Universidad Nacional Autónoma de México

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

INVESTIGACIÓN Y DESARROLLO DE UN MÉTODO PARA DIMENSIONAR MICRO-PIEZAS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

ELÉCTRICA-INSTRUMENTACIÓN

P R E S E N T A

MIGUEL HERNÁNDEZ ACOSTA

T U T O R E S

DR. ERNST KUSSUL - CCADET

DRA. TETYANA BAYDYK - CCADET

Ciudad Universitaria, D.F., México. Septiembre 2013

Jurado asignado

Presidente:	Dra. Medina Gómez Lucia
Secretario:	Dra. Baydyk Mykolaivna Tetyana
Vocal:	Dr. Kussul Ernst Mikhailovich
1^{er} Suplente:	Dr. Avendaño Alejo Maximino
2^{o} Suplente:	Dra. Montiel Sánchez María Herlinda

Lugar donde se realizó la tesis:
Grupo Académico de Computación Neuronal del Departamento de Tecnologías de la Información en el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM).

Tutores de tesis:

Dr. Ernst Kussul

Dra. Tetyana Baydyk

El trabajo de investigación fue desarrollado en los laboratorios del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM) gracias al apoyo de una beca para estudios de Posgrado del Consejo Nacional de Ciencia y Tecnología (CONACYT) y de una beca PAPIIT IN110510-3, así como apoyo del proyecto PAPIIT IN119610.

Dedicatoria

Dedico esta tesis a todas aquellas personas que hicieron posible este trabajo y más, en especial a mi familia, profesores y amigos que me han apoyado en mis proyectos y locuras.

A mi tía con cariño, respeto y una gran admiración.

Михаил

Дорогая Тетя!
Мы всегда будем помнить Вас!

♣ *Терпение и труд всё перетрут* ♣

Agradecimientos

Le doy gracias a Dios por las cualidades que me ha dado, principalmente por el libre albedrío, con el cual me ha facultado para tomar las decisiones de mi futuro. Ademas de darme la fortaleza para alcanzar mis metas y objetivos.

Muchas gracias a mi familia por su gran amor y apoyo incondicional.

Agradezco a mis tutores, los doctores Tetyana Baydyk y Ernst Kussul por su paciencia, opiniones, sugerencias, apoyo, enseñanzas y consejos. Ha sido un privilegio contar con su ayuda y guía.

A mis sinodales: Dra. Medina, Dra. Baydyk, Dr. Kussul, Dr. Avedaño y Dra. Montiel: muchas gracias por su tiempo dedicado a la revisión y asesoramiento en la realización de éste trabajo.

A mis profesores por su tiempo y enseñanzas que me trasmitieron en el desarrollo de mi formación.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) que me brindo el apoyo económico en mis estudios de maestría, igualmente al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) de la UNAM por la beca PAPIIT IN110510-3, así como el apoyo del proyecto PAPPIIT IN119610.

A nuestra casa de estudios, la Universidad Nacional Autónoma de México (UNAM), por haberme dado la oportunidad de ingresar al posgrado y cumplir otro gran proyecto de vida.

Por último, a todas aquellas personas que han sido mi soporte y compañía durante esta etapa.

Уважаемые Докторы Татьяна Байдык и Эрнст Куссуль!

Я благодарю Вас за все усилия, терпение, понимание и знания.

Miguel Hernández Acosta.

Prólogo

En esta tesis se presenta la investigación y el desarrollo de un método para dimensionar micro piezas. El método se enfoca para el área de micromecánica, en la situación en que el sistema de control debe de obtener información del tamaño de las micro piezas. El caso de estudio específico es la medición de micro pistones, los cuales son utilizados en motores térmicos que transfieren la energía calorífica de un concentrador solar a energía eléctrica.

La tesis se encuentra dividida en 2 partes con un total de 6 capítulos. La primera parte consta del capítulo 1 al 3 y contempla la investigación y los fundamentos para desarrollar el método (introducción, antecedentes y estado del arte, asimismo el clasificador neuronal *LIRA*). En la segunda parte se encuentra compuesta del capítulo 4 al 6, en donde se exponen la implementación, el desarrollo, las pruebas y los resultados del método.

Finalmente se tienen tres apartados mas, en donde se presentan las conclusiones sustentadas en el análisis de los resultados, el trabajo a futuro propuesto y el apéndice con el código fuente de algunos procedimientos de los programas y las características u hoja de especificaciones del equipo utilizado.

Resumen

Hoy en día la miniaturización de objetos se ha convertido en un tema muy importante de investigación. El uso cotidiano de las pequeñas piezas parece ser desapercibido, en cambio su fabricación a llegado a innovar y crear nuevas tecnologías.

Una de las tendencias en la manufacturación de micro piezas, es a base de tecnología de micro equipo (MET, por sus siglas en Inglés), con la cual se va reduciendo el tamaño de la fábrica hasta llegar a las micro fábricas.

En las micro fábricas se requiere que los procesos se automaticen; ya que al reducir el tamaño de los componentes, se complica su manipulación, por lo que es mas difícil de ejecutar el proceso por parte de un operador. Por eso es necesario desarrollar un sistema de visión computacional que defina el tamaño de las micro piezas.

La definición del tamaño de micro piezas por medio de un sistema de visión computacional se realiza a través de un procesamiento de imágenes, con el cual se reconoce un borde del objeto. El borde se determina con la extracción de los contornos y finalmente se realiza un conteo de los pixeles para determinar el tamaño del objeto.

En la presente tesis se analizaron distintos métodos de extracción de contornos, tanto convencionales, como heurísticos; con los cuales se desarrolló un método para dimensionar micro piezas, principalmente micro pistones.

Se prestó especial atención a aquellos métodos que están basados en redes neuronales artificiales, por su carácter adaptativo en los sistemas automatizados y por trabajos anteriores (como es el caso del clasificador *LIRA*).

La prueba del método de dimensionamiento de micro piezas se hizo a través del desarrollo de varios programas, los cuales muestran la eficiencia del método creado.

Abstract

Nowadays the miniaturization of objects has become an important research topic. The pieces themselves are usually overlooked, on the other hand its manufacturing has enjoyed great attention, leading to innovation on the field and the creation of new technologies.

One trend regarding the manufacturing of micropieces is based on the MET technology, in which the size of the factory is reduced from a normal factory scale to a microfactory scale.

One key requirement of the microfactories is that the processes must be automated; owing to the fact that by reducing the size of the components it becomes more difficult for the operator to handle (perform his tasks). Therefore it is necessary to develop a computer vision system that identifies the size of the micropieces.

The sizing of micropieces using a computer vision system is performed through image processing. First the edge of the object is identified by extracting the contours, then the size is calculated counting from the left to the right side of the edge.

In the present thesis different contour extraction methods were analyzed, both conventional as well as heuristics. Based on that analysis a method for sizing micropieces, mainly micropistons, was developed.

Special attention was given to those methods based on artificial neural networks, due to their adaptive nature in automated systems and also from previous works, such as the *LIRA* classifier.

Finally to show the efficiency of the proposed method several benchmarks were performed.

Índice general

Índice de figuras

Índice de tablas

Parte I

MARCO TEÓRICO

Capítulo 1

Introducción

En la manufacturación de micro piezas se necesita una automatización eficiente de los procesos. El reto para obtener una alta eficiencia con la automatización de los procesos recae en la inspección visual de las micro piezas, para poder corregir los parámetros que controlan la micro fábrica. La inspección visual se apoya en los sistemas de visión computacional, los cuales permiten hacer un análisis detallado de los objetos, extrayendo sus características. Una de las características usualmente requerida es la dimensión de la pieza. A continuación se exponen los motivos para crear un método para el dimensionamiento de micro piezas basado en redes neuronales artificiales.

1.1. Antecedentes

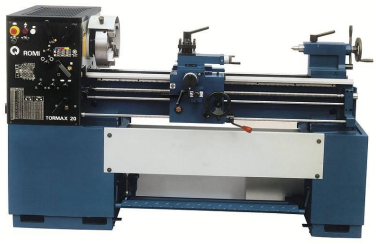
En el ámbito de la manufactura existen procesos que se realizan por medio de un operador y otros que se encuentran automatizados. Los procesos en los que interviene un operador se hacen mas difíciles de ejecutar a medida que los componentes que se manipulan reducen su tamaño. Por eso es que se crean micro fábricas [2].

Las micro fábricas pueden producir equipo miniaturizado, tal como máquinas - herramienta, manipuladores, dispositivos de ensamble, instrumentos de medición, etc. Para crear una micro fábrica se han seguido dos métodos, uno es usando equipo convencional [8] y el otro es por medio de equipo micro mecánico basado en tecnología de micro electrónica (MEMS[1], por sus siglas en Inglés) [2].

[1]MicroelectroMEchanical Systems

En el caso del primer método, el equipo convencional utilizado son los tornos, fresadoras, taladros, pulidoras, prensas, rectificadoras y máquinas de control numérico (CNC[2], por sus siglas en Inglés), ver figura 1.1. Estos equipos tienen desventajas para manufacturar piezas y componentes pequeños (menores a 200mm [9–12]), entre estas desventajas están el consumo de energía, el espacio que abarca el equipo y el desperdicio de material. Cabe mencionar que es muy costoso el obtener una buena precisión con el equipo convencional.

[2]Computer Numerical Control



(a) Torno.

(b) Fresadora.

(c) Taladro.

(d) Pulidora.

(e) Prensa.

(f) Rectificadora.

(g) CNC.

Figura 1.1: Máquinas - Herramienta convencionales.

Por otro lado, el segundo método es más avanzado, la tecnología de MEMS permite producir dispositivos por debajo de los 100 μm, ver figura 1.2, [4]. Sin embargo, esta tecnología se limita a la creación de componentes con materiales que son compatibles con la tecnología de silicio y en la creación de dispositivos con forma de 2 o 2.5[3] dimensiones [5], debido a que la tecnología utilizada y los procesos de fabricación son similares a la producción de circuitos integrados y de semiconductores.

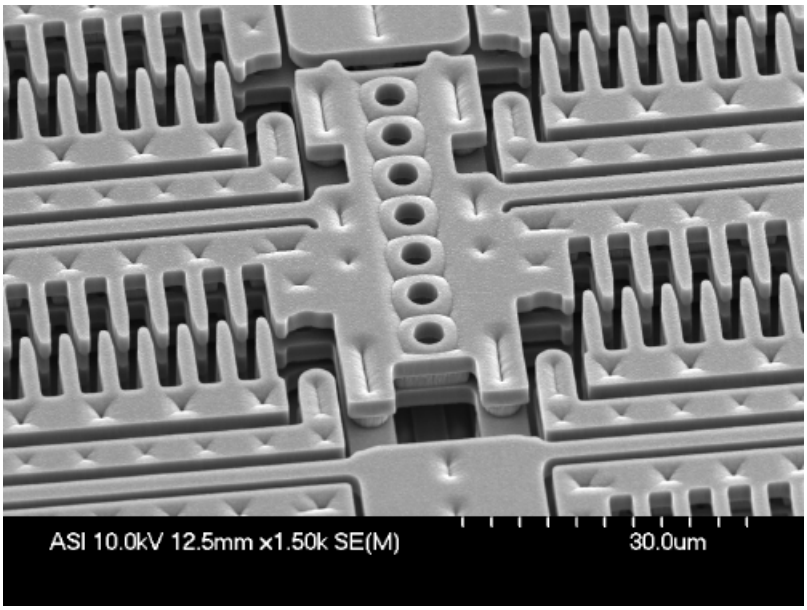


Figura 1.2: MEMS Actuador electrostático.

Una propuesta alternativa es la creación de micro fábricas a través de la tecnología basada en tecnología de micro equipo (MET[4], por sus siglas en Inglés) y ensamble de micro dispositivos, los cuales pueden ser producidos por generaciones, en donde cada nueva generación es producida por la generación anterior, como se muestra en la figura 1.3 [6].

[3]Perspectiva 3/4 o Pseudo-3D, la estructura es en 3 dimensiones, pero por su escala en profundidad, también es considera como 2 dimensiones

[4]Micro Equipment Technology

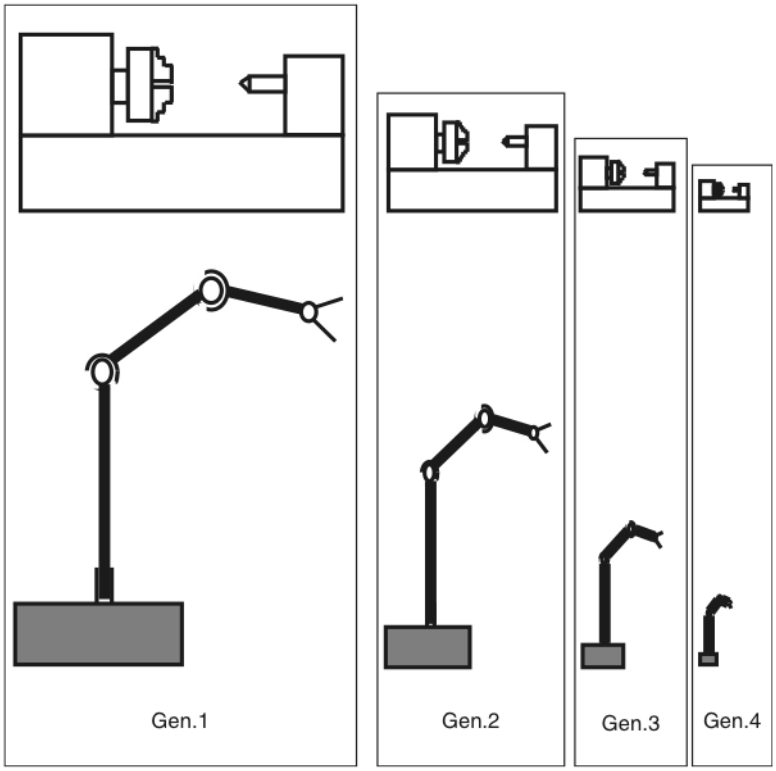


Figura 1.3: Las generaciones sucesivas de micro equipo.

La primera generación sería producida por el equipo convencional de gran escala, la segunda generación sería producida por el equipo de la primera generación, teniendo como resultado una reducción en el tamaño del equipo, y así sucesivamente para cada nueva generación. A este enfoque de manufacturación mecánica de micro dispositivos se le denomina como MET [5].

En países como Japón se han desarrollo prototipos de micro fábrica del tamaño de una mesa, con dimensiones de 625 mm de largo por 490 mm de ancho y 380 mm de alto, como se presenta en la figura 1.4 [7]. Una de estas micro fábricas desarrolladas en el laboratorio japonés está constituida por un micro torno, un micro manipulador, una micro prensa y una micro fresadora, como se muestra en la figura 1.5. La micro fábrica japonesa es capaz de manufacturar piezas experimentales de mínimo 60μm y fabricar un producto de 900μm.

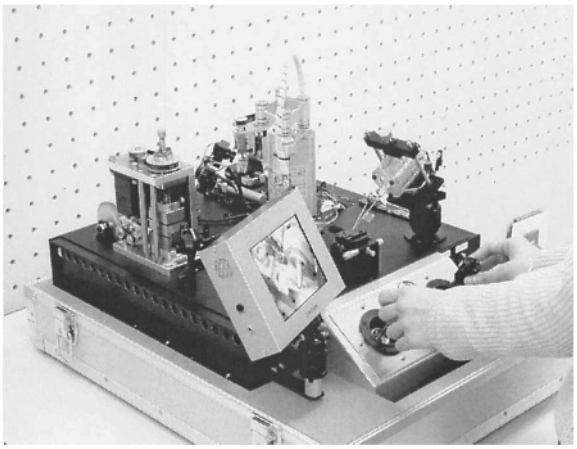
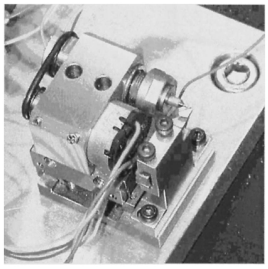
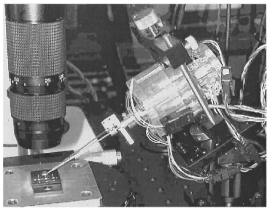


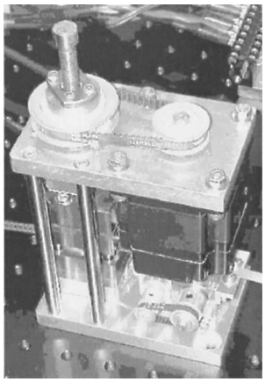
Figura 1.4: Micro fábrica japonesa portable de mesa.



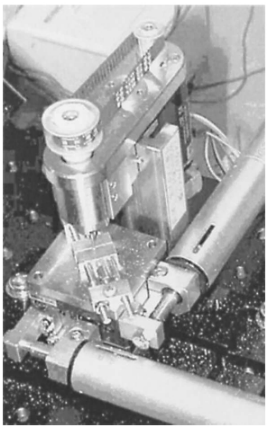
(a) Micro torno.



(b) Micro manipulador.



(c) Micro prensa.



(d) Micro fresadora.

Figura 1.5: Micro Máquinas - Herramienta.

En los laboratorios del CCADET[5] se han desarrollado y construido prototipos de micro máquina - herramienta, capaz de tornear, barrenar, taladrar y rectificar, con solo cambiar las herramientas y el software que la controla. En la figura 1.6 se muestra el segundo prototipo de máquina - herramienta creado. El prototipo tiene dimensiones de 130 mm de largo por 160 mm de ancho y 85 mm de alto [5].

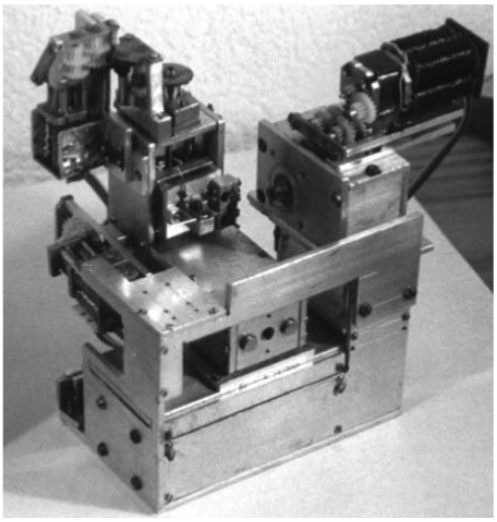


Figura 1.6: Segundo prototipo de Máquina - Herramienta.

Cabe destacar que una tarea primordial es la automatización de los procesos de manufactura. La automatización permite que un gran número de micro máquinas - herramienta se aprovechen en conjunto para tener una micro fábrica [1, 3]. Para lograr la automatización deseada, se considera que solo un operador va a manipular la micro fábrica, por lo tanto todos los problemas durante la producción deben ser resueltos automáticamente, ya que en este caso el operador no puede ayudar mucho y que el operador solo debe de intervenir para resolver problemas muy complejos o inusuales [5]. Puesto que la manufacturación de micro piezas es afectada por varias perturbaciones y es imposible considerar por adelantado todas las perturbaciones existentes y las que se pueden generar, se propone utilizar sistemas adaptativos, de tal forma que pueda aprender y ajustar los parámetros de la micro fábrica. Por eso se propone desarrollar un sistema de visión computacional, con el cual se pretende definir los tamaños de las micro piezas.

[5]Centro de Ciencias Aplicadas y Desarrollo Tecnológico, UNAM

Uno de los procesos que se requiere automatizar, es el proceso de medición de tamaños de micro piezas. El proceso de medición es considerado como una inspección de calidad intermedia de la producción, para determinar si el equipo necesita reajustar automáticamente los parámetros de control y evaluar la calidad de la pieza fabricada. Existen diferentes métodos para dimensionar piezas, uno de ellos es por medio de sistemas de visión computacional. El método para dimensionar las micro piezas que se propone en el presente trabajo se realiza por medio de imágenes digitales y se encuentra basado en tecnología que utiliza redes neuronales artificiales.

El método contribuye a la construcción de una micro fábrica para la manufacturación de motores térmicos. Los motores térmicos son ampliamente utilizados para la transformación de energía calórica en energía mecánica y eléctrica. Dos clases de motores se emplean en las tecnologías de energías renovables, estos son el motor térmico de ciclo Stirling y de ciclo Rankine. El motor térmico de ciclo Rankine se usa principalmente en plantas solares de gran escala, en cambio el de ciclo Stirling se puede usar en plantas solares pequeñas [13].

El motor térmico Stirling, figura 1.7, presenta muchas ventajas; una de ellas es la alta eficiencia, así como el tiempo de vida mayor, pero sus desventajas son: necesita una diferencia de temperaturas muy altas y materiales de construcción mas costosos. Para reducir las desventajas del motor térmico de ciclo Stirling, se está investigando en el CCADET el motor Ericsson, el cual contiene propiedades muy prometedoras [13].

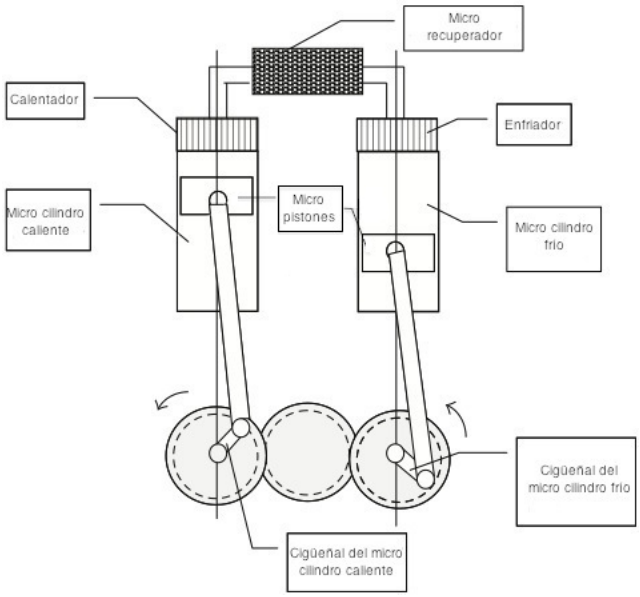


Figura 1.7: Esquema del Motor Stirling.

La fuente de energía térmica del motor Ericsson, proviene de un concentrador solar. En el CCADET se desarrollaron dos prototipos de concentradores solares.

Para construir motores Ericsson eficientes es necesario manufacturar micro piezas con gran precisión. El problema principal para medir a través de imágenes digitales es encontrar los contornos de los bordes de la pieza que la limiten con respecto al entorno y el fondo de la imagen. Para discernir estos contornos es necesario usar los métodos de reconocimiento de imágenes más avanzados [14] incluso los sistemas adaptativos que pueden entrenarse mediante algunas imágenes de ejemplo [15, 16].

Las imágenes que se utilizaron fueron obtenidas a través de un microscopio metalúrgico trinocular, modelo NJF-120A (ver apendice D), para crear una base de datos de imágenes inicial de 15 imágenes digitales.

En este trabajo se investigan las redes neuronales artificiales y se prueban para desarrollar un método de extracción de contornos y de esta manera resolver la tarea de obtener el borde de las micro piezas. La aplicación en la que se enfoca el trabajo de investigación, es el reconocimiento de contornos de micro pistones, específicamente para crear micro pistones y micro cilindros para motores térmicos, los cuales serán usados para la transformación de energía solar de los concentradores solares en energía eléctrica.

Existen diferentes investigaciones para obtener contornos por medio de redes neuronales artificiales [16–21], debido a que en el procesamiento de imágenes y en visión computacional es necesario el contorno para distintos análisis en la imagen, como la amplificación [21], la segmentación, la dimensionalidad y el reconocimiento de patrones [14]. Estos algoritmos se distinguen, de los filtros y operadores convencionales (Sobel, Robert's, Prewitt, Schwartz), en su adaptabilidad y su no linealidad [17], permitiendo que se apliquen en tareas de manufactura y de tiempo real [18].

El entrenamiento de la red neuronal artificial se realiza con diferentes técnicas de enseñanza. La técnica más popular es la retro-propagación (llamada en Inglés *"Backpropagation"*) [22]; la cual ajusta los pesos de las conexiones de la red neuronal repetidamente, para minimizar la diferencia entre la salida y el resultado deseado [23].

La diferencia entre la salida y el resultado deseado es considerado como el error de la red neuronal. La reducción del error radica en realizar dos recorridos a través de las diferentes capas de la red neuronal.

El primer recorrido es hacia adelante, aplicando en la capa de entrada un patrón, que propaga su efecto a través de las diferentes capas, produciendo el vector de salida. En este proceso los pesos de las relaciones entre las neuronas son fijos y no se modifican [24].

El segundo recorrido es hacia atrás, en donde el error es propagado a cada una de las capas de la red neuronal en dirección contraria. Al propagar el error, los pesos se modifican, de tal forma que al pasar de nuevo el patrón se corrija el error.

Una parte importante en ciertos métodos es la binarización de la imagen [20], pero para obtener mayores características del objeto bajo análisis en la imagen, el método propuesto consta de una red neuronal artificial *LIRA* (LImited Receptive Area), en su variante de escala de grises [15].

El método desarrollado se basó en la investigación para obtener los contornos mediante el algoritmo de la red neuronal artificial *LIRA* en escala de grises y de esa manera definir el borde del objeto y poder dimensionarlo.

1.2. Objetivos generales

En el desarrollo del presente trabajo se fijaron los siguientes objetivos generales:

- Investigar y desarrollar un método basado en redes neuronales artificiales para dimensionar micro piezas. Para emplearlo en el área de micro mecánica, especialmente en la fabricación de motores térmicos.
- Analizar los métodos y filtros convencionales para extraer contornos en una imagen.
- Desarrollar un clasificador neuronal para reconocer los contornos externos de una imagen.
- Experimentar con los parámetros del clasificador neuronal.

1.3. Objetivos de la tesis

Se plantearon los siguientes objetivos para el desarrollo de la tesis:

- Investigar y desarrollar un método, utilizando algoritmos de redes neuronales artificiales.
- Simular un sistema con el método desarrollado, generando los experimentos necesarios, para su posterior análisis.
- Reducir los costos de manufactura, desarrollando el método de dimensionamiento de micro piezas.

1.4. Justificación

Los motores térmicos para el concentrador solar, requieren una precisión suficiente en la manufacturación. Esta precisión se podría lograr mediante equipo especializado muy costoso. Otra opción puede ser como las cámaras inteligentes que existen en el mercado [25] o software [26], [27] que utiliza tanto métodos convencionales [14] como algoritmos cerrados y desconocidos; y que demanda de una programación especializada. Para reducir esos costos de manufacturación de las piezas, se usan los métodos convencionales en ambientes de programación mas estandarizados, como es en el lenguaje de programación C [28]. Pero estos métodos generan contornos excesivos y no deseados. Por eso se usan los algoritmos adaptativos, aprovechando sus ventajas de no linealidad y capacidad de aprendizaje. Se han hecho varias investigaciones con algoritmos en la obtención de contornos, pero la mayoría de ellas utilizan imágenes de prueba generales, como la imagen de prueba Lenna [29]. A diferencia de las pruebas realizadas con esos algoritmos, en este trabajo de tesis se utilizan imágenes de los micro pistones, los cuales no tienen un tratamiento previo, emulando lo mas posible a una sistema de manufactura habitual.

El algoritmo adaptativo seleccionado ha demostrado en trabajos previos que se puede aplicar para diferentes tareas, como el reconocimiento de imágenes para ensamblar micro piezas [16]. De esta forma se puede investigar éste algoritmo, compararlo con los métodos convencionales y los otros algoritmos adaptativos generales, aplicarlo a una tarea específica de manufactura de micro piezas y desarrollar un método para dimensionar los micro pistones.

1.5. Metodología

En el desarrollo del trabajo se aplicó la siguiente metodología:

- Crear la base de datos de imágenes para probar el método de reconocimiento de contornos de micro piezas.
- Probar los filtros convencionales para extraer contornos del objeto en análisis.
- Marcar manualmente las imágenes de la base de datos de imágenes, indicando los bordes reales que un operador puede apreciar.

- Desarrollar y programar el clasificador neuronal para reconocer los contornos del objeto.
- Entrenar el clasificador neuronal con la base de datos de imágenes marcadas.
- Examinar la posibilidad de reconocer bordes usando como ejemplo la base de datos de imágenes que no fueron utilizadas en el proceso de entrenamiento.
- Analizar los resultados obtenidos con los métodos y filtros convencionales.
- Analizar la posibilidad de uso del método desarrollado para aplicaciones en mediciones de micro-pistones y micro-cilindros.

