propuesto y su función dentro de la microfábrica en desarrollo. Por último, en la Sección 1.3 se describe el sistema de visión técnica para reconocimiento y localización de piezas que se propone.

1.1. Investigación en el Laboratorio de Micromecánica y Mecatrónica

Desde 1998, el Laboratorio de Micromecánica y Mecatrónica (LMM) del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM), encabezado por el Dr. Ernst Kussul, trabaja en el desarrollo de microfábricas y las tecnologías relacionadas. Diversos artículos especializados dan cuenta de la producción y los resultados alcanzados [1–6]. El LMM desde su constitución y hasta el momento ha tenido diversas líneas de investigación. Las principales son las siguientes:

- Desarrollo de microtecnología mecánica para aplicaciones en células de producción.
- Investigación y desarrollo de sistemas de control para micromáquinas herramientas y micro-manipuladores.
- Investigación de sistemas de control basados en redes neuronales con aplicaciones en sistemas micromecánicos.
- Investigación y desarrollo de un sistema de visión computacional para microequipo.
- Investigación y desarrollo de manipuladores de baja inercia.

Las bases de estas investigaciones y desarrollos fueron planteadas en [7]. La propuesta general se denomina Tecnología de Microequipo o MET (por sus siglas en inglés). Esta propuesta se ha introducido en la Sección ?? y se ha hecho de acuerdo a lo expuesto en [8]. Mediante el uso de

generaciones sucesivas de microequipo capaz de reproducirse así mismo, pero con dimensiones más pequeñas, se puede llegar en pocas generaciones a equipo de dimensiones micrométricas partiendo de equipo miniatura de unos cuantos centímetros de tamaño, véase Tabla 1.1.

Tabla 1.1: Generaciones de microequipo de acuerdo a su escala de reducción.

Generación	Escala de reducción (ER)		
	ER=2	ER=4	ER=8
1	*100 mm	*100 mm	*100 mm
2	50 mm	25 mm	12.5 mm
3	25 mm	6.25 mm	1.5 mm
4	12.5 mm	1.5 mm	0.2 mm
5	6 mm	0.4 mm	25 μ m
6	3 mm	0.1 mm	3 μ m
7	1.5 mm	25 μ m	\triangle
8	0.8 mm	6 μ m	—
9	0.4 mm	1.5 μ m	—
10	0.2 mm	\triangle	—
* Valor inicial igual para todos los casos.			
\triangle Límite teórico de micromaquinado.			

Además del objetivo principal de crear otra generación más pequeña para cada una de las generaciones MET, existen muchas otras aplicaciones diversas. Estas aplicaciones tienen que ver con la posibilidad de producción de piezas y equipos diversos de dimensiones correspondientes a cada generación MET.

Las principales áreas tecnológicas de aplicación son la medicina, la microelectrónica y la industria espacial. Las posibilidades son muy alentadoras ya que no sólo se tendrá la capacidad de producir piezas y equipos de dimensiones micrométricas, sino de cualquier dimensión intermedia entre lo micro y lo meso.

Sin embargo, para crear la primera generación MET existen diversos problemas de ingeniería que es necesario resolver. A medida que nuevas generaciones sean creadas nuevos problemas asociados a sus dimensiones deberán enfrentarse debido a la miniaturización progresiva. Con el objetivo general de desarrollar el microequipo en el LMM se llevan a cabo diversas investigaciones. Estas investigaciones tienen que ver con diseño mecánico, precisión mecánica, ingeniería térmica, resistencia de materiales, interfaces, sistemas mecatrónicos e inteligencia artificial entre otros. Todas estas investigaciones son necesarias para alcanzar los objetivos mencionados para el LMM.

Como avance concreto en el objetivo general del LMM se ha desarrollado y caracterizado un microcentro de maquinado con dimensiones de $130 \times 160 \times 85 \text{ mm}^3$. Este centro de maquinado ha permitido manufacturar objetos de entre $50 \mu\text{m}$ y 5 mm . Estas piezas tienen geometrías tridimensionales complejas, como es el caso de tornillos, engranes y ejes graduados que han sido producidos. Estas piezas producidas son de materiales diversos como acero, plásticos varios o latón [4]. Adicionalmente se han diseñado y construido otros prototipos de la primera generación MET como es el caso de un par de manipuladores. El centro de maquinado y los manipuladores se muestran en la Figura 1.1.

Los conceptos MET y microfábrica están ampliamente relacionadas en la perspectiva del LMM ya que la viabilidad de esta tecnología depende de la capacidad de construir una microfábrica altamente automatizada de primera generación. Esta microfábrica deberá ser capaz de producir las piezas y luego las máquinas de una siguiente generación más pequeña como se expuso en la Sec. ???. Con el objetivo de lograr la automatización y la flexibilidad de la microfábrica se ha argumentado que una función muy importante es la de un sistema automático de manejo de piezas (Sec. ???). En la siguiente sección se expone la propuesta de dicho sistema enmarcando sobre éste los objetivos del presente trabajo.



Figura 1.1: Máquinas de la primera generación MET. Izquierda, microcentro de maquinado. Centro, manipulador de topología común. Derecha, manipulador basado en paralelogramos. Un disco magnético de $3\frac{1}{4}$ " se ubica en la imagen para dar cuenta de las dimensiones.

1.2. Sistema automático de manejo de piezas

En una fábrica convencional los materiales y las piezas producidas pertenecen a un proceso masivo. Por lo general este proceso es poco o nada flexible durante el tiempo de producción. Esta forma de trabajo genera gran cantidad de productos de un mismo tipo siendo la forma dominante en el mundo desarrollado al día de hoy. El concepto de microfábrica no es solamente una fábrica de tamaño reducido, sino un nuevo concepto de producción [9]. En el concepto de producción de la microfábrica se busca hacer de la producción algo flexible. Lo anterior significa que los productos pueden variar en diseños y geometrías entre muchos otros aspectos en un mismo lote de producción. Lo anterior hace que el material suministrado así como los productos intermedios y terminados del proceso no sean fácilmente manejables mediante equipos rígidos especialmente adaptados a un sólo producto. Es por ello que un sistema automático de manejo de materiales y piezas debe ser desarrollado para la microfábrica. Independientemente de ello, un sistema así podrá ser utilizado además en las fábricas convencionales.

Diversas operaciones de una microfábrica requieren un sistema automático de manejo de materiales y piezas, entre las que sobresalen:

- El suministro y manejo de materia prima de estado sólido.

- La manipulación de los productos intermedios y finales.
- La manipulación de piezas y partes para el microensamble.
- El remplazo y cambio de herramientas.
- La manipulación y remplazo de partes para labores de mantenimiento.

Dada la amplia variedad de materiales y objetos a ser manejados y a las funciones requeridas, un sistema de tales características deberá ser altamente flexible y deberá contar con algunas capacidades atribuibles a la inteligencia, como es el caso de la identificación de texturas para diferenciar el material o la determinación de la posición de ciertos objetos aleatoriamente ordenados con miras en su manipulación.

El problema descrito se aborda en el presente trabajo proponiendo un sistema automático de manejo de piezas enfocado a la tarea de reconocimiento de piezas aleatoriamente distribuidas. Un sistema capaz de resolver esta compleja tarea será fundamental para el desarrollo de los diversos sistemas de manipulación de materiales y partes para una microfábrica.

La meta de un sistema automático de manejo de piezas es que sea capaz de reconocer una determinada pieza aleatoriamente ubicada en una cierta área de trabajo además de su posición y orientación respectiva, de tal forma que pueda ser tomada con un manipulador para ser movida a alguna otra posición en el espacio según los requerimientos del proceso respectivo. Un sistema que cumpla con las expectativas mencionadas debe ser flexible, es decir, debe ser capaz de manejar piezas de diversas formas y con múltiples dimensiones.

Como solución al problema planteado en el presente trabajo se tiene un Sistema Automático de Manipulación de Piezas (SAMP). Este sistema se ha propuesto para formar parte en una celda de ensamble en una microfábrica. El sistema propuesto se muestra en la Figura 1.2. El SAMP está compuesto por dos cámaras, un manipulador, un sistema de control del manipulador (SCM), un sistema inteligente de manipulación (SIM) y un sistema de visión técnica (SVT).

En congruencia con la MET, el diseño del SAMP utiliza componentes de bajo costo y de manufactura simple. La parte mecánica del sistema es el manipulador, el cuál debe ser diseñado con MET para permitir su consecuente decremento en dimensión, un diseño se ha propuesto en [3]. Las dos cámaras del diseño son cámaras *web* de bajo costo y baja resolución. Para la segunda y siguientes generaciones MET, estas cámaras deberán emplear lentes para reducir su área de enfoque sin deteriorar la calidad de imagen. El resto de los componentes, los tres subsistemas propuestos, son sistemas lógicos implementados mediante software y hardware. Estos subsistemas no requieren ser escalados en las sucesivas generaciones MET.

Cada uno de los tres subsistemas del SAMP tiene una función específica. El SIM es el control central del sistema de manipulación. Es el encargado de comunicarse con el exterior para recibir tareas y devolver información del estado de éstas así como del estado general del SAMP, también intercomunica los otros dos subsistemas. El SCM es la parte lógica del manipulador, es el responsable de convertir los requerimientos dados para el SIM en términos de coordenadas espaciales a el espacio de estados de la topología particular del manipulador así como realizar la planeación de trayectorias de éste. Convirtiendo los requerimientos en señales eléctricas para mover los actuadores del manipulador. El SCM también devuelve el estado del manipulador, tanto espacial (posición, orientación y estado del efector) como lógica (banderas de error, de éxito u otros estados). El SVT es la parte inteligente del SAMP, es el responsable de devolver una posición espacial ante el requerimiento de localizar cierto tipo de pieza en el área de trabajo. Esto es, dada la necesidad de manipular una o más piezas, el SIM pide al SVT buscar éstas en el área de trabajo y devolver la posición de cada una de ellas para que éstas puedan ser requeridas físicamente al manipulador mediante el SCM. El usuario del SAMP puede ser un humano u otro sistema, e.g. sistema de manufactura o centro de control de la microfábrica.

Existen dos tareas comunes a ser resueltas por el SAMP. Una es colocar un cierto tipo de pieza ubicada en el área de trabajo en una micromáquina herramienta, la otra tarea es realizar un ensamble sencillo utilizando dos piezas del área de trabajo. Para la primera tarea el SIM pedirá al SVT localizar la pieza solicitada, en caso de que exista alguna de estas piezas en el área de trabajo,



Figura 1.2: Sistema automático de manejo de piezas.

el SVT deberá devolver al SIM la ubicación espacial de la misma. El SIM enviará entonces esta posición y orientación al SCM junto con los datos de destino para la pieza. La posición destino puede ser conocida por el SIM o no, según ésta sea de una máquina previamente ubicada o no. La segunda tarea es más complicada ya que requiere que el manipulador tenga capacidad para ensamble, y de no conocerse la geometría exacta de las piezas a ensamblar, se requerirá de un sistema de visión por computadora basado en la propuesta dada en [10]. Esta segunda tarea requerirá además que la tarea particular de ensamble este previamente programada de modo que el manipulador pueda colocar la primera pieza en la posición y lugar correctos para posteriormente ensamblarle la segunda pieza. Una vez que se logre desarrollar un sistema capaz de realizar la segunda tarea, nada impide que éste pueda realizar ensambles con más piezas.

Dadas las características y requerimientos funcionales de cada uno de los sistemas que componen el SMAP, en el presente trabajo se ha desarrollado el SVT, ya que es el sistema que propone más retos y mayor dificultad, además de constituir una tarea que no está resuelta satisfactoriamente según se revisó en la Sección ??.

Dados los grandes retos que presenta este sistema, para realizar las investigaciones orientadas al desarrollo del SVT se procuró aislar el problema de los demás subsistemas y componentes del SAMP sin perder la problemática fundamental que se presenta. Por esto, el trabajo se hace sobre la parte del SAMP que contiene el SVT y una sola cámara además de su área de trabajo. Es decir, este trabajo está basado en la tarea de reconocimiento de piezas y sus posiciones en ambiente de microfábricas. En la siguiente sección final de este capítulo se aborda en detalle el SVT.

1.3. Subsistema de visión técnica (SVT)

En las Secciones ?? y ?? se han descrito los requerimientos generales para el Sistema de Visión Técnica (SVT) para una microfábrica. Se tiene que un SVT para localización de piezas debe ser

congruente con los principios de flexibilidad de una microfábrica. Para lo cual debe ser capaz de trabajar con diversos tipos de piezas, además adaptarse a nuevas piezas en poco tiempo por lo que debe minimizarse el tiempo de procesamiento previo requerido para el reconocimiento de nuevas piezas.

Como se explicó en la sección anterior, el propósito del SVT es reconocer objetos y la posición de éstos. Este sistema está compuesto de tres elementos además de la cámara. Estos elementos son una interfase, un localizador de objetos y un controlador para la cámara (véase Fig. 1.3). La Interfase es la unidad que se comunica con el exterior, el SAMP (Véase Fig. 1.2) además de intercomunicarse con los otros dos elementos del sistema. A través de la Interfase el SVT recibe ordenes y envía información. El localizador de objetos es el elemento principal del SVT, su función es identificar y localizar un objeto requerido en el campo de visión de la cámara, devolviendo sus coordenadas. El controlador de la cámara se encarga de capturar imágenes del área de trabajo cuándo así lo requiera el localizador de objetos.

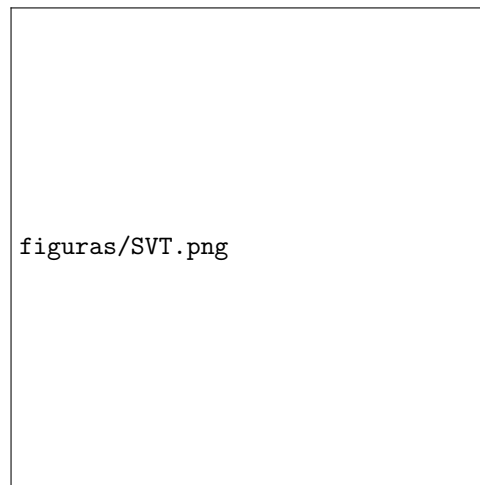


Figura 1.3: Sistema de Visión Técnica.

El objetivo es encontrar las coordenadas de las piezas con una precisión aceptable para su posterior manipulación. Esta tarea debe ser eficiente en cuanto al uso del tiempo y el uso de recursos (equipo, memoria y tiempo de procesador, etc.), de tal manera que el sistema pueda ser utilizado en línea de producción (*on-line*) además de ser económico. En la Fig. 1.4 se muestra el resultado esperado por el SVT ante la solicitud de localizar un tornillo dentro del área de trabajo. Se busca que el SVT localice el primer objeto del tipo requerido y devuelva su posición y su orientación. Lo ideal es que esto se haga respecto al centro de gravedad de la pieza. La posición y orientación deben tener precisión tal, de forma que permitan la manipulación de la pieza mediante un robot.

Para el desarrollo del SVT han sido hechas las siguientes consideraciones sobre las piezas a reconocer y a ubicar.

- Se encuentran en un área de trabajo igual al campo visual de la cámara.
- Se localizan en un mismo plano¹.
- Están aleatoriamente distribuidas y orientadas.
- Pueden estar juntas e incluso una sobre otra, este caso no es general pero fue considerado para los experimentos.

¹Esta consideración no es del todo cierta para una de las bases de datos utilizadas, ya que las piezas están amontonadas y por ende, en distintos planos; sin embargo, la distancia entre éstos es relativamente pequeña.

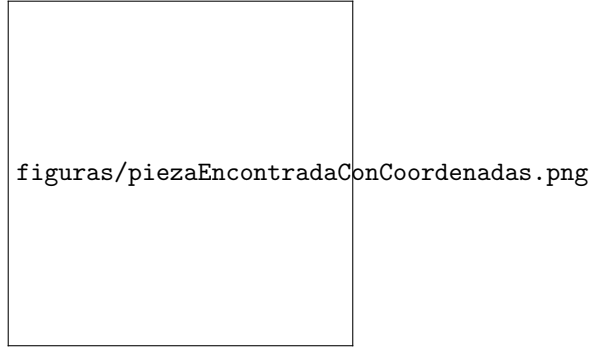


Figura 1.4: Ejemplo de un tornillo reconocido. El sistema devuelve las coordenadas (x, y, θ) con respecto del origen de la imagen.

- Tienen sección transversal predominantemente redonda, plana o una combinación de ambas características (en un caso), esto facilita la tarea de reconocimiento ya que piezas así dan por lo común una vista superior igual sin importar su rotación respecto de su eje principal. La vista superior es importante porque es la perspectiva de la cámara del SVT.

El SVT se probó con diversos conjuntos de piezas reales sin ningún tipo de tratamiento especial. Las piezas escogidas son predominantemente mecánicas las cuales se encuentran comúnmente en una línea de producción. Se utilizaron piezas tales como tornillos de diversas clases, tuercas, arandelas y otras piezas maquinadas. En la Fig. 1.5 se muestran algunas de las piezas utilizadas. En congruencia con un ambiente de ensamble y producción, estas piezas en ocasiones presentan superficies sucias, tienen colores oscuros, brillo heterogéneo y a veces tienen sombras. Todas las anteriores son características que complican la tarea de reconocimiento.



Figura 1.5: Algunas de las diversas piezas que se utilizaron en las investigaciones y experimentos al desarrollar el SVT.

El siguiente capítulo se dedica a la parte medular del SVT, que es el sistema de localización de piezas. Se describirán las bases de datos de imágenes utilizadas para la experimentación y se explicarán los métodos utilizados para la investigación y desarrollo del sistema identificador de piezas además de explicar el método particular de localización propuesto.

Capítulo 2

Clasificador Neuronal de Permutación de Códigos (PCNC)

El capítulo que inicia está dedicado al segundo clasificador neuronal utilizado para la implementación del SVT. Este clasificador llamado PCNC ha sido introducido en la Sección ?? teniendo diferencias sustanciales con el clasificador LIRA pero también algunas similitudes. A continuación en la siguiente sección se describe detenidamente la estructura del PCNC y los procesos que lleva a cabo. En la sección posterior se aborda el proceso de entrenamiento del mismo. En la Sección 2.3 se explican las distorsiones probadas con las base de datos utilizadas con el PCNC mientras que la Sección 2.4 y última dedica unas líneas a la discusión sobre este clasificador respecto a la tarea de la que se ocupa.

2.1. Estructura

El PCNC está basado en la estructura genérica del paradigma *Associative Projective Neural Networks* (APNN) descrita en [11, 12]. Dicho paradigma incluye al *clasificador de umbral aleatorio* [13, 14], al *clasificador neuronal de subespacio aleatorio* [15], al clasificador LIRA [16] y al *clasificador neuronal de permutación de códigos* (PCNC¹) [17]. El PCNC al igual que el clasificador neuronal LIRA trabaja con imágenes en escala de grises.

La estructura del PCNC consta de tres partes que trabajan en forma seriada, estas son un extractor de propiedades, un codificador y un clasificador neuronal (Fig. 2.1).