1.6. Contribución de este trabajo

Este trabajo contribuye con la investigación de tecnologías para la solución de problemas en aplicaciones de visión computacional basados en algoritmos avanzados, como lo son las redes neuronales artificiales.

Además de reducir los costos de fabricación de micro piezas, al no utilizar equipo comercial muy costoso, como las cámaras inteligentes Cognex [25] o sistemas de visión de Labview [26]. También se mantienen bajos costos al automatizar la micro fábrica y no depender tanto de un operador.

1.6.1. Alcances y limitaciones

La investigación y el desarrollo del método se realizó de tal manera que cumpliera con los siguientes requerimientos:

- Crear un método de dimensionamiento para el campo de trabajo de la manufactura, específicamente para micro fábricas.
- Reconocer tres clases dentro de la imagen digital.
- Configurar los parámetros del clasificador neuronal (número de neuronas, iteraciones de entrenamiento, tamaño de la muestra y de las ventanas dentro de la muestra).

Durante el desarrollo del trabajo se presentaron las siguientes limitantes:

- El tiempo de entrenamiento dependiente del número de pixeles a entrenar, mientras más pixeles se tengan, más información se tiene de la clase y mejor es el reconocimiento.
- El tiempo de reconocimiento dependiente del tamaño de la imagen, si la imagen es muy grande el tiempo de ejecución del reconocimiento se incrementa.
- El hardware inadecuado para la tarea, se efectuaron simulaciones, pero se propone en el trabajo a futuro cambiarlo.

1.7. Organización del trabajo

A continuación, en la tabla 1.1 se proporciona la reseña de como se encuentra organizado el presente trabajo.

Parte	Capítulo	Descripción
I	Introducción	En este capítulo se explica un panorama general del problema y los antecedentes de la investigación, así como los objetivos, el marco teórico y su contribución.
	Estado del arte	El capítulo describe el estado del arte de las técnicas para la extracción de los contornos.
	Clasificador neuronal *LIRA*	Este capítulo está dedicado a mostrar el clasificador neuronal propuesto para el desarrollo del método para la tarea de dimensionar micro pistones.
II	Descripción de la base de las imágenes	En este capítulo se exponen las propiedades, el tratamiento previo de las imágenes digitales de los micro pistones.
	Experimentos y resultados	El capítulo presenta los experimentos y resultados realizados con los métodos convencionales y el clasificador *LIRA*.
	Medición	El último capítulo brinda la aplicación de la red neuronal artificial en el dimensionamiento de micro piezas.

Tabla 1.1: Descripción de capítulos

Por último se tienen las conclusiones de la tesis, con el posible trabajo a futuro. En el apéndice se tiene el código fuente de algunos procedimientos del software desarrollado para la investigación y las especificaciones del equipo utilizado (Torno, micrómetro, cámara y microscopio).

Capítulo 2

Estado del arte

En este capítulo se presenta el estado del arte sobre como extraer los contornos y el borde de objetos en imágenes digitales. La obtención de los contornos los he clasificado utilizando métodos convencionales y algoritmos de redes neuronales artificiales. En la primera clasificación de la investigación se enfocó principalmente en el operador Sobel y el algoritmo de Schwartz, aunque también se mencionan el operador Roberts y el operador Prewitt. Se seleccionaron estos métodos debido a que otros métodos se basan en ellos y el principio es el mismo, como el operador Canny [30], el operador Prewitt [31], el operador Roberts [32]. La otra clasificación muestra cinco técnicas con los algoritmos de obtención de contornos basadas en redes neuronales artificiales.

2.1. Métodos convencionales

Los métodos convencionales investigados fueron principalmente el operador Sobel y el algoritmo de Schwartz, los cuales a continuación se presentan. Estos métodos se programaron y probaron; los experimentos y resultados se exponen en el capítulo 5.

Adicionalmente se explican los operadores Robert's y Prewitt, pero que no fueron programados, ya que son muy similares al operador Sobel.

2.1.1. Operador Sobel

El operador Sobel permite la detección de contornos y de esta forma distinguir objetos del fondo de una imagen u objetos de otros objetos en la imagen. El operador se basa en el principio en que exista un cambio abrupto en la intensidad de los pixeles y así poder definir un contorno. Los contornos se extraen con cuatro diferentes mapas, el mapa horizontal, el mapa vertical y dos mapas en diagonal de $\pm 45°$ [33].

En el procesamiento de una imagen, el operador Sobel es técnicamente un operador diferencial discreto [34], el cual calcula una aproximación del gradiente de la función de intensidad de la imagen. El resultado del operador Sobel en cada pixel de la imagen corresponde al vector gradiente o a la norma del vector, indicando la dirección del cambio en la intensidad o el color de la imagen, como se muestra en la figura 2.1 [35].

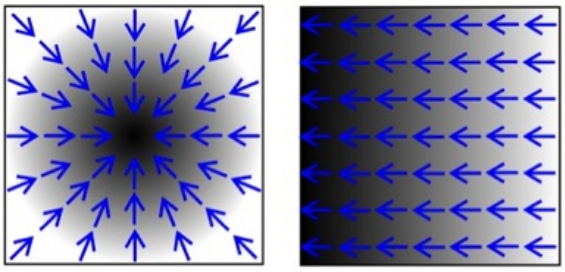


Figura 2.1: Dos tipos de gradientes.

En una función continua dada por la expresión 2.1

$$f(x, y) \tag{2.1}$$

su gradiente se expresa como en la ecuación 2.2

$$\nabla f(x, y) = [G_x, G_y] = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right) \tag{2.2}$$

su magnitud 2.3

$$|\nabla f(x, y)| = \left[G_x^2 + G_y^2\right]^{\frac{1}{2}} \tag{2.3}$$

y la dirección 2.4

$$\theta(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.4}$$

Pero una imagen se contempla como una función discreta, debido a que la función de intensidad de una imagen digital sólo se conoce mediante puntos discretos. Para poder obtener el gradiente de una función discreta se utiliza una convolución con un filtro para efectuar una aproximación.

La convolución discreta en una imagen se define como en la expresión 2.5.

$$(A \star M) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} A_{ij} M_{ij} \tag{2.5}$$

donde A es la imagen, M es el filtro que se aplica, A_{ij} es el brillo de cada pixel en la imagen A y M_{ij} es cada elemento del filtro.

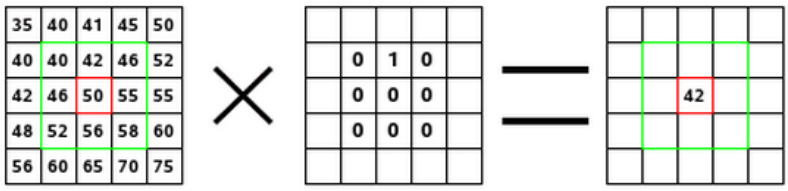


Figura 2.2: Ejemplo de convolución en una imagen.

El gradiente de la imagen se genera con la convolución de la imagen y cuatro filtros por separado, uno en dirección horizontal, otro en vertical y otros dos en diagonales. Con las imágenes resultantes de la aplicación de los filtros, se emplea la operación "OR" para crear la imagen final con los contornos en las direcciones deseadas (vertical, horizontal y las diagonales). Los filtros están constituidos por cuatro máscaras de 3×3.

Las máscaras y sus características

Las máscaras de los filtros se definen como en la expresión 2.6.

$$M = \begin{bmatrix} m_{i-1,j-1} & m_{i,j-1} & m_{i+1,j-1} \\ m_{i-1,j} & m_{i,j} & m_{i+1,j} \\ m_{i-1,j+1} & m_{i,j+1} & m_{i+1,j+1} \end{bmatrix} \tag{2.6}$$

donde cada coeficiente m está determinado para aproximar la derivada y de esa manera obtener el gradiente.

Para determinar los coeficientes de las máscaras, se parte de la función continua de la expresión 2.1, en donde el gradiente en dirección x esta dada por la ecuación 2.7, mientras que el gradiente en dirección y se indica en la ecuación 2.8.

$$\nabla_x f(x,y) = \frac{\partial f(x,y)}{\partial x} = \Delta_x = \frac{f(x+d_x,y) - f(x,y)}{d_x} \tag{2.7}$$

$$\nabla_y f(x,y) = \frac{\partial f(x,y)}{\partial y} = \Delta_y = \frac{f(x,y+d_y) - f(x,y)}{d_y} \tag{2.8}$$

Dado que el desplazamiento en la imagen es discreto, el gradiente de un pixel se aproxima con los pixeles adyacentes, por lo que $d_x = d_y = 1$ y de igual manera la posición del pixel se indica con i, j; por lo tanto las ecuaciones 2.7 y 2.8 se modifican, generando las expresiones 2.9 y 2.10.

$$\Delta_x \approx f(i+1,j) - f(i,j) \tag{2.9}$$

$$\Delta_y \approx f(i,j+1) - f(i,j) \tag{2.10}$$

En la definición de la convolución en la expresión 2.5 se determinó que A es la imagen a procesar, de manera que se redefinen las ecuaciones 2.9 y 2.10 y se obtienen las ecuaciones 2.11 y 2.12.

$$\Delta_x \approx A_{i+1,j} - A_{i,j} \tag{2.11}$$

$$\Delta_y \approx A_{i,j+1} - A_{i,j} \tag{2.12}$$

Para obtener una mejor aproximación del gradiente en cada pixel, se toman en cuenta los pixeles vecinos. En el caso del gradiente horizontal, se suman los resultados del cálculo del gradiente con los pixeles de la derecha e izquierda, por lo que se obtiene la ecuación 2.13. Para el gradiente vertical es similar, pero con los pixeles inferior y superior (ver ecuación 2.14).

$$\Delta_x \approx (A_{i-1,j} - A_{i,j}) + (A_{i,j} - A_{i+1,j}) = A_{i-1,j} + 0(A_{i,j}) - A_{i+1,j} \tag{2.13}$$

$$\Delta_y \approx (A_{i,j-1} - A_{i,j}) + (A_{i,j} - A_{i,j+1}) = A_{i,j-1} + 0(A_{i,j}) - A_{i,j+1} \tag{2.14}$$

Las ecuaciones descritas 2.13 y 2.14 se representan o equivalen a los vectores 2.15 y 2.16.

$$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{2.15}$$

$$\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{2.16}$$

Al aplicar la convolución en la imagen con los vectores 2.15 y 2.16 se localizan los cambios de brillo en cada pixel (horizontalmente y verticalmente por separado), pero son muy sensibles al ruido. Ésta sensibilidad se reduce al extender el gradiente de forma que se involucren los pixeles en diagonal, como se muestran en las matrices 2.17 y 2.18.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.17}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.18}$$

Adicionalmente para robustecer el operador, se enfatiza solo la línea y la columna que se esta procesando, multiplicándolas por dos. Así los coeficientes de las máscaras quedan de la siguiente manera: la máscara horizontal (X) se muestra en la matriz 2.19, la máscara vertical (Y) se aprecia en la matriz 2.20, la primera máscara en diagonal se indica en la matriz 2.21 y la otra máscara en diagonal se observa en la matriz 2.22.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.19}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.20}$$

$$G_{45^\circ} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \tag{2.21}$$

$$G_{-45^\circ} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \tag{2.22}$$

Cabe destacar que las máscaras tienen ciertas propiedades:

- Son matrices cuadradas de 3×3.
- El determinante es 0.

$$|G| = 0 \tag{2.23}$$

- Son matrices singulares (no existen sus matrices inversas, porque no tienen determinantes).
- G_y es la transpuesta de G_x.

$$G_y = G_x^T \tag{2.24}$$

- G_{45° es simétrica.
- G_{-45° es antisimétrica.
- Al pixel que se le calcula el gradiente se localiza en el centro de la máscara.
- G_x se puede separar como:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{2.25}$$

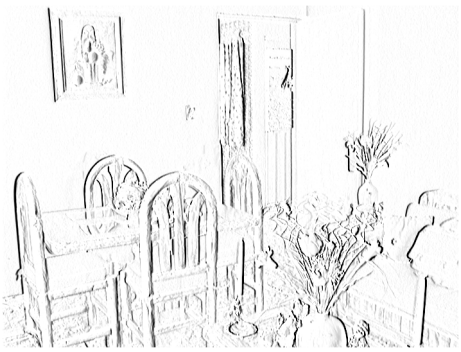
- G_y se puede separar como:

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{2.26}$$

Ejemplos de la aplicacion del operador Sobel

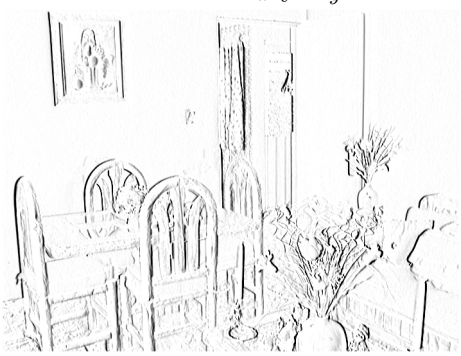
En la figura 2.3 se observa la aplicación del operador Sobel en una imagen de ejemplo para la extracción de los contornos.

(a) Imagen original en escala de grises.



(b) Imagen resultante del gradiente calculado por el operador Sobel sobre la imagen original, empleando solo las máscaras G_x y G_y.

(c) Cálculo del gradiente X con el operador Sobel, aplicando la máscara G_x.



(d) Cálculo del gradiente Y con el operador Sobel, aplicando la máscara G_y.

Figura 2.3: Ejemplo de la aplicación del operador Sobel.

2.1.2. Operador Roberts Cross

El operador Roberts Cross, usualmente denominado como operador Roberts, es un operador diferencial, el cual es utilizado como un detector de bordes. Se basa en el mismo principio que el operador Sobel, al hacer una aproximación de la magnitud del gradiente pero con los pixeles adyacentes en diagonal, indicando así si existe un punto de borde; sin embargo no puede señalar su orientación [36]. Las imágenes a las que se pueden aplicar el operador Roberts pueden ser binarias o en escala de grises, aunque es obtienen mejores resultados con imágenes binarias.

De acuerdo con el creador del operador, un detector de borde debe de estar compuesto por las siguientes propiedades:

- el borde debe de estar bien definido.
- el fondo debe de contener el menor ruido posible.
- la intensidad del borde debe de corresponder lo mas cercano posible a la percepción del humano.

Con este criterio, se propusieron las ecuaciones 2.27 y 2.28.

$$y_{i,j} = \sqrt{x_{i,j}} \tag{2.27}$$

$$z_{i,j} = \sqrt{(y_{i,j} - y_{i+1,j+1}) + (y_{i+1,j} - y_{i,j+1})} \tag{2.28}$$

donde x es el valor de intensidad inicial en la imagen, z es la derivada calculada y i,j representan la ubicación en la imagen.

El resultado de esta operación resaltará los cambios de intensidad de los pixeles en sentido diagonal.

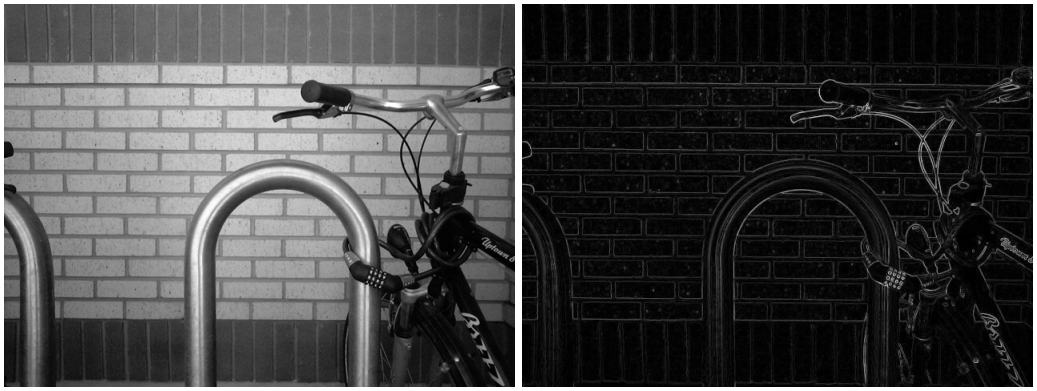
Del mismo modo que el operador Sobel, para poder aplicar el operador Roberts en una imagen es necesario convolucionar la imagen con máscaras que nos representen las ecuaciones 2.27 y 2.28, las cuales se muestran con las matrices 2.29 y 2.30.

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.29}$$

$$G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.30}$$

Los coeficientes de las máscaras del operador Roberts se generan de la misma forma que los coeficientes del operador Sobel (ver sección 2.1.1), con diferencias mínimas, ya que las máscaras de Roberts son mas pequeñas.

Una de las características de este operador es su simplicidad, ya que las máscaras son pequeñas y solo contienen enteros, sin embargo es muy sensible al ruido.



(a) Imagen original en escala de grises.

(b) Imagen resultante del gradiente calculado por el operador Roberts sobre la imagen original, empleando las máscaras G_x y G_y.

Figura 2.4: Ejemplo de la aplicación del operador Roberts.

2.1.3. Operador Prewitt

Otro de los detectores de borde que se basan en el cálculo del gradiente es el operador Prewitt, con el cual se resaltan los cambios abruptos en la imagen, por consiguiente es muy probable que esa parte en la imagen represente un borde, así como su orientación [31].

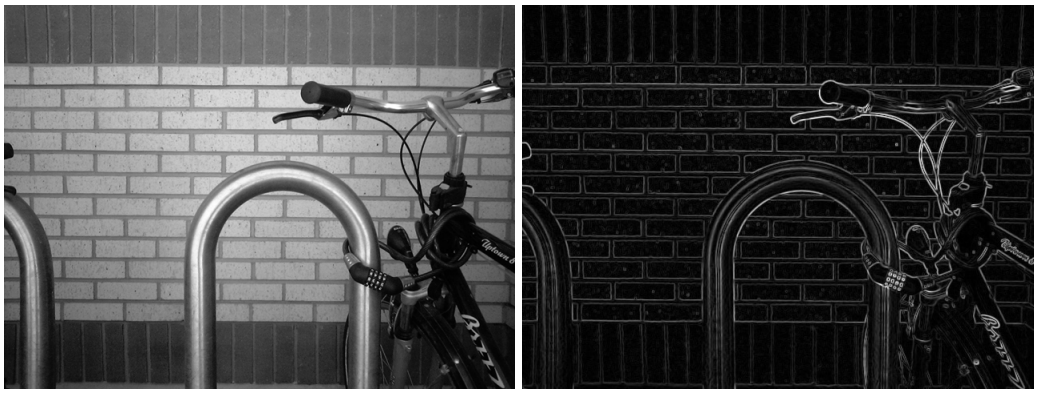
El operador Prewitt es muy similar al operador Sobel, ya que toma en cuenta los pixeles adyacentes para hacerlo más inmune al ruido; la diferencia radica en los coeficientes de las máscaras, los cuales no enfatizan la línea o columna que se procesa.

Las máscaras del operador Prewitt están compuestas por matrices de 3×3, las cuales se convolucionan con la imagen original para calcular las aproximaciones de las derivadas, una en dirección horizontal con la matriz 2.31 y la otra en dirección vertical con la matriz 2.32.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.31}$$

$$G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.32}$$

Los coeficientes de las máscaras del operador Prewitt se generan de igual manera que los coeficientes de las máscaras del operador Sobel (ver sección 2.1.1), pero sin enfatizar la línea o columna que se procesa.



(a) Imagen original en escala de grises.

(b) Imagen resultante del gradiente calculado por el operador Prewitt sobre la imagen original, empleando las máscaras G_x y G_y.

Figura 2.5: Ejemplo de la aplicación del operador Prewitt.

2.1.4. Algoritmo de Schwartz

El algoritmo de la patente de Schwartz [37] describe como adquirir información y características dentro de una imagen. Las características que se pueden determinar son los contornos, el ángulo de orientación del contorno, así como el brillo dentro de la región de análisis. Este contorno lo define como la transición de un área iluminada (blanca) a un área obscura (negra).

El primer paso del algoritmo es comprimir la región lo más posible, de tal forma que contenga el contorno. La región debe de poderse subdividir en un número de sectores simétricos e igualmente distribuidos. La región más pequeña posible se define como un área cuadrada de 2×2 pixeles. Tomando esta pequeña región se crea una ventana como se muestra en la figura 2.6.

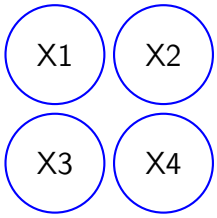


Figura 2.6: Área de 2×2 pixeles.

A continuación, tomando el valor de cada pixel se calcula el brillo que se representa como y_1 y y_2, por medio de la expresión 2.33 y 2.34

$$y_1 = X_1 + X_4 \tag{2.33}$$
$$y_2 = X_2 + X_3 \tag{2.34}$$

Con los valores adquiridos de y_1 y y_2, se hace la comparación con una constante experimental (CE), con la cual definirá si existe o no existe un contorno en la imagen. Si el absoluto de la diferencia es mayor a CE, por lo tanto si existe un contorno, como lo muestra la desigualdad 2.35.

$$|y_1 - y_2| > CE \tag{2.35}$$

donde $CE \in \mathbb{Z}$

Se calculan las diferencias entre los pixeles cruzados con las expresiones 2.36 y 2.37.

$$\delta_1 = X_1 - X_4 \tag{2.36}$$
$$\delta_2 = X_3 - X_2 \tag{2.37}$$

El siguiente paso es el calcular los valores mínimos y máximos de esas diferencias de acuerdo a las ecuaciones 2.38 y 2.39.

$$\Delta_1 = max(|\delta_1|, |\delta_2|) \tag{2.38}$$
$$\Delta_2 = min(|\delta_1|, |\delta_2|) \tag{2.39}$$

Por último se obtiene la variable y del contorno (si existe) con la fórmula 2.40.

$$y = \frac{\Delta_2}{\Delta_1} < 1 \tag{2.40}$$

Para obtener el ángulo se hace referencia a la tabla 2.1

ϕ	$0 - \frac{\pi}{4}$	$\frac{\pi}{4} - \frac{\pi}{2}$	$\frac{\pi}{2} - \frac{3}{4}\pi$	$\frac{3}{4}\pi - \pi$	$\pi - \frac{5}{4}\pi$	$\frac{5}{4}\pi - \frac{6}{4}\pi$	$\frac{6}{4}\pi - \frac{7}{4}\pi$	$\frac{7}{4}\pi - 2\pi$
$y(ángulo)$	↘	↗	↘	↗	↘	↗	↘	↗
δ_1	+	+	+	−	−	−	−	+
δ_2	−	+	+	+	+	−	−	−
$\delta_1 - \delta_2$	+	+	−	−	+	+	−	−
	$\frac{\pi}{4} - y$	$\frac{\pi}{4} + y$	$\frac{3}{4}\pi - y$	$\frac{3}{4}\pi + y$	$\frac{5}{4}\pi - y$	$\frac{5}{4} + y$	$\frac{7}{4}\pi - y$	$\frac{7}{4}\pi + y$

Tabla 2.1: Referencia para el ángulo del contorno

En la tabla se indica el sentido de la flecha, la cual si es hacia arriba se debe de sumar el valor obtenido en la variable y de la formula 2.40, mientras que la flecha hacia abajo se resta.

2.2. Redes neuronales artificiales

Además de los métodos convencionales investigados expuestos en el capítulo 2.1, se presenta también la clasificación de las redes neuronales artificiales.

Las redes neuronales artificiales son una tecnología, que junto con los algoritmos evolutivos y la lógica difusa, son parte de una rama de las ciencias de la computación que se enfoca en el estudio del aprendizaje o razonamiento de un agente no vivo, comúnmente llamada esta rama como inteligencia artificial (AI).

La inteligencia artificial es de gran interés debido a la *capacidad de solucionar problemas* por parte de los agentes no vivos. Ésta característica es aprovechada en el campo de la visión computacional, en donde los paradigmas computacionales no se pueden resolver linealmente.

2.2.1. Técnicas de extracción de contornos basadas en redes neuronales artificiales

Se han desarrollado diferentes técnicas, basadas en redes neuronales artificiales, para extraer los contornos en una imagen digital. Sin embargo no se ha podido desarrollar exitosamente una en la que pueda ser usada universalmente y aplicada en cualquier tarea.

A continuación se muestran algunas de éstas técnicas, las cuales fueron seleccionadas por sus características y la forma en que extraen los contornos de la imagen.

Detección de bordes con redes neuronales BP

La detección de bordes es una herramienta muy importante en el procesamiento de imágenes, es una etapa de pre-procesamiento para la segmentación, detección de movimiento y compresión de imagen. Generalmente se utilizan técnicas para la detección de contornos basadas en los operadores Roberts y Sobel. La mayoría de las técnicas existentes son eficientes pero con grandes limitaciones, como el tiempo de cómputo.

Con las redes neuronales no solo se pueden mejorar las aproximaciones existentes, también se pueden desarrollar nuevas técnicas, de hecho ya han sido utilizadas para la segmentación y para la amplificación de las imágenes. Una red neuronal muy utilizada es la red neuronal de retro-propagación, (BP

por sus siglas en Inglés "BackPropagation") [21], demostrando que una red neuronal BP de 3 capas es capaz de implementar clasificaciones arbitrarias.

La primer técnica analizada propone detectar el borde con algoritmos basados en redes neuronales BP. Ésta técnica clasifica los elementos del contorno en imágenes binarias dentro de 18 categorías de patrones de contornos predefinidas.

La red neuronal BP se construye con tres capas, la capa de entrada de 9 neuronas, la capa intermedia de 8 neuronas y la capa de salida de 18 neuronas.

Cada una de las 35 neuronas contiene una función de transferencia. La función de transferencia que se eligió fue la sigmod, ya que es ideal para la salida con los valores booleanos. La salida de la red neuronal pasa a través de una función de transferencia competitiva, para asegurar que existe un solo resultado con valor 1 en todo el vector y que el resto de los valores sean 0.

Después de entrenar la red neuronal con los patrones de contornos predefinidos, se aplica y clasifica cualquier contorno en las 18 categorías.

Los contornos predefinidos se muestran en la Figura 2.7.

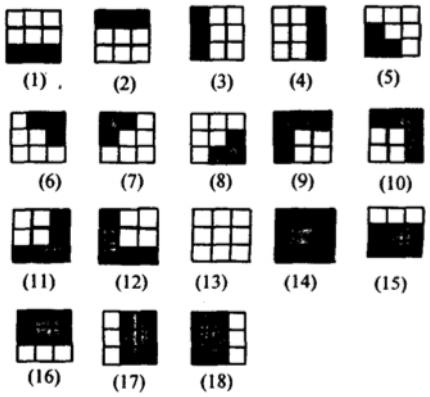


Figura 2.7: 18 patrones binarios de contornos.

Para el caso en que la imagen se encuentre en escala de grises, primeramente se debe de binarizar la imagen, para posteriormente clasificarla dentro de las 18 categorías y así poder entrenar la red neuronal con estos patrones, junto a sus versiones de ruido. Después de entrenar la red neuronal, ésta puede reconocer los patrones de entrada como un patrón dentro de las categorías.

Los resultados de ésta técnica se obtuvieron utilizando una ventana de 9 pixeles (ver figura 2.8), se pasó la ventana por toda la imagen sin traslaparse.