De forma muy general el PCNC inicia su trabajo cuando una imagen en escala de grises es presentada a la entrada del extractor de propiedades, las propiedades extraídas por este último son presentadas al codificador que las transforma en un vector binario de gran dimensión, vector que por último es dado al clasificador neuronal de una capa para ser procesado por éste, ya sea para propósitos de entrenamiento, de prueba o para reconocimiento de alguna clase previamente entrenada.

2.1.1. Extractor de propiedades

El extractor de propiedades (Fig. 2.2) inicia su trabajo con una imagen en escala de grises. Éste selecciona sobre la imagen una serie de puntos específicos (Fig. 2.3). Muchas formas de selección de estos puntos específicos pueden ser utilizadas, lo importante es que estos puntos representen las propiedades de la imagen que intervengan en su clasificación o diferenciación entre otras imágenes distintas a ser reconocidas. Dos métodos para definir los puntos específicos son por umbral de brillo y por extracción de contornos. Ambos métodos requieren de la selección de un umbral de brillo determinado B . Para el primer método, los puntos específicos de la imagen serán todos aquellos

¹PCNC son las siglas de *permutative code neural classifier*.

píxeles cuyo brillo b_{ij} sea mayor que B . Para el método de contorno se seleccionan todos aquellos puntos en donde el gradiente del brillo, es decir los contornos, sean mayores que el umbral B ; con esta última selección por umbral se logra eliminar considerablemente el ruido de la imagen de contorno.

Para el presente trabajo, de acuerdo a la naturaleza de las imágenes y las piezas que éstas contienen descritas en el Capítulo ??, se ha utilizado el método de extracción de contornos mediante un operador Sobel [18]. El método de umbral de brillo aplicado en imágenes relativamente grandes como las utilizadas en este trabajo (100×100 o 150×150) resulta en grandes cantidades de puntos específicos que implican el incremento masivo de recursos necesarios para el procesamiento de éstos y para las posteriores etapas del PCNC.

Para cada punto específico se define un rectángulo de dimensión $w \times h$ en cuyo centro está precisamente este punto (Fig. 2.4). Desde dentro de éste rectángulo se extraen múltiples propiedades de la imagen mediante el procedimiento explicado a continuación. Un conjunto de puntos positivos p y negativos n determinan cada una de las propiedades dentro del rectángulo. Estos puntos se distribuyen aleatoriamente dentro del rectángulo y su número es fijo para toda la estructura. Cada punto P_{rs} tiene asociado un umbral T_{rs} el cuál se define aleatoriamente en el rango:

$$T_{min} \leq T_{rs} \leq T_{max} \quad (2.1)$$

Los puntos positivos serán activos siempre que su brillo sea:

$$b_{rs} \geq T_{rs}. \quad (2.2)$$

Los puntos negativos serán activos siempre que su brillo sea:

$$b_{rs} \leq T_{rs}. \quad (2.3)$$

Una determinada propiedad existirá en el rectángulo si todos los puntos tanto positivos como negativos se encuentran activos.

Se procura que ninguno de estos rectángulos de búsqueda de propiedades en la imagen salga de la misma. Debe recordarse que una característica de las imágenes que se busca reconocer en este trabajo (imágenes normalizadas) no tienen puntos de interés en las orillas. Sin embargo si se elijen relativamente grandes los parámetros de ventana w y h con respecto a las dimensiones de la imagen W y H se tendrá más probabilidad de que existan rectángulos que salgan de los límites de la imagen, esto se hace más probable si existen puntos específicos cerca de las orillas. Considerando lo anterior se expande la imagen $w/2$ píxeles a cada lado y $h/2$ píxeles arriba y abajo con color blanco, es decir, significando ausencia de todo punto específico posible y evitando que cualquiera de estos rectángulos salga de la imagen ampliada.

Se utilizan muchas propiedades distintas $F_i \mid i \in [1, S]$. Donde S es por lo general del orden de unidades de millar. El extractor de propiedades examina las S propiedades para cada uno de los puntos específicos definidos. Todas las propiedades extraídas de todos los puntos específicos definidos son entregados al codificador.

2.1.2. Codificador

El codificador (Fig. 2.5) transforma las propiedades dadas por el extractor de propiedades a un vector binario:

$$V = \{v_i \mid v_i = \{0, 1\}, i \in (1, N)\}, \quad (2.4)$$

Para cada propiedad extraída F_k el codificador crea un vector adicional binario:

$$U_k = \{u_i \mid u_i = \{0, 1\}, i \in (1, N)\}, \quad (2.5)$$

Este vector contiene K 1's, donde $K \ll N$, al menos mil veces menor. Un procedimiento aleatorio que se explica más tarde es utilizado para elegir las posiciones de los unos en el vector U para cada

propiedad F_k . Este procedimiento genera la lista de posiciones de unos para cada característica y salva todas estas listas en memoria no volátil. El vector U_k es llamado *máscara* de la propiedad F_k . En la siguiente etapa del proceso de codificación se hace necesario transformar el vector auxiliar U al nuevo vector U^* el cual corresponde a la propiedad localizada en la imagen. Esta transformación se hace mediante permutaciones de los componente del vector U . El número de permutaciones depende de la localización de la propiedad en la imagen. Las permutaciones correspondientes a las direcciones horizontal (X) y vertical (Y) son permutaciones diferentes. Una permutación de m elementos P^m puede ser representada como un vector m -dimensional. Para aplicar esta permutación a un vector éste debe ser de dimensión m . En términos formales:

$$P^m(V) = V' \mid V' = \{v'_i\}, \quad v'_{p_i} = v_i, \quad P, V, V' \in \Re^m \quad (2.6)$$

Lo cual significa que el resultado de aplicar la permutación P^m a un vector V resulta en un nuevo vector V' cuyos componentes se definen tomando cada componente v_i de V y colocándolo en la posición p_i de V' , es decir, a la que apunta el índice correspondiente del vector de permutación P^m . En la Fig. 2.6 se ilustra un ejemplo gráfico simple a este respecto.

2.1.2.1. Codificación de las propiedades

El problema a resolver es obtener códigos binarios de las propiedades extraídas los cuales tengan correlación fuerte si la distancia entre las localizaciones de las propiedades es pequeña y tengan correlación débil o no tengan ninguna si esta distancia es grande. Por ejemplo, si una misma propiedad F_k se extrae tanto en un extremo de la pieza en la imagen como en el otro extremo entonces éstas deben ser codificadas como vectores binarios distintos U_{k1}^* y U_{k2}^* , existiendo entre estos correlación débil o ausencia alguna de correlación. En el caso de que la misma propiedad sea encontrada en puntos vecinos entonces estas deben ser codificadas con los mismos vectores U_{k3}^* y U_{k4}^* . Esta propiedad que se acaba de describir permite que el sistema de reconocimiento sea insensible a pequeños desplazamientos de los objetos en la imagen.

Para codificar la localización de la propiedad F_k en la imagen es necesario definir y dar valor a la distancia de correlación D_c . Sea la propiedad F_k detectada en dos puntos distintos $P_1(x_1, y_1)$ y $P_2(x_2, y_2)$, sean U_{P1}^* y U_{P2}^* los vectores binarios que codifican a F_k para estos puntos respectivamente y sea d la distancia euclidiana entre estos puntos dada por:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2.7)$$

Se requiere que:

$$d < D_c \Rightarrow U_{P1}^* \text{ y } U_{P2}^* \quad \text{estén correlacionados y,} \quad (2.8)$$

$$d \geq D_c \Rightarrow U_{P1}^* \text{ y } U_{P2}^* \quad \text{no estén correlacionados.} \quad (2.9)$$

Para buscar cumplir con estas propiedades se calculan los siguientes valores para una propiedad detectada F_k en un punto $P(i, j)$:

$$\begin{aligned} X &= i/D_c, \\ E(X) &= \text{int}(X), \\ R(X) &= i - E(X) \cdot D_c, \end{aligned} \quad (2.10)$$

$$\begin{aligned} Y &= j/D_c, \\ E(Y) &= \text{int}(Y), \\ R(Y) &= j - E(Y) \cdot D_c, \end{aligned} \quad (2.11)$$

$$p_x = \text{int} \left(\frac{R(X)}{D_c} N \right), \quad (2.12)$$

$$p_y = \text{int} \left(\frac{R(Y)}{D_c} N \right), \quad (2.13)$$

donde $\text{int}()$ es la función entero. Por lo tanto $E(X)$ y $E(Y)$ son las partes enteras de X y Y respectivamente y $R(X)$ y $R(Y)$ son las partes fraccionarias de X y Y respectivamente.

La máscara (vector U_k) de la propiedad F_k se considera como un código de esta propiedad localizado en el punto origen de la imagen: $O(0,0)$. Se definen las permutaciones \mathbf{P}_x y \mathbf{P}_y como requisito para obtener los códigos correspondientes a las propiedades existentes en cualquier punto de la imagen fuera del origen. En general, para obtener el código de la propiedad F_k perteneciente al punto $P(i,j)$ se procede de la siguiente forma:

1. Primero se trata el desplazamiento horizontal, para lo cual se aplica $E(X)$ veces la permutación \mathbf{P}_x al vector U_k , después se aplica la misma permutación una vez más pero solamente a los primeros p_x componentes del vector U_k .
2. Segundo, se trata el desplazamiento vertical de forma análoga. Se aplica la permutación \mathbf{P}_y $E(Y)$ veces con una permutación adicional para los primeros p_y componentes de U_k .

2.1.2.2. Ejemplo de permutaciones

El procedimiento anterior se ilustra con un ejemplo para facilitar su comprensión. La dimensión del vector U y los valores x y y del punto P se eligen pequeños con el propósito de dar claridad al proceso. Tomemos como parámetros del PCNC $D_c = 6$ y $N = 8$ y sean \mathbf{P}_x y \mathbf{P}_y dos permutaciones de dimensión N .

Supóngase detectada la propiedad F en el punto $P(10,14)$ y sea el vector U la máscara correspondiente a esta propiedad. La tarea consiste en codificar U en un nuevo vector U^* atendiendo a las coordenadas del punto P . Como primer paso se aplican (2.10), (2.11), (2.12) y (2.13) resultando: $E(X) = 1$, $E(Y) = 2$, $p_x = 5$ y $p_y = 2$. Se aplica a continuación $E(X)=1$ permutación \mathbf{P}_x al vector U (Fig. 2.7) y luego al resultado (U_1) se aplica una permutación adicional únicamente a los primeros $p_x = 5$ componentes obteniendo con esto el vector U' , todas las componentes que no sean definidas mediante la permutación parcial se copian del vector anterior correspondiente no importante si fueron permutados o no. Con el proceso anterior tenemos las siguientes trayectorias de ejemplo: $u_1 \rightarrow u_{1,3} \rightarrow u'_4$, $u_2 \rightarrow u_{1,7} \rightarrow$ se elimina y $u_8 \rightarrow u_{1,5} \rightarrow u'_8, u'_5$. Tenemos que para el primer ejemplo el valor de u_1 es permutado dos veces y termina en u'_4 . Para el segundo caso u_2 se permuta una sola vez a $u_{1,7}$ y luego se ignora porque a la componente 7 es menor que $p_x = 5$ y por que la componente correspondiente en U' ya ha sido ocupada por u_7 . Para el tercer caso u_8 se permuta las dos veces pero adicionalmente es copiado a la componente u'_5 pues luego de permutar los primeros 5 elementos de U' queda vacía.

En este ejemplo se han dado los tres casos posibles que pueden ocurrir, sin embargo debe tenerse en cuenta que de acuerdo a la naturaleza práctica de los vectores máscaras (U) que tienen dimensión N muy grande y sus componentes son mayoritariamente ceros, los casos especiales de eliminación ocurren muy rara vez como se explicará.

Una vez habiendo realizado las permutaciones X , realizamos las permutaciones Y con el mismo procedimiento, sólo que ahora lo haremos con el vector U' y aplicaremos la permutación \mathbf{P}_y y los parámetros $E(Y)$ y p_y . En la Fig. 2.8 se muestra gráficamente este procedimiento. El resultado de esta permutación es el resultado de ambas permutaciones aplicadas sobre el vector máscara U de la propiedad F y le llamamos vector U^* . Este vector codifica la propiedad F en la ubicación del punto $P(10,14)$ de la imagen correspondiente.

2.1.2.3. Propiedades de las permutaciones

Consideremos ahora las propiedades de las permutaciones descritas. Supóngase que la propiedad F_k ha sido detectada en dos puntos $P_1(x_1, y_1)$ y $P_2(x_2, y_2)$ tales que $x_1 \neq x_2$ y $y_1 \neq y_2$. Sea U_k la máscara correspondiente para esta propiedad. Se definen d_x y d_y como:

$$d_x = |x_2 - x_1|, \quad (2.14)$$

$$d_y = |y_2 - y_1| \quad (2.15)$$

Suponiendo que $d_x \neq 0$, luego de realizar las permutaciones horizontales (\mathbf{P}_x) los vectores correspondientes U_1 y U_2 serán distintos. Sea Δn la diferencia en el número de 1's entre los vectores. Puede mostrarse que el valor promedio de Δn puede calcularse de forma aproximada con:

$$\Delta n \approx \frac{K}{D_c} d_x \quad (2.16)$$

donde K es el número de unos del vector auxiliar binario U de la propiedad F_k . Luego de las permutaciones verticales (\mathbf{P}_y) los vectores resultantes correspondientes U_1^* y U_2^* tendrán diferencias que pueden ser estimadas por:

$$\overline{\Delta n} \approx K \left(1 - \left(1 - \frac{d_x}{D_c} \right) \left(1 - \frac{d_y}{D_c} \right) \right), \quad 1 > \overline{\Delta n} > 0. \quad (2.17)$$

Por lo anterior los vectores U_1^* y U_2^* estarán correlacionados solamente si se cumple $d_x < D_c$ y $d_y < D_c$. La correlación se incrementará si d_x y d_y se decrementan.

Puede verse en la Fig. 2.9 que distintos componentes del vector U pueden pretender quedar en la misma posición luego de realizar las permutaciones. Por ejemplo, luego de las permutaciones \mathbf{P}_x el componente u_8 ocupa dos posiciones en U' o luego de aplicar ambas permutaciones el componente u_6 termina en dos posiciones y el u_8 en tres. Estos eventos son indeseables y no son un problema para el PCNC pues la probabilidad de que tales eventos indeseables sucedan está relacionada inversamente a la dimensión de N y a la relación N/K y atendiendo a los valores grandes de N y pequeños para K la probabilidad de estos eventos es menor de 0.01 % [19].

2.1.2.4. Vector código V

Una vez calculados todos los vectores U_r^* de todas las propiedades detectadas en la imagen se crea el vector código definido como:

$$V = \left\{ v_i \mid v_i = \bigwedge u_{ri}^*, i \in (1, N) \right\} \quad (2.18)$$

donde \bigwedge es el símbolo de la disyunción, u_{ri}^* es el i -ésimo componente del vector U_r^* , vector que es el corresponde a la propiedad detectada F_r .

Al utilizar números aleatorios independientes para la generación de las máscaras se logra que este proceso de codificación produzca representaciones mayoritariamente independientes para todas las propiedades. La única pero débil influencia entre las distintas propiedades aparece cuando se absorbe algún 1 en la disyunción, Eq. (2.18).

2.1.2.5. Adelgazador dependiente de contexto (CDT)

Para poder reconocer alguna pieza en una imagen se hace necesario utilizar combinación de propiedades, es decir, combinar la existencia de ciertas propiedades con la ausencia de otras. Para lograr este propósito de combinación de propiedades se ha utilizado con éxito el CDT² o *Adelgazador*

²Llamado así por ser las siglas en inglés de Context Dependent Thinning.

dependiente de contexto. [1]. El CDT ha sido desarrollado en base al proceso de normalización de vectores [20]. Si bien existen diversos procedimientos de implementación del CDT en este trabajo se utiliza el procedimiento ilustrado en la Fig. 2.10, el cuál requiere de un número entero q como parámetro de entrada y consiste en lo siguiente:

1. Se genera una nueva permutación \mathbf{Q} de dimensión N la cuál es independiente de \mathbf{P}_x y \mathbf{P}_y .
2. Se prueba cada componente v_i del vector V , si $v_i = 0$ no se hace nada si $v_i = 1$ se considera la trayectoria de este componente individual durante q permutaciones \mathbf{Q} . Si esta trayectoria pasa por al menos un elemento 1 del vector V , el valor de v_i se hace 0.
3. Al vector resultante se le llama V' .

Un ejemplo gráfico de lo anterior se tiene en la Fig. 2.10 donde $q = 4$. La permutación \mathbf{Q} se representa por las flechas. Para el elemento v_1 se sigue la trayectoria indicada por la permutación la cuál es: $v_1 \rightarrow v_7 \rightarrow v_8 \rightarrow v_5$, si cualquiera de las componentes v_7 , v_8 o v_5 es igual a 1 entonces $v_1 = 0$. Lo mismo se hace para todos los elementos de V , construyéndose un nuevo vector sin alterar el vector original.

El número q es un parámetro de reconocimiento del sistema. Una vez que se aplica el CDT al vector V se ha cumplido con el proceso correspondiente del codificador. El vector V' de dimensión N representa el código de la imagen presentada originalmente al extractor de propiedades. Este resultado está listo para ser pasado al clasificador neuronal.

2.1.3. Clasificador neuronal

En la Sección ?? se ha mencionado y hecho referencia al perceptrón de una capa de Rosenblatt. Este perceptrón posee muy buena convergencia sin embargo requiere que en el espacio paramétrico las clases tengan separabilidad lineal. Para obtener esta separabilidad lineal, las etapas anteriores del PCNC, el extractor de propiedades y el codificador, convierten el espacio paramétrico de una imagen que es representado por el brillo de todos los píxeles de ésta, a un espacio paramétrico de mayor dimensión. En general se tiene un espacio paramétrico de dimensión $W \cdot H$ convertido a otro de dimensión N , donde W y H son el ancho y el alto en píxeles de las imágenes a procesar por el PCNC y N es el parámetro del PCNC igual a la dimensión del vector V' . Este procedimiento descrito mejora considerablemente la separabilidad lineal del espacio paramétrico que representa la imagen y el objeto que esta contiene.

En la Fig. 2.11 se presenta de nuevo la estructura del clasificador neuronal LIRA pero dividiéndolo. Obsérvese que las primeras tres capas S , I y A cumplen la función de un extractor de propiedades al ser las entradas respectivas de cada grupo de la capa I seleccionadas aleatoriamente dentro de un rectángulo posicionado aleatoriamente en la imagen (Sec. ??). En cambio, las salidas de la capa A y la capa R en su totalidad, cumplen la función propia del clasificador neuronal cuya estructura está basada como ya se explicó en el perceptrón. De lo que se trata es de utilizar esta segunda parte de la estructura junto con el extractor de propiedades y el codificador (Fig. 2.1). Es decir, respecto del clasificador LIRA, se sustituyen las capas S , I y A por los dos bloques previamente descritos del PCNC. Debe notarse que con esta sustitución, la capa A del clasificador neuronal pasa a contener exactamente al vector V' por lo cuál debe tener N elementos.

Una vez que se ha descrito con todo detalle la estructura del PCNC se explicará su proceso de entrenamiento.