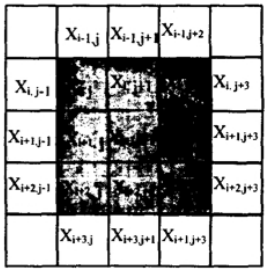
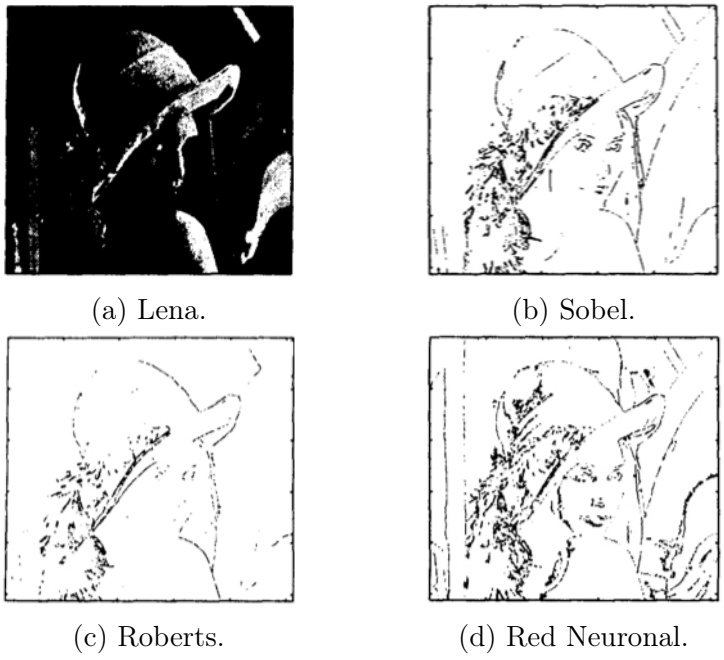


Figura 2.8: Ventana de 9 pixeles.

La imagen con la que se hicieron los experimentos, los resultados del método y el comparativo con los operadores Roberts y Sobel se muestran en la figura 2.9.



(a) Lena. (b) Sobel.

(c) Roberts. (d) Red Neuronal.

Figura 2.9: Detección de borde para la imagen Lena.

La red neuronal proveé mejores resultados que los operadores Sobel y Roberts, dando como ventaja el uso de las redes neuronales para esta aplicación, debido a sus 18 clases de reconocimiento. No obstante la imagen en la que se probó su funcionamiento es una imagen arbitraria y binarizada, por lo que se pierden muchos atributos de los objetos dentro de la imagen.

Detección de bordes utilizando redes neuronales

Así como se tienen métodos para extraer los contornos basados en redes neuronales para aplicaciones arbitrarias, también se tienen aplicaciones muy específicas. Tal es el caso particular para imágenes de LADAR[1] [17].

Se investigó el uso de las redes neuronales en la detección del contorno para imágenes de LADAR, por ser una herramienta muy poderosa. Las redes neuronales actúan como un filtro no lineal que tiene integrada la habilidad para adaptar los umbrales, los cuales se encuentran en función de los pesos entre las conexiones y la función de transferencia.

Como ya se mencionó las redes neuronales se pueden entrenar mediante retro-propagación; o diseñar la red neuronal desde cero, pero se deben de seleccionar sus pesos de acuerdo a las características que se desean para la detección del contorno.

Para el caso en la que la red neuronal es entrenada, específicamente con el método de retro-propagación, tiene la ventaja de que se realiza el entrenamiento (o "aprendizaje") con los datos que se esperan encontrar en el procesamiento y así generalizar la red neuronal. Primeramente se propone el concepto de que el borde es un objeto de la imagen, por lo tanto se debe de entrenar la red neuronal para reconocer un objeto, otorgando a la salida un +1 cuando el objeto ha sido reconocido y -1 cuando no se reconoció. Como los valores de la salida de la red neuronal son ± 1, se usa la función de transferencia sigmoid tangente hiperbólica. No se considera ninguna variación en la rotación o algún desplazamiento, por lo tanto el objeto es reconocido solo si se encuentra en el centro dentro de la ventana de entrada.

La forma en la que se prepara el conjunto de imágenes de entrenamiento, es tomar el objeto que se desea reconocer y pasarlo con un desplazamiento de punto por punto en la ventana que será la entrada de la detección del patrón. Parte del objeto estará afuera de la ventana de entrada, hasta que el objeto este centrado en la ventana, lo cual solo en esa posición la red neuronal dará una respuesta +1 (Ver figura 2.10).

[1] Siglas en Inglés de Laser Detection and Ranging

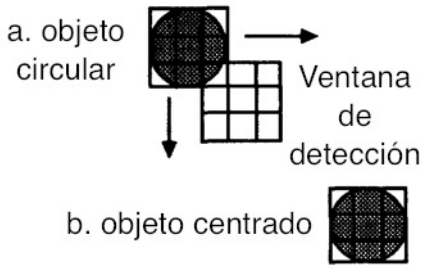


Figura 2.10: Desplazamiento de un objeto en la ventana.

El número de patrones de entrenamiento se determina por el tamaño del objeto y el tamaño de la ventana. Por lo tanto se puede definir que si el objeto es de $n \times n$, la ventana es $m \times m$ y el objeto cabe dentro de la ventana; habrá mínimo $(m + n - 1)^2$ patrones de entrenamiento y se añaden algunos casos especiales (como por ejemplo todo blanco, todo negro y en gris).

El otro método para el desarrollo de la red neuronal para la detección de contornos es el diseño de la red neuronal desde cero. Este método necesita que los pesos se los den directamente a la red neuronal, para crear una función deseada.

Por ejemplo se considera la red neuronal de la figura 2.11, que consiste en 5 entradas y se le da un peso de "bias" con la que crea la función sigmoid, dándole el ancho del cambio con el que la imagen original se cambia a binaria, por lo tanto el peso de "bias" determina el umbral.

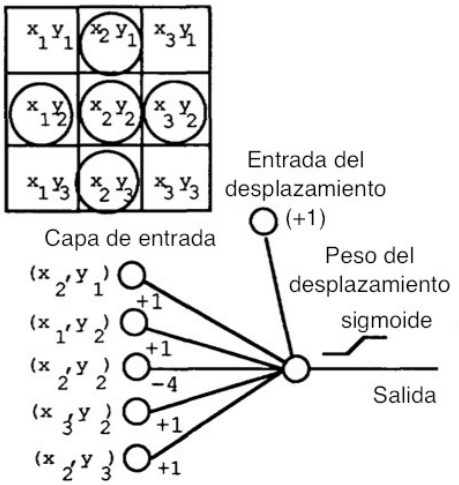
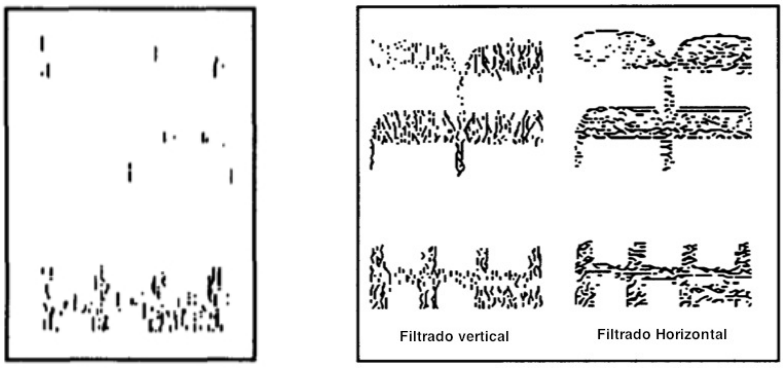


Figura 2.11: Red neuronal simple de detector de Borde.

Se obtuvieron varios resultados con los dos métodos para el desarrollo de redes neuronales. Con el método de entrenamiento por retro-propagación, se pueden crear redes neuronales y entrenarlas para detectar cambios positivos o negativos en la imagen, verticales y horizontales. Combinando estas redes neuronales de forma muy variada se puede crear una detección con diferentes efectos. El producto resultante depende de la configuración de la red neuronal; por ejemplo en una imagen de un puente, se ve que una ventana de 2×2 se define mejor que con una ventana de 7×7 (ver figura 2.12).



(a) Usando ventana 7×7. (b) Usando ventana 2×2.

Figura 2.12: Imagen de Ladar filtrada.

Con la red neuronal diseñada desde cero se obtuvo una imagen muy definida, detectando los bordes, incluso mejor que la red neuronal de entrenamiento.

En los dos métodos no se tuvo ningún problema con el ruido de la imagen, éste fue filtrado con éxito.

Las desventajas que presenta esta técnica es que es una aplicación muy específica, con imágenes especializadas de Ladar y que solo puede reconocer 2 clases (contorno o no contorno).

Redes neuronales para el tratamiento de imágenes: Nuevo algoritmo de detección de bordes

Otra técnica para extraer contornos basadas en redes neuronales artificiales fue utilizada para una aplicación de tiempo real [18], para detección de contornos en video.

El algoritmo creado aprovecha dos operadores de ventanas circulares concéntricas para revelar los contornos como puntos de cruce por cero de una función que solo depende de los valores mínimo y máximo en las ventanas. El método consiste en una detección preliminar del contorno basada en una plantilla de parámetros predefinidos para las características del hardware (precisión del chip 8-bits). Después de obtener la detección preliminar del borde, se aplican las ventanas circulares, las cuales el punto de inicio es en la transición de luminosidad en los puntos flexibles y que un conjunto de pixeles sea constante en su iluminación.

Los resultados obtenidos se muestran en la figura 2.13, exponiendo una segmentación.

Figura 2.13: Imagen de entrada Izquierda; Imagen obtenida derecha.

La técnica que se propuso tiene un excelente desempeño en velocidad de ejecución, ya que se estimó en un 333 frame/s, el cual excede los normalmente 30 frame/s.

Su objetivo fundamental fue la segmentación, pero se tuvo que usar procesamiento previo para poder segmentar las imágenes del video. En nuestra tarea se busca la identificación del micro pistón, para poder dimensionarlo.

La técnica confirma el amplio uso de las redes neuronales en diferentes aplicaciones, dando como pauta para extender las aplicaciones de las redes neuronales, e incursionar en nuestra tarea.

Nueva detección de bordes usando redes neuronales BP sobre umbral binarizado

Una de las características fundamentales en las imágenes digitales y en su procesamiento, análisis y reconocimiento de patrones es el borde, donde la precisión y la confiabilidad para su detección afectará directamente la comprensión objetiva del mundo.

La detección del borde con algoritmos que se basan en redes neuronales tienen mucho auge en los últimos años, en especial las redes neuronales de retro-propagación. Una nueva técnica basada en redes neuronales de retro-propagación, se propuso con la finalidad de tratar de crear un algoritmo arbitrario, que funcione en aplicaciones arbitrarias.

En el procesamiento de imágenes y en visión computacional, la detección de bordes es un proceso que captura distintas propiedades de los objetos, tales como discontinuidades, características geométricas y físicas.

La red neuronal propuesta consta de 16 posibles patrones de borde de imágenes binarizadas y entrenando los patrones en la red neuronal de retro-propagación se obtienen mejores resultados que si se aplicaran técnicas tradicionales [20].

Es muy difícil diseñar un algoritmo general para la detección de bordes que se desempeñe bien en cualquier tarea, es por eso que se han concebido una variedad de detectores de bordes, que disciernen en su propósito, propiedades matemáticas y propiedades del algoritmo.

La gran mayoría de los detectores de bordes se pueden clasificar en 3 grandes categorías: Búsqueda, Cruce por cero y por Umbral. Los detectores de la categoría de Búsqueda, procesan la imagen usando derivada de primer orden (magnitud del gradiente), para después buscar el máximo de la dirección local, para estimar la orientación del borde. Los de la categoría de Cruce por cero, buscan el cruce por cero en imágenes generadas por la segunda derivada. Los de la categoría del Umbral, se pueden dividir en 2 tipos los locales y los globales; solo difieren en el área de la imagen, si se toma toda la imagen o solo una parte.

La técnica se muestra en la figura 2.14, y consiste en primero binarizar las imágenes de escala de grises con el método de Otsu's; después utilizar una ventana de 2×2 pixeles, para clasificarlas en 16 categorías (como lo muestra en la figura 2.15). Ya que se tienen las imágenes binarizadas y en ventanas de 2×2, se entrena la red neuronal con los patrones de las 16 categorías ya predefinidas. Con la red neuronal entrenada, se puede reconocer el borde en

las 16 categorías

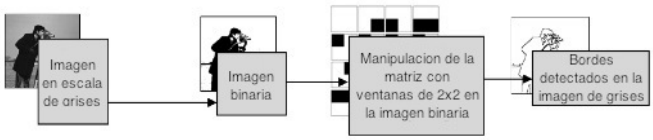


Figura 2.14: Técnica propuesta.

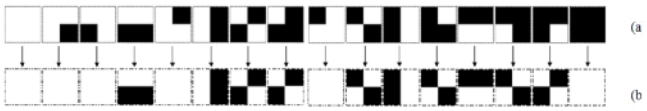


Figura 2.15: (a) Todas las posibles entradas de patrones, (b) todas las posibles salidas.

Los resultados se pueden apreciar en las figuras 2.16 y 2.17, comparando la técnica propuesta con los operadores de Canny, Roberts, Prewitt y Sobel. Se puede observar que el operador Roberts tiene una mala continuidad en los contornos; el operador Sobel, Prewitt y Canny tienen distorsiones.

(a) Imagen en escala de grises.

(b) Imagen binarizada.

(c) Resultado del operador Canny.

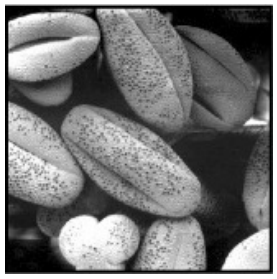
(d) Resultado del operador Sobel.

(e) Resultado del operador Roberts.

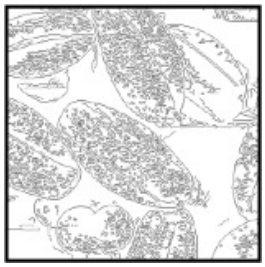
(f) Resultado del operador Prewitt.

(g) Resultado de la técnica propuesta.

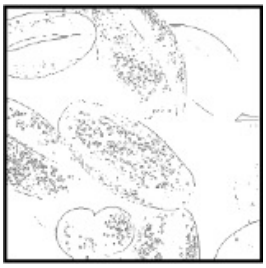
Figura 2.16: Camarógrafo.



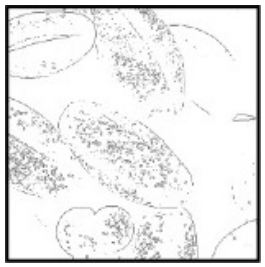
(a) Imagen en escala de grises.



(b) Resultado del operador Canny.



(c) Resultado del operador Sobel.



(d) Resultado del operador Roberts.

(e) Resultado del operador Prewitt.

(f) Resultado de la técnica propuesta.

Figura 2.17: Granos de café.

Ésta técnica es una de las que más se aproxima a la extracción de contornos para nuestra tarea, como se ve en las dos imágenes de resultados los objetos se encuentran muy bien definidos, con posibilidad de poder dimensionarlos por medio de la imagen. El único inconveniente que tiene esta técnica, es que se necesita binarizar la imagen, por lo que se pierde de nuevo caractesísticas del objeto bajo análisis en la imagen.

Detección de bordes basado en redes neuronales con operación de modo de pulso y precisión de punto flotante

La última técnica que se explica en éste trabajo de investigación es para una aplicación arbitraria, pero con una red neuronal con operación de modo de pulso y una precisión de punto flotante, el cual actúa como operador Canny [19].

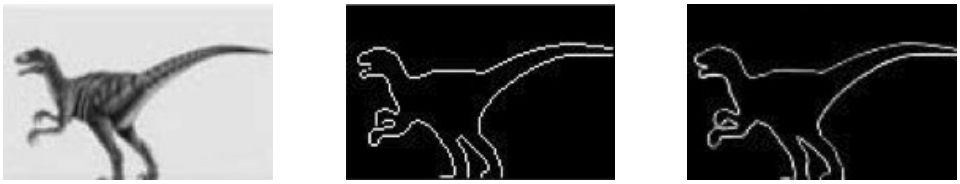
La detección del borde con esta técnica reduce significativamente los datos y filtra información menos importante, conservando propiedades estructurales significativas; es por eso que es muy utilizado y eficiente en imágenes médicas.

Las redes neuronales se han convertido en una atractiva solución para la aproximación de funciones basadas en modulación de frecuencia, en lugar de los métodos tradicionales como el Laplaciano o el operador Canny; pero para tener una mejor detección del borde, se modifica el algoritmo de retro-propagación adaptándolo al hardware. La modificación se realiza en la arquitectura de la red con operación de modo del pulso, usando operaciones de punto flotante para la activación de la función. El diseño se implementó en un sistema virtex II PRO XC2VP7 Xilinx FPGA. La meta es marcar los puntos en una imagen digital, en la que la intensidad de la iluminación tiene un cambio abrupto.

Las redes neuronales multi-capas son usadas en aplicaciones con entradas de alta resolución, haciendo que el aprendizaje no sea posible si el intervalo de los pesos es reducido. Para la técnica propuesta, usa un número de punto flotante para los valores de los pesos, mientras que los niveles de entrada y salida son representados por la frecuencia del pulso de la señal, permitiendo de esta manera que se tenga el amplio intervalo que necesita la red neuronal.

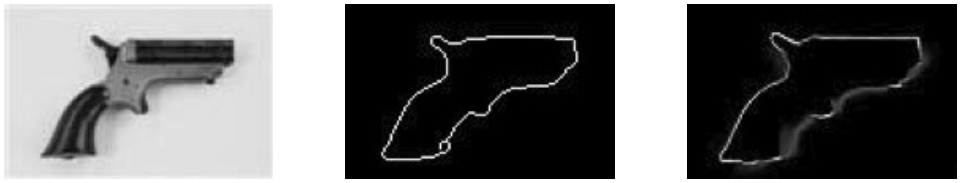
El modelaje de una red neuronal debe de ser simple, para poder ser implementado en un sistema FPGA[2]; ya que si las funciones son complejas, se incrementa drásticamente el tamaño de la red neuronal y aumenta el consumo de recursos del hardware. De acuerdo al algoritmo Canny, el filtro optimo para obtener una aproximación del borde en una imagen se usa la primera derivada de una función gaussiana. Este algoritmo se entrenó a la red neuronal y probada con diferentes imágenes como manos, mamografías o dinosaurios. El entrenamiento de la red neuronal se realiza por medio de una máscara de 3×3 que escanea toda la imagen.

Los resultados obtenidos se observan tanto en las figuras 2.18, 2.19 y 2.20.



(a) Entrada. (b) Canny. (c) Red neuronal.

Figura 2.18: Resultados de la imagen dinosaurio.



(a) Entrada. (b) Canny. (c) Red neuronal.

Figura 2.19: Resultados del fusil.

(a) Entrada. (b) Canny. (c) Red neuronal.

Figura 2.20: Resultados de la mano.

[2]Siglas en Inglés de Field Programmable Gate Array

2.3. Discusión

Como regla general, los métodos para extraer contornos en imágenes generan contornos que no corresponden al borde de los objetos, por lo que hay muchos contornos que se extraen en la imagen (como se aprecia en el capítulo 5 con las imágenes resultantes).

Para hacer las mediciones de los objetos es necesario encontrar algunos métodos que distingan contornos internos del objeto y contornos que corresponden al borde del objeto. El análisis de los métodos de extracción de contornos muestra que como regla general los operadores usan filtros con pequeñas ventanas que contienen un tamaño de algunos pixeles por algunos pixeles. Las ventanas de este tamaño no tienen suficiente información para distinguir el borde del objeto. Los métodos para discernir el borde deben de basarse en ventanas con un tamaño mayor que tengan suficiente información para reconocer el borde del objeto. El objetivo general de la siguiente parte del trabajo es probar la posibilidad de utilizar ventanas mas grandes para entrenar el sistema y distinguir el borde real. Esta tarea es novedosa y la meta de la investigación es obtener las primeras estimaciones de los problemas que posiblemente se tengan que enfrentar con el uso de grandes ventanas.

Para realizar estas nuevas investigaciones es necesario encontrar el método que pueda trabajar con ventanas de gran tamaño. Ya dijimos que los métodos basados en operadores como Sobel, Schwartz y otros no son buenos, porque no permiten trabajar en tiempo real con ventanas grandes. Para la investigación de estos problemas elegimos el clasificador neuronal *LIRA*, que puede trabajar con ventanas suficientemente grandes, como fue mostrado en trabajos previos [38] y [39]. Durante este trabajo fue necesario preparar los programas que permitan entrenar el clasificador *LIRA* con grandes ventanas. En esta investigación elegimos ventanas de 101×101 pixeles, la cual se definió a base de experimentos, del tamaño de la imagen y para centrar el pixel que se procesa. La tarea más importante fue mostrar que en tiempo real si es posible organizar el proceso de entrenamiento del clasificador *LIRA* para realizar en un futuro el reconocimiento del borde real.

Los programas del clasificador *LIRA* con ventanas de 101×101 fueron escritos en lenguaje de programación *Visual* `C#` y los parámetros de estos programas fueron investigados en proceso de trabajo. Los resultados de estas investigaciones son presentados en los capítulos siguientes.

Capítulo 3

Clasificador neuronal *LIRA*

El presente capítulo se dedica a describir el clasificador neuronal LIRA.

En primer lugar se expone el origen del clasificador, la estructura que lo conforma y las interconexiones de las capas dentro de la estructura.

En seguida se detalla en una sección el método de entrenamiento y en otra, la forma en que se calculan los errores.

En la última sección se tienen los algoritmos de cada una de las etapas del clasificador para ser programado; desde la selección de las imágenes para los conjuntos de entrenamiento y prueba, pasando por la construcción de la estructura a través del procedimiento de la máscara y la codificación de las imágenes al aplicar la estructura en ellas, hasta el entrenamiento del clasificador con el conjunto de imágenes de entrenamiento y reconocimiento con el conjunto de imágenes de prueba.

3.1. Estructura

El clasificador neuronal *LIRA* [38] es una red neuronal basada en los principios del perceptrón de Rosenblatt [40]. Rosenblatt desarrolló e investigó el perceptrón de tres capas, en el cual cada capa tenía una función específica. La primera capa es llamada la capa *S-layer* y corresponde a la retina, en nuestra tarea es un sensor o una entrada de una imagen. La segunda capa se le nombra *A-layer* y consiste en la extracción de características en el subsistema. La última capa es conocida como *R-layer* y consiste en que es la salida del sistema, donde cada neurona de la capa corresponde a cada clase. El clasificador neuronal *LIRA* tiene una capa adicional, ésta es una capa

intermedia entre la capa A y la capa S. La estructura del clasificador *LIRA* y sus conexiones se muestran en la figura 3.1.

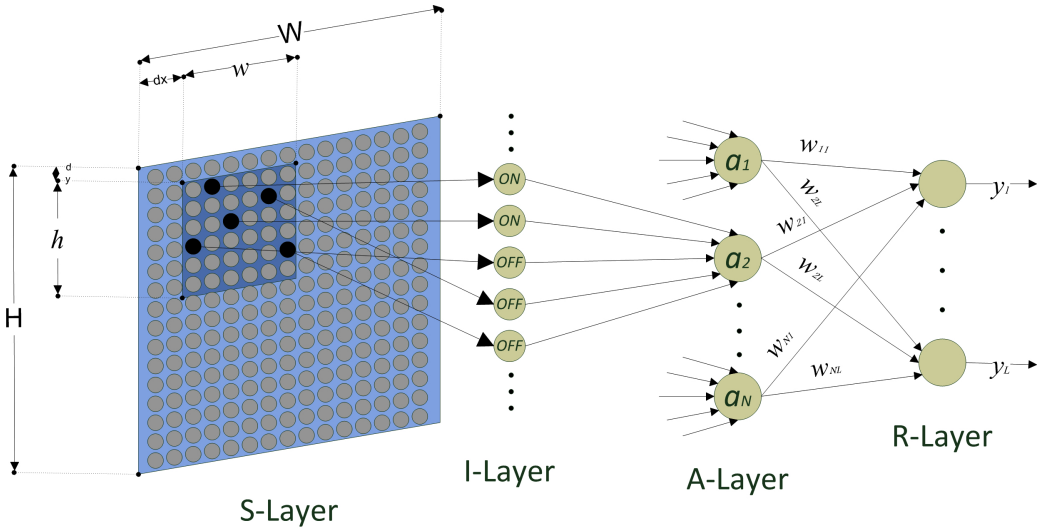


Figura 3.1: Estructura del clasificador neuronal *LIRA*

Existen dos variantes del clasificador neuronal *LIRA*, la primera es el clasificador *LIRA_binary* y fue desarrollado para imágenes binarias, por lo tanto solo puede tener como entrada a la red neuronal una imagen en blanco y negro. La otra variante es el clasificador neuronal *LIRA_grayscale* [39]. Este clasificador fue desarrollado para tener como entrada en la red neuronal imágenes en escala de grises. En este trabajo proponemos usar el clasificador *LIRA_grayscale* en la tarea de reconocer el borde de las micro piezas.

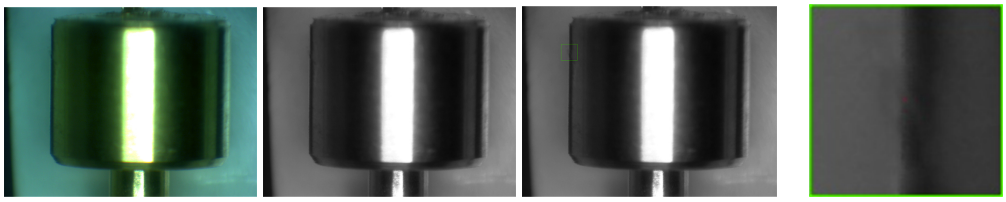
Las conexiones entre cada una de las capas en la red neuronal del clasificador *LIRA_grayscale* son diferentes. Las capas S y A están conectadas a través de la capa I, estas conexiones no son entrenables. En primer lugar se define aleatoriamente un rectángulo con un área de $(h \times w)$ dentro de *S-layer* (ver figura 3.1); dentro de esta área, se seleccionan arbitrariamente M neuronas (pixeles de la imagen de entrada). Cada una de estas M neuronas están conectadas directamente a una neurona de la capa I.

La capa I está compuesta por dos tipos de neuronas, neuronas del tipo *ON* y neuronas del tipo *OFF*. Las neuronas *ON* son activadas cuando la entrada de cada neurona es mayor a un umbral generado aleatoriamente; por otro lado, las neuronas *OFF* son activadas solo cuando la entrada es menor que otro umbral generado aleatoriamente. La salida de las neuronas de la capa I están conectadas a la capa A.

Las neuronas de la capa A tienen dos estados, activada o inhibida. Cada neurona de la capa A será activada solo si todas las entradas conectadas desde la capa I están activadas.

Las últimas conexiones entre la capa A y la capa R constan en que cada salida de las neuronas de la capa A están conectadas a cada neurona de la capa R, y cada conexión tiene un peso específico. Al principio, todos los pesos están establecidos a un valor predeterminado, y se va modificando conforme al procedimiento de entrenamiento. Esos pesos son muy importantes para poder distinguir cada clase en la imagen y para tener un mejor desempeño de clasificación.

Antes de ejecutar el algoritmo de *LIRA_grayscale*, es necesario realizar algunos procedimientos preliminares con las imágenes, como se muestra en la figura 3.2. La imagen se convierte a una imagen en escala de grises; después se localiza cada pixel que se encuentra bajo análisis. Cerca de ese pixel, se selecciona un área de $W \times H$. Esa área es usada como la entrada para la capa S y así poder ejecutar el algoritmo *LIRA_grayscale*, como se presenta en la figura 3.3.



(a) Imagen de entrada. (b) Imagen en escala de gris. (c) Localización del pixel para entrenamiento. (d) Área seleccionada.

Figura 3.2: Etapas preliminares de preparación de imágenes.