2.2. Proceso de entrenamiento

De acuerdo a las similitudes entre las estructuras del PCNC y del clasificador neuronal LIRA se tiene que la única parte de ambos que varía durante el proceso de entrenamiento son las conexiones entre las capas A y R , por lo tanto el proceso de entrenamiento descrito para el clasificador LIRA (Sec. ??) funciona igual para el PCNC y por lo tanto es aplicado. El proceso de codificación de las

imágenes aplicado a LIRA es válido para el PCNC por lo que se aplica también. Esto se debe a que cada imagen tendrá un vector V' que codifica sus propiedades extraídas por lo cual pueden usarse éstas a través de V' en lugar de la imagen con propósitos de entrenamiento ahorrando importantes recursos de cómputo y de tiempo en el proceso.

2.3. Distorsiones

De forma similar que con el clasificador neuronal LIRA y con el objetivo de ampliar el conjunto de imágenes destinadas para el entrenamiento del PCNC, se consideró la aplicación de distorsiones sobre las imágenes normalizadas originales. Atendiendo a que el PCNC no es sensible a desplazamientos cartesianos de la imagen, sólo se aplicaron distorsiones angulares (Fig. 2.12). Estas distorsiones aplicadas sobre las imágenes, representan pequeñas variaciones sobre la exacta alineación de las piezas con respecto al eje horizontal que pasa por el centro de la imagen. Por esto se realizaron experimentos con diversas distorsiones angulares. Estos experimentos se explican en el siguiente capítulo, en la Sección 3.1.3.

Para realizar estas distorsiones se ha utilizado el mismo método de creación de imágenes distorsionadas que para el clasificador LIRA. Se consideró la creación de tales distorsiones en pares, con un mismo valor absoluto angular, pero con signos opuestos.

2.4. Discusión

Se ha visto que el clasificador PCNC es igual que el LIRA en cuando a su método de clasificación, variando en ambos la forma en que se crean o extraen las propiedades de la imagen a clasificar. Es entonces en los procesos de extracción de propiedades y de codificación donde está la diferencia entre el PCNC y LIRA. Esta diferencia tiene dos características de importancia en el PCNC, una de éstas es la capacidad de aplicar y probar diversos métodos de extracción de puntos específicos y no limitarse únicamente a niveles de brillo a través de un umbral, abriendo paso de esta manera a las posibilidades que ofrece el preprocesamiento de las imágenes a clasificar; la otra característica medular es la consideración explícita que lleva a cabo el codificador sobre la posición de las propiedades encontradas en la imagen.

La principal desventaja del PCNC sobre el clasificador LIRA está en sus mayores requerimientos de cómputo debido a la necesidad de realizar gran cantidad de cálculos con vectores durante el proceso de codificación de cada imagen. Comparando la etapa de extracción de propiedades de LIRA y del PCNC se observa que ambos ubican ventanas aleatoriamente en la imagen de entrada, el número de puntos de cada una de ellas puede ser similar, sin embargo, mientras la cantidad total de ventanas aleatorias en LIRA será igual al número de grupos en la capa I , que es un parámetro de este clasificador del orden de 100000 para la tarea de reconocimiento que nos ocupa [21], se tiene que en el PCNC el número de tales ventanas será el número de puntos específicos extraídos de la imagen multiplicado por el parámetro S , y dado que este parámetro está entre 1000 y 10000 [19], tenemos que para el caso de $S = 1000$ con 100 puntos específicos igualamos los requerimientos del clasificador LIRA y en el orden en que este último número sea rebasado lo serán los requerimientos del PCNC respecto de LIRA en términos generales. Si consideramos que las imágenes normalizadas con que se ha trabajado (Sec. ??) tienen dimensión de 100×100 como mínimo, igual a 10000 píxeles se tiene que el número de puntos específicos para igualar los requerimientos de LIRA deberán ser de aproximadamente el 1 %. Esto lleva a intentar mejorar el método de selección de puntos específicos o a tener que trabajar con las imágenes normalizadas reducidas cierta escala. Por lo anterior, de usarse el tamaño original de las imágenes, el método de selección de puntos específicos por umbral arrojará gran porcentaje de puntos respecto al total de píxeles de la imagen por lo que los requerimientos computacionales y de tiempo crecerán tanto que se violarán los principios de economía establecidos para las microfábricas y el SVT (Sec. ?? y 1.3). El inconveniente anterior se resolvió utilizando el umbral de gradiente de brillo (extracción de contornos), lo cual ha reducido el número de puntos específicos significativamente sin que se haya

traducido en reducción de las propiedades que posibilitan la adecuada clasificación de las imágenes. Prueba de lo anterior se presenta en el capítulo siguiente y último.

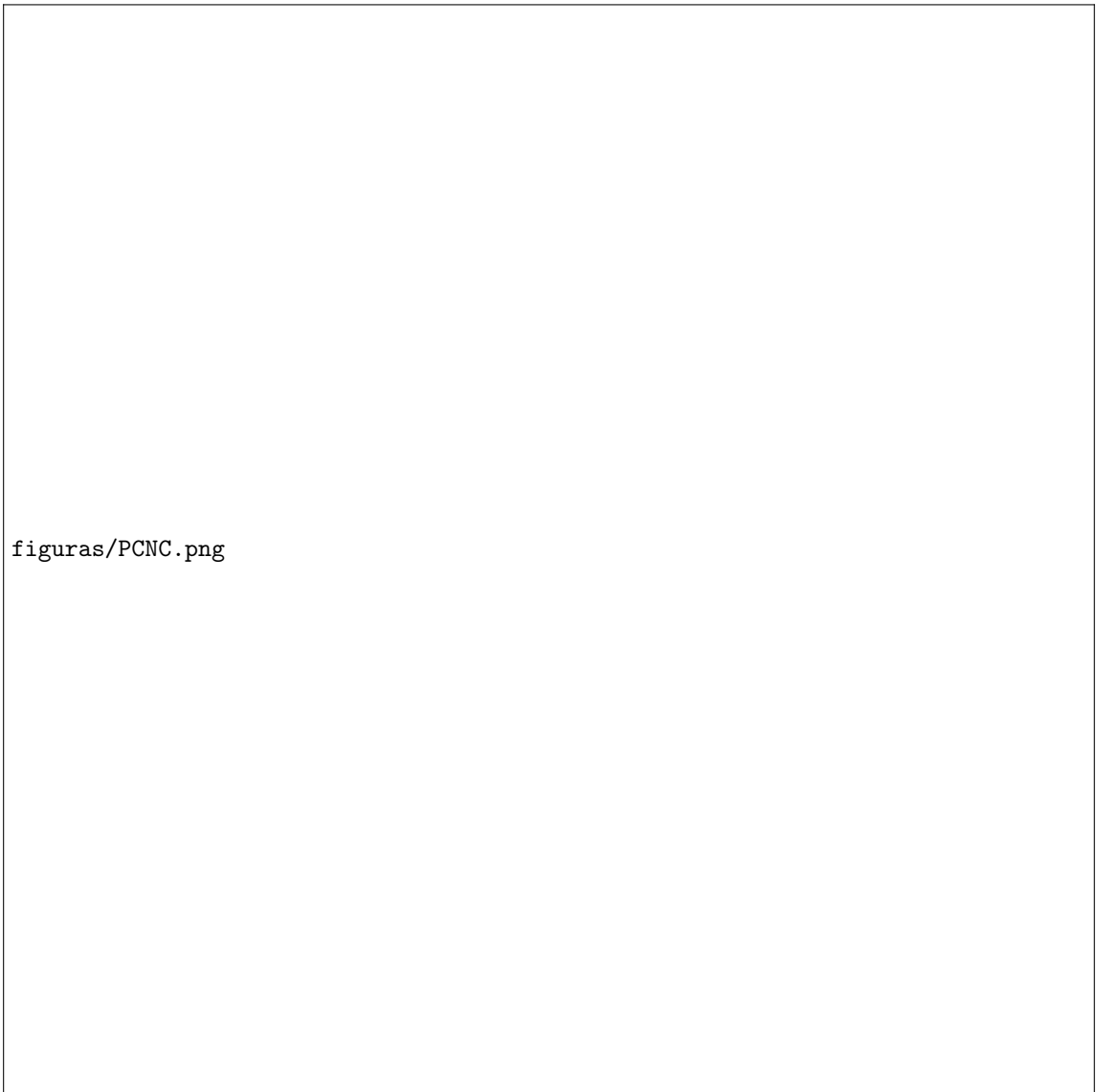


Figura 2.1: Diagrama a bloques de la estructura del PCNC mostrando los elementos principales de intercambio entre ellos.



Figura 2.2: Procesos del extractor de propiedades, su interrelación, así como sus entradas y salidas.

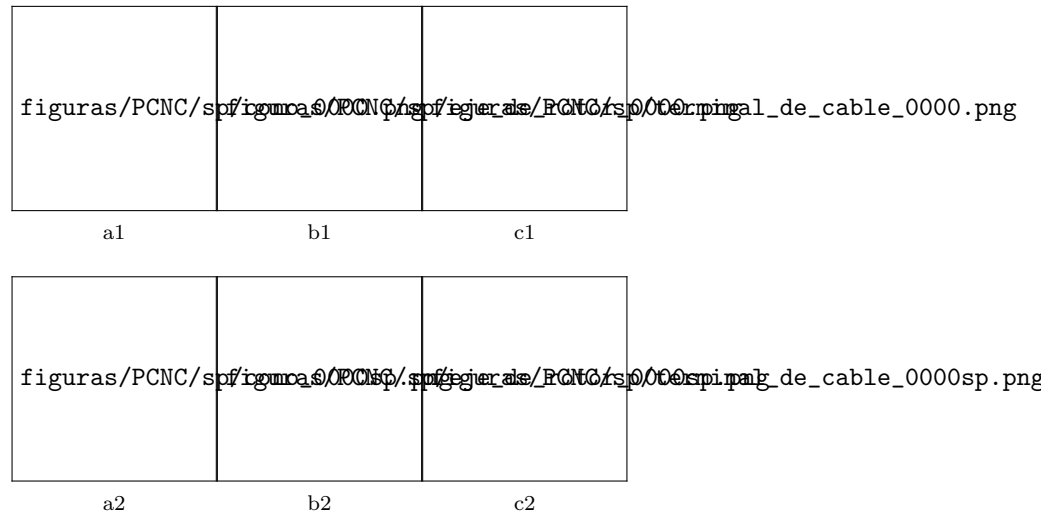


Figura 2.3: Selección de puntos específicos con el método de umbral de bordes. Se parte de una imagen normalizada y se aplica un operador de extracción de bordes Sobel. Se seleccionan aquellos puntos resultantes que son mayores que el umbral predefinido B . Fila superior. Imágenes normalizadas originales. Fila inferior. Resultado de la selección de los puntos específicos sobre las imágenes respectivas de arriba, todos los píxeles negros en la imagen son puntos específicos seleccionados.



Figura 2.4: Extracción de propiedades en la imagen. Primero se definen S propiedades. Cada una mediante p puntos positivos y n puntos negativos aleatoriamente distribuidos sobre un rectángulo de $w \times h$ píxeles. A la izquierda se ilustran ejemplos de estas propiedades con $p = 4$ y $n = 5$. En el cuadro principal se muestran los puntos específicos de una imagen. Para cada uno de estos puntos se prueba la existencia de las S propiedades. En la imagen se muestra el proceso de búsqueda de la propiedad F_k .

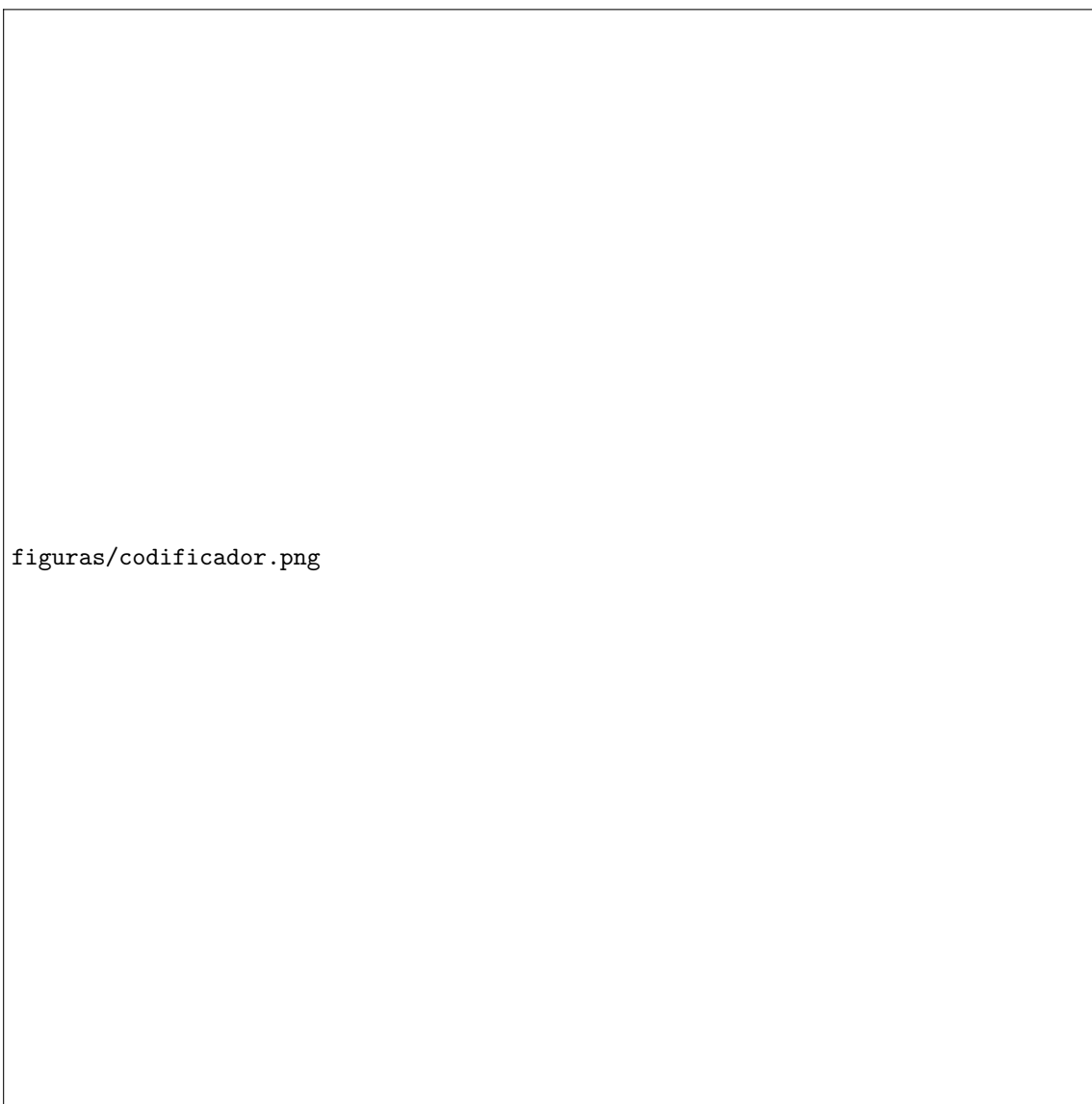


Figura 2.5: Principales procesos que lleva a cabo el codificador así como sus entradas y salidas.

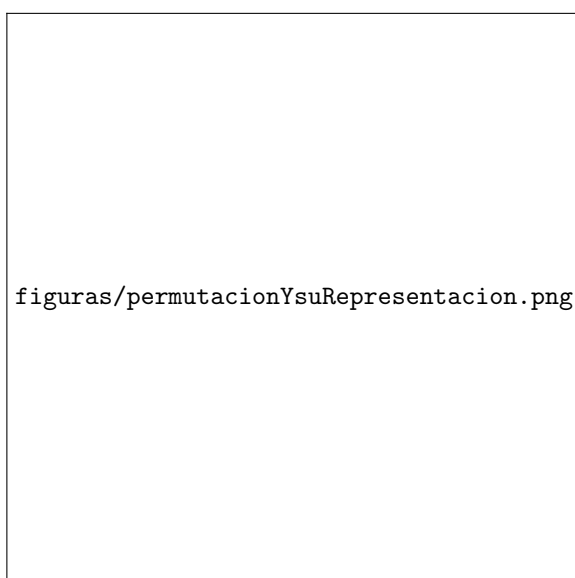


Figura 2.6: Permutación y su representación sobre un vector. Se ilustra una permutación P^6 como un vector. El resultado de aplicar esta permutación sobre un vector V se ilustra a la derecha. Cada elemento v_i de V es reordenado en la posición señalada por la componente correspondiente de P^6 , dando como resultado un nuevo vector V' con los mismos elementos de V pero en orden distinto.

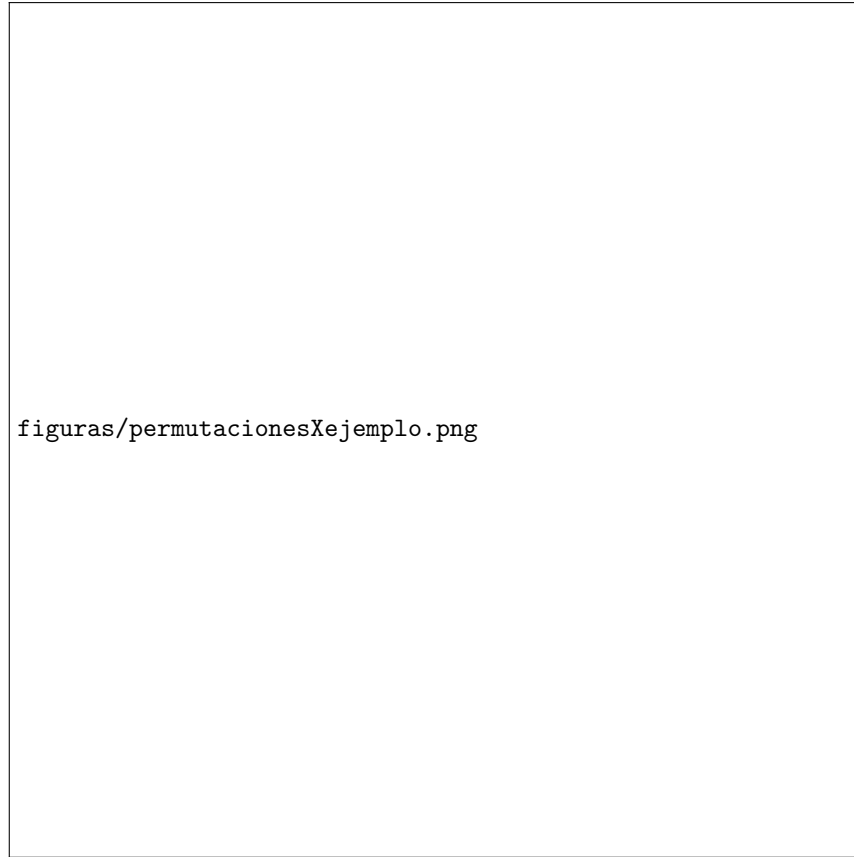


Figura 2.7: Ejemplo de permutaciones $\mathbf{P}_{\mathbf{x}}$ sobre el vector U con $E(X) = 1$ y $p_x = 5$. Se aplica una vez la permutación a todo el vector y una vez más sólo a los primeros cinco componentes del mismo. En este sentido las líneas punteadas representan cambios que no deben realizarse.

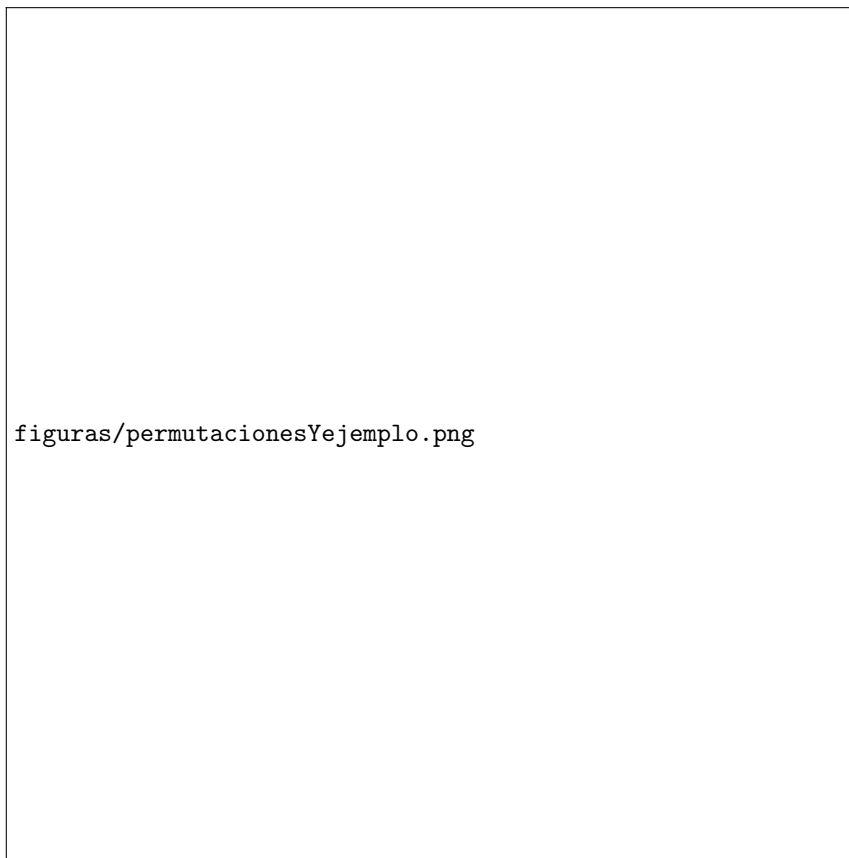


Figura 2.8: Ejemplo de permutaciones \mathbf{P}_Y aplicado sobre el vector resultante de las permutaciones \mathbf{P}_X con $E(Y) = 2$ y $p_y = 2$. La permutación se aplica dos veces a todo el vector y una vez más solamente a los primeros dos elementos del mismo. Las líneas punteadas de la última permutación indican cambios que no deben realizarse.

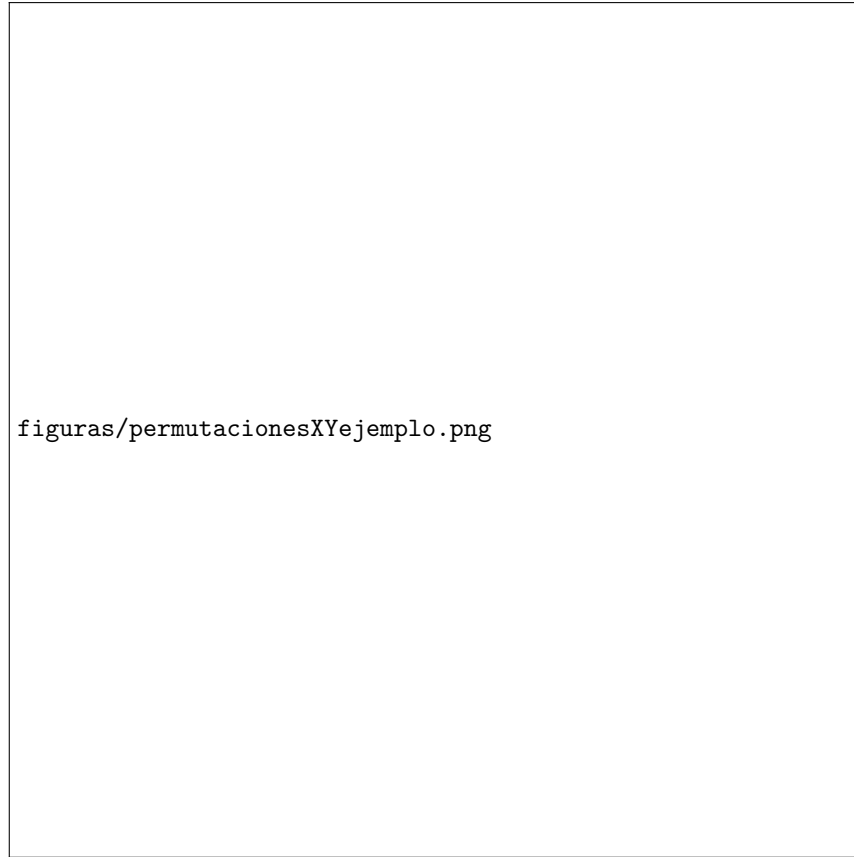


Figura 2.9: Resultados obtenidos al aplicar las permutaciones $\mathbf{P}_{\mathbf{x}}$ sobre el vector U y luego la permutación $\mathbf{P}_{\mathbf{y}}$ a ese resultado U' . En la figura se muestra el orden final de las componentes del vector U dentro del vector final U^* .



Figura 2.10: Aplicación de q permutaciones \mathbf{Q} sobre el vector V .

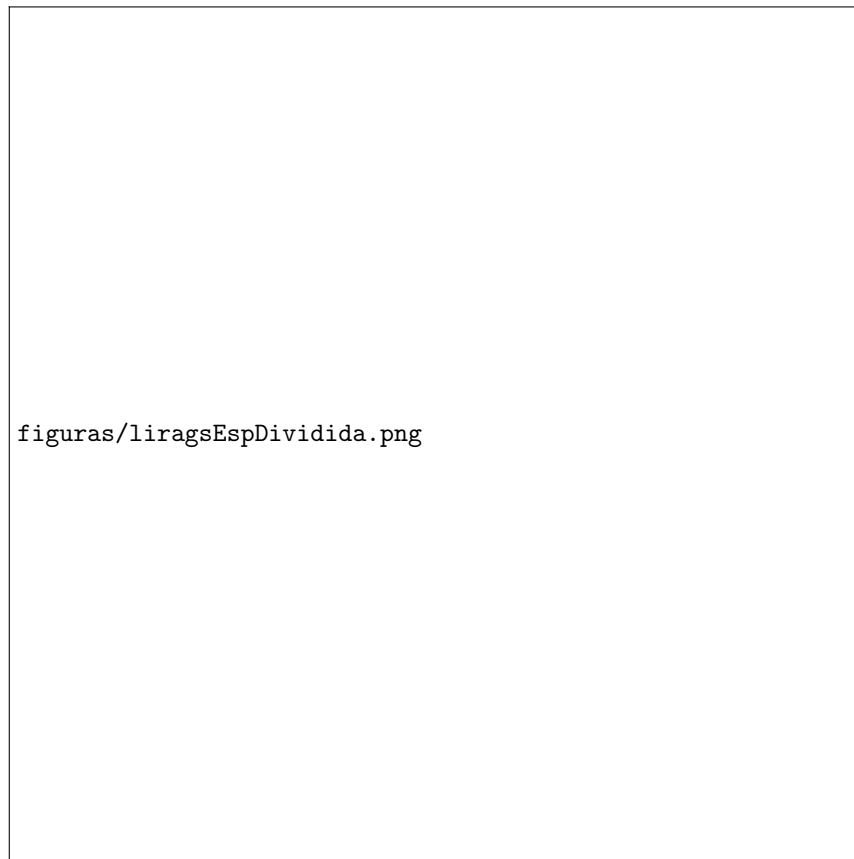


Figura 2.11: Clasificador neuronal LIRA dividido en dos partes según su función partículas: extractor de propiedades y clasificador.

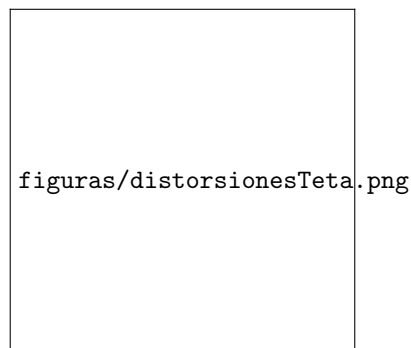


Figura 2.12: Ejemplo de una distorsión angular para las imágenes del conjunto de entrenamiento destinadas al PCNC.

Capítulo 3

Experimentos y resultados

Este capítulo final se dedica a los experimentos realizados con los clasificadores LIRA y PCNC sobre las bases de datos descritas en la Sección ???. La primera sección de este capítulo se dedica a los experimentos con el clasificador neuronal LIRA y la segunda a los experimentos con el PCNC. La tercer sección compara los resultados de ambos clasificadores y discute sobre de éstos. La cuarta y última sección se dedica a la tarea de búsqueda y localización de piezas.

En este trabajo un experimento significa una serie de pruebas con el clasificador LIRA o PCNC encaminadas a probar o concluir una propiedad u objetivo particular del mismo.

Para ambos clasificadores neuronales se realizaron múltiples experimentos con las bases de datos descritas. Los experimentos llevados a cabo han sido objetivos, sistemáticos y estadísticamente convincentes. Los experimentos son objetivos por que parten de un plan predeterminado y bien definido para su realización el cuál se explica en breve. Son sistemáticos por que se han hecho de forma ordenada y organizados en etapas, utilizando cuando es necesario los resultados obtenidos en los experimentos previos en los subsecuentes. Y por último los experimentos son estadísticamente convincentes por que se ha tenido cuidado de ejecutar múltiples pruebas agrupadas en experimentos con metodología idéntica.

Todos los experimentos descritos en este trabajo se ejecutaron en un computador con procesador Intel® Pentium® 4 a 2.80 GHz con 512 KB de memoria caché y 512 MB de memoria RAM con sistema operativo GNU/Linux kernel 2.6.17-11-generic.

3.1. LIRA

Primero que nada es importante señalar que dada la estructura del clasificador LIRA (Sec. ??), el orden de magnitud en los valores prácticos de algunos de sus parámetros ($W \times H$, $w \times h$ y N), y por el carácter aleatorio de su construcción es sumamente improbable que dos estructuras idénticas LIRA sean creadas, esto considerando idénticos parámetros de creación. Por lo anterior es importante mencionar cuando un experimento utilizó un mismo clasificador LIRA (misma creación) y cuando se utilizaron distintos clasificadores con idénticos parámetros. Adicionalmente debido al proceso de entrenamiento del clasificador LIRA (Sec. ??), se tiene que un mismo clasificador puede estar entrenado con diversos conjuntos de entrenamiento y diversos ciclos de entrenamiento, por lo cuál aún copias idénticas de un mismo clasificador pueden presentar respuestas distintas. Tener presente lo anterior ha sido importante para la planeación de los experimentos realizados y comprende varios resultados que se mostrarán más adelante.

Cada una de las bases de datos con imágenes empleadas consiste en dos conjuntos fijos de imágenes, uno para entrenamiento y otro para prueba. Estos conjuntos se seleccionaron aleatoriamente de la base de datos respectiva. Adicionalmente, las bases de datos descritas pueden tener imágenes para propósitos especiales, las cuales se explican en los experimentos que las ocupan. Para ciertos experimentos realizados se utilizaron los conjuntos fijos de entrenamiento, mientras que para otros ambos conjuntos se seleccionaron aleatoriamente de entre toda la base de datos respectiva. Todos

los experimentos realizados con el clasificador LIRA utilizaron las bases de datos A, B y D con excepción de los experimento sobre distorsiones y sobre conjunto ampliado de entrenamiento.

Los experimentos realizados, su justificación, metodología y resultados se abordan en las secciones siguientes.

3.1.1. Experimentos preliminares

Las primeras pruebas con el clasificador LIRA han sido reportadas en [21]. Estas pruebas utilizaron la base de datos α descrita en la Sección ?? . Se probaron varias combinaciones de parámetros para el clasificador considerando los mejores parámetros reportados en [22]. En este grupo de pruebas no se utilizaron distorsiones para el conjunto de imágenes de entrenamiento. En la Tabla 3.1 se muestran los resultados de este experimento. El mejor resultado obtenido en el experimento fue con una ventana LIRA de 15×15 píxeles, 175 000 neuronas en la capa A, cuatro neuronas ON y tres neuronas OFF en cada grupo, el parámetro η del clasificador igual a 1.0 y el parámetro de entrenamiento T_E igual a 0.15. Para estos parámetros el porcentaje de neuronas activas en la capa A fue 0.164 %. Para esta prueba el porcentaje de reconocimiento correcto fue de 94 %. El mejor desempeño obtenido para múltiples combinaciones de parámetros fue alcanzado con 40 ciclos de entrenamiento por lo que los mismos se utilizaron para todas las pruebas del experimento descrito.

Tabla 3.1: Experimento preliminar con nueve pruebas para la sintonización de parámetros LIRA con la base de datos α .

No. de experimentos	1	2	3	4	5	6	8	9
Tamaño de la ventana ($w \cdot h$)	12 · 12	15 · 15	15 · 15	10 · 10	13 · 13	17 · 17	15 · 15	10 · 10
Constante LIRA (η)	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Neuronas capa A (N) (miles)	175	175	175	175	175	175	200	200
Neuronas ON por grupo (p)	3	3	4	4	4	5	4	4
Neuronas OFF por grupo (n)	4	4	3	3	3	3	3	3
Neuronas activas (%)	0.06	0.09	0.16	0.12	0.13	0.08	0.17	0.15
Porcentaje de reconocimiento (%)	68	89	94	93	85	88	89	89

Luego de múltiples pruebas preliminares y otras adicionales se determinó que el parámetro de entrenamiento T_E igual a 0.15 es el que mejor rendimiento ofreció para todos los casos y bases de datos por lo que se fijó al valor dado.

Los experimentos de las secciones siguientes se realizaron teniendo como base el mejor conjunto de parámetros obtenido en este experimento preliminar, es por ello que en lo sucesivo se les hará referencia a este conjunto de parámetros como parámetros base.

3.1.2. Unicidad

Al inicio de esta sección se ha hecho mención de la característica única de cada construcción de un determinado clasificador LIRA, aún con los mismos parámetros. Para estudiar cuál es el comportamiento de tales clasificadores construidos así se realizó el experimento descrito a continuación. Este experimento consistió en una serie de pruebas con distintas construcciones LIRA utilizando los mismos parámetros y utilizando los mismos conjuntos fijos de entrenamiento y prueba, siendo entrenadas con el mismo número de ciclos de entrenamiento. Estas consideraciones llevan a que la única variable del experimento es la estructura única de cada construcción LIRA. Este experimento se describe en primer lugar ya que para experimentos posteriores donde se utilizan distintas construcciones del clasificador LIRA además de la variable que se pretenda estudiar se tendrá inevitablemente la variabilidad de la estructura particular de cada construcción LIRA. Por esta razón a este experimento se le llama de unicidad, derivado del hecho de que cada clasificador LIRA es único.

En la Tabla 3.2 se muestran los resultados obtenidos para 10 pruebas realizadas con los parámetros base¹: $w = 15$, $\eta = 1,0$, $N = 175000$, $p = 4$ y $n = 3$ y 40 ciclos de entrenamiento. Adicionalmente se presenta el promedio de reconocimiento obtenido y la desviación estándar para mejor comprensión de estos datos.

Tabla 3.2: Resultados del experimento con 10 construcciones LIRA para cada una de las tres bases de datos. Cada una de las 10 corridas se hizo con idénticos parámetros, mismos conjuntos fijos de entrenamiento y prueba, y mismo número de ciclos de entrenamiento. Todos los resultados se dan en unidades porcentuales que indican el reconocimiento logrado.

LIRA:	1	2	3	4	5	6	7	8	9	10	\bar{x}	σ
BD-A	87	86	91	89	86	90	81	90	93	84	87.7	3.14
BD-B	93	90	90	92	91	91	92	91	93	91	91.4	1.02
BD-D	87	89	88	89	84	88	89	88	89	89	88.0	1.48

...

Tabla 3.3: Resultados de pruebas del clasificador LIRA con variación del parámetro p sobre las bases de datos A, B y D.

Parámetro	p	2	3	4	5	6
BD-A	%	85	88	89	83	86
BD-B	%	79	69	94	82	85
BD-D	%	86	85	85	89	86

Tabla 3.4: Resultados de pruebas del clasificador LIRA con variación del parámetro n sobre las bases de datos A, B y D.

Parámetro	n	1	2	3	4	5	6
BD-A	%	35	78	89	86	83	64
BD-B	%	73	82	94	83	88	80
BD-D	%	85	86	85	80	83	78

El análisis de los datos obtenidos da muestra de la relación de cada uno de los parámetros del clasificador LIRA sobre su resultado. Se ha visto que los parámetros w y N influyen en el resultado para cada base de datos en forma mas o menos uniforme mientras que p , n y η no lo hacen.

Ahora bien, debido a la interdependencia que los parámetros juegan en el desempeño del clasificador LIRA, debe dudarse que los mejores parámetros obtenidos de forma individual garanticen que un conjunto formado por los mismos sea el que mejor resultados proporcione. Para esto es necesario realizar otro experimento en donde se prueben diversos conjuntos de parámetros que tengan como base los resultados obtenidos tanto previamente como en esta sección. Por lo tanto se han probado tres conjuntos de parámetros sobre la base de datos A, el primer conjunto es el de los parámetros obtenidos en la Sección 3.1.1, el segundo es el de los parámetros que dieron el mejor resultado en el experimento de parámetros independientes y el tercer conjunto es el construido con los parámetros que fueron mejores en los experimentos individuales. Para este experimento se han hecho cinco distintas construcciones LIRA para cada conjunto de parámetros, para obtener el promedio de reconocimiento correcto para cada uno en lugar de conformarse con un sólo resultado, pues como se explicó en la Sección 3.1.2 diversas construcciones LIRA con idénticos parámetros dan resultados con cierta variación. Nótese que en la Tabla ?? se reporta el mejor resultado obtenido en estos experimentos de sintonización. Este resultado es de 92 % de reconocimiento correcto. En la Tabla

¹Tanto por el clasificador LIRA como para el PCNC la ventana respectiva es cuadrada, por lo que el valor del parámetro h es igual al de w . Por esta razón de aquí en adelante se da sólo el valor para w .

Tabla 3.5: Resultados de pruebas del clasificador LIRA con variación de los parámetros p y n sobre las bases de datos A, B y D para distintos valores de $p + n$.

Parámetros p/n , $p + n = 6$	1/5	2/4	3/3	4/2	5/1	-	-
BD-A (%):	63	83	82	78	33	-	-
BD-B (%):	82	88	91	92	84	-	-
BD-D (%):	77	85	85	87	80	-	-
Parámetros p/n , $p + n = 7$	1/6	2/5	3/4	4/3	5/2	6/1	-
BD-A (%):	33	73	90	89	66	43	-
BD-B (%):	80	89	88	94	91	87	-
BD-D (%):	74	81	84	85	83	81	-
Parámetros p/n , $p + n = 8$	1/7	2/6	3/5	4/4	5/3	6/2	7/1
BD-A (%):	43	58	80	83	86	72	55
BD-B (%):	78	80	87	83	86	85	82
BD-D (%):	72	82	81	85	87	78	80

3.6 se resumen las pruebas realizadas para este experimento con las cinco corridas para cada uno de los 3 conjuntos de parámetros descritos, los resultados obtenidos y el promedio obtenido para cada caso.