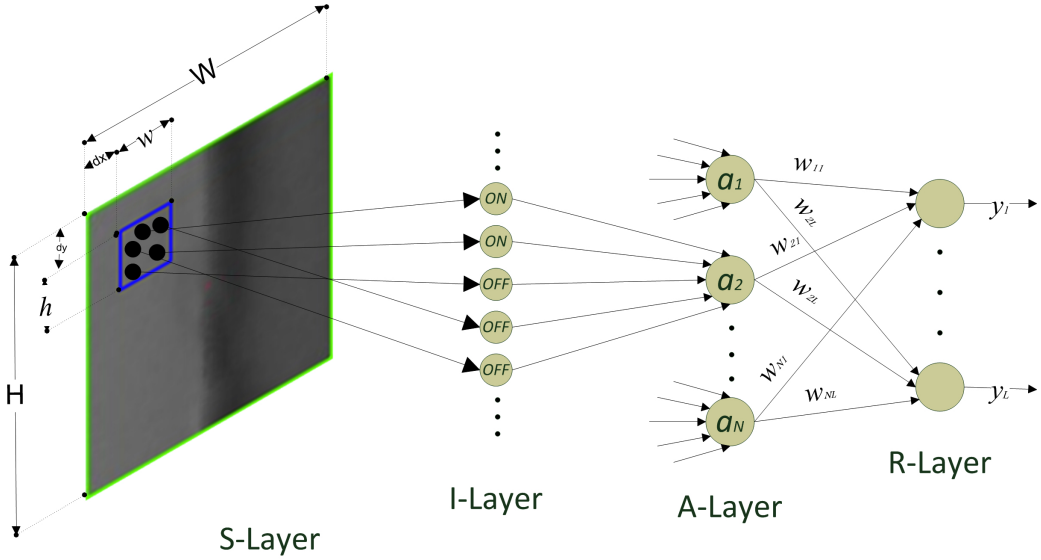


Figura 3.3: Estructura de *LIRA_grayscale* para micro pistones

3.2. Algoritmo de entrenamiento

El algoritmo de entrenamiento es un método supervisado. Esto significa que el clasificador neuronal en proceso de entrenamiento tiene que "saber" a que clase pertenece la imágen. Se inicia seleccionando la primera imagen del conjunto de entrenamiento. Se extraen las características de la imagen. Se calcula la excitación de cada neurona de la capa R. El resultado de cada neurona se define de acuerdo a la ecuación 3.1.

$$y_i = \sum_{k=1}^{N} w_{ki} a_k \tag{3.1}$$

donde y_i es la salida de la i-esima neurona de la capa R; a_k es la salida k-esima neurona de la capa A; w_{ki} es el peso de la conexión entre la k-esima neurona de la capa A y la i-esima neurona de la capa R.

Ademas se añade una condición mas, la cual hace que la red neuronal sea mas efectiva y robusta. Esta condición se aplica a la neurona ganadora de la capa R, de acuerdo a la ecuación 3.2.

$$y_r = y_r(1 - T_e) \tag{3.2}$$

Después de que la neurona ganadora ha sido seleccionada, esta neurona representa a la clase ganadora.

Si la clase ganadora corresponde con la clase deseada, no se realiza ninguna acción; en cambio si la clase ganadora no corresponde a la clase deseada, los pesos son modificados de acuerdo al criterio de la expresión 3.3.

$$\begin{aligned} \forall k, w_{kr}(t+1) &= w_{kr}(t) + C \\ \forall k, w_{kg}(t+1) &= w_{kg}(t) - C \end{aligned} \tag{3.3}$$

donde $w_{kr}(t)$ y $w_{kr}(t+1)$ son los pesos de las conexiones entre la k-esima neurona de la capa A y la r-esima neurona de la capa R de las neuronas de la clase deseada (neuronas reales) antes y después de las modificaciones; $w_{kg}(t)$ y $w_{kg}(t+1)$ son los pesos de las conexiones entre la k-esima neurona de la capa A y la g-esima neurona de la capa R de las neuronas ganadoras antes y después del ajuste; C es una constante para cambiar el peso.

Para obtener mejores resultados y distinguir el número de errores entre el reconocimiento de clases, el algoritmo de entrenamiento debe de ser iterativo. Un ciclo en la iteración de entrenamiento se realiza cuando se probaron todas las imágenes del conjunto de entrenamiento, y el número de errores se calculó. El número de iteraciones se encuentra pre-definido en el programa.

3.3. Cálculos de errores

En la red neuronal, el algoritmo de entrenamiento y el procedimiento de reconocimiento en la imagen, es necesario conocer su tasa de error. La tasa de error define el porcentaje del número de pixeles no reconocidos, del total de los pixeles que se analizaron.

El cálculo de la tasa de error requiere de ciertas variables dependiendo de la etapa del clasificador. En la etapa de entrenamiento, las variables principalmente son el número de errores y el total del número de pixeles que se entrenan en cada iteración del conjunto de imágenes para entrenamiento. Para la etapa del reconocimiento, las variables son el número total de errores y el número total de pixeles del conjunto de imágenes de prueba.

Éstas variables se ingresan a la ecuación 3.4.

$$\%error = (100 * Errores)/TP \tag{3.4}$$

en donde *%error* es la tasa de error, *Errores* es el número de pixeles que erró el sistema en clasificar de cada iteración en la etapa de entrenamiento ó el número total de pixeles que no fueron reconocidos correctamente para la etapa de reconocimiento, y *TP* es el total de pixeles analizados de cada iteración en la etapa de entrenamiento ó el total de pixeles del conjunto de imágenes de prueba.

La tasa de error es graficada en la etapa de entrenamiento, con el objetivo de observar si existe convergencia a cero, como lo muestra la figura 3.4, demostrando que el sistema es capaz de "aprender". En la gráfica en el eje de las abscisas se representan las iteraciones y en las ordenadas el número de errores.

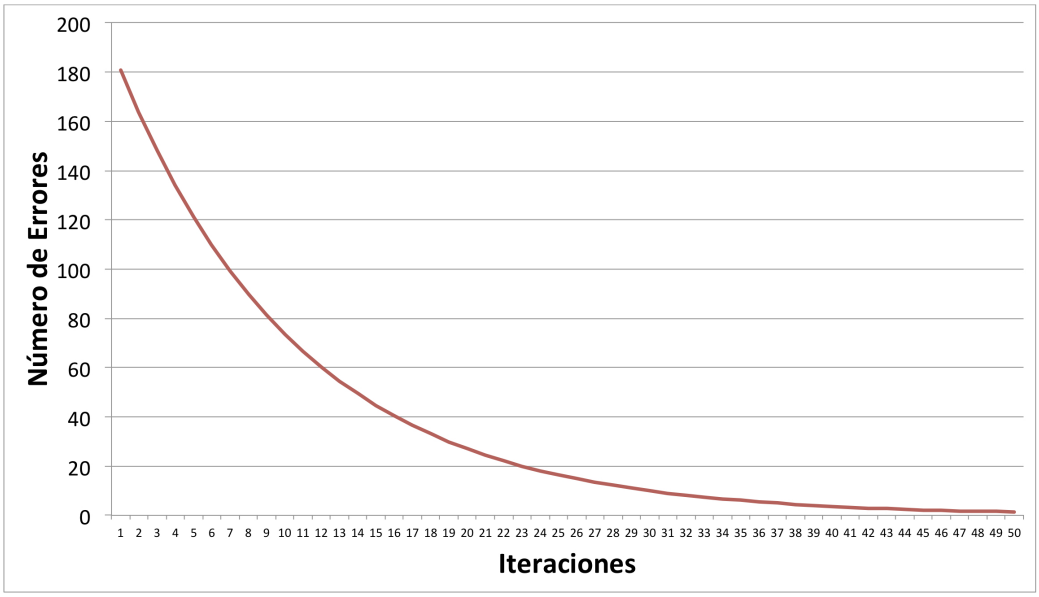


Figura 3.4: Ejemplo de gráfica de errores.

3.4. Algoritmo

A continuación en este capítulo se muestra cada parte del algoritmo que conforma a la red neuronal artificial *LIRA*, sus diagramas de flujo, así como el nombre del botón en donde fue implementado en el programa.

3.4.1. Conjunto de imágenes para entrenamiento y prueba

La red neuronal artificial *LIRA* necesita dos conjuntos de imágenes, uno de los cuales se utiliza para el entrenamiento y el otro para realizar la prueba de la red neuronal. Los dos conjuntos se definen mediante la base de imágenes previamente obtenida, descrita en el capítulo 4. De esta base de imágenes se seleccionan cierto número de imágenes para el conjunto de entrenamiento y el resto de las imágenes se utilizan para crear el conjunto con el que se prueba el sistema.

Existen diferentes formas de elegir el conjunto de imágenes para entrenamiento. La selección se realiza por ejemplo en forma de las primeras n imágenes, patrón de ajedrez o aleatoriamente.

La selección de las imágenes en forma de las primeras n imágenes, se realiza tomando en cuenta las primeras imágenes para entrenamiento.

En el caso de la forma de ajedrez, se efectúa tomando la primera imagen para prueba, la segunda para entrenamiento y se van intercalando una a una, entre prueba y entrenamiento.

La última opción es en forma aleatoria, se realiza escogiendo aleatoriamente las imágenes, hasta que se tenga el número de imágenes para entrenamiento deseadas.

En las tres formas se crea un vector $\vec{V}$, donde $\vec{V} = \{v_i | v_i = \{0, 1\}, i \in (0, 15)\}$, esto quiere decir que si el índice del vector es 5 (i=5) y el vector es 1 (V_5=1), la imagen 6 es de entrenamiento. Este vector se almacena en un archivo para su posterior utilización.

Su algoritmo por pasos se realiza de la siguiente manera, como se muestra en el diagrama de flujo de la figura 3.5.

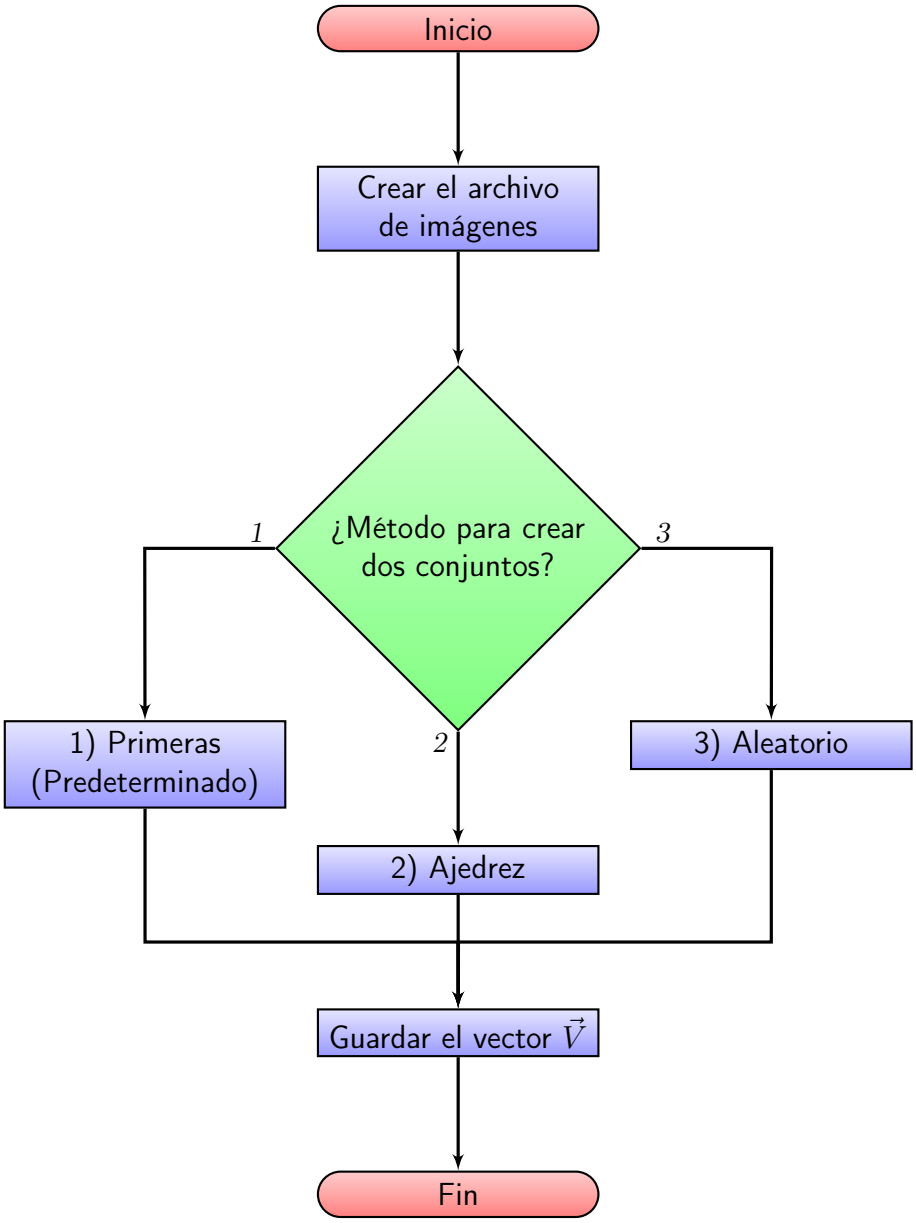


Figura 3.5: Procedimiento de selección del conjunto de entrenamiento y prueba

En el programa este algoritmo se encuentra implementado en el botón "Conjuto de Prueba y Entrenamiento"

3.4.2. Máscara

En este proceso se crea la estructura de la red neuronal artificial. Primeramente se define el tamaño de la red neuronal artificial como lo indica la expresión 3.5.

$$N = \big(N_1 + N_2\big) \cdot N_A \tag{3.5}$$

donde N es Netsize, N_1 el número de neuronas ON, N_2 el número de neuronas OFF, N_A numero de neuronas en la capa A.

Después se genera un vector con umbrales aleatorios $\vec{U}$, donde $\vec{U} = \{u_k | u_k = \{0 - 255\}, k \in (0, NetSize)\}$.

Ya que se tiene el vector $\vec{U}$ con los umbrales, se define la posición de la muestra con las coordenadas $\vec{P}(x, y)$, las cuales se localizan en la parte superior izquierda; como se señala en la figura 3.6. Dentro de esa muestra de tamaño $W \times H$ se obtiene el vector $\vec{P}1(x1, y1)$, con las posiciones aleatorias de las ventanas (pequeñas $w \times h$) que contienen las neuronas ON y OFF de la capa I. Cada una de las posiciones del vector $\vec{P}1$ de esas pequeñas ventanas corresponden a la parte superior izquierda.

El último paso de la construcción de la estructura, es el generar el vector $\vec{P}2(x2, y2)$ que contiene las coordenadas de las posiciones aleatorias $x2$ y $y2$ para las neuronas ON y OFF.

El algoritmo está realizado de acuerdo al diagrama de flujo de la figura 3.7

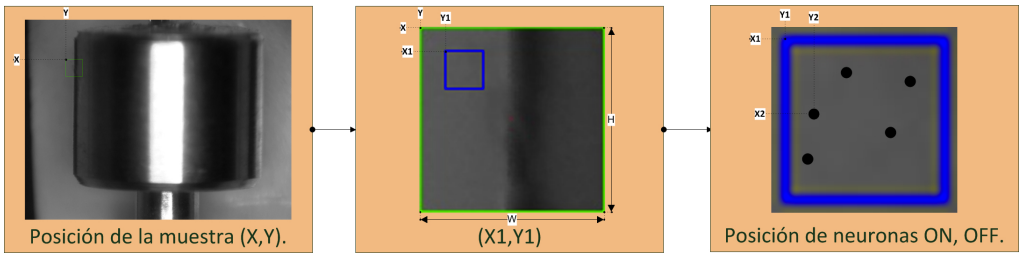


Figura 3.6: Posicionamiento de ventanas y neuronas “ON” y “OFF”

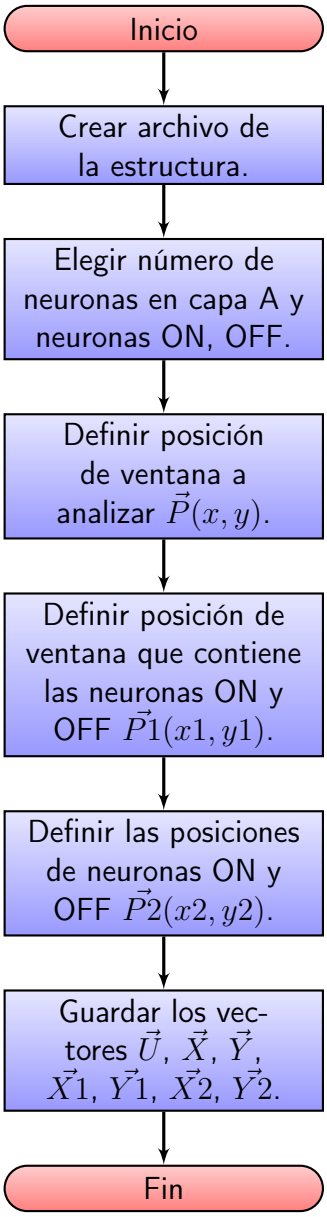


Figura 3.7: Procedimiento de la Máscara

En el programa el algoritmo se encuentra implementado en el botón "Mascara".

3.4.3. Codificación de Imágenes

Ya que se tiene la definición de la red neuronal artificial *LIRA*, creada con el procedimiento de la máscara (ver sección 3.4.2), es necesario codificar la imagen.

La codificación de la imagen consiste en aplicar la estructura de *LIRA*, comenzando con el área seleccionada, esta área se considera como la capa S. Dentro de la capa S se posicionan las ventanas que contienen las neuronas "ON" y "OFF".

Ya que se tiene cada ventana posicionada, se ubican los pixeles con las coordenadas de la neuronas "ON" y "OFF". De cada uno de los pixeles se extrae el valor de los brillos, para después ingresarlos a las entradas de las neuronas "ON" y "OFF".

Las salidas de las neuronas "ON" y "OFF" se activan o inhiben de acuerdo a su umbral correspondiente del vector $\vec{U}$. Las neuronas "ON" se activan cuando la entrada es mayor al umbral y se inhibe cuando es menor. En cambio, las neuronas "OFF" se activan cuando la entrada es menor al umbral y se inhibe cuando es mayor.

Posteriormente de tener las salidas de todas las neuronas "ON" y "OFF", éstas se introducen en las entradas correspondientes a cada una de las neuronas a en la capa A. Cada neurona de la capa A se excita si y solo si todas las entradas de esa neurona se encuentran activadas, de lo contrario se inhibe.

Las neuronas de la capa A se asignan al vector $\vec{A}$, el cual se guarda para su posterior uso en el entrenamiento de la red neuronal y en el reconocimiento de los pixeles y su clasificación.

Cabe señalar que el pixel bajo análisis siempre se encuentra en el centro de la ventana de análisis, el cual en el entrenamiento (ver 3.4.4) se supervisa que sea contorno, fondo u objeto.

Este algoritmo se aprecia en el diagrama de flujo de la figura 3.8, y su implementación se realizó en el botón "Begin Coding" del programa.

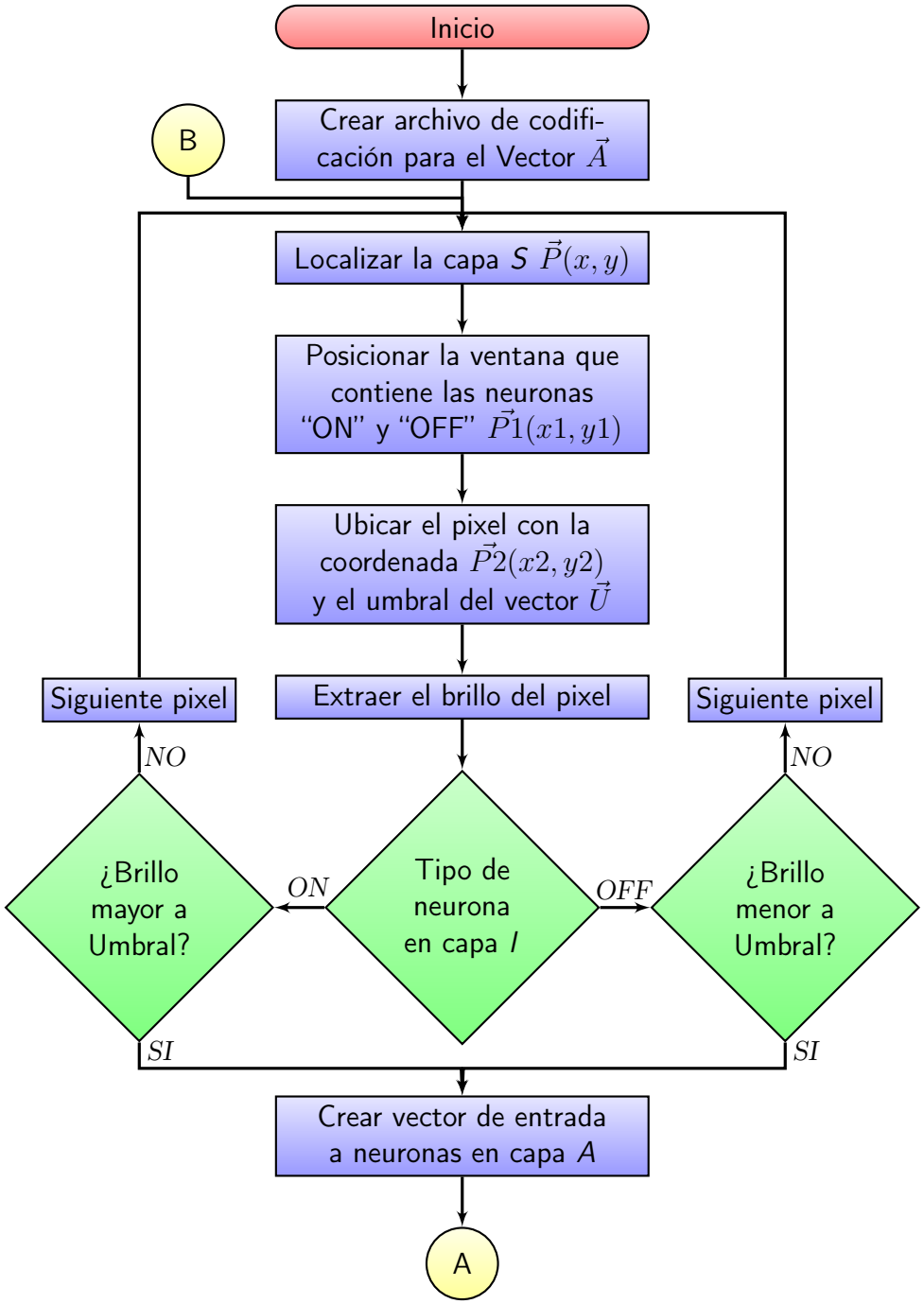


Figura 3.8: Procedimiento de la Codificación

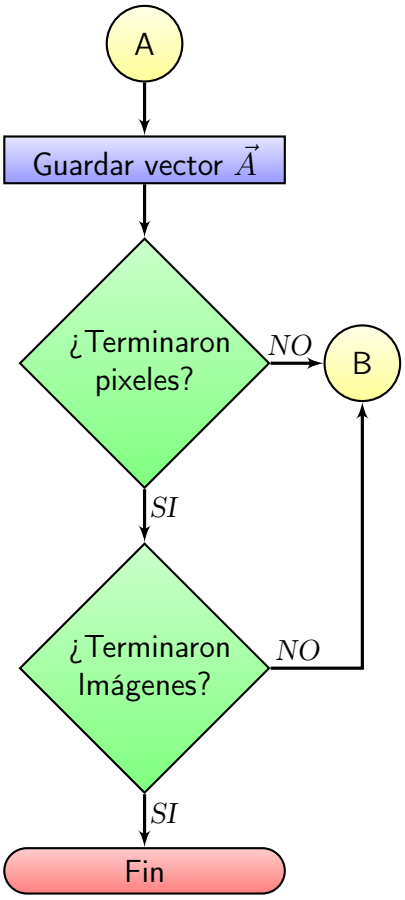


Figura 3.9: Continuación del procedimiento de la Codificación

3.4.4. Entrenamiento de la Red Neuronal

El entrenamiento de la red neuronal artificial *LIRA* utiliza la codificación de las imágenes previamente obtenidas en el procedimiento de la codificación (ver sección 3.4.3) y el vector $\vec{V}$ con el conjunto de entrenamiento y prueba antes creado (ver sección 3.4.1).

El entrenamiento se basa en hacer que la neurona ganadora sea de la clase correcta (contorno, fondo u objeto), como se describe en la sección 3.2. Para seleccionar a la neurona ganadora se toman en cuenta los pesos w_{kr} entre las conexiones de la capa A y la capa R.

Los pesos se relacionan entre la neurona de la capa A y la neurona de la capa R por medio de una matriz. La matriz se modifica de acuerdo a la expresión 3.3 en cada iteración, hasta que se tenga el número de iteraciones deseadas o el número de errores sea menor a uno propuesto.

La modificación de los pesos se describe en la tabla 3.1.

Pertenece a:	Reconocido como:	Acción a ejecutar
contorno	contorno	nada
fondo	fondo	nada
objeto	objeto	nada
contorno	fondo	modificar la matriz con los pesos, incrementando las relaciones de la clase contorno y reduciendo las relaciones de las clases fondo y objeto.
fondo	objeto	modificar la matriz con los pesos, incrementando las relaciones de la clase fondo y reduciendo las relaciones de las clases contorno y objeto.
objeto	contorno	modificar la matriz con los pesos, incrementando las relaciones de la clase objeto y reduciendo las relaciones de las clases contorno y fondo.
contorno	objeto	modificar la matriz con los pesos, incrementando las relaciones de la clase contorno y reduciendo las relaciones de las clases objeto y fondo.
objeto	fondo	modificar la matriz con los pesos, incrementando las relaciones de la clase objeto y reduciendo las relaciones de las clases contorno y fondo.
fondo	contorno	modificar la matriz con los pesos, incrementando las relaciones de la clase fondo y reduciendo las relaciones de las clases contorno y objeto.

Tabla 3.1: Relación para la modificación de pesos

En el diagrama de flujo de la figura 3.10 se muestra como fue implementado en el programa, en el botón con el nombre "Begin Train".

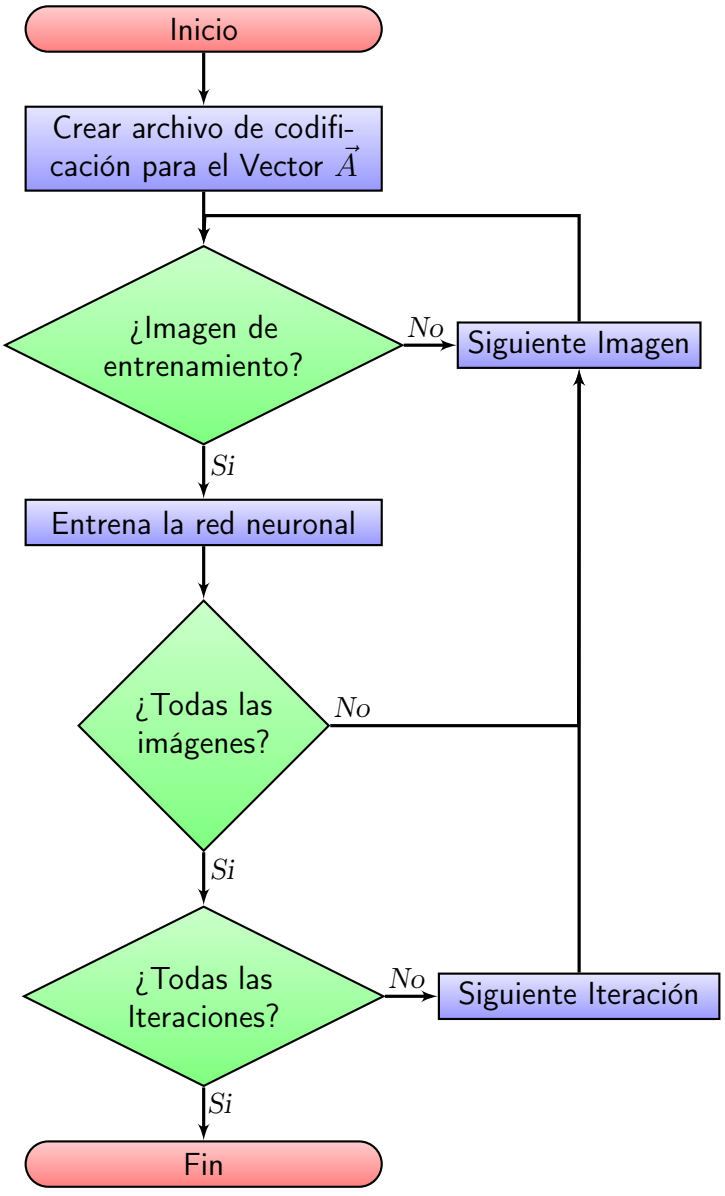


Figura 3.10: Procedimiento de Entrenamiento

Parte II

IMPLEMENTACIÓN Y DESARROLLO

Capítulo 4

Descripción de la base de imágenes

La tarea en la que se enfocó esta investigación requería de varias imágenes. El número de imágenes depende de la robustez deseada en el sistema, mientras más imágenes se tengan, se extraen más características de la imagen y mejor será el entrenamiento y prueba del clasificador neuronal, incrementando así el porcentaje de reconocimiento.