Tabla 3.6: Resumen de resultados de pruebas para obtener el mejor conjunto de parámetros LIRA para la base de datos A. \bar{x} es el promedio y σ es la desviación estándar.

Parámetros	Valores					Construcciones/(%)					\bar{x}	σ
	w	η	N	p	n	1	2	3	4	5	(%)	(%)
Base	15	1.0	175 000	4	3	81	89	91	92	90	88.6	3.93
Mejor en pruebas	15	0.9	175 000	4	3	85	91	87	88	91	88.4	2.33
Mejores aislados	15	0.9	200 000	3	4	78	85	89	85	90	85.4	4.22

...

Para el parámetro D_c que es la distancia de correlación del codificador del PCNC los resultados obtenidos se muestran en la Tabla 3.7. Se tiene que para la base de datos A hubo un empate con D_c igual a 2 y 5, mientras que para la base de datos B el mejor resultado fue con $D_c = 6$ y para la base de datos D existió también un empate para D_c igual a 4 y 8. Estos resultados nos dicen, para la tarea particular que se trata, que la distancia de correlación debe tener un valor pequeño menor que 10 pero mayor que uno.

Tabla 3.7: Resultados de pruebas del PCNC con variación del parámetro D_c sobre las bases de datos A, B y D.

Parámetro	D_c	1	2	4	5	6	8	10	15	20	25
BD-A	%	70	93	91	93	92	90	90	88	87	74
BD-B	%	84	91	93	94	95	92	89	91	82	82
BD-D	%	77	72	85	82	83	85	78	82	72	76

El parámetro q es un factor del CDT que combina las distintas propiedades encontradas para una imagen determinada (Sec. 2.1.2.5). El resultado de las pruebas con este parámetro se muestran en la Tabla 3.8. El resultado de variar este parámetro sobre el resultado del PCNC es bastante irregular por lo que sólo puede entenderse tomando en cuenta las posibles variaciones por el concepto de unicidad (Sec. ??). Así se tiene que para la base de datos A el mejor resultado se tuvo para $q = 5$, para la base de datos B para $q = 10$ mientras que para la base de datos D el mejor resultado se obtuvo al no aplicar el CDT, esto es $q = 0$.

Tabla 3.8: Resultados de pruebas del PCNC con variación del parámetro q sobre las bases de datos A, B y D.

Parámetro	q	0	1	2	3	4	5	6	7	8	9	10	11	12	13
BD-A	%	90	92	93	93	92	96	93	94	87	88	86	91	93	93
BD-B	%	94	93	94	90	94	94	92	92	89	90	97	95	93	93
BD-D	%	85	80	82	83	80	84	80	82	78	80	80	79	83	82

Dados los anteriores resultados tenemos que el mejor conjunto de parámetros para la base de datos A se dio para los parámetros base pero con $q = 5$ (Tabla 3.8). Considérese también el conjunto de parámetros construido tomando los mejores parámetros que aisladamente dieron mejores resultados en los experimentos considerando además su eficiencia. Así, tomando estos dos conjuntos de parámetros junto con los parámetros base se realizó el siguiente experimento con cinco creaciones distintas para cada conjunto de parámetros, para de esta forma obtener resultados estadísticamente confiables. En la Tabla 3.9 se muestran estos resultados donde se ve que los mejores parámetros obtenidos aisladamente también constituyeron el mejor conjunto de parámetros. Adicionalmente es de notar que este conjunto de parámetros logró la menor desviación estándar en sus respuestas.

Tabla 3.9: Resumen de resultados de pruebas para obtener el mejor conjunto de parámetros PCNC para base de datos A. \bar{x} es el promedio y σ es la desviación estándar.

Parámetros:	Parámetros								Construcciones/(%)					\bar{x}	σ
	w	p	n	S	N	K	D_c	q	1	2	3	4	5	(%)	(%)
Base	10	5	4	1000	300000	20	5	2	93	85	93	93	92	91.2	3.12
Mejor en pruebas	10	5	4	1000	300000	20	5	5	95	90	86	90	88	89.8	2.99
Mejores aislados	12	4	3	2500	200000	20	5	5	93	93	94	88	94	92.5	2.24

El mismo procedimiento descrito se utilizó para obtener los mejores parámetros para el clasificador PCNC aplicado a las bases de datos B y D obteniendo un porcentaje de reconocimiento de 97% para la base de datos B con los parámetros $w = 10$, $p = 4$, $n = 3$, $S = 1500$, $N = 300000$, $K = 20$, $D_c = 6$ y $q = 10$. Para la base de datos D el mejor resultado obtenido fue de 91% de reconocimiento con los parámetros $w = 11$, $p = 4$, $n = 3$, $S = 2000$, $N = 400000$, $K = 15$, $D_c = 4$ y $q = 0$. Este último resultado con $q = 0$ implica que este PCNC no utilizó el CDT para lograr el mejor desempeño.

3.1.3. Distorsiones

Con el objetivo de mejorar la capacidad de reconocimiento del clasificador PCNC se realizó un par de pruebas empleando distorsiones de las imágenes originales del conjunto de entrenamiento para ampliarlo. La prueba se hizo de la misma forma que para el clasificador LIRA (Sec. ??). En la Tabla 3.10 se resumen estas pruebas.

Los resultados obtenidos con las distorsiones son similares a los obtenidos para el clasificador LIRA, sin embargo considerando el mejor porcentaje de reconocimiento del PCNC las distorsiones no contribuyeron a mejorar el comportamiento del PCNC y si se considera además el tiempo adicional requerido para crear las distorsiones y para el entrenamiento de éstas la conclusión es que su aplicación no es conveniente para la tarea que nos ocupa. La explicación a esto se da por el hecho que los clasificadores son entrenados para el reconocimiento de imágenes normalizadas y el agregar distorsiones sólo distrae la memoria del clasificador hacia casos que se apartan de tal normalización. A medida que los clasificadores reconozcan una pieza ligeramente desviada de su estado normalizado (Sec. ??) con respuesta similar a su estado normalizado entonces no podrá aplicarse tal clasificador para la correcta identificación del ángulo de orientación o posición cartesiana de tal imagen como se verá más adelante en la Sec. 3.3.

Tabla 3.10: Experimento usando distorsiones para aumentar el conjunto de entrenamiento del PCNC aplicado sobre la base de datos A.

	Número de prueba	
	1	2
Número de distorsiones	6	12
distorsiones horizontales	+1, -1	+2, +1, -1, -2
distorsiones verticales	+1, -1	+2, +1, -1, -2
distorsiones angulares	+1°, -1°	+2°, +1°, -1°, -2°
Ciclos de entrenamiento	30	30
Porcentaje de reconocimiento (%)	93	91

3.1.4. Ciclos de entrenamiento

Se realizó la misma prueba que para LIRA con los ciclos de entrenamiento. Se crearon dos PCNCs con idénticos parámetros para cada una de las base de datos utilizadas. En este experimento se tuvo una respuesta más rápida para alcanzar el reconocimiento máximo respecto a LIRA. Por esta razón las pruebas se hicieron con pasos de 5 en lugar de 10 ciclos de entrenamiento. En la Tabla 3.11 se muestran los resultados obtenidos. Para la base de datos A el reconocimiento máximo se alcanzó desde los 15 ciclos de entrenamiento, sin embargo se dio un resultado notable a solo 10 ciclos para la segunda prueba, pues se alcanzó un resultado de 93 % pero luego con ciclos adicionales cayó esta respuesta y no volvió a repetirse. Para la base de datos B la respuesta fue uniforme y se tuvo a los 15 ciclos para la prueba 3 y a los 25 ciclos para la prueba 4. Para la base de datos D sucedió algo similar pero con 25 y 30 ciclos para la primera y segunda de sus pruebas (pruebas 5 y 6 respectivamente). Se deduce que con 30 ciclos de entrenamiento se obtienen resultados correctos.

Tabla 3.11: Experimento con diversos ciclos de entrenamiento sobre seis clasificadores PCNC. Para cada base de datos los parámetros son los mismos. Las pruebas se realizaron en las bases de datos A, B y D.

Número de prueba:	No. de ciclos de entrenamiento					
	5	10	15	20	25	30
BD-A 1:	66	88	94	94	94	94
BD-A 2:	76	93	90	90	90	90
BD-B 3:	69	90	96	96	96	96
BD-B 4:	67	89	94	97	97	97
BD-D 5:	39	59	69	81	88	88
BD-D 6:	51	61	64	79	84	86

3.1.5. Aleatoriedad de los conjuntos para entrenamiento y prueba

Una prueba muy importante es la de poder entrenar y probar el clasificador sobre conjuntos de entrenamiento y prueba variables. Por ello se realizó un experimento con estos conjuntos tomados aleatoriamente de toda la base de datos en la misma proporción que se hizo originalmente, esto es, 50 % para las base de datos A y B y 72 % para la base de datos D para los conjuntos respectivos de entrenamiento y el resto para los conjuntos de prueba respectivamente. Para cada base de datos distinta se utilizó un PCNC distinto, pero para cada una de las diez pruebas de cada base de datos se utilizó el mismo PCNC. Los resultados obtenidos mostrados en la Tabla 3.12 dan cuenta de la estabilidad del PCNC en cuanto a una misma estructura particular. Los porcentajes para cada base de datos variaron poco, lo que se ve en la desviación estándar obtenida para cada caso.

Tabla 3.12: Resultados de pruebas con el PCNC con conjuntos de entrenamiento y prueba seleccionados aleatoriamente para las bases de datos A, B y D. Para cada base de datos se utilizó un mismo clasificador.

Prueba No.:	1	2	3	4	5	6	7	8	9	10	\bar{x}	σ
BD-A:	94	92	93	93	90	90	91	93	91	94	92.1	1.45
BD-B:	96	96	97	94	95	96	95	97	95	95	95.6	0.92
BD-D:	90	90	89	91	95	88	87	92	89	90	90.1	2.12

3.1.6. Mejores clasificadores PCNC

En esta sección se describen a detalle los resultados obtenidos con los mejores PCNC obtenidos para las bases de datos utilizadas. Primero que nada se muestra en la Tabla 3.13 un resumen de los parámetros utilizados para estos clasificadores así como la respuesta obtenida por ellos a detalle. Para todos ellos se utilizaron 30 ciclos de entrenamiento. En la Tabla se muestra además de los parámetros empleados en cada base de datos y el resultado porcentual exacto obtenido en el reconocimiento, un desglose del número de errores en cada caso por cada una de las clases utilizadas así como la contribución porcentual de cada clase en el error total de reconocimiento. Consúltense las Tablas ?? y ?? para interpretar las clases para cada base de datos. Tómese en cuenta que los percentiles totales no suman 100 % debido a la aproximación a dos dígitos decimales.

Tabla 3.13: Resumen de resultados de los mejores clasificadores PCNC para las bases de datos utilizadas. T, ciclos de entrenamiento. R, porcentaje de reconocimiento.

Base de datos:	Parámetros									T	R	Errores por clase/(%)							
	w	p	n	S	N	K	D_c	q				1	2	3	4	5	6	7	8
A	10	5	4	1000	300000	20	5	5	30	96.87	4/2.50	0	0	0	0	0	0	1/0.63	0
B	10	4	3	1500	300000	20	6	10	30	97.80	1/0.37	1/0.37	0	3/1.10	0	1/0.37	0	-	-
C	11	4	3	2000	400000	15	4	0	30	91.43	6/5.71	0	1/0.95	1/0.95	1/0.95	0	0	-	-

3.1.6.1. Base de datos A

El PCNC logró obtener un porcentaje de error para la base de datos A mayor a 96 %. Solo 5 imágenes de un total de 160 no fueron reconocidas correctamente. Estas imágenes se repartieron en sólo dos clases (base de tubito y tornillo cabeza cónica) mientras que para las otras seis clases el reconocimiento fue del 100 % (cono, eje de rotor, no pieza central, terminal de cable, tornillo allen y tornillo cabeza redonda). En la Fig. 3.1 se presentan dos ejemplos de cada clase de entre las imágenes reconocidas y en la Fig. ?? se presentan todas las imágenes que no se reconocieron correctamente.

...

3.1.6.2. Base de datos B

En la base de datos B existió el mejor resultado obtenido en el presente trabajo. Este resultado fue de 97.80 % de reconocimiento correcto. Dos ejemplos de cada una de las clases tomadas del grupo de 267 imágenes bien reconocidas se muestran en la Fig. ?. Las imágenes mal reconocidas se distribuyeron en cuatro de las siete clases de la base de datos B de la siguiente forma: un tornillo plano, tres tornillos allen grandes, una no pieza central y una arandela. Las clases tornillo allen chico, tornillo gota y tuerca se reconocieron sin errores. En la Fig. ?? se muestran las seis imágenes mal reconocidas.

...

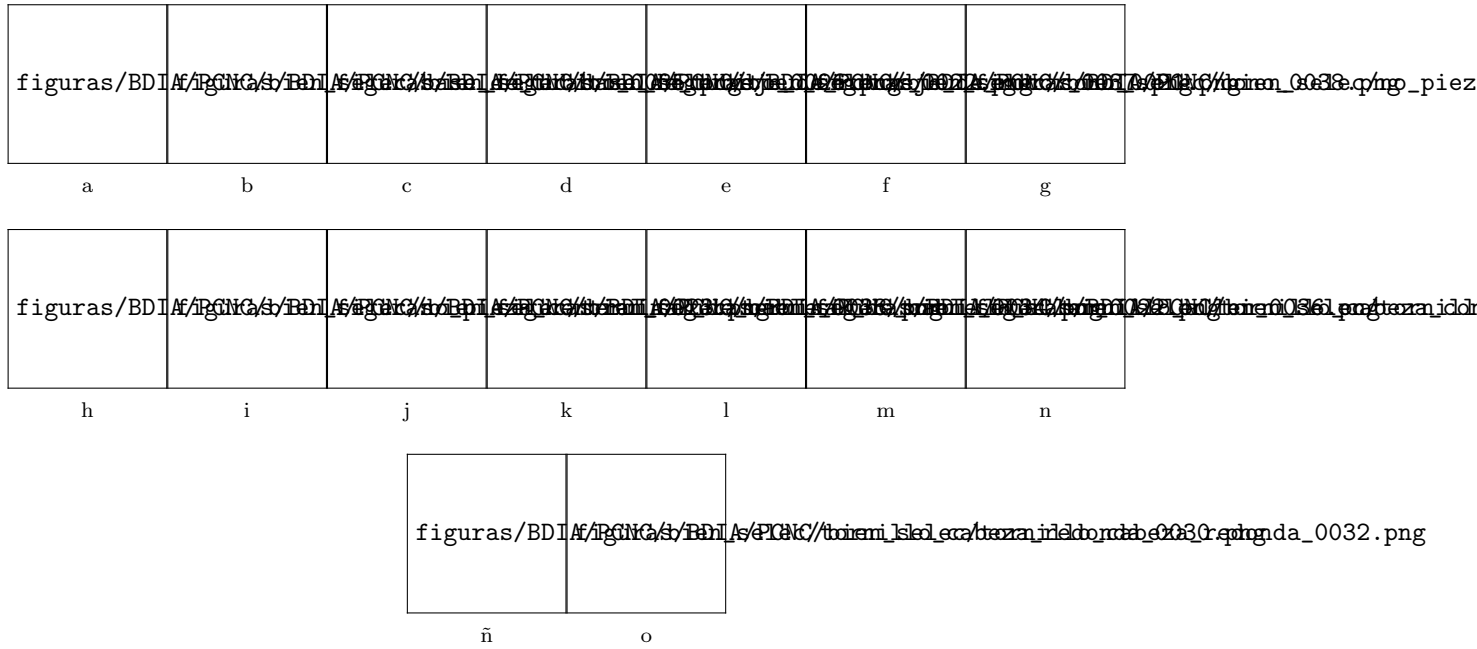


Figura 3.1: Ejemplos de imágenes correctamente reconocidas con el mejor PCNC para la base de datos A. Se muestran dos ejemplos por cada clase.

3.1.7. Prueba con conjunto de entrenamiento ampliado

La misma prueba que se aplicó para LIRA con un conjunto de entrenamiento ampliado de la base de datos D, se aplicó para el PCNC. Esta ampliación consiste en añadir las imágenes extras descritas en la Sección ?? . El resultado de entrenar al PCNC con 537 imágenes agregadas al conjunto de entrenamiento resultó en un porcentaje de reconocimiento sobre el conjunto de prueba de 93 %. Los errores fueron cinco, distribuidos en una no pieza central, dos tornillos allen grandes, un tornillo gota y un tornillo plano.

3.2. Comparación de clasificadores

El propósito de esta sección es comparar los resultados obtenidos de ambos clasificadores en las distintas pruebas realizadas con el objetivo de facilitar la elección de uno u otro atendiendo a la aplicación que se requiera.

3.2.1. Tiempos

El porcentaje de reconocimiento máximo que sobre una base de datos de imágenes logre un clasificador es el parámetro principal para medir su eficiencia. Sin embargo el consumo de recursos que se tenga es un factor muy importante también y en especial el tiempo. Es por ello que se muestran y comparan los tiempos empleados por ambos clasificadores utilizados para una misma base de datos y en un mismo equipo de cómputo. Con esto se tiene una clara y objetiva comparación entre los clasificadores, considerando además el resultado obtenido en cuanto al porcentaje de reconocimiento. Estos datos son presentados en la Tabla 3.14.

Todos estos tiempos dependen de los parámetros de los clasificadores, como se explicó el parámetro N para el clasificador LIRA y los parámetros S y N para el PCNC son los que más impactan en los tiempos de ejecución de las diversas tareas de estos clasificadores. Los resultados presentados en la tabla son para el mejor clasificador para cada uno de ellos para la base de datos A. Tenemos que

Tabla 3.14: Tiempos empleados por los clasificadores LIRA y PCNC sobre la base de datos A.

Clasificador	Reconocimiento (%)	Tiempos por tarea					
		Creación o carga	Entrenamiento (160 imágenes) codificación	Entrenamiento entrenamiento	Prueba (160 imágenes)	Guardar en disco	Reconocer una imagen
LIRA	93 %	3.08s	70s	154s	110s	0.65s	0.687s
PCNC	97 %	1.25s	198s	148s	222s	0.67s	1.387s

los tiempos de creación, carga y salvado de estos clasificadores es pequeño y considerando que en un trabajo continuo de reconocimiento sólo serán cargados una vez entonces este tiempo es mínimo y no impacta el desempeño de ninguno de los clasificadores. Lo mismo aplica para el tiempo de guardado. Para el caso del entrenamiento dividido en el tiempo de codificación y de entrenamiento de los códigos, se tiene que para el PCNC el tiempo de codificación es casi de tres veces el respectivo para LIRA, mientras que los tiempos de entrenamiento de códigos es muy similar. Si se considera que el entrenamiento de una determinada base de datos a utilizar se hace una sola vez, entonces, al menos para este caso, la ventaja de cuatro puntos porcentuales más de reconocimiento que tiene el PCNC pagará el costo en el tiempo del entrenamiento. Respecto al tiempo de prueba, el PCNC requiere un tiempo doble que LIRA. El mismo argumento que para el tiempo de entrenamiento puede esgrimirse para los tiempos de prueba. Sin embargo, los tiempos de reconocimiento de una imagen particular tienen la misma proporción entre los clasificadores. Es este tiempo el factor clave que dependiendo de la aplicación y uso particular que se requiera debe ser tomado en cuenta junto con el porcentaje de reconocimiento de cada clasificador para escoger uno de ellos. En general si la tarea no es crítica en tiempo debiera emplearse el PCNC debido a su poder superior de reconocimiento, mientras que si la tarea particular exige tiempo críticos y a su vez puede tolerar más errores entonces el clasificador LIRA debe ser utilizado.

3.2.2. Estabilidad en la creación de estructuras

En los experimentos de unicidad de LIRA y del PCNC se crearon y probaron diez clasificadores para cada uno con idénticos parámetros, ciclos de entrenamiento y conjuntos de entrenamiento y prueba. De estos experimentos la variable resultante de importancia es la desviación estándar. Téngase en cuenta que la desviación estándar es una medida de la variación de los resultados respecto a su promedio, por lo que a menor valor mejor confiabilidad se tiene en los resultados para un conjunto de parámetros dada a la hora de construir un clasificador con ellos. Los resultados presentados en las Tablas 3.2 y ?? se resumen y comparan en la Tabla 3.15. Se tiene que LIRA fue superior en estabilidad para las bases de datos B y D, mientras que para la base de datos A tuvo una estabilidad menor que el PCNC. En base a estos resultados no puede declararse ningún ganador entre los dos tipos de clasificadores probados en cuanto a esta característica.

Tabla 3.15: Comparación en la desviación estándar obtenida entre los clasificadores LIRA y PCNC sobre las tres bases de datos utilizadas.

Base de datos:	Clasificador	
	LIRA	PCNC
A	3.14 %	2.24 %
B	1.02 %	1.58 %
D	1.48 %	3.75 %

3.2.3. Parámetros

Para ambos clasificadores se hicieron una serie de experimentos variando un sólo parámetro a la vez para estudiar la forma en que varían los resultados obtenidos y para encontrar el mejor clasificador para cada base de datos. Sin embargo para cada una de estas pruebas es necesario construir un

nuevo clasificador por lo que siempre se tiene añadido al resultado obtenido la incertidumbre dada por la unicidad. Dado el argumento anterior y el estudio de las estructuras LIRA y PCNC se concluye que la búsqueda de parámetros para ambos clasificadores utilizados requiere de múltiples pruebas con cada base de datos a utilizar.

3.2.4. Ciclos de entrenamiento

De los experimentos hechos con los clasificadores para encontrar el mejor número de ciclos de entrenamiento se tiene que el clasificador LIRA requirió entre 30 y 70 ciclos de entrenamiento para alcanzar un valor estable mientras que el PCNC sólo requirió entre 15 y 30 ciclos. Esta diferencia no influye significativamente en el rendimiento total de los clasificadores pues ambos utilizan codificación del conjunto de entrenamiento para evitar codificar cada nuevo ciclo de entrenamiento. Siendo el tiempo de entrenar cada ciclo adicional para ambos clasificadores muy similar como puede verse en la tabla 3.14

3.2.5. Confiabilidad

Una vez construido un clasificador debe conocerse que tan confiable y estable es éste para reconocer objetos. Para analizar esta propiedad sobre las bases de datos y conocer en qué medida varía una construcción única de un determinado clasificador se realizaron los experimentos con conjuntos aleatorios de entrenamiento y prueba. El resumen y la comparación de estos experimentos se tiene en la Tabla 3.16

Se tiene que el PCNC fue en todos los casos más confiable que el clasificador LIRA. Para las bases de datos A y B superó por más del doble a LIRA. Este resultado es de lo más importante por que da cuenta de la capacidad de los clasificadores de ser entrenados y probados con imágenes aleatoriamente seleccionadas.

Tabla 3.16: Resumen de los experimentos de confiabilidad para los clasificadores LIRA y PCNC sobre las tres bases de datos utilizadas. Los resultados se expresan como la desviación estándar obtenida sobre las diez pruebas realizadas para cada base de datos y cada clasificador.

Base de datos:	Clasificador	
	LIRA	PCNC
A	3.40 %	1.45 %
B	2.15 %	0.92 %
D	3.37 %	2.12 %

3.2.6. Resultados

Luego de diversos experimentos se encontraron los mejores parámetros para cada clasificador y para cada base de datos. El desempeño del PCNC fue superior al clasificador LIRA para todas las bases de datos utilizadas. Sólo en la base de datos D este resultado fue casi igual. Un resumen de estos resultados se presenta en la Tabla 3.17.

Tabla 3.17: Resultados obtenidos por los clasificadores LIRA y PCNC sobre las bases de datos A, B y D. Los resultados expresan el porcentaje de reconocimiento sobre la base de datos respectiva.

Base de datos:	Clasificador	
	LIRA	PCNC
A	93.75 %	96.87 %
B	94.14 %	97.80 %
D	90.47 %	91.43 %

Del análisis y comparación de los resultados obtenidos por los mejores clasificadores para las bases de datos tenemos que para la base de datos A ambos clasificadores reconocieron el 100 % de las clases 2 a 6 (Tabla ??), es decir, las clases cono, eje de rotor, no pieza central, terminal de cable y tornillo allen. Para la base de datos D las clases 3 y 7 (tornillo allen chico y tuerca) fueron también reconocidas sin errores por ambos clasificadores. En cuanto a las base de datos B y D que están construidas con el mismo tipo de piezas y el mismo número de clases pero de complejidad distinta, el clasificador LIRA no tuvo errores para la clase 4 (tornillo allen grande) mientras que el PCNC no tuvo errores para ninguna de estas dos bases de datos para la clase 3 (tornillo allen chico).

Comparando los errores obtenidos por cada clasificador para cada una de las base de datos utilizadas tenemos que las cinco imágenes mal reconocidas con el PCNC para la base de datos A se encuentran dentro de las diez mal reconocidas por LIRA para esta misma base de datos. Es decir, estas cinco imágenes no pudieron ser reconocidas por ninguno de los clasificadores (Fig. ??).

Para la base de datos B la comparación da resultados distintos ya que sólo una imagen no pudo ser reconocida por ambos clasificadores Fig. ?? ??.

Por último, para la base de datos D cuatro imágenes no pudieron ser reconocidas por ninguno de las clasificadores. Estas imágenes son las mostradas en las Figs.: ?? ??, ??, ??, ?? que corresponden a las Figs: ?? ??, ??, ?? y ?? respectivamente.

Por otra parte ambos clasificadores no tuvieron problema en reconocer las piezas que ubicándose cerca del extremo de la escena² poseen una parte blanca que rellena la falta de imagen original. Ejemplos de estas imágenes bien reconocidas se tienen en la Fig. ?? b, Fig. ?? ??, Fig. 3.1 n y Fig. ?? ??.

Otra cualidad muy importante de ambos clasificadores ha sido poder reconocer piezas parcialmente obstruidas como puede verse en las Figs. ?? ??, ?? y Fig: ?? ??.

3.3. Búsqueda y reconocimiento de posición

Uno de los objetivos del presente trabajo es la creación de un sistema automático de localización de piezas o SVT (Sec. 1.3). En este punto del trabajo ya se han expuesto dos métodos de identificación de piezas constituidos por los clasificadores LIRA y PCNC. Cualquiera de estos clasificadores ha demostrado, en los experimentos expuestos en las secciones anteriores, resultados suficientemente buenos por lo que cualquiera de ellos puede ser usado como base en el método de localización de piezas. Una vez que el clasificador seleccionado es entrenado con una base de datos particular que contiene imágenes de las piezas con la que se desea trabajar, puede aplicarse el método de localización de piezas sobre una imagen o escena particular. Dicho método se ha expuesto en la Sección ??.

Como ejemplo se tomó el mejor clasificador LIRA para la base de datos A entrenado con las siete clases de esta base de datos y se realizaron distintas búsquedas para localizar una determinada pieza en alguna imagen dada.

En la Fig. 3.2 mostramos dos ejemplos de imágenes reconocidas. Una de las imágenes contiene dos conos reconocidos (conos 1, 2 en la Fig. 3.2(a). En la Fig. 3.2 (b) la imagen contiene tres terminales de cable reconocidas (1, 2, 3).

El pequeño cuadro blanco en la figura representa el lugar donde la pieza solicitada ha sido reconocida y el cuadro grande representa la ventana correspondiente de búsqueda asociada a esta posición.

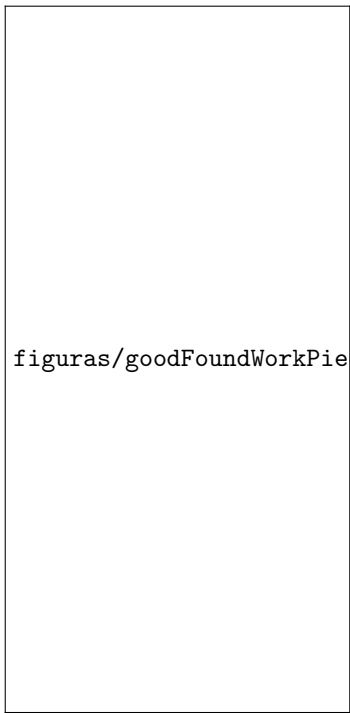
Al pie de las imágenes puede leerse las coordenadas de la posición y la orientación encontradas por el sistema para cada una de las piezas reconocidas.

El clasificador neuronal es un método flexible, esto significa que el sistema reconoce una pieza fuera de su centro, pero suficientemente cerca del cuerpo de la pieza.

Algunas veces el sistema puede reconocer una misma pieza varias veces. En la Fig. 3.3(a) se muestra un reconocimiento múltiple de una pieza (casos 1 y 3).

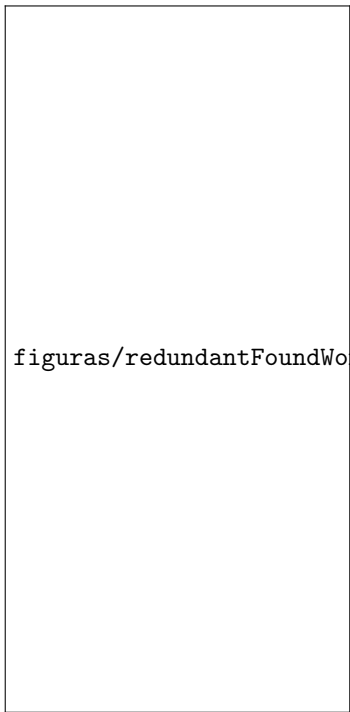
Esto sucede por que el sistema puede encontrar una pieza lejos de su centro, por ejemplo las marcas 2 y 3 en la Fig 3.3(a) y 1, 7 en la Figura 3.3(b).

²Se entiende por escena la imagen original de la cuál se recortaron las imágenes para formar las bases de datos o sobre la cuál se pretende localizar una pieza determinada.



figuras/goodFoundWorkPieces.jpg

Figura 3.2: Localización y reconocimiento de piezas.



figuras/redundantFoundWorkPieces.jpg

Figura 3.3: Localización y reconocimiento de piezas con redundancia.

En otros casos, esto pasa porque los parámetros de búsqueda (Δx , Δy , $\Delta\theta$) tienen valores muy grandes, por ejemplo la marca 1 en la Fig. 3.3(a) y las marcas 2, 4, 8 y 9 en la Figura 3.3(b).

La marca 4 de la Fig. 3.3(a) y la marca 6 de la Fig. 3.3(b) tienen suficiente precisión para el trabajo de manipulación.

Dos ejemplos de imágenes con piezas que se tocan una a otra se presentan en la Fig. 3.4. En la Fig. 3.4(a) el sistema encontró tres piezas (obsérvese que el punto de localización está lejos del centro de estas piezas). En la Fig. 3.4(b) el sistema presenta redundancia. En este caso el sistema puede filtrar los resultados considerando la mejor respuesta obtenida en la salida del clasificador LIRA, lo cual significa que la pieza con las marcas 2 y 3 se considerará reconocida en la marca número 2 que está mucho mejor posicionada con respecto al centro de la pieza.



Figura 3.4: Localización y reconocimiento de piezas que se tocan.

Los resultados obtenidos son buenos para permitir la manipulación, sin embargo el método de búsqueda debe ser mejorado para incrementar la precisión de la localización y del ángulo de la pieza a ser reconocida.

De los resultados presentados se tiene que en el sistema desarrollado existen dos problemas específicos de reconocimiento que son:

1. La redundancia en el reconocimiento y
2. La precisión del localizador de piezas.

Respecto a la redundancia en el reconocimiento se tiene que esto sucede debido a que los clasificadores utilizados tienen suficiente flexibilidad como para reconocer una pieza aún lejos de su centro. Esto en primera instancia al momento de la localización representa una ventaja, sin embargo cuando se quiere obtener la localización exacta de la pieza entonces la misma característica representa un problema. Una forma de resolver este problema es comparando las respuestas neuronales de los clasificadores de la salida ganadora y seleccionando la que mayor valor tenga, la cuál será a su vez el punto más cercano al centro de la pieza. Sin embargo esta situación resta eficiencia al método de localización por lo que más estudios deben hacerse al respecto para reducir este problema.

Con respecto a la precisión del localizador, ésta depende tanto de la estructura particular del clasificador que se utilice, como de su entrenamiento así como de los parámetros propios del localizador. Si los parámetros que definen el avance de la ventana de búsqueda se reducen la precisión de búsqueda aumentará sin embargo mayor tiempo de cómputo será necesario; por el contrario, si estos parámetros se aumentan la precisión baja y el tiempo aumenta. Dado lo anterior es importante encontrar los valores óptimos para los parámetros del localizador. Además de lo anterior pueden introducirse mejoras al método de localización como la realización de una búsqueda fina una vez que se ha localizada una región donde exista una pieza de interés.

Considerando los resultados de los experimentos presentados con el localizador así como los dos problemas que en ocasiones se presentan tenemos que el sistema es un excelente paso inicial para la localización e identificación de piezas; sin embargo más investigación y desarrollo debe de hacerse en esta área particular para mejorar las prestaciones del sistema de localización.

Conclusiones

En el Laboratorio de Micromecánica y Mecatrónica se desarrolla microequipo de bajo costo para producir dispositivos micromecánicas. Una característica que permite el bajo costo es la utilización de algoritmos adaptivos para compensar la mayor precisión respecto a los equipos de alto costo. Entre los algoritmos adaptivos destacan los basados en visión por computadora.

...

Para la implementación de los dos clasificadores neuronales, para el localizador y para la creación de las bases de datos de las imágenes se desarrollaron diversos módulos de software. Todo este software constituyó una herramienta indispensable para la investigación y los experimentos realizados. El software se desarrolló mediante un lenguaje de Programación Orientado a Objetos, C++ estándar, desde el sistema operativo GNU/Linux.

Apéndices

Apéndice A

Software

Con el objetivo de contar con una herramienta para la investigación y los experimentos sobre los clasificadores neuronales LIRA y PCNC, así como para el localizador y la creación de las bases de datos, se crearon varios paquetes de software. El software se desarrolló mediante un lenguaje de Programación Orientado a Objetos (POO). Los lenguajes orientados a objetos permiten contar con módulos de software reutilizables, flexibles y de fácil mantenimiento. Esto hace posible que la investigación y desarrollo sean más eficientes en lo que a software se refieren y pone a disposición inmediata aquellos módulos de software que han sido probados con buenos resultados ya sea para un sistema completo, un sistema alternativo o para realizar otro tipo de experimentos.

El software desarrollado en primera instancia se compone de cuatro módulos: Optik, RNA, Localizador e Interfaz. El módulo Optik se encarga de la creación de bases de datos de imágenes, el módulo RNA es la implementación del clasificador neuronal LIRA, el módulo Localizador busca una pieza determinada en una imagen y la Interfaz es el módulo encargado de la intercomunicación entre los demás módulos así como con el usuario. En la Fig. A.1 se muestra un diagrama a bloques de estos módulos así como las interconexiones entre éstos y los archivos que se manejan, también se listan las funciones disponibles para el usuario. Se describen a continuación con más detalle cada uno de los módulos en los se hará referencia continua a la figura mencionada.

El software desarrollado posee muchas ventajas de la programación orientada a objetos (POO). Se utilizó el lenguaje de programación C++. En un inicio se usó el Ambiente integral de desarrollo Borland^{MR} y con ello algunos objetos predefinidos por Borland^{MR} fueron utilizados para la manipulación de imágenes y la creación de las interfaces gráficas.

Más tarde se decidió trasladar el código existente a C++ estándar por lo que se abandonó el uso de dicho ambiente de desarrollo. El uso del C++ estándar posibilita que el código pueda ser trasladado fácilmente a cualquier plataforma y a sistemas embebidos, lo anterior es especialmente importante para nuestro proyecto. Además, los costos de desarrollo se reducen al no depender de software comercial haciendo que el sistema sea más económico, característica que es una de las pautas más importantes en todo el trabajo desarrollado. Bajo esta nueva pauta de trabajo se desarrolló también el software PCNC.

La aplicación Optik está siendo desarrollado para GNU/Linux [REF] y las demás aplicaciones para el clasificador LIRA y el PCNC se desarrollaron en C++ estándar en este mismo sistema operativo. También el localizador de piezas. Si bien no se les desarrolló interfaz gráfica, esto se hizo premeditadamente debido a las siguientes razones:

1. Las buenas prácticas de programación moderna enseñan que la interfaz gráfica debe separarse totalmente de la implementación del software particular.
2. El poder que ofrece el hecho de poder correr el software desde la línea de comandos cuyo ejemplo más práctico se da en los archivos de procesamiento por lotes que posibilitaron la ejecución de decenas de experimentos en una sola orden y
3. La visión de que en algún momento del desarrollo futuro el sistema pueda ser embebido a un microcontrolador o algún otro hardware especializado.

A.1. Módulo Optik

Optik es un software que genera bases de datos de imágenes de objetos destinadas para entrenamiento y pruebas del clasificador neuronal LIRA. Se parte de imágenes generales de múltiples objetos ordenados aleatoriamente, con ayuda de marcas colocadas por el usuario, genera una base de datos de imágenes normalizadas mediante un proceso de extracción. imágenes normalizadas se refiere a que éstas tienen iguales características: contienen una pieza centrada y con orientación fija de 0° respecto a su eje mayor, son de dimensión constante y tienen una ventana circular que facilita la rotación (Fig. A.2). Las imágenes extraídas son nombradas de acuerdo al tipo de pieza correspondiente y a un número consecutivo. También el sistema crea un archivo MRK que contiene los datos de todas las marcas realizadas.