Se creó una base de imágenes inicial de 15 micro pistones, para adaptar el clasificador *LIRA* a la tarea de dimensionamiento de micro piezas.

4.1. Micro pistones y sus características

Los micro pistones fueron manufacturados con ayuda de un torno *Sherline*®, modelo 4410 (ver figura 4.1 y especificaciones en el apéndice B). Éstos micro pistones se hicieron con diferentes diámetros, los cuales tienen un intervalo de 0.1mm. El diámetro inicial es de 8.0mm hasta llegar a una medida cercana de 8.5mm (ver tabla 4.1).

Se fabricaron 15 micro pistones, los cuales se clasificaron en 5 grupos de 3 pistones (ver tabla 4.1). El criterio que se tomó para su clasificación fue la dimensión del diámetro, donde cada grupo tiene casi el mismo diámetro con pequeñas variaciones de centésimas, debido a la inexactitud del equipo con el que se fabricaron.

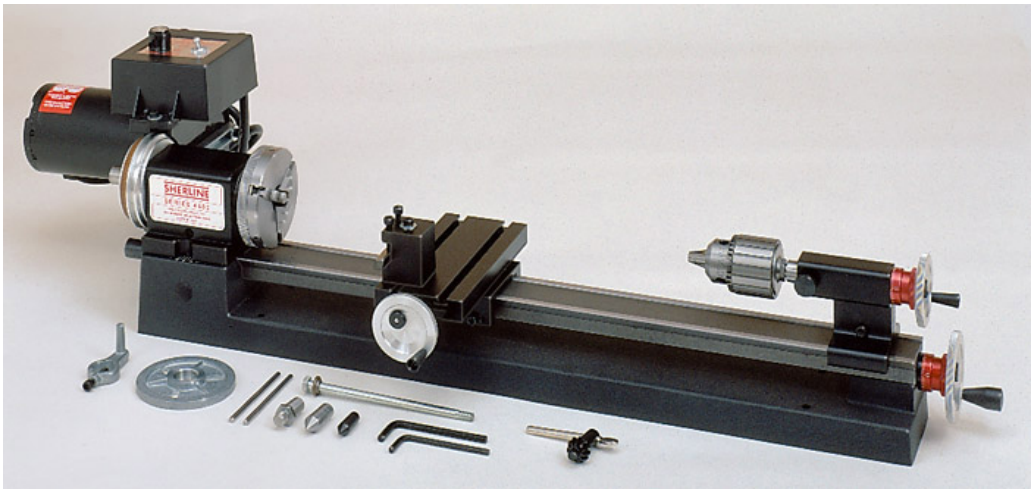


Figura 4.1: Torno Sherline.

Pistón	Diámetro medido [mm]	Grupo
1	8.43	1
2	8.43	1
3	8.46	1
4	8.35	2
5	8.32	2
6	8.33	2
7	8.22	3
8	8.20	3
9	8.23	3
10	8.11	4
11	8.16	4
12	8.15	4
13	8.04	5
14	8.03	5
15	8.01	5

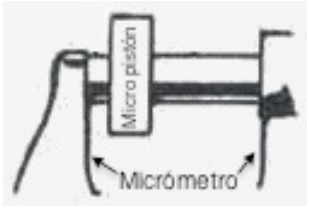
Tabla 4.1: Clasificación de los pistones

4.1.1. Medición de los micro pistones

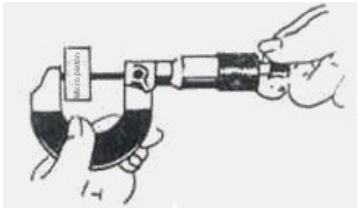
La medición de cada micro pistón se hizo con un micrómetro digital *Starrett®*, modelo 756, con una resolución de ±0.001mm. El procedimiento para medir los micro pistones es directo, ya que el micrómetro consta de un cuerpo con un tope fijo y otro móvil provisto de una cabeza micrométrica; así como de un LCD [1] que proyecta la lectura del tamaño medido.

Cada uno de los micro pistones se midieron de la siguiente manera:

1. Se colocó el micro pistón entre las puntas en forma de disco del micrómetro, como se ilustra en la figura 4.2a.
2. Se ajustó el tornillo hasta que el micro pistón entrara en contacto con los discos y se ejerciera una pequeña presión de medición adecuada.
3. Se tomó la lectura del tamaño del micro pistón.
4. Se liberó el micro pistón después de completada la medición.



(a) Ubicación del micro pistón en el micrómetro.



(b) Ajuste del micrómetro al micro pistón que se está midiendo.

Figura 4.2: Medición con micrómetro.

4.2. Imágenes y sus propiedades

Posteriormente de haber medido los distintos micro pistones, se adquirieron las imágenes de cada uno de ellos (ver figura 4.3), con ayuda de la cámara *AmScope®* modelo MD700 (ilustrada en la figura 4.4 y sus especificaciones en el apéndice C) del microscopio metalúrgico trinocular *NJF-120A* (ver figura 4.5 y especificaciones en el apéndice D).

[1] Sigla en Inglés "Liquid Crystal Display"

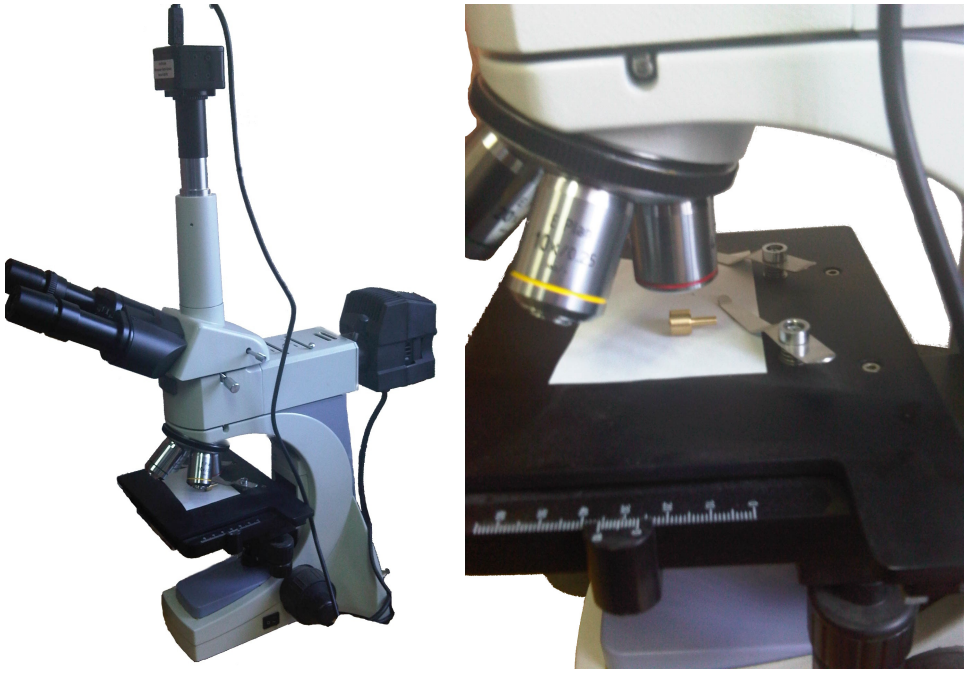


Figura 4.3: Adquisición de imagen del micro pistón.

(a) Cámara y sus accesorios.

(b) Cámara montada en microscopio.

Figura 4.4: Cámara AmScope.

Figura 4.5: Microscopio metalúrgico trinocular NJF-120A.

Las imágenes capturadas, como se ilustra en los ejemplos de la figura 4.6, tienen una resolución de 1600 x 1200 pixeles[2], es la máxima resolución que la cámara puede otorgar para capturar los más posibles detalles del pistón. El formato con que se adquirieron las imágenes fué en BMP[3], con un modelo de color RGB[4].

Se eligió este formato, desarrollado por *Microsoft*© e *IBM*©, por su forma de almacenamiento sin comprimir, el cual es un mapa de bits, es decir en un archivo de imagen de gráficos, con pixeles almacenados en forma de tabla de puntos que administra los colores, como colores reales o usando una paleta indexada.

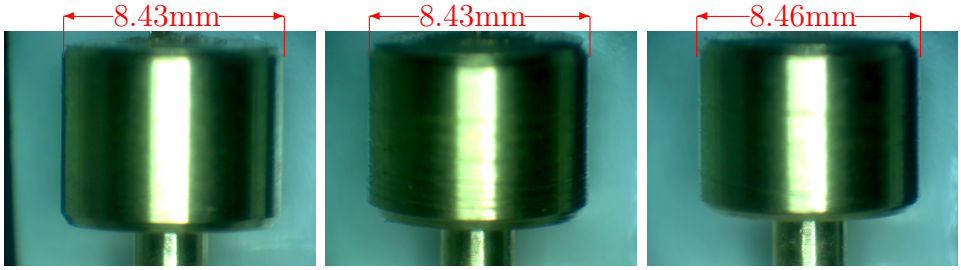


Figura 4.6: Ejemplo de imágenes adquiridas de los pistones.

[2]1920000 pixeles, aproximadamente 2 megapixeles.
[3]Mapa de Bits, por sus siglas en Inglés BitMaP
[4]Siglas en Inglés de Red, Green, Blue

4.3. Tratamiento de las imágenes

Uno de los primeros requerimientos, para el entrenamiento y prueba del clasificador neuronal, es el marcado del perímetro del objeto en las imágenes, es decir el borde del micro pistón. Las 15 imágenes fueron marcadas, ya que se tienen tres formas de seleccionarlas, descrito el procedimiento en la sección 3.4.1.

El marcado del micro pistón es a criterio del entrenador del clasificador neuronal, tanto su grosor y la posición en donde se define la frontera entre el objeto y el fondo. Esta etapa se debe al tipo de entrenamiento para la red neuronal, el cual es de BP [5].

La forma en que se hace el marcado de la imagen es por medio de un tratamiento a la imagen, descrito en la sección 4.3.1.

4.3.1. Marcado de imágenes

El programa que se utilizó para marcar el borde de los pistones fue *GIMP*[6]; por su versatilidad en la manipulación de imágenes. Con este programa se convirtió la imagen de color a escala de grises, posteriormente se marca el borde del micro pistón, para finalmente seccionar la imagen en clases, como lo son el fondo, el borde y el objeto.

El marcado del contorno se realizó de la siguiente manera:

- Se abre la imagen del objeto a marcar con el programa GIMP, mediante la opción "Abrir" del menú "Archivo" (Archivo->Abrir, ver figura 4.7).

[5]Backpropagation (retro-propagación, es decir supervisado)

[6]GNU Image Manipulation Program, programa de edición de imágenes digitales en forma de mapa de bits

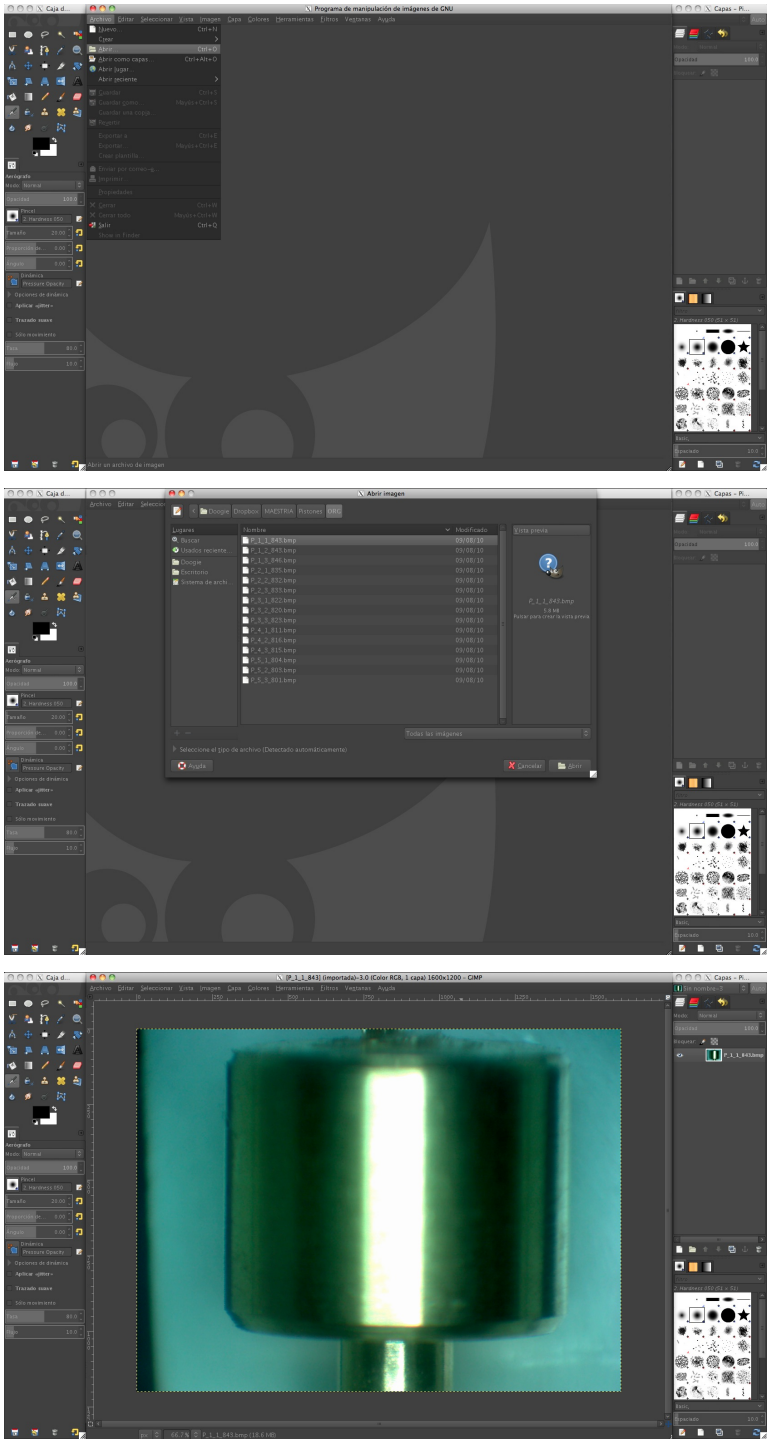


Figura 4.7: Abrir imagen en GIMP.

- Se cambia el modo a escala de grises. Para realizar este tratamiento se selecciona la opción "Escala de grises" del submenú "Modo" del menú de "Imagen" (Imagen->Modo->Escala de grises, ver figura 4.8).

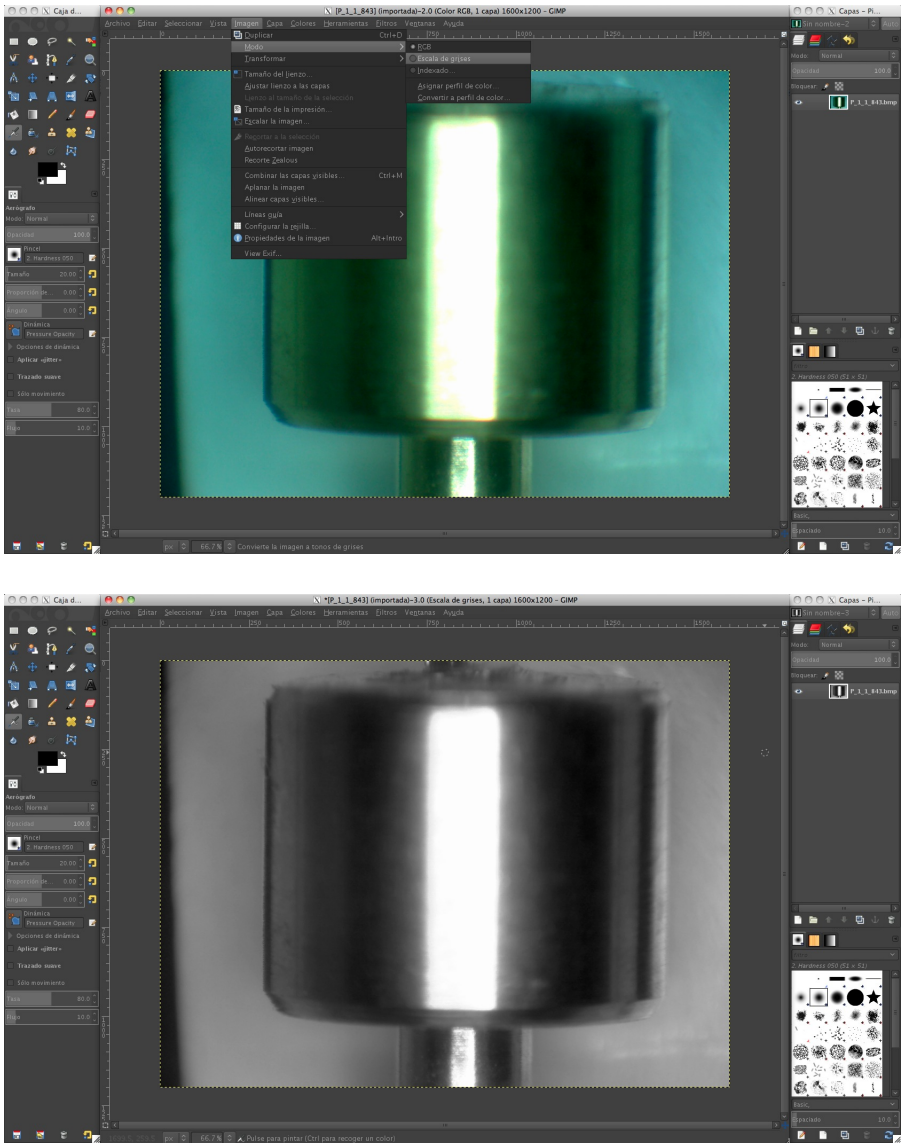


Figura 4.8: Cambio de la imagen a escala de grises.

- Se selecciona el contorno del objeto con ayuda de la herramienta "rutas", que se encuentra en el menú "Herramientas" (Herramientas->Rutas, ver figura 4.9). Su precisión depende del número de puntos con el cual se marca el contorno y del ángulo que se le da al arco entre los puntos. Para agregar mas puntos, se presiona la tecla "control".

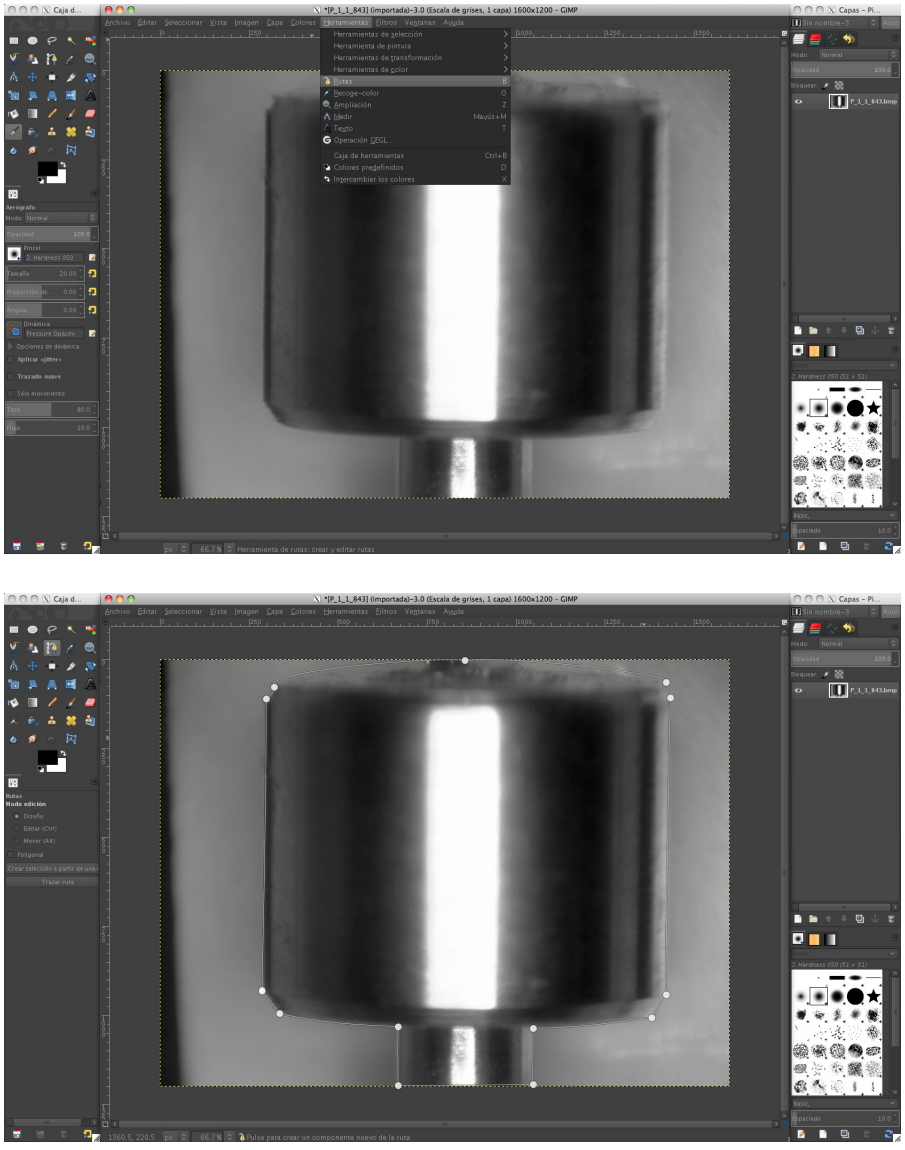


Figura 4.9: Delineamiento de la ruta.

- Se cierra la "ruta" del contorno del objeto por medio de la opción "A partir de una ruta" del menú "Seleccionar" (Seleccionar->A partir de una ruta, ver figura 4.10).

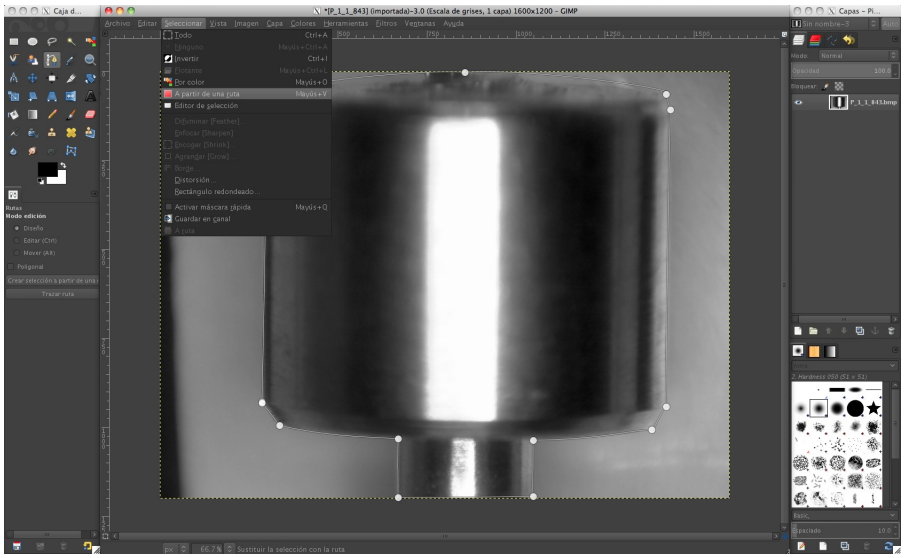


Figura 4.10: Cierre de ruta.

- Se resalta el borde definiendo su ancho en pixeles, a través de la opción "Borde" del menú "Seleccionar" (Seleccionar->Borde, ver figura 4.11).

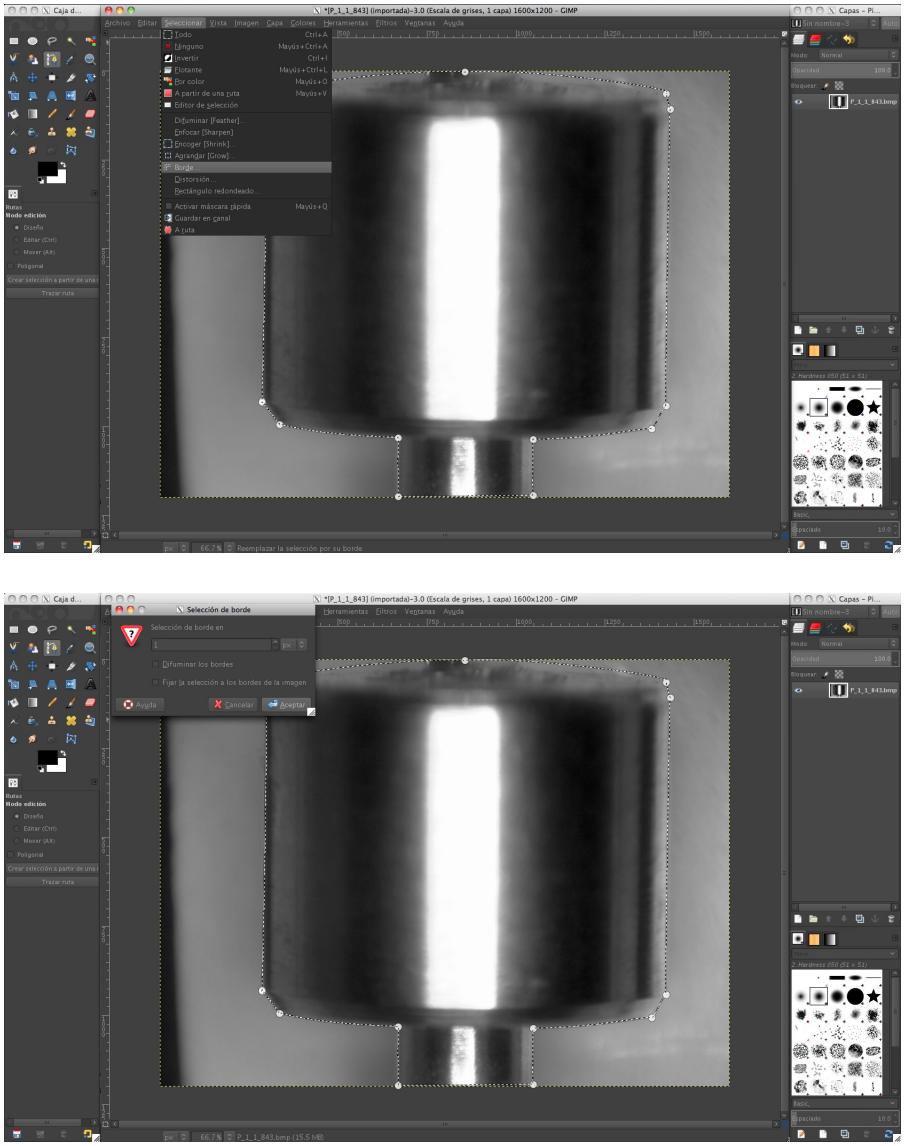


Figura 4.11: Especificación del espesor del borde.

- Se resalta el borde de un color que se contraste con el objeto y el fondo, se realiza eliminando el interior de la selección del borde y rellenando con la "herramienta de relleno" (ver figura 4.12).

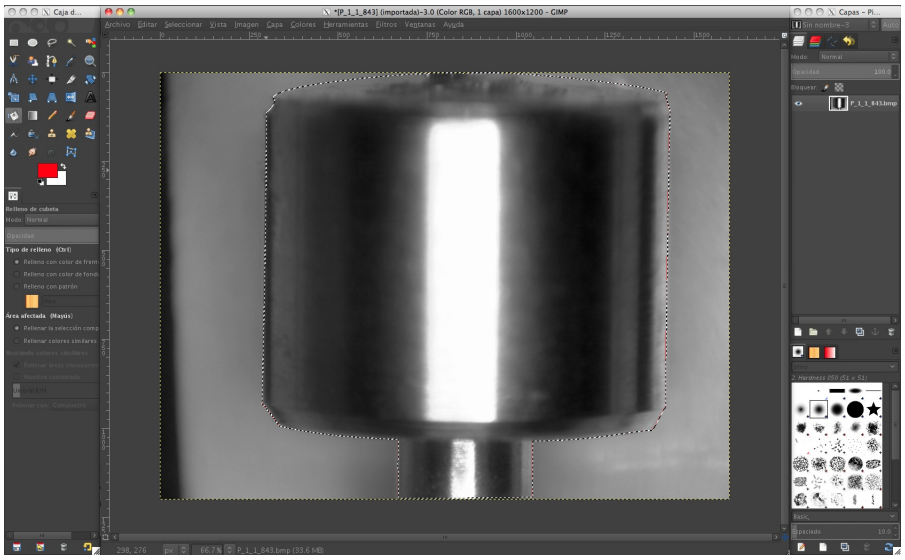


Figura 4.12: Resaltado del borde.