Antes de el proceso de colocación de muestras, deben definirse los nombres y otras propiedades de los distintos tipos de piezas con que se va a trabajar (i. e. dimensiones, peso y material), éstos datos son de utilidad al sistema para calcular automáticamente el centro de la pieza además de futuras operaciones. Esta información se almacena en el sistema en un archivo PZA y puede ser cargada cuando se requiera. Optik además realiza algunas tareas de preprocesamiento de imágenes, como extracción de contornos y conversión de imágenes de color a escala de grises.

Debido al hecho de que las marcas en las escenas son puestas por el usuario mediante el ratón, las imágenes muestras para las bases de datos no están centradas ni orientadas exactamente. Este no es un problema ya que los clasificadores utilizados son capaces de llevar a cabo la tarea de reconocimiento con muestras imperfectamente centradas o giradas, sobre todo en conjuntos grandes de entrenamiento.

Este software en primera instancia se elaboró con Borland^{MR} C++, luego se decidió en colaboración con otros colegas pasarlo a sistema operativo GNU/Linux [23].

A.2. Módulo RNA

El módulo RNA implementa al clasificador neuronal LIRA, es decir, es la realización de la red neuronal completa, neurona a neurona, sus interconexiones y toda su funcionalidad. Este es el módulo más elaborado, su eficiencia es crítica, por lo que fue necesario cuidar dos aspectos fundamentales, memoria y velocidad. Un clasificador neuronal puede necesitar cientos de miles de neuronas y una magnitud mayor de interconexiones además de ser indispensable tiempos de ejecución total aceptables, tanto en las fases de entrenamiento como en las de prueba, siendo crítico en una aplicación real de reconocimiento (módulo Localizador). Para cuidar la velocidad se decidió programar este módulo y por extensión todo OptikRNA con lenguaje C++, es decir, se utilizó la velocidad y el poder de C combinado con las ventajas de la POO. Se utilizaron punteros, estos permiten operar a bajo nivel mejorando la velocidad y realizar las interconexiones entre las distintas clases utilizadas más eficientemente. Para cuidar la eficiencia evitando operaciones de punto flotante se utilizaron únicamente números enteros. Este módulo se constituye de diversas clases, las principales, junto con su relación de herencia se muestran en la Fig. A.3.

Existe una súper clase llamada RNA la cual construye con las clases descritas antes y otras menores no mencionadas el clasificador neuronal LIRA. Esta clase ofrece las mismas funciones que tiene el clasificador Lira descritas en la Sección 2 además de otras necesarias para su implementación y funcionamiento, entre las principales están: creación, codificación, entrenamiento, reconocimiento y borrado de memoria (pesos a cero). La clase RNA controla completamente a todo el módulo y es con la cuál se comunica realmente el módulo Interfaz.

A.3. Módulo Localizador

El módulo Localizador se encarga de buscar una pieza requerida en una imagen fuente y definir la posición de ésta. La pieza que será capaz de localizar este módulo debe estar en el conjunto de piezas con que el clasificador se haya entrenado previamente. La imagen fuente no tiene por que

ser una imagen ocupada para la extracción de imágenes normalizadas, puede ser cualquier imagen siempre que contenga el objeto a buscar en la misma escala en que existe en las imágenes ocupadas. Sobre una imagen dada, este módulo aplica un algoritmo de búsqueda que consiste en ir tomando subimágenes empezando por el centro y desplazándose en forma espiral (Fig. 5b). Para cada posición se pide al módulo RNA identificar la pieza requerida y si no se encuentra se rota un cierto ángulo y se repite el proceso hasta encontrar la pieza o cuando una rotación es completada, si no se encuentra, se continua el desplazamiento espiral hasta encontrar la pieza buscada o alcanzar los límites de la imagen. Los desplazamientos lineales y angulares pueden ser definidos por el usuario.

El módulo Localizador constituye una aplicación práctica concreta en microensamble y puede funcionar independientemente de los módulos Optik e Interfaz con el fin de ser acoplado a sistemas automáticos de manipulación de piezas. En la Fig. 5c se muestra un resultado de la búsqueda de una pieza ("tornillo de cabeza redonda") sobre una imagen que contiene diversas piezas. El sistema despliega las coordenadas de la pieza así como la orientación. Cuando la pieza se localiza lejos de su centro, la orientación es errónea, por esta razón este módulo debe ser mejorado.

A.4. Interfaz

El módulo Interfaz es el único módulo que se comunica con el usuario, además intercomunica los demás módulos para administrarlos. Este módulo en si es una Interfaz Gráfica de Usuario (IGU), cuenta con todas las funciones disponibles de OptikRNA y con áreas para desplegar imágenes y resultados. En la Fig. A.4 se muestra ésta interfaz. El área llamada "parámetros del clasificador.^{es}" a donde el usuario ingresa los parámetros para la creación de un clasificador neuronal LIRA en particular. Abajo está el panel de funciones básicas de la RNA, desde aquí se envían los comandos para el módulo RNA, como crear, guardar, cargar y reconocer. En el área de mensajes se despliega información general del clasificador cargado. En el panel de funciones avanzadas existen funciones para el módulo RNA como entrenar, probar, asignar bases de datos y entrenar-probar automáticamente. En el área de resultados se muestran los resultados de las pruebas o el reconocimiento de piezas. Por último, desde el panel del módulo Localizador accedamos a las funciones y parámetros de este módulo. En la figura referida se muestra un clasificador cargado junto con su información general y los resultados de una prueba aplicada. En el área vacía es donde se despliega la imagen utilizada por el Localizador. Las funciones del módulo Optik están en otra IGU que no se muestra. El usuario trabaja con el módulo de software de redes neuronales que implementa el clasificador LIRA, esto lo hace a través de la IGU Rna (Fig. A.4). Con la ayuda de esta interfaz el usuario puede crear el clasificador neuronal LIRA, entrenar, probar y usar el clasificador así como utilizar el localizador de piezas. La IGU Rna con un clasificador entrenado ya cargado es mostrada en la Fig. A.4. En el cuadro de texto de la izquierda se muestra información sobre el clasificador. Los resultados se dan en el cuadro de texto de la derecha: el porcentaje de reconocimiento obtenido y los nombres de los archivos de las muestras que el sistema no pudo reconocer.

En el centro de la Fig. A.6 se muestra una escena en la cuál se ha llevado a cabo el procesamiento con el localizador de piezas. Un clasificador previamente entrenado y probado ha sido ya cargado. En esta misma imagen se muestra también en el cuadro de texto de la derecha, una marca sobre la pieza localizada así como las coordenadas y orientación asociada a la misma.

A.5. Implementación de LIRA

Our neural classifier software is based on integer numbers operations in order to avoid large time effort that is necessary for floating point operations. The most general classes were made, that means that the same software modules can be used to create different neural network topologies. The user interface was made specially for LIRA neural classifier. It is not hard to use this software modules to construct other types of neural networks or make changes to the current topology, e. g. add more layers.

The most important classes created for artificial neural network realization were: Dentrita, Neuron, NeuralSet and Rna.

The Dentrta class is a basic one. It has only two attributes, stimulus and weight. Dentrta instances are used widely by neuron objects in order to create fully functional neurons. In Fig. A.7 the Neuron class and its derived classes are shown. The neuron types used in LIRA classifier are ON, OFF, AND and adding neurons. The ON and OFF neurons are one input (OI) neurons and the rest are several input (SI) neurons. The Neuron class is the fundamental part of the neural network software.

In order to construct neuron sets a general NeuralSet class was created. Rna is the class that combines all mentioned classes. This class we use to construct a LIRA neural network. Examples of the parameters are the number of neurons or groups in each layer, the number of ON and OFF neurons in each group, the input vector and the output classes. The Rna object contains information about itself. It is possible to store the complete neural network including all its internal parameters, to load a previously stored neural network and to perform training and recognition processes.

The neural classifier is controlled by an object called Rna-Interface. That interrelates the user by means the GUI with the databases used for training and testing of the classifier.

A.6. Software de línea de comandos

A.6.1. *lira2007*

A continuación se presenta la ayuda propia del software *lira2007* que ha sido usado para implementar y experimentar con el clasificador LIRA así como su interacción con las bases de datos.

Example to use follow()-functions:

USAGE:

```
--help, -h
    display this help.
```

*** Setup ***

```
--create adn='<RNA name (string)> <tamVectEnt> <ImageWidth> <windowWidth> <windowHeight> <numGroup> <elemXGroup> <numNeuON> <numNeuOFF> <eta (double)> <numClasses>', you can use -c instead of --create. Non especified are (int).
```

For example use:

```
lira -c adn='liraSUN 22500 150 15 15 170000 7 4 3 1.0 8'
    Create a Lira Neural Classifier from scrath.
```

```
--load <filename.rna>, -l <filename.rna>
    load a neural classifier from specified file.
```

```
--info, -i
    displays info about the loaded classifier.
```

```
--classes-file <filename.ent>, -cf <filename.ent>
    assign the classes file to LIRA classifier.
```

*** Preparing ***

```
--train-dir <dir>, -td <dir>
    assign the training directory, default is "./train".
```

```
--code-dir <dir>, -cd <dir>
    assign the code directory, default is "./code".
```

```
--code, -k
    code all the images in the train directory, they will put in the code directory
```

```
--reset, -kill
    reset the internal connections of the loaded ANN, Erase its memory
```

```
--train num, -t num
    train using the coded images from the given code dir using "num" cycles.
```

Default is 40

--prove-dir <dir>, -pd <dir>
 assign the prove directory, default is "./prove".

--images-dir <dir>, -id <dir>
 assign the images directory, default is "./images".

--train-percentage num, -tp num
 Percentage of images in images-dir to be selected for the training set.
 Default is 50%

*** Using ***

--prove, -p
 proves the classifier with the images files stored in the "proving directory".

--recognize <filename.png>, -r <filename.png>
 recognize a class in the given normalized image file.

pos='<x (int)> <y (int)> <0 (double)>', Parameters especification for recognized a big (not normalized) image in certain point and orientation. All values should be (int).
 You might specified a "filename.png" by -r to look for.
 Example use:
 lira -l lira.rna -r imagen.png pos='10 20 -45.0'
 An image file called "cut.png" will be created with the used subimage.

*** Ending***

--save, -s
 Save the current classifier to a .rna file.

--close, -q
 (NOW FAIL!) Save the current classifier to a .rna file.
 close the current classifier loaded in memory.

--output, -o
 Print values from the output layer.

*** Search ***

--searchImage <filename.png>, -si <filename.png>
 Search for some recognized clase.

--searchClass <className>, -sc <className>
 Class to be search by the Search command. Use "prueba" to make a prove and "cualquiera" to search anyone. Default is anyone

--searchQuantity num, -sq num
 Number of objects to search for. Default is 1.

--searchStep num, -ss num
 Step in pixels for the searcher to jump (/x & /y). Default is 20.

--searchStepAngle num, -sa num
 Angle in degrees for the searcher to jump. Default is 45.

*** Tools ***

--distortion dis='<dx> <nx> <dy> <ny> <d0> <n0>', you can use -d instead of -distortion. All parameters should be (int).
 You might specified the "training dir".
 For example use:
 lira -td ./trainimages -d dis='5 2 5 2 5 2'

A.6.2. pcnc2007

A continuación se presenta la ayuda propia del software *pcnc2007* que ha sido usado para implementar y experimentar con el PCNC así como su interacción con las bases de datos.

Example to use follow()-functions:

USAGE:

```
--help, -h
    display this help.
```

*** Setup ***

```
--create adn='<PCNC name (string)> <method> <Tmin> <Tmax> <w> <h> <p> <n> <S>
<N> <K> <Dc> <q> <numClasses>', you can use -c instead of --create. Non espe
cified are (int).
```

For example use:

```
peco -c adn='PCNCprueba 0 1 32000 5 5 3 2 500 1000 12 8 5 8'
    Create a Permutative Code Neural Classifier (PCNC) from scrath.
--load <filename.pcnc>, -l <filename.pcnc>
    load a PCNC from specified file.
--info, -i
    displays info about the loaded PCNC.
--classes-file <filename.ent>, -cf <filename.ent>
    assign the classes file to PCNC.
```

*** Preparing ***

```
--train-dir <dir>, -td <dir>
    assign the training directory, default is "./train".
--code-dir <dir>, -cd <dir>
    assign the code directory, default is "./code".
--code, -k
    code all the images in the train directory, they will put in the code
directory
--reset
    reset the internal connections of the loaded PCNC, Erase its memory
--train num, -t num
    train using the coded images from the given code dir using "num" cycles
. Default is 40
--prove-dir <dir>, -pd <dir>
    assign the prove directory, default is "./prove".
--images-dir <dir>, -id <dir>
    assign the images directory, default is "./images".
--train-percentage num, -tp num
    Percentaje of images in images-dir to be selected for the training set.
Default is 50%
```

*** Using ***

```
--prove, -p
    proves the PCNC with the images files stored in the "proving directory".
--recognize <filename.png>, -r <filename.png>
    recognize a class in the given normalized image file.
```

*** Ending***

```
--save, -s
```

```

        Save the current PCNC to a .pcnc file.
--close, -q
        (NOW FAIL!) Save the current PCNC to a .pcnc file.
        close the current PCNC loaded in memory.

```

A.6.3. Potencial para la experimentación

Como un ejemplo del uso ventajoso del software de línea de comandos se presenta uno de los múltiples archivos de procesamiento por lotes¹ empleado para realizar decenas de experimentos con cambio de parámetros de forma automática, es decir, en una sola corrida.

```

#!/bin/bash

#Decenas de pruebas para estudiar comportamiento de parámetros
#La base es el Mejor:
#../bin/pcnc2007 -c adn='PCNCmejorBD-A 1 0 65535 10 10 5 4 1000 300000 20 5
  2 8' -cf BD-A.ent -td ../muestras/entrenamiento -cd ../muestras/codigo
-pd ../muestras/prueba -k -t 40 -p -s

#####PARAMETRO W
#Inicializacion
PARAM=W
PARw=10
PARp=5
PARn=4
PARS=1000
PARN=300000
PARK=20
PARDc=5
PARq=2
#Usando "for" para probar distintos parámetros
for PARw in 05 08 09 11 12 15 20;
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARw" *****"
    echo "*****"
    #Hace todo de una vez, también salva
    ../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARw' 1 0 65535 '$PARw' '$PARw'
    '$PARp' '$PARn' '$PARS' '$PARN' '$PARK' '$PARDc' '$PARq' 7' -cf BD-D.ent -td
    ../muestrasBD-D/entrenamiento -cd ../muestrasBD-D/codigo -pd ../mues
    trasBD-D/prueba -k -t 30 -p -s

# ../bin/pcnc2007 -c adn='PCNCparam'$PARAM' 1 0 65535 '$PARw' '$PARw' '$PAR
p' '$PARn' '$PARS' '$PARN' '$PARK' '$PARDc' '$PARq' 7' -cf BD-D.ent -i
done

#####PARAMETRO K
#Inicializacion
PARAM=K
PARw=10
PARp=5

```

¹Más conocidos por su denominación en inglés: *scripts*

```

PARn=4
PARS=1000
PARN=300000
PARK=20
PARDc=5
PARq=2
#Usando "for" para probar distintos parámetros
for PARK in 05 10 15 25 30;
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARK" *****"
    echo "*****"
    #Hace todo de una vez, también salva
    ../../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARK' 1 0 65535 '$PARw' '$PARw'
    '$PARp' '$PARn' '$PARS' '$PARN' '$PARK' '$PARDc' '$PARq' 7' -cf BD-D.ent -td
    ../../muestrasBD-D/entrenamiento -cd ../../muestrasBD-D/codigo -pd ../../mues
    trasBD-D/prueba -k -t 30 -p -s
done

#*****PARAMETRO Dc
#Inicializacion
PARAM=Dc
PARw=10
PARp=5
PARn=4
PARS=1000
PARN=300000
PARK=20
PARDc=5
PARq=2
#Usando "for" para probar distintos parámetros
for PARDc in 02 04 06 08 10 15 20 25;
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARDc" *****"
    echo "*****"
    #Hace todo de una vez, también salva
    ../../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARDc' 1 0 65535 '$PARw' '$PARw'
    '$PARp' '$PARn' '$PARS' '$PARN' '$PARK' '$PARDc' '$PARq' 7' -cf BD-D.ent -td
    ../../muestrasBD-D/entrenamiento -cd ../../muestrasBD-D/codigo -pd ../../mues
    trasBD-D/prueba -k -t 30 -p -s
done

#*****PARAMETRO q
#Inicializacion
PARAM=q
PARw=10
PARp=5
PARn=4
PARS=1000
PARN=300000
PARK=20
PARDc=5

```



```

PARq=2
#Usando "for" para probar distintos parámetros
for PARq in 00 01 03 04 05 10;
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARq" *****"
    echo "*****"
    #Hace todo de una vez, también salva
    ../../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARq' 1 0 65535 '$PARw' '$PARw'
    '$PARp' '$PARn' '$PARs' '$PARn' '$PARK' '$PARdc' '$PARq' 7' -cf BD-D.ent -td
    ../../muestrasBD-D/entrenamiento -cd ../../muestrasBD-D/codigo -pd ../../mues
    trasBD-D/prueba -k -t 30 -p -s
done

*****PARAMETROS p y q
#Inicializacion
PARAM=PyN
PARw=10
PARp=5
PARn=4
PARs=1000
PARn=300000
PARK=20
PARdc=5
PARq=2
#Usando "for" para probar distintos parámetros
for PARpn in "3 6" "4 5" "6 3" "7 2" "2 5" "3 4" "4 3" "5 2" "6 1" "4 7" "5 6"
    "6 5" "7 4" "8 3" "9 2";
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARpn" *****"
    echo "*****"
    #Variable auxiliar para nombrar sin espacios
    PARpnNom=${PARpn/ /y}
    #Hace todo de una vez, también salva
    ../../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARpnNom' 1 0 65535 '$PARw' '$P
    ARw' '$PARpn' '$PARs' '$PARn' '$PARK' '$PARdc' '$PARq' 7' -cf BD-D.ent -i -td
    ../../muestrasBD-D/entrenamiento -cd ../../muestrasBD-D/codigo -pd ../../mues
    trasBD-D/prueba -k -t 30 -p -s
done

*****PARAMETRO N
#Inicializacion
PARAM=N
PARw=10
PARp=5
PARn=4
PARs=1000
PARn=300000
PARK=20
PARdc=5

```

```

PARq=2
#Usando "for" para probar distintos parámetros
for PARN in 100000 200000 250000 400000 500000;
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARN" *****"
    echo "*****"
    #Hace todo de una vez, también salva
    ../../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARN' 1 0 65535 '$PARw' '$PARw'
    '$PARp' '$PARn' '$PARs' '$PARn' '$PARK' '$PARDc' '$PARq' 7' -cf BD-D.ent -td
    ../../muestrasBD-D/entrenamiento -cd ../../muestrasBD-D/codigo -pd ../../mues
    trasBD-D/prueba -k -t 30 -p -s
done

#*****PARAMETRO S
#Inicializacion
PARAM=S
PARw=10
PARp=5
PARn=4
PARS=1000
PARN=300000
PARK=20
PARDc=5
PARq=2
#Usando "for" para probar distintos parámetros
for PARS in 0400 0600 0800 1200 1500 2000 2500 3000 4000 5000 6000 10000;
do
    echo "*****"
    echo "***** CORIDA: "$PARAM:$PARS" *****"
    echo "*****"
    #Hace todo de una vez, también salva
    ../../bin/pcnc2007 -c adn='PCNCparam'$PARAM'$PARS' 1 0 65535 '$PARw' '$PARw'
    '$PARp' '$PARn' '$PARs' '$PARn' '$PARK' '$PARDc' '$PARq' 7' -cf BD-D.ent -td
    ../../muestrasBD-D/entrenamiento -cd ../../muestrasBD-D/codigo -pd ../../mues
    trasBD-D/prueba -k -t 30 -p -s
done

#Crea el resumen de resultados
grep "El porcentaje" PCNCparam*.proveresults >> resumen.proveresults

#FIN

# LocalWords:  PARw

```

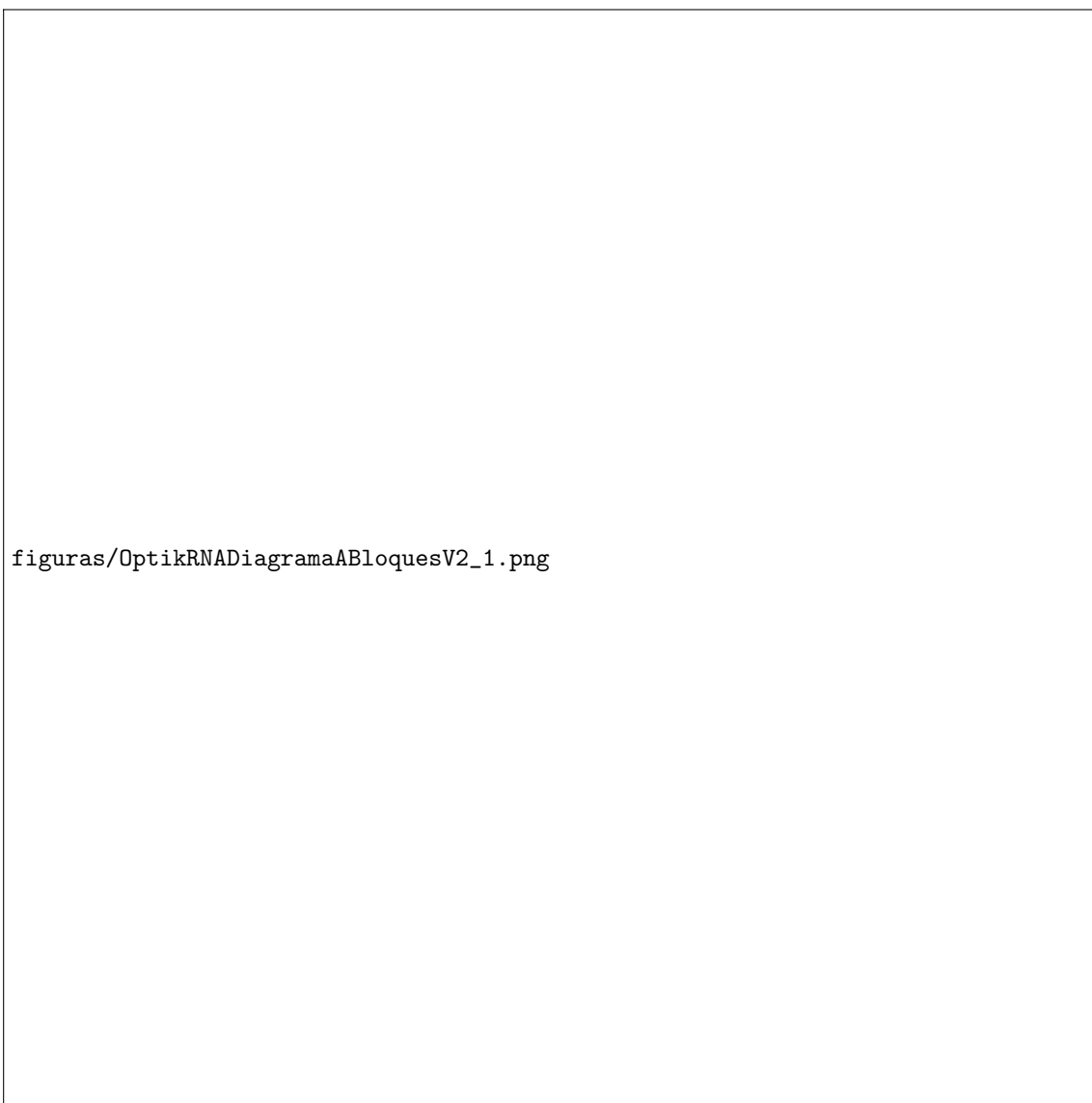


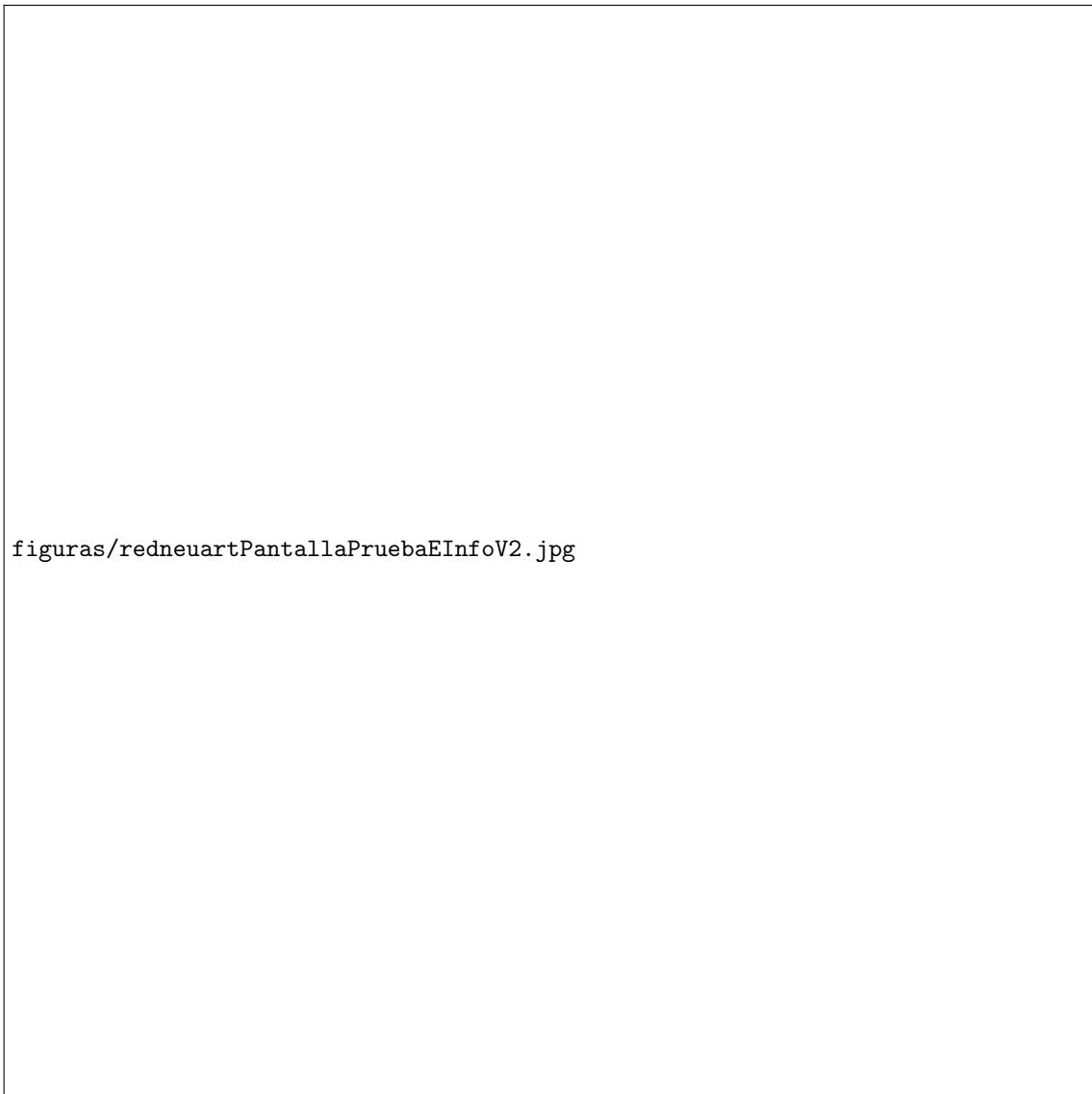
Figura A.1: Diagrama a bloques del software OptikRNA. Se muestran los cuatro módulos que lo componen, los archivos utilizados y sus principales funciones así como las intercomunicaciones entre todos estos elementos.



Figura A.2: IGU Optik, la cual permite marcar y extraer las muestras de imágenes a partir de las imágenes escena.

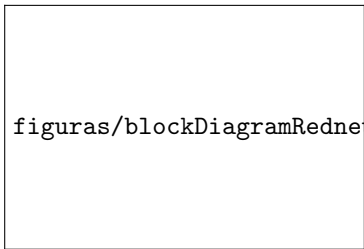


Figura A.3: Clases principales del módulo RNA y su relación de herencia. a) Clases a nivel neurona.
b) Clases a nivel capa.



figuras/redneuartPantallaPruebaEInfoV2.jpg

Figura A.4: Interfaz gráfica de usuario del software RNA.



figuras/blockDiagramRedneuartOptikV2.jpg

Figura A.5: Diagrama a bloques del software OptikRNA.



Figura A.6: IGU Rna. Localizador de piezas.

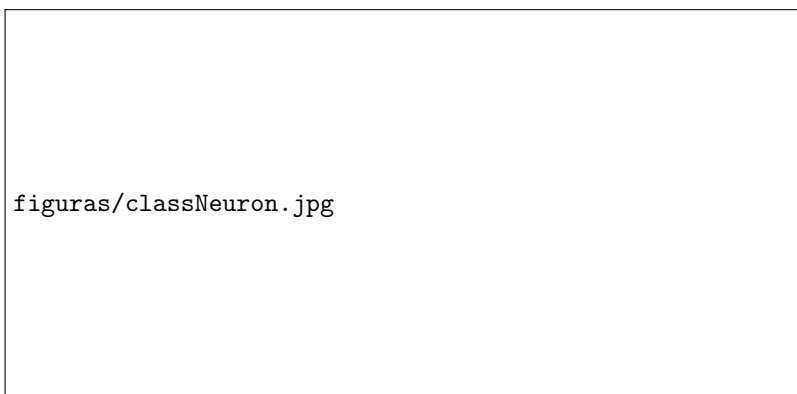


Figura A.7: Clase neurona y sus clases derivadas.

Apéndice B

Publicaciones

El presente trabajo dió lugar a las siguientes publicaciones y congresos nacionales e internacionales:

1. G.K. Toledo, E. Kussul, T. Baidyk. Neural classifier LIRA for recognition of micro work pieces and their positions in the processes of microassembly and micromanufacturing. The Seventh All-Ukrainian International Conference on Signal/Image Processing and Pattern Recognition, UkrObraz2004, Kiev, Ukraine, October 2004, pp. 1720.
2. G. Toledo, E. Kussul, T. Baidyk. Clasificador neuronal LIRA para reconocimiento de piezas de trabajo y sus posiciones en los procesos de microensamble y micromanufactura. Memorias del XIX Congreso de Instrumentación SOMI, Pachuca, Hidalgo, México, 25-29, octubre, 2004.
3. Ernst Kussul, Tatiana Baidyk, Gengis Kanhg Toledo Ramírez. Investigación y desarrollo del sistema de visión técnica para aplicaciones en micromaquinado y microensamble. Reporte técnico, CCADET-UNAM, 15 de junio de 2005, 29 p.
4. Gengis K. Toledo-Ramírez. Software orientado a objetos para investigación de redes neuronales. Memorias del XX Congreso de Instrumentación SOMI, León, Guanajuato, México, 24-28, Octubre, 2005.
5. Gengis K. Toledo, Ernst Kussul, Tatiana Baidyk. Neural classifier for micro work piece recognition. Image and Vision Computing. Elsevier. Vol 24/8, 2006, pp 827-836.
6. Gengis K. Toledo-Ramírez, Ernst Kussul, Tatiana Baidyk. Object oriented software for micro work piece recognition in microassembly. Journal of Applied Research and Technology. Vol 4/1, 2006, pp 59-74.
7. Josué Enríquez Zárate, Zaizar Betanzos Cortés, Gengis Kanhg Toledo Ramírez. Sistema Generador de Bases de Datos para Imágenes: una aplicación con Gambas. I Symposium de Linux de la Mixteca, 2-4, Marzo, 2006. http://www.utm.mx/gulmix/symposium_2006/ponencia.html
8. Gengis K. Toledo, Ernst Kussul, Tatiana Baidyk. Clasificación de piezas mediante un clasificador neuronal artificial. Memorias del XXI Congreso de Instrumentación SOMI, Ensenada, Baja California, México, 22-25, Octubre, 2006.
9. Gengis Kanhg, Kussul Ernst, Baidyk Tatiana. Reconocimiento de piezas mediante un clasificador neuronal con permutación de códigos. Artículo aceptado para el XXII Congreso de Instrumentación SOMI, Monterrey, Nuevo León, México, 1-4, Octubre, 2007.

Asi mismo al momento de terminar este trabajo una nueva publicación internacional está en proceso y un nuevo trabajo está postulándose para un congreso internacional.

Bibliografía

- [1] D. Rachkovskij and E. Kussul. Binding and normalization of binary sparse distributed representations by context-depending thinning. *Neural Computation*, 13:411–452, 2001.
- [2] T. Baidyk and E. Kussul. Application of neural classifier for flat image recognition in the process of microdevice assembly. In *Proceeding of IEEE International Joint Conference on Neural Networks*, volume 1, pages 160–164, Honolulu, Hawaii, USA, May 12-17 2002. doi:10.1109/IJCNN.2002.1005462.
- [3] E. Kussul, T. Baidyk, L. Ruiz-Huerta, A. Caballero, G. Velasco, and L. Kasatkina. Development of micromachine tool prototypes for microfactories. *Journal of Micromechanical and Microengineer*, 12(6):795–812, October 2002.
- [4] E. Kussul, T. Baidyk, L. Ruiz, A. Caballero, and G. Velasco. Development of low cost microequipment. In *International Symposium on Micromechatronics and Human Science*, pages 125–134, Nagoya, Japan, October 2002.
- [5] G.K. Toledo, E. Kussul, and T. Baidyk. Neural classifier for micro work piece recognition. *Image and Vision Computing, Elsevier.*, 24(8):827–836, 2006.
- [6] F. Lara-Rosano, E. Kussul, T. Baidyk, L. Ruiz, A. Caballero, and G. Velasco. Artificial intelligence systems in micromechanics. In *The first IFIP International Conference on Artificial Intelligence Applications and Innovations.*, pages 1–10, Toulouse, France, August 2004. Ed. By Max Bramer, Boston/Dordrecht/London, Kluwer Academic Publishers.
- [7] Ernst Kussul, Dmitri Rachkovskij, Tatyana Baidyk, and Semion Talayev. Micromechanical engineering: a basis for the low cost manufacturing of mechanical microdevices using microequipment. *Journal of Micromechanics and Microengineering*, 6(4):410–425, August 1996. doi:10.1088/0960-1317/6/4/008.
- [8] Ernst Kussul. Micromechanics and perspectives of neurocomputing. neuron-like networks and neurocomputers. Technical report, Institute of Cybernetics, Kiev, Ukraine, 1993.
- [9] Yuichi Okazaki, Nozomu Mishima, and Kiwamu Ashida. Microfactory - concept, history, and developments. *Journal of Manufacturing Science and Engineering*, 126:837–844, 2004.
- [10] T. Baidyk, E. Kussul, O. Makeley, A. Caballero, L. Ruiz, G. Carrera, and G. Velasco. Flat image recognition in the process of microdevice assembly. *Pattern Recognition Letters, Elsevier*, 25(1):107–118, January 2004.
- [11] E. M. Kussul, D. A. Rachkovskij, and T. N. Baidyk. On image texture recognition by associative-projective neurocomputer. In *Proc. of the ANNIE’91 conference “Intelligent engineering systems through artificial neural networks”*, pages 453–458. ASME Press, C.H. Dagli and S. Kumara and Y.C. Shin, 1991.
- [12] E. M. Kussul, D. A. Rachkovskij, and T. N. Baidyk. Associative-projective neural networks: architecture, implementation, applications. pages 463–476, Nimes, France, Nov. 4-8 1991.

-
- [13] E. Kussul and T. Baidyk. Neural random threshold classifier in OCR application. pages 154–157, Kiev, Ukraine, 1994.
 - [14] E.Kussul., T. Baidyk, V. Lukaviteh, and D. Rachkovskij. Adaptive high performance classifier based on random threshold neurons. *Cybernerics and Systems'94*, pages 1687–1695, 1994. in R. Trappl (Ed.) Singapore: World Scientific Publishing Co. Pte. Ltd.
 - [15] E. Kussul, T. Baidyk, L. Kasatkina, and V. Lukovich. Rosenblatt perceptrons for handwritten digit recognition. pages 1516–1520, Washington. USA, July 15-19 2001.
 - [16] E. Kussul and T. Baidyk. Improved method of handwritten digit recognition tested on MNIST database. In *15-th International Conference on Vision Interface*, pages 192–197, Calgary, Canada, May 27-29 2002.
 - [17] Ernst M. Kussul and Tatiana Baidyk. Permutative coding technique for handwritten digit recognition. In *International Joint Conference on Neural Networks*, volume 3, pages 2163–2168, Portland, Oregon, USA, July 20-24 2003.
 - [18] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. presented at a talk at the Stanford Artificial Project in 1968, unpublished but often cited, orig. in *Pattern Classification and Scene Analysis*, Duda, R. and Hart, P., John Wiley and Sons, 1973, pp 271-2, 1968.
 - [19] Ernst M. Kussul, Tatiana Baidyk, Donald C. Wunsch II, Oleksandr Makeyev, and Anabel Martín. Permutative coding technique for image recognition systems. *IEEE Transactions on neural networks*, 17(6):1566–79, november 2006.
 - [20] S Artikutsa, T. Baidyk, E Kussul, and D Rachkovskij. Texture recognition with the neuro-computer. Preprint 91-8 20, Institute of Cybernetics, Ukraine, 1991. (in Russian).
 - [21] G.K. Toledo, E. Kussul, and T. Baidyk. Neural classifier LIRA for recognition of micro work pieces and their positions in the processes of microassembly and micromanufacturing. In *The Seventh All-Ukrainian International Conference on Signal/Image Processing and Pattern Recognition, UkrObraz2004*, pages 17–20, Kiev, Ukraine, October 2004.
 - [22] E. Kussul and T. Baidyk. Improved method of handwritten digit recognition tested on MNIST database. *Image and Vision Computing, ELSEVIER*, 22(12):971–981, October 2004. doi:10.1016/j.imavis.2004.03.008.
 - [23] Josué Enríquez-Zárate, Zaizar Betanzos-Cortés, and Gengis Kanhg Toledo-Ramírez. Sistema generador de bases de datos para imágenes: una aplicación con gambas. Ixtepec, Oaxaca. México., Marzo 2-4 2006. I Simposium de Linux de la Mixteca.