- Se invierte la selección para resaltar cada clasificación, eliminando el interior del objeto y del fondo, esto se realiza con la herramienta "Invertir" del menú "Selecionar" (Seleccionar->Invertir, ver figura 4.13).

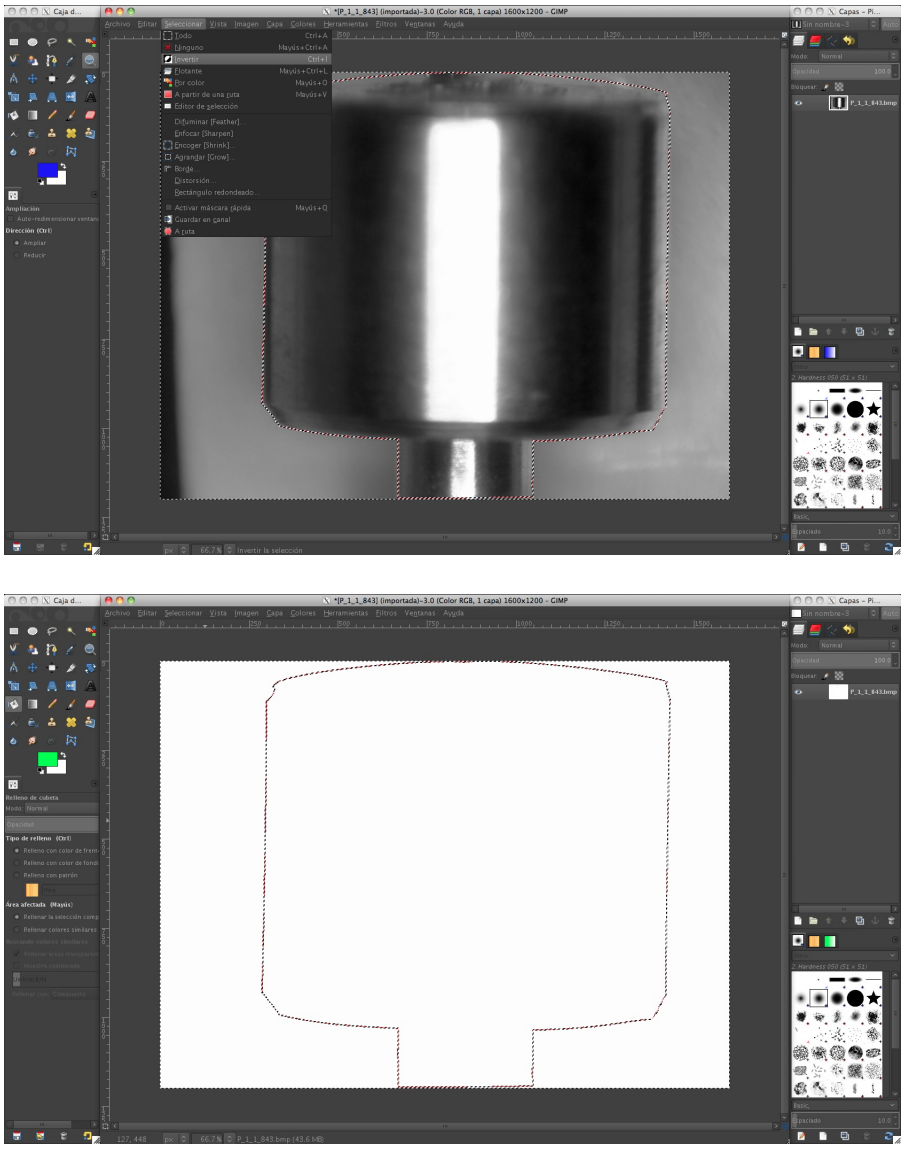


Figura 4.13: Invertir selección y eliminación del interior de la selección.

- Se rellena el objeto y el fondo de distintos colores (ver figura 4.14).

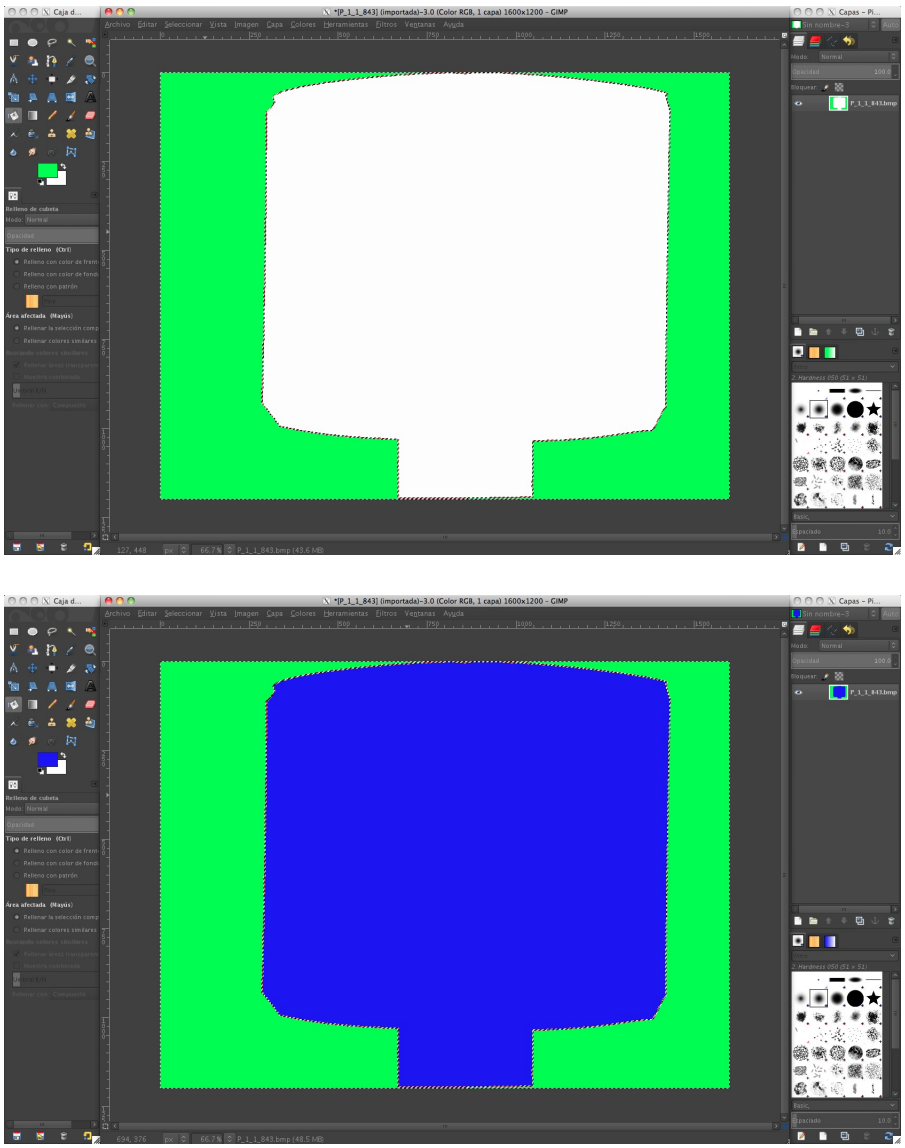


Figura 4.14: Rellenado del objeto y del fondo.

- Se quita la selección con la opción "Ninguno" del menú "Seleccionar" (Seleccionar->Ninguno, ver figura 4.15).

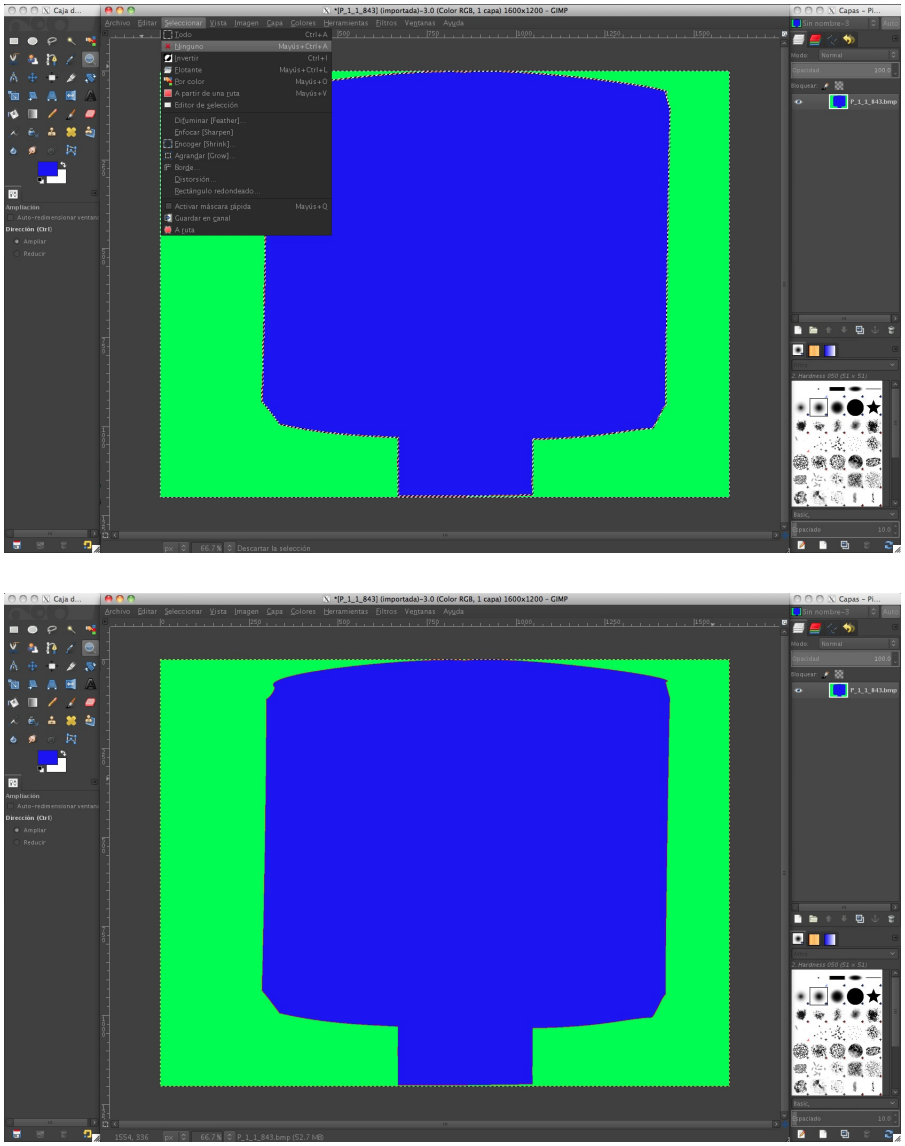


Figura 4.15: Quitar la selección.

- Finalmente se guarda la imagen con la opción "Exportar" del menú "Archivo" (Archivo->Exportar, ver figura 4.16)

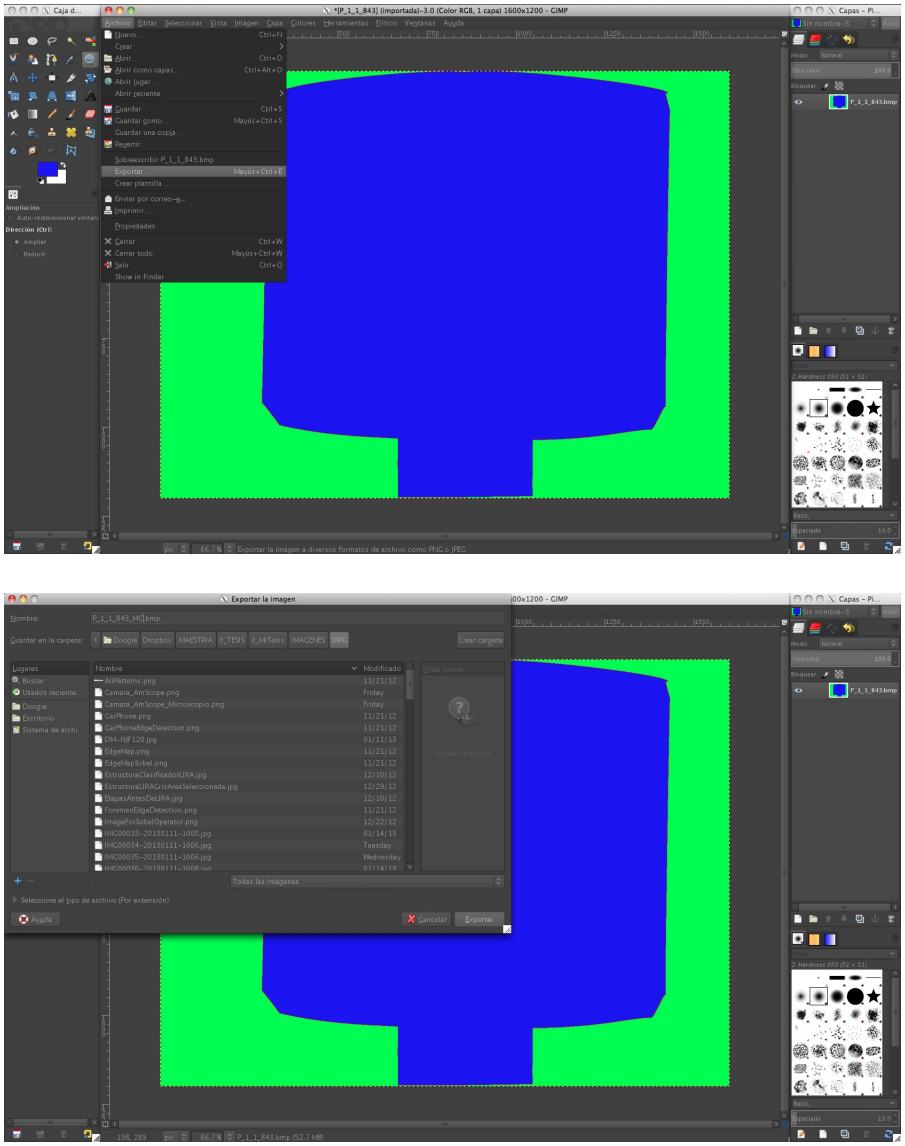


Figura 4.16: Guardado de la imagen.

El programa GIMP tiene su propio formato de manejo de imágenes, pero mantiene compatibilidad con otros y es por eso que no se guarda directamen-

te, si no que se exporta al formato BMP.

En la figura 4.17 se muestran imágenes guardadas como respaldo de la conversión de las imágenes a escala de grises y marcado del borde.

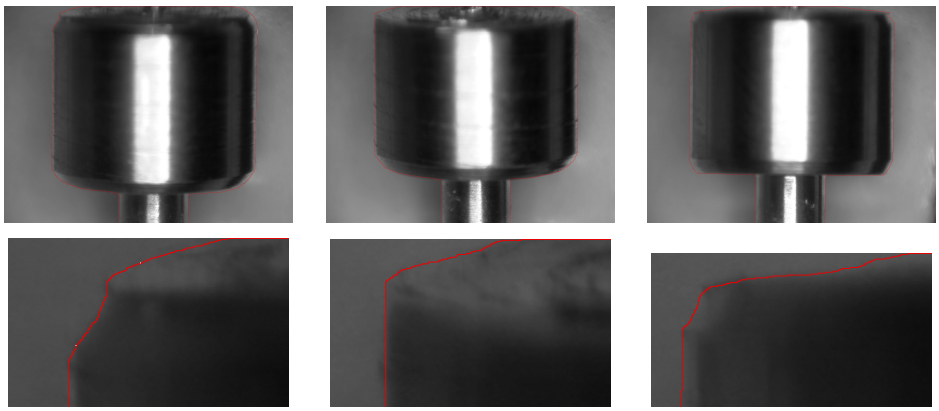


Figura 4.17: Imágenes en escala de grises con el borde marcado en rojo.

Las imágenes segmentadas resultantes se ilustran en la figura 4.18. Cada segmento representa una clase, los colores que se tomaron fueron el verde para el fondo, rojo para el borde y azul para el objeto.

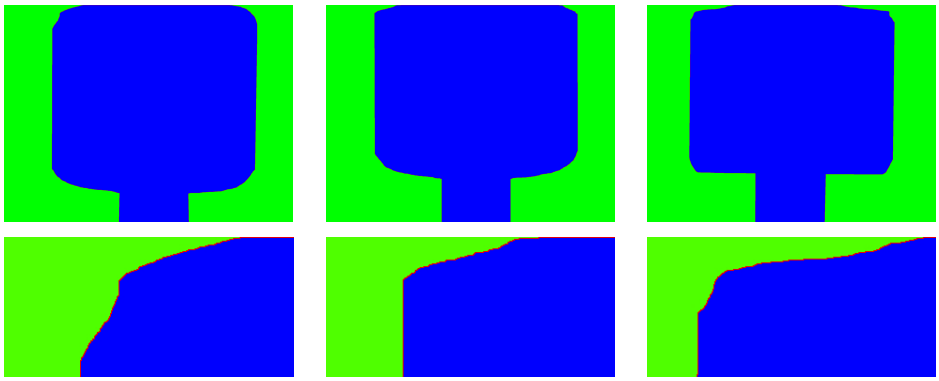


Figura 4.18: Imágenes segmentadas resultantes.

4.3.2. Localización de pixeles

En seguida de segmentar las imágenes, se obtuvieron las coordenadas de los pixeles que fungieron como muestras para el entrenamiento y prueba del clasificador neuronal.

La localización de cada uno de los pixeles de cada imagen se realizó con un programa creado que identifica, conforme al color, el pixel de la imagen segmentada (ver figura 4.19).

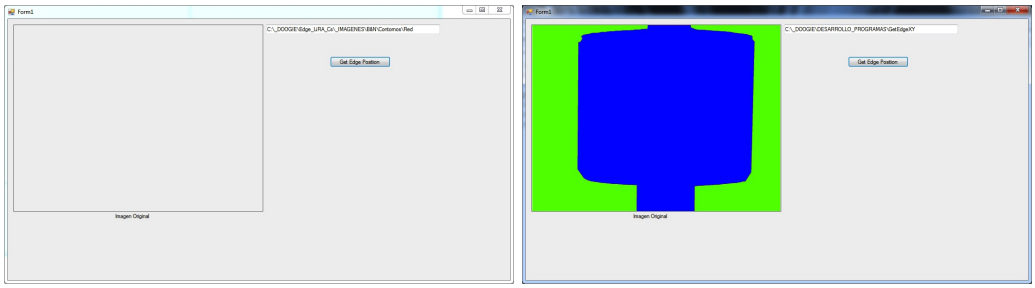


Figura 4.19: Rellenado del objeto y del fondo.

El algoritmo del programa para la obtención de las coordenadas de los pixeles se basa en el diagrama de flujo de las figuras 4.20 y 4.21 (el código fuente del procedimiento se encuentra en el apéndice A.1).

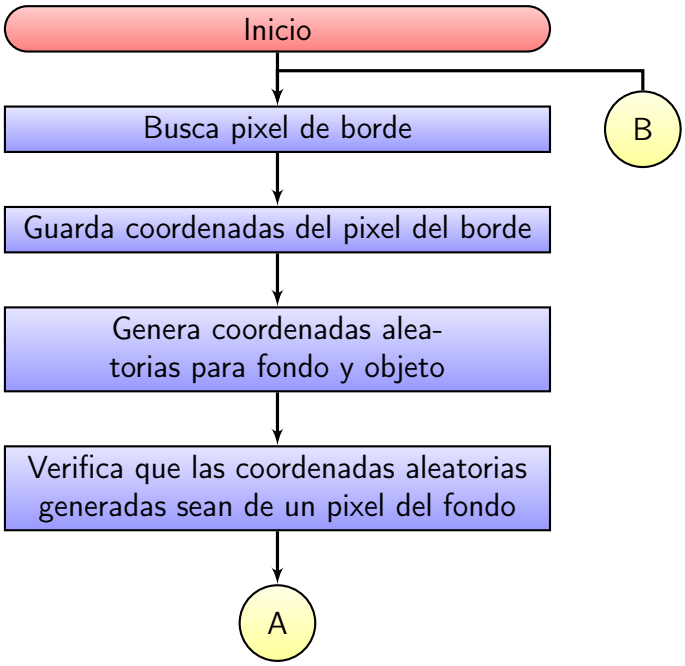


Figura 4.20: Procedimiento de obtención de coordenadas de los pixeles

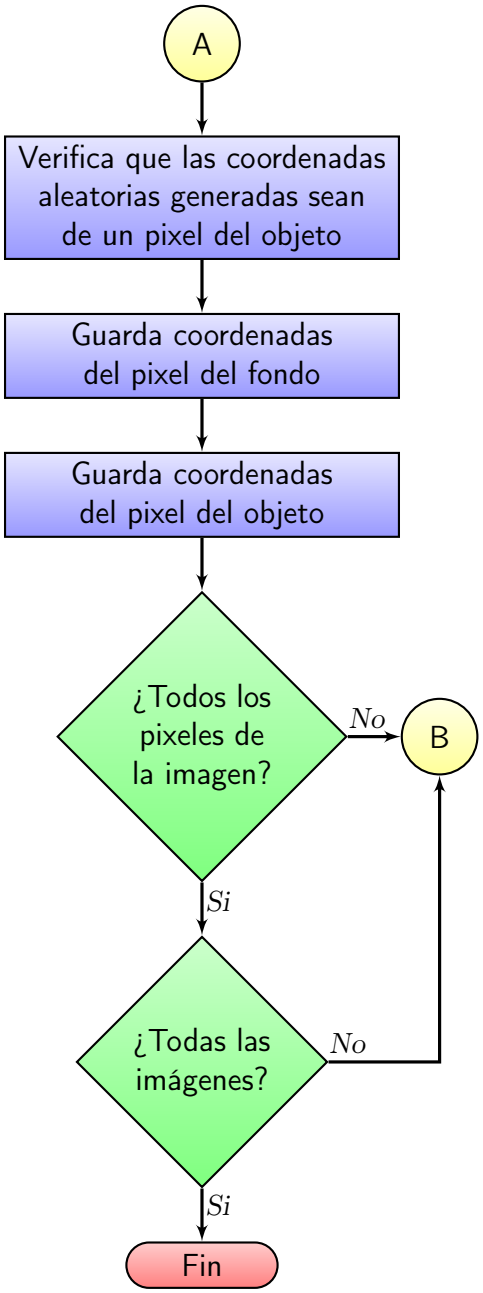


Figura 4.21: Continuación del procedimiento de obtención de coordenadas de los pixeles

El programa genera un archivo por cada clase y de cada imagen con las coordenadas de los pixeles, además de un archivo general con la cantidad de pixeles del borde de cada imagen.

El manejo del programa es sencillo, solo se le indica la ruta de la carpeta en donde se encuentran las imágenes segmentadas y se oprime el botón "Get Edge Position", con el cual automáticamente crea los archivos con los datos (coordenadas y cantidad de pixeles).

El resultado del número de pixeles se aprecia en la tabla 4.2, incluyendo el número total de pixeles.

Imagen	Número de pixeles del borde	Número total de pixeles
1	3177	9531
2	3026	9078
3	3015	9045
4	3021	9063
5	2937	8811
6	2999	8997
7	2990	8970
8	2976	8928
9	2986	8958
10	2949	8847
11	2997	8991
12	2953	8859
13	2910	8730
14	2953	8859
15	2970	8910

Tabla 4.2: Número de pixeles de las 15 imágenes

El número total de pixeles es la suma del número de pixeles de las 3 clases, esto significa que incluyen el número de pixeles del borde, del fondo y del objeto.

La cantidad de pixeles del fondo están en función del número de pixeles del borde; es decir que es la misma cifra y si cambia la cantidad de pixeles del borde, también lo hacen los del fondo. Igualmente la cantidad de pixeles del objeto.

Capítulo 5

Experimentos y resultados de la extracción de contornos

En definitiva, se probaron los tres métodos seleccionados (Sobel, Schwartz y el clasificador neuronal *LIRA*) para extraer los contornos y definir el borde de los micro pistones; y sus resultados se reflejaron en los experimentos hechos, gracias al desarrollo de varios programas. El siguiente capítulo se dedica a presentar los resultados de esos experimentos.

Los programas se crearon en el entorno de desarrollo integrado (IDE[1]) *Visual Studio 2010 Professional®*. El lenguaje de programación utilizado fue *Visual* `C#`, por su interfaz amigable, programación OOP[2] y creación de proyectos para ambientes visuales (como lo es el sistema operativo *Windows® XP*).

Estos programas fueron ejecutados en dos sistemas computacionales diferentes, las características de cada sistema se aprecian en la tabla 5.1.

Característica	Sistema 1	Sistema 2
Procesador	i7 @ 2.8GHz	i7 @ 2.66GHz
Memoria	4GB	8GB
Sistema operativo	Windows® 7 Professional @ 32 bits	Windows® 7 Professional @ 64 bits

Tabla 5.1: Sistemas computacionales.

[1]Siglas en Inglés de *Integrated Development Environment*
[2]Siglas en Inglés de *Object-Oriented Programming*

En primer lugar se implementó el operador Sobel en un programa que muestra y genera las imágenes con los contornos extraídos. Se describe el programa y el análisis de sus resultados.

En seguida se implementó el algoritmo de Schwartz en un programa independiente al del operador Sobel, en el que también muestra las imágenes con los contornos extraídos por este método, con su respectivo análisis de los resultados.

Y para finalizar el capítulo, se presenta el clasificador neuronal *LIRA*, su configuración, implementación, prueba y análisis, para el reconocimiento de los pixeles.

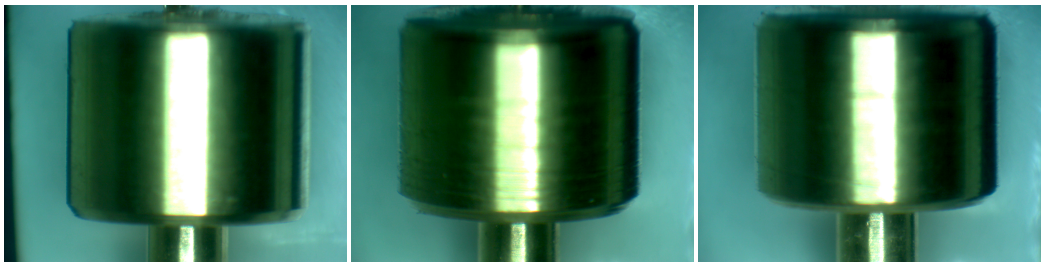
5.1. Operador Sobel

El operador Sobel es una herramienta convencional útil para extraer los contornos dentro de una imagen y fué probado con las imágenes de los micro pistones, con la finalidad de definir el borde del micro pistón y de esa forma poder dimensionarlo.

Inicialmente se utilizaron las tres diferentes imágenes de un mismo grupo de pistones, como lo indica la tabla 5.2.

Nombre de la Imagen	Grupo	Micro pistón	Medición	Figura
P_1_1_843	1	1	8.43	5.1a
P_1_2_843	1	2	8.43	5.1b
P_1_3_846	1	3	8.46	5.1c

Tabla 5.2: Relación de imágenes del pistón



(a) Micro pistón 1. (b) Micro pistón 2. (c) Micro pistón 3.

Figura 5.1: Imágenes de los micro pistones.

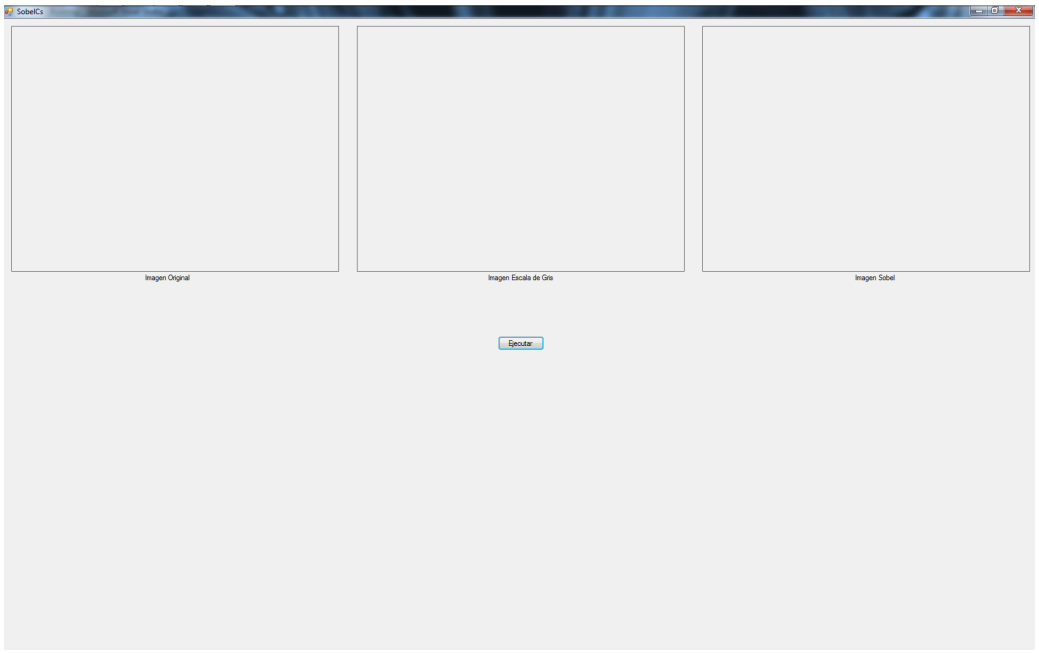
5.1.1. Descripción del programa de Sobel

El programa se diseñó de tal manera que fuera con un ambiente visual amigable, simple y automático, el código fuente del procedimiento se puede apreciar en el apartado del apéndice A.2.

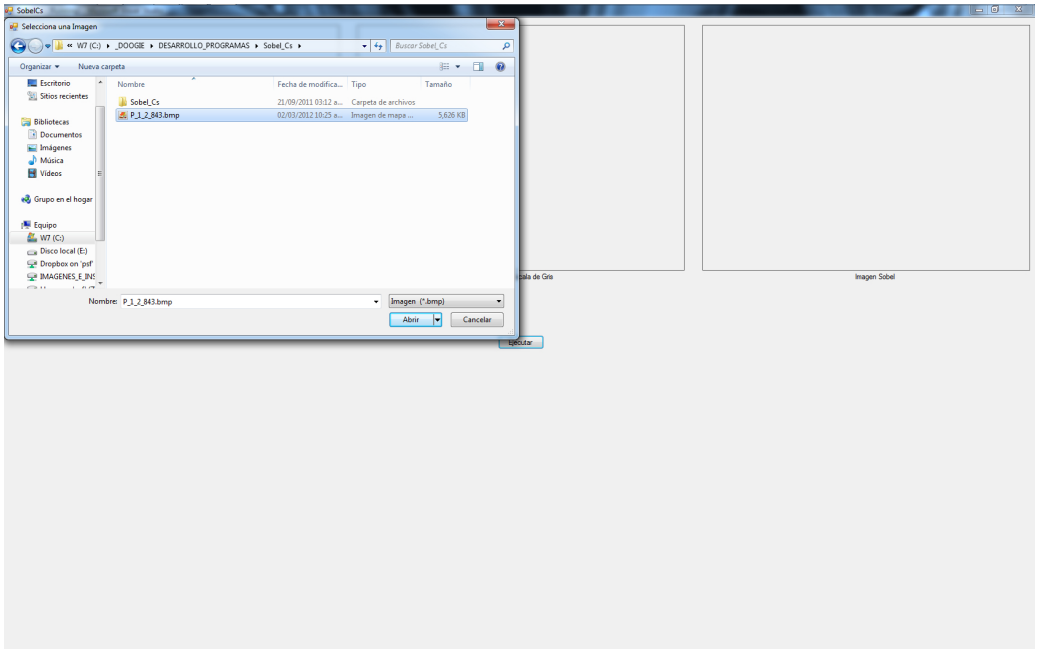
El diseño del programa consta de solo una ventana, la cual se compone de tres cuadros de figuras y un solo botón que ejecuta el operador Sobel.

Cada cuadro de figura muestra una imagen diferente (como se ilustra en la figura 5.2), de izquierda a derecha en el primer cuadro de figura se tiene la imagen original (figura 5.2a), en el segundo es la imagen en escala de gris (figura 5.2b) y en el tercero es la imagen resultante del operador Sobel (figura 5.2c).

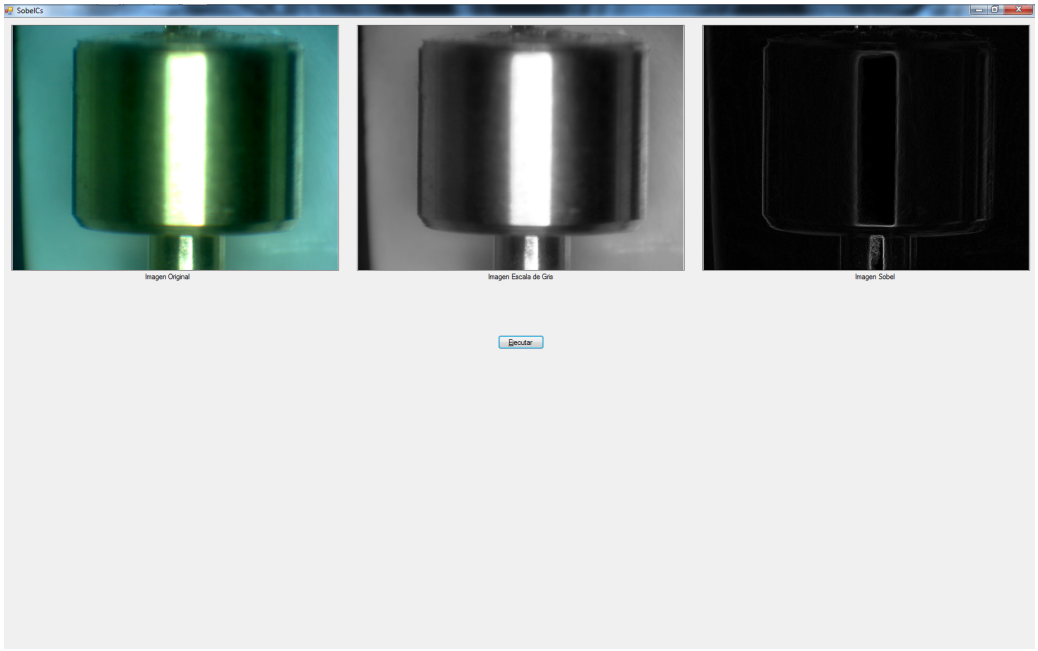
El botón abre un cuadro de dialogo, en donde se puede seleccionar la imagen a procesar. Después de ser seleccionada la imagen, automáticamente se ejecuta el operador Sobel, dando como resultado la imagen con los contornos extraídos.



(a) Al iniciar.



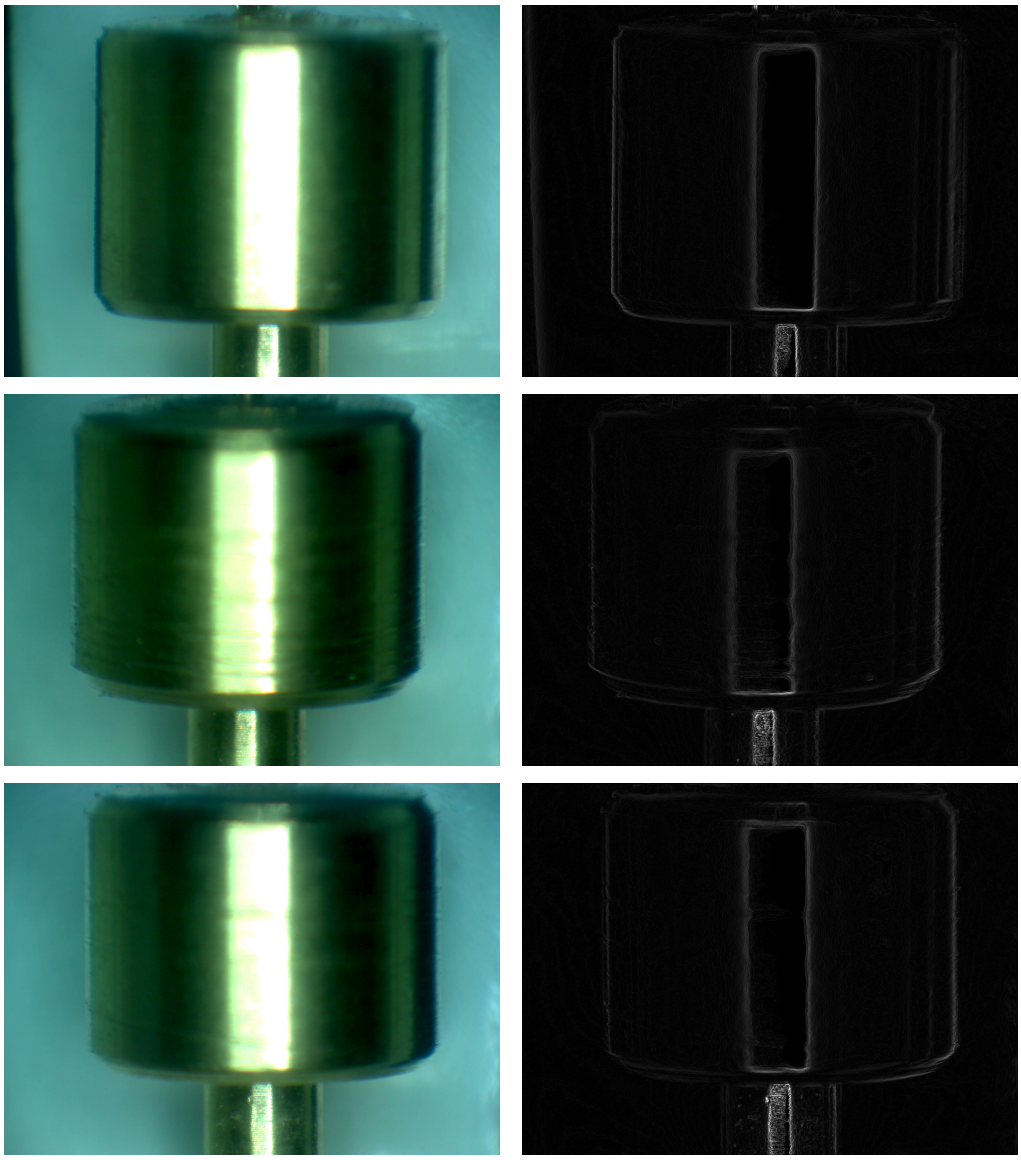
(b) Abriendo una imagen.



(c) Con una imagen procesada.

Figura 5.2: Programa Sobel.

La imagen creada por el programa es una imagen binaria y se guarda en un archivo con formato BMP; en la figura 5.3 se pueden apreciar las imágenes resultantes en la columna de la figura 5.3b, a partir de las imágenes originales de la columna de la figura 5.3a.

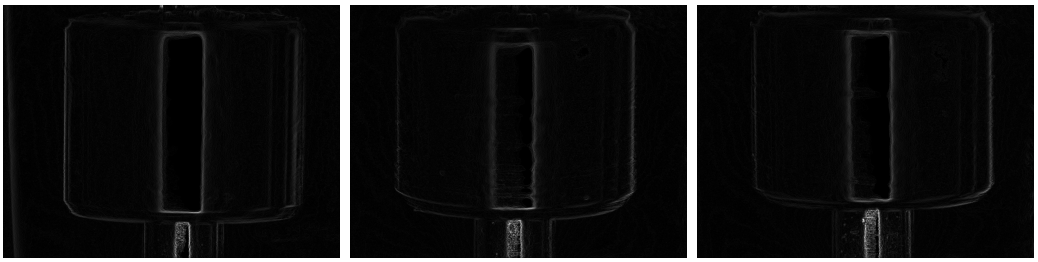


(a) Originales (b) Sobel resultantes

Figura 5.3: Imágenes de pistones.

5.1.2. Resultados

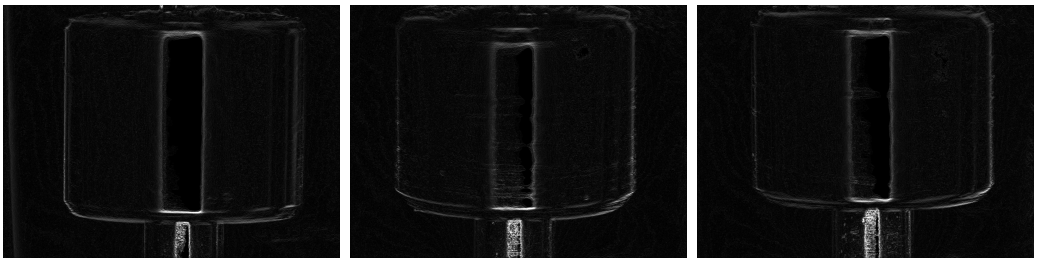
Se procesaron las 15 imágenes de los 15 micro pistones, de las cuales se generaron las imágenes resultantes. En la figura 5.4 se ilustran, como ejemplo, las imágenes resultantes de uno de los grupos de los micro pistones.



(a) Micro pistón 1. (b) Micro pistón 2. (c) Micro pistón 3.

Figura 5.4: Resultados de Sobel con 2 máscaras del grupo 1 de micro pistones.

Los experimentos anteriormente presentados se realizaron con las máscaras del operador Sobel verticales y horizontales . En primer lugar se procesó la imagen con la máscara de la ecuación 2.20, en seguida se aplicó de nuevo a la imagen original la máscara de la ecuación 2.19, para por último generar la imagen resultante final con la operación "OR" con las imágenes resultantes de aplicar las ecuaciones 2.20 y 2.19. Además, con el mismo procedimiento, se realizó el experimento con las 4 máscaras (horizontal, vertical y las dos diagonales; ecuaciones 2.20, 2.19, 2.21 y 2.22). Las imágenes de uno de los grupos se muestran en la figura 5.5.



(a) Micro pistón 1. (b) Micro pistón 2. (c) Micro pistón 3.

Figura 5.5: Resultados de Sobel con 4 máscaras del grupo 1 de micro pistones.

5.1.3. Análisis

La gran ventaja que tiene el operador Sobel es por su fácil entendimiento e implementación, lo que lo hace que sea un método ampliamente usado en diferentes aplicaciones, pero su principal desventaja es que no es posible configurarlo, ya que las máscaras son simétricas; y si se modifican de otra forma, se denomina de diferente manera al operador.

Se apreció que aunque se aplique el operador a cada grupo específico de los micro pistones con características similares en 3 imágenes, la definición de los contornos depende de la iluminación con la que se adquirió cada imagen, ya que la intensidad de cada pixel está en función de la iluminación. Por lo que el borde del objeto dentro de la imagen no se encuentra bien definido.

Para obtener con alta precisión la medición de un objeto por medio de imágenes, es necesario encontrar un borde bien definido. La definición del borde se reduce a medida que se extraen contornos innecesarios. En las imágenes creadas por el operador Sobel (tanto con 2 máscaras como con 4 máscaras) contienen contornos que dificultan la definición del borde.

Otra de sus desventajas se encuentra en la frontera, ya que no es posible calcular la existencia del borde en los pixeles que se encuentran en la primera fila y columna, así como en la última fila y columna de la imagen; esto es ocasionado a que el tamaño de la máscara es de una ventana de 3×3 y el gradiente calculado es del pixel central de la ventana.

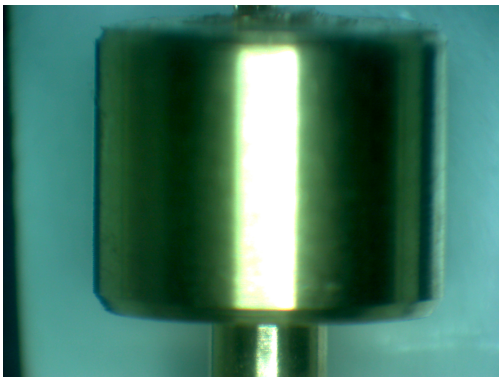
Es importante señalar que el operador Sobel fue también implementado en MATLAB®, sin tener variación alguna en los resultados, tanto en la extracción de los contornos, como de las imágenes binarias generadas.

5.2. Algoritmo de Schwartz

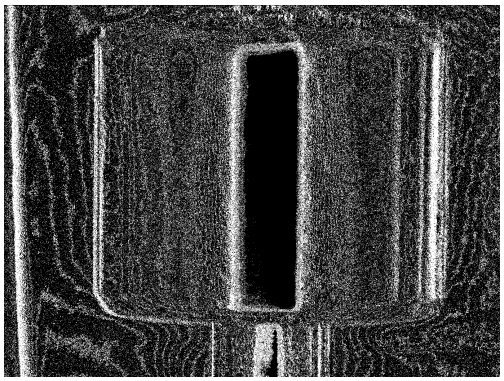
El segundo método convencional probado fue el algoritmo de Schwartz, utilizando inicialmente las mismas imágenes de prueba de los micro pistones que se emplearon con el operador Sobel (ver tabla 5.2). Se eligió el mismo grupo de imágenes para tener un punto de comparación entre los dos métodos. Con este método se intenta definir (de igual manera que con el operador Sobel) el borde del micro pistón.

El algoritmo de Schwartz contiene un parámetro configurable, con el que se define la sensibilidad de detección de los contornos en la imagen. El parámetro se denomina como *CE*, refiriéndose como la constante experimental.

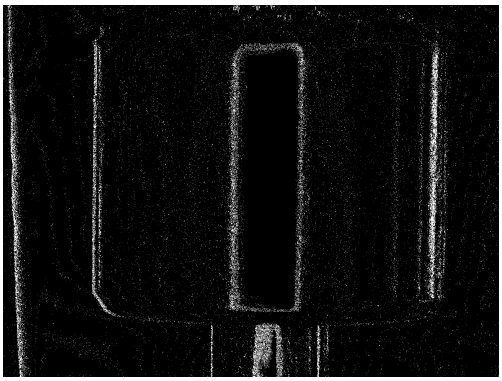
Para los experimentos se determinaron 3 constantes experimentales, 1, 3 y 5; con las cuales se tiene una mejor observación de las diferencias al generar las imágenes binarias resultantes, como se ilustra en las imágenes de ejemplo de la figura 5.6. De éstas 3 *CE* se analiza si es posible seleccionar la que obtenga mejores resultados, para el procesamiento de las imágenes de los micro pistones.



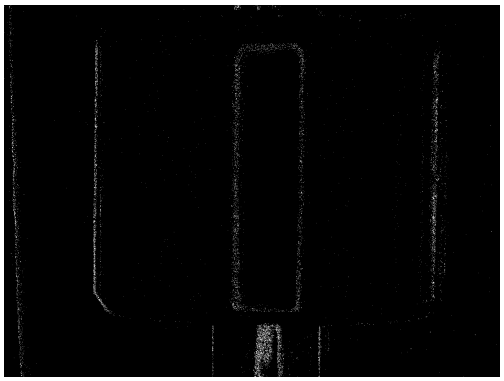
(a) Imagen original.



(b) Constante CE de 1.



(c) Constante CE de 3.



(d) Constante CE de 5.

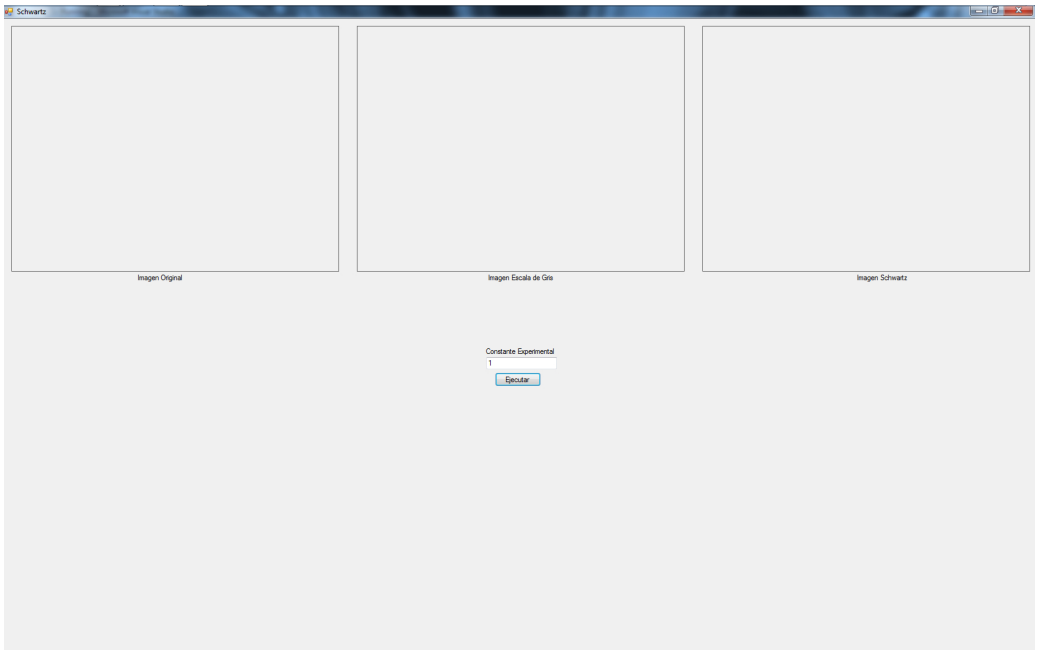
Figura 5.6: Imágenes resultantes de la imagen del pistón 1 del grupo 1.

5.2.1. Descripción del programa de Schwartz

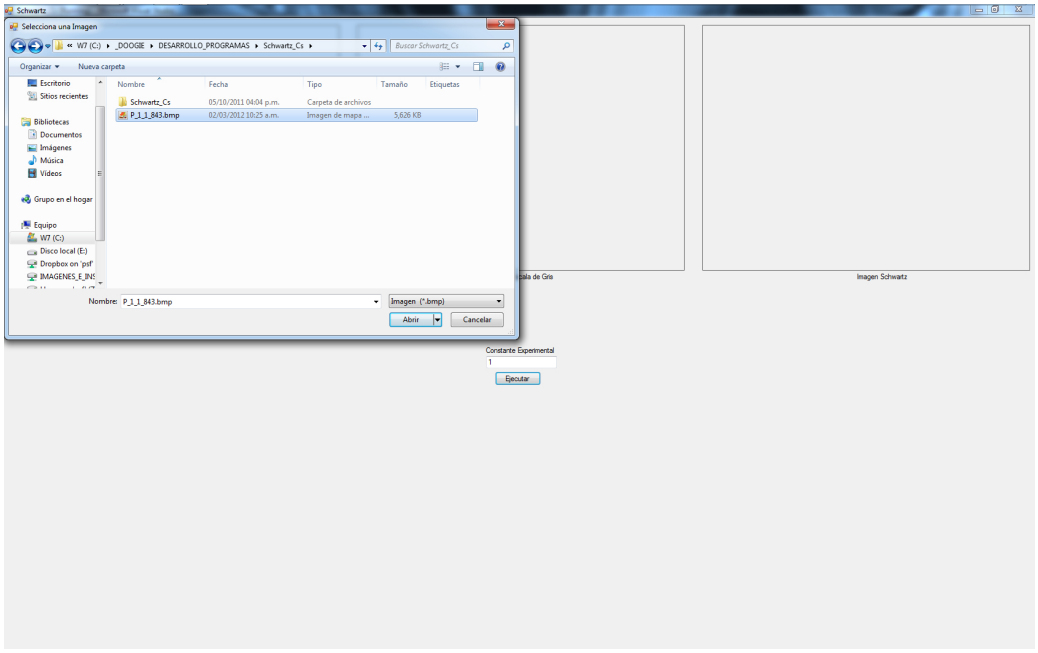
Al igual que el programa del operador Sobel (ver sección 5.1), se diseñó un programa para el algoritmo de Schwartz con características similares, solo que se agregó un campo mas en la ventana principal, en donde se ingresa el parámetro de la constante experimental.

El programa desarrollado se muestra en la figura 5.7, con los 3 cuadros de figura, el botón que ejecuta el algoritmo de Schwarzt y el cuadro de texto, para ingresar el valor de la constante experimental.

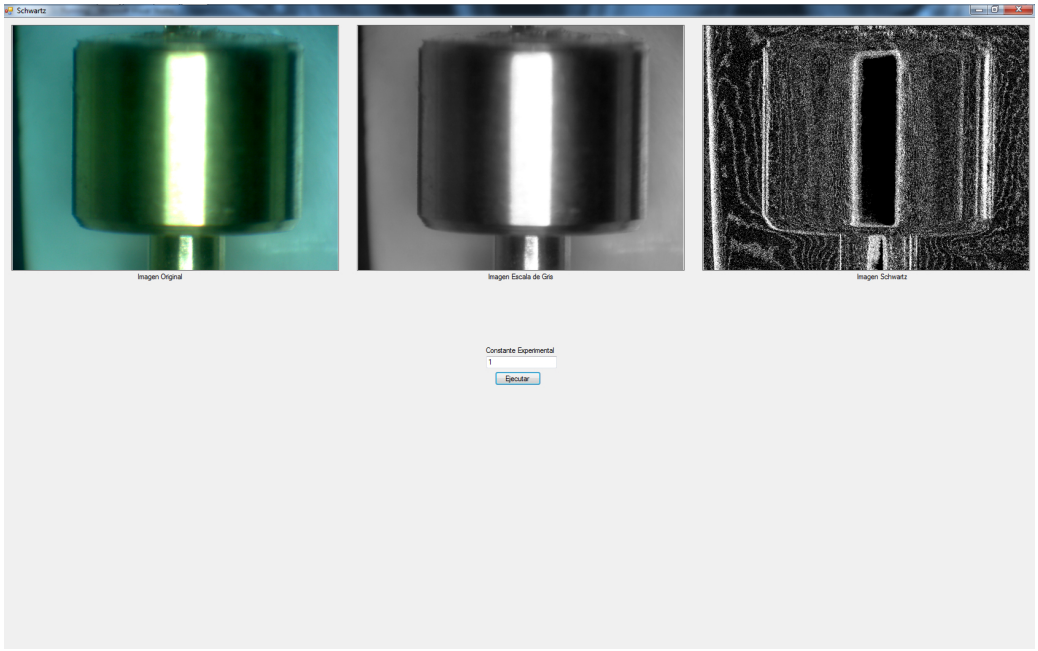
El funcionamiento del programa de Schwartz también es muy similar al del programa del operador Sobel. La gran diferencia es que primeramente se establece el valor de la constante experimental en el cuadro de texto, después se oprime el botón "Ejecutar", el cual abre un cuadro de dialogo y se selecciona la imagen a procesar. En seguida se ejecuta automáticamente el algoritmo de Schwartz, con el que se genera una imagen binaria con los contornos extraídos y la guarda en formato BMP.



(a) Al iniciar.



(b) Abriendo una imagen.

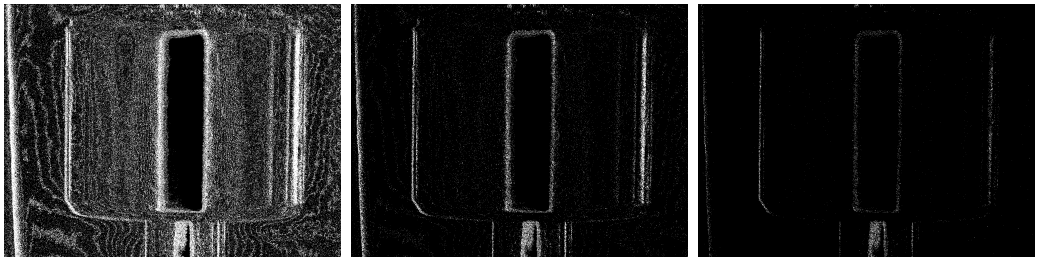


(c) Con una imagen procesada.

Figura 5.7: Programa Schwartz.

5.2.2. Resultados

Se realizaron pruebas con toda la base de imágenes (las 15 imágenes de los micro pistones), con las 3 constantes experimentales previamente determinadas. Con cada experimento se generó una imagen resultante, un ejemplo de esas imágenes resultantes se muestran en la figura 5.8, en donde se presentan 3 imágenes del mismo micro pistón con las variantes de la constante experimental.



(a) Valor de CE=1. (b) Valor de CE=3. (c) Valor de CE=5.

Figura 5.8: Micro pistón 1 del grupo 1.

5.2.3. Análisis

El algoritmo de Schwartz presenta mejores ventajas para adquirir los contornos de un objeto que si se utiliza un operador típico, como lo es el Sobel. Una ventaja con respecto al operador Sobel es que no tiene problemas en la frontera de la imagen, debido a que no necesita una máscara por separado como filtro para realizar la convolución en pequeñas ventanas de 3×3.

Este algoritmo es configurable, ya que se puede ajustar la sensibilidad a través de la constante experimental y eliminar contornos no deseados; pero por desgracia la extracción de algunos contornos es muy difuso, por lo que con algunas *CE* extrae contornos innecesarios y con otras *CE* omite contornos. La *CE* que dio mejores resultados fue la $CE = 5$, pero no le fue posible reducir los contornos internos y los externos se empiezan a difuminar.

Los recursos computacionales son menores a los del operador Sobel, por la reducción de operaciones que realiza al hacer el cálculo de los contornos del objeto en la imagen.

Así como el operador Sobel fue implementado en dos entornos de desarrollo, igualmente se hizo el mismo procedimiento para el algoritmo de Schwartz, observando que tampoco existió alguna variación en los resultados.

5.3. Clasificador *LIRA*

El tercer y último método probado para extraer los contornos fue con la red neuronal *LIRA*. Se desarrolló un programa con el que crea los subconjuntos de imágenes de entrenamiento y prueba de la base de imágenes, la estructura, el entrenamiento, la prueba de la red neuronal, así como el reconocimiento de los pixeles en una imagen; generando así una imagen con los pixeles reconocidos, la cual es empleada en el programa de medición (ver capítulo 6).

Las pruebas se hicieron con la base de imágenes divida en 2 subconjuntos: el subconjunto de imágenes de entrenamiento y el subconjunto de imágenes de prueba. La división se realizó con la opción 1 del algoritmo de "Conjunto de imágenes para entrenamiento y prueba" (ver sección 3.4.1), la cual separa las primeras imágenes para el conjunto de entrenamiento y el resto de la base de imágenes para probar el sistema.

Para realizar los experimentos con el clasificador neuronal, éste cuenta con varios parámetros configurables. Estos parámetros definen las opciones del clasificador. Los parámetros que se utilizaron para el clasificador se muestran en la tabla 5.3.

Parámetro	experimento 1	experimento 2	experimento 3	experimento 4
Clases	3	3	3	3
Neuronas A	1024	1024	1024	1024
Neuronas ON	2	2	2	2
Neuronas OFF	3	3	3	3
Tamaño ventana $(W \times H)$	151×151	101×101	151×151	151×151
Tamaño ventana $(w \times h)$	51×51	21×21	51×51	51×51
TDS	0.15	0.15	0.15	0.15
Ciclos de entrenamiento	10	10	40	100

Tabla 5.3: Parámetros de la red neuronal.

Además de los parámetros del clasificador, se puede configurar el funcionamiento del programa por los valores de las propiedades de la tabla 5.4.

Propiedades	Valor
Número total de imágenes	15
Número de imágenes de entrenamiento	7
Imágenes de entrenamiento	Primeras
Ecualizador de imagen	0.65 (65 %)

Tabla 5.4: Propiedades del programa.

5.3.1. Descripción del programa de LIRA

El programa desarrollado, de ahora en adelante llamado solamente "programa *LIRA*", se dedica a construir y entrenar el clasificador, además de generar la imagen con los pixeles reconocidos.

La técnica con la que se desarrolló el "programa *LIRA*" fue con programación orientada a objetos, con la finalidad de crear una *class*[3] con la que se pudiera configurar la estructura del clasificador y que la *class* pudiera servir en futuras investigaciones.

El "programa *LIRA*" se configuró para 3 clases, pixeles del borde, pixeles del fondo y pixeles del objeto. Éste se conforma de varias secciones: "Configuración", "Resultados", "Proceso" y recuadros de figuras multi-propósito (ver figura 5.9).

En la sección de la configuración se encuentran los parámetros de las imágenes, en donde se define el número de imágenes en la base de imágenes, el número de imágenes que se usan para el entrenamiento del clasificador y el ecualizador de la imagen (con el cual ajusta el brillo de la imagen). Igualmente en esta sección se incluyen las rutas donde se encuentran los archivos de datos y la base de imágenes. También contiene los parámetros de la "RED NEURONAL", como lo son: el número de clases, número de neuronas en la capa asociativa "A", número de neuronas ON, número de neuronas OFF, ancho de la ventana de análisis W, largo de la ventana de análisis H, ancho de la ventana w, largo de la ventana h. Se agregó una parte mas a esta sección para futuras investigaciones y desarrollo llamada "Codificación", en donde se pueden agregar las distorsiones de las imágenes. Por último se tiene la

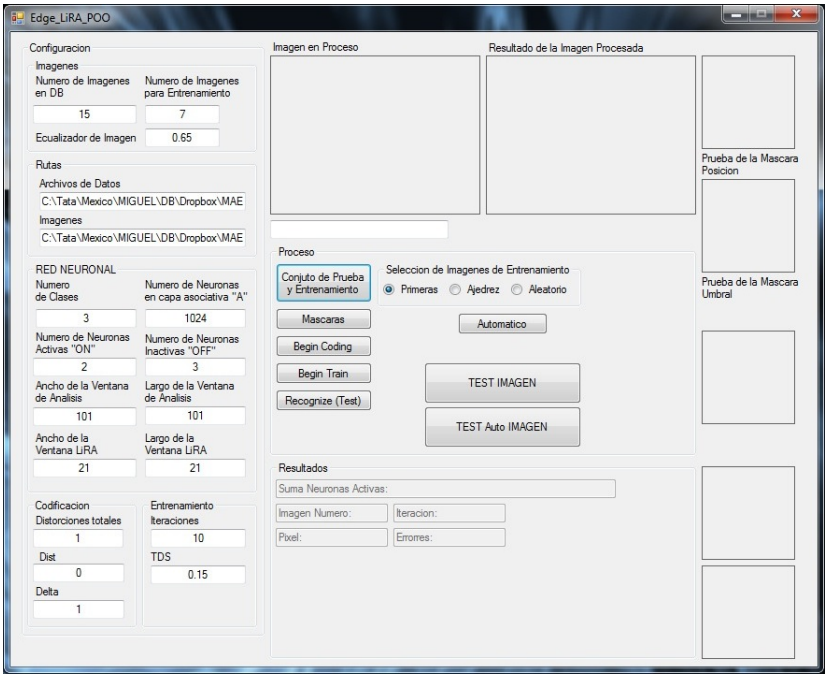
[3]Clase de OOP

subsección de "Entrenamiento", en donde están el número de iteraciones y el factor de decremento (TDS).

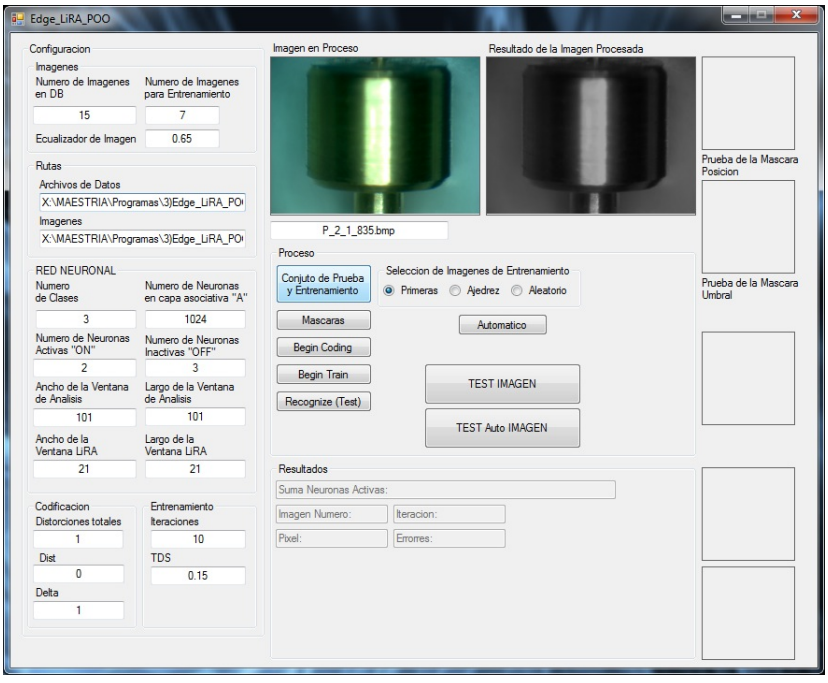
La sección de "Proceso" incluye los botones en donde se ejecutan los algoritmos de la red neuronal de la sección 3.4, empezando por el "Conjunto de Prueba y Entrenamiento", "Mascaras", "Begin Coding", "Begin Train", "Recognize" y un "Automatico" que ejecuta todos los procedimientos anteriores. Dentro de esta sección se encuentra una subsección de "Selección de imágenes", con la que determina la forma de separar la base de imágenes en subconjuntos. Finalmente se tienen 2 botones (llamados "TEST IMAGEN" y "TEST Auto IMAGEN") con los que hacen el reconocimiento de los pixeles de una imagen seleccionada, o de las imágenes de una carpeta y genera la imagen con los pixeles clasificados.

Debajo de la sección de "Proceso" se localiza la sección de "Resultados", en donde se puede visualizar la suma de las neuronas activas, la imagen bajo tratamiento, el número de pixel siendo procesado, la iteración del entrenamiento y el número de errores.

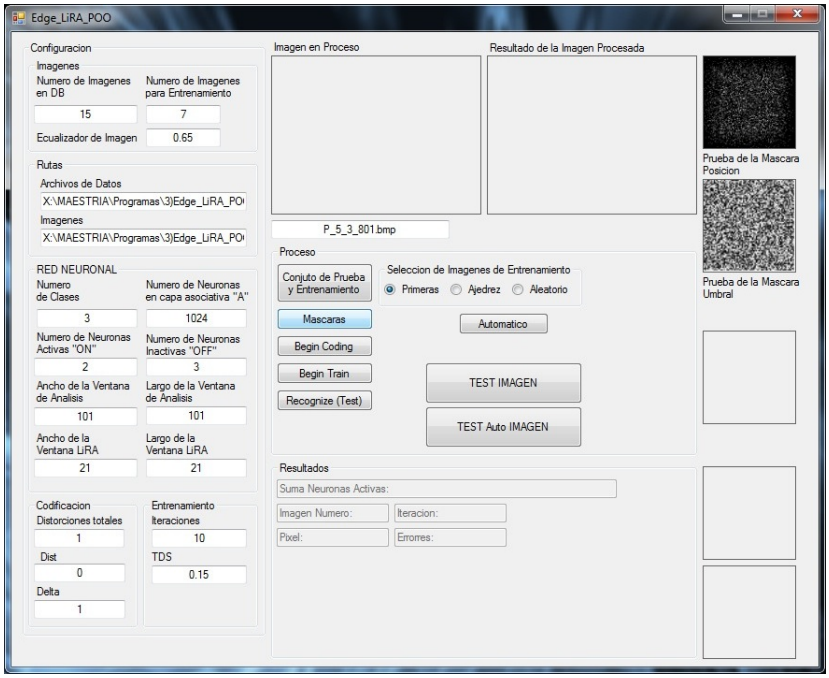
Existen diferentes cuadros de figuras en toda la ventana para distintos propósitos, desde mostrar las imágenes que están siendo procesadas, hasta las pruebas de la creación de las máscaras y pruebas de la codificación.



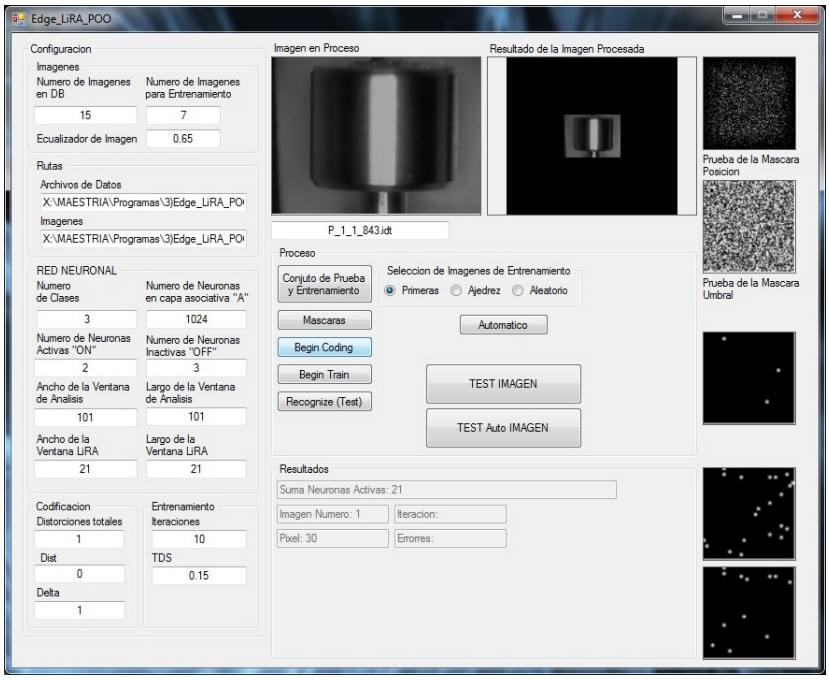
(a) Al iniciar.



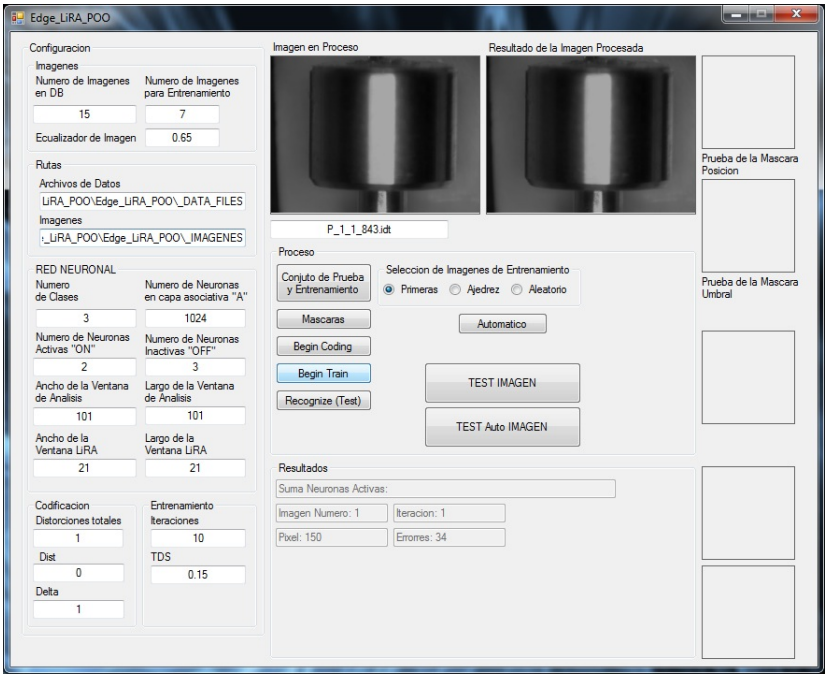
(b) Creando los 2 conjuntos.



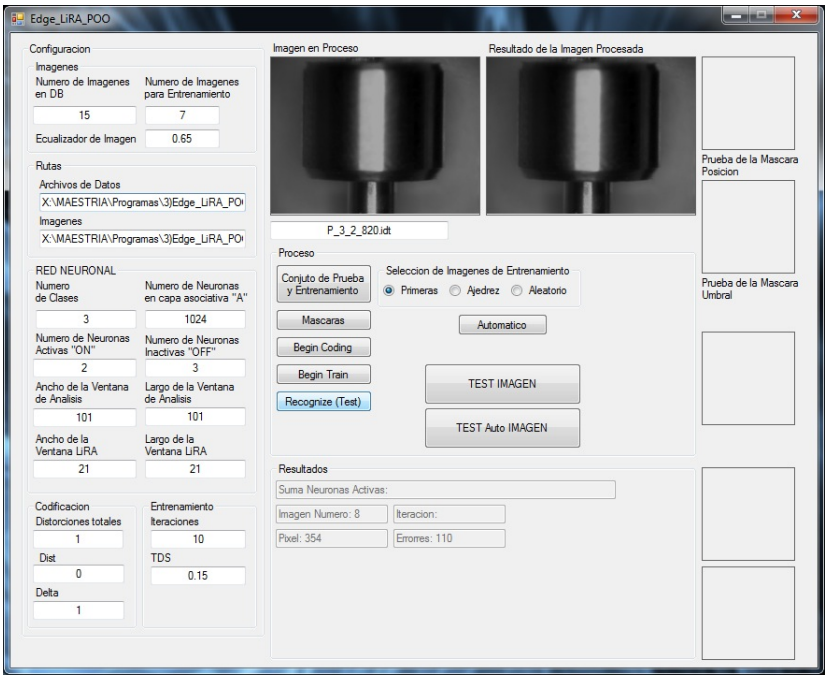
(c) Creando la estructura de *LIRA*.



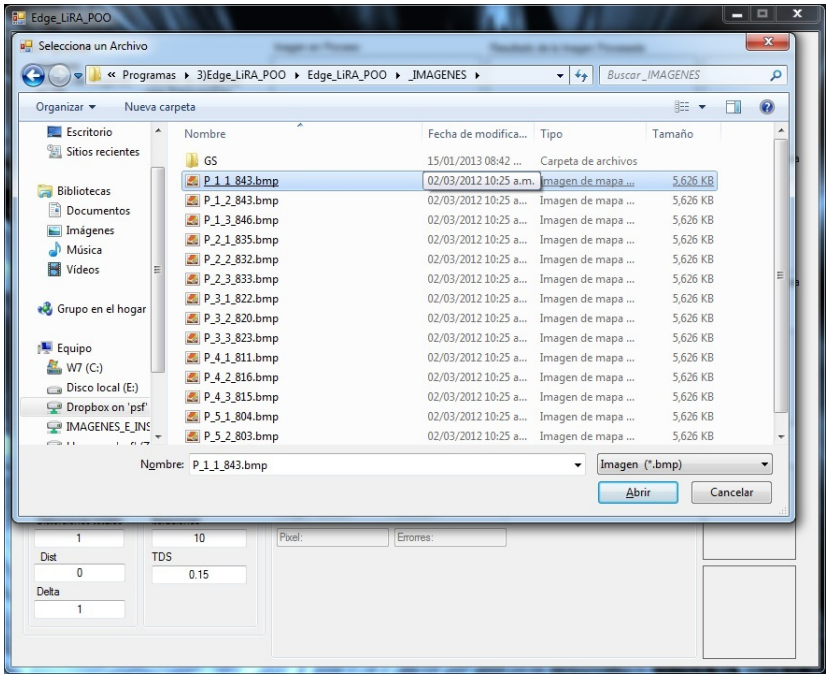
(d) Codificando las imágenes.



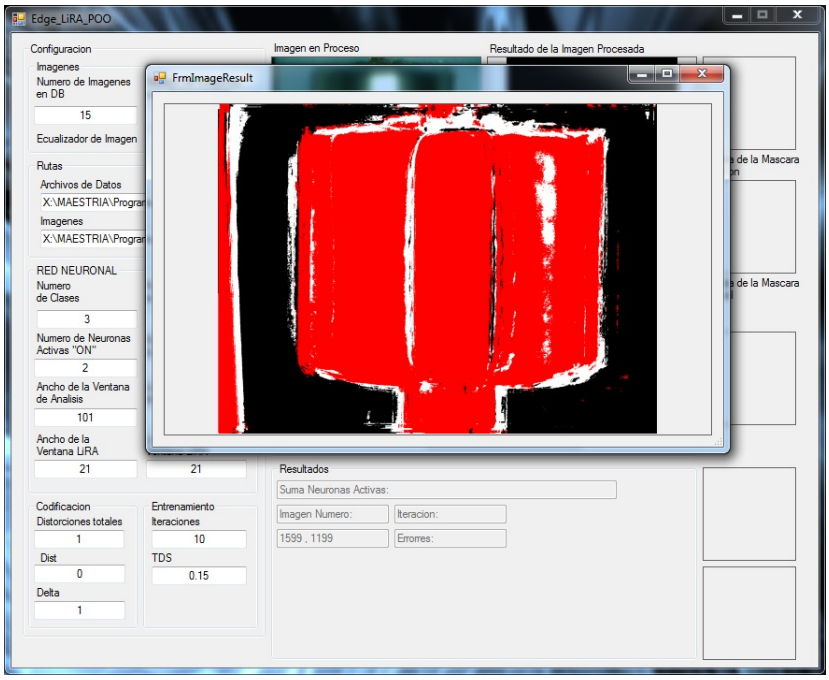
(e) Entrenamiento de *LIRA*.



(f) Prueba de *LIRA*.



(g) Abriendo una imagen a reconocer.



(h) Imagen reconocida.

Figura 5.9: Programa *LIRA*.

5.3.2. Resultados

El "programa *LIRA*" entrega dos tipos de resultados, el primer tipo son datos que nos muestra la efectividad del clasificador, como el número de pixeles reconocidos y la tasa de error del reconocimiento (tanto de las iteraciones, como de la prueba de reconocimiento del conjunto de prueba del sistema); el segundo tipo es gráfico el cual genera imágenes con los pixeles reconocidos, siendo estos identificados en colores (rojo para el objeto, negro para el fondo y blanco para el borde).

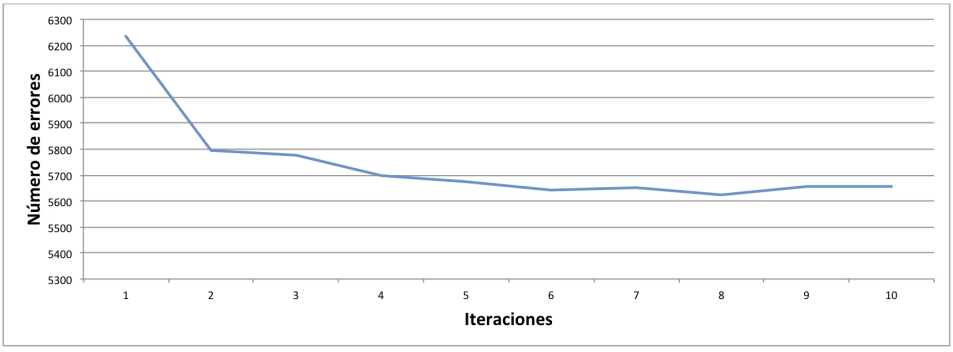
Los datos que se obtuvieron en el proceso de prueba del clasificador *LIRA* ("Recognize") se pueden apreciar en la tabla 5.5.

Experimento	Pixeles	Número de errores	Porcentaje reconocido
E1	101439	30826	69.61 %
E2	101439	24681	75.66 %
E3	101439	27816	72.57 %

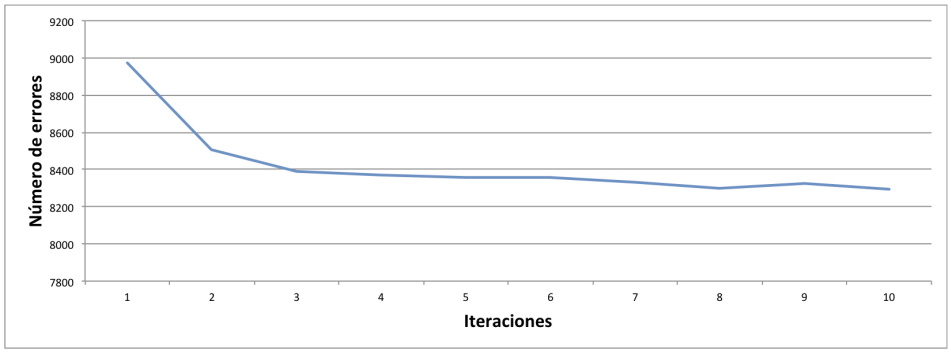
Tabla 5.5: Resultados de reconocimiento.

Cabe mencionar que el experimento 4 es el mismo que el experimento 3, pero se incrementó el número de iteraciones en el entrenamiento.

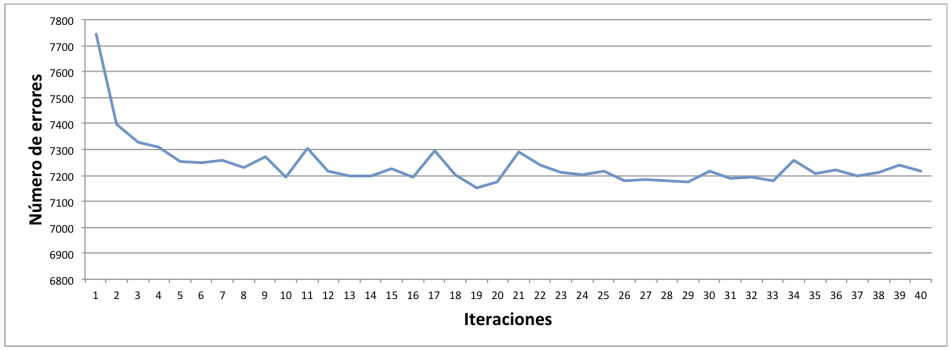
En el proceso de entrenamiento se generaron gráficas con los datos que obtuvimos de cada iteración, las cuales se muestran en la figura 5.10.



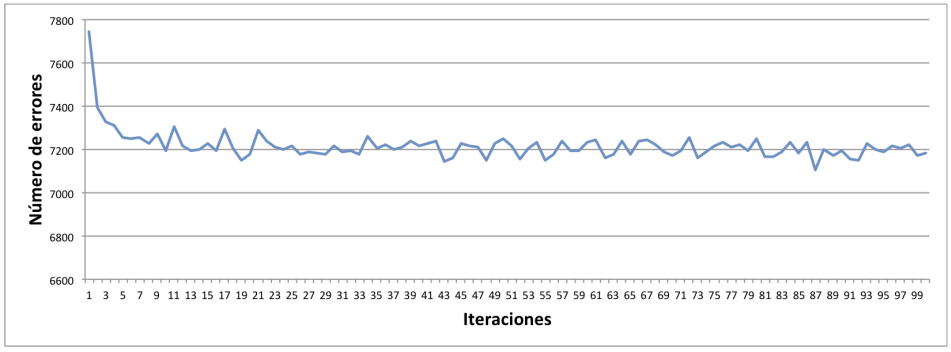
(a) Experimento 1.



(b) Experimento 2.



(c) Experimento 3.



(d) Experimento 4.

Figura 5.10: Errores de entrenamiento (Iteraciones vs. Errores).

A continuación se muestran algunos de los resultados gráficos en las imágenes generadas en la figura 5.11.

(a) Micro pistón 1. (b) Micro pistón 2. (c) Micro pistón 3.

Figura 5.11: Micro pistones del grupo 1.

5.3.3. Análisis

Los datos obtenidos por los experimentos muestran que existe una convergencia (como se puede apreciar en la figura 5.10), por lo que el clasificador neuronal es capaz de "aprender".

Debido al carácter aleatorio del clasificador neuronal, se realizaron varios experimentos con parámetros similares, para construir diferentes estructuras. Es importante señalar que independientemente de la aleatoriedad del clasificador se observan en las gráficas la reducción de los errores.

Las imágenes generadas por la red neuronal muestran la clasificación de cada pixel, en las que se aprecia el mismo micro pistón en color rojo (identificado como el objeto en la imagen), el fondo en negro y el borde en blanco, por lo que es posible refinar la identificación de los pixeles manipulando la configuración de la red neuronal y de esa manera tener una mejor definición del borde.

Aunque sea muy difusa la frontera entre las tres clases, el clasificador neuronal identificó en todos los experimentos mas del 69 % de los pixeles, por lo que es posible incrementar este porcentaje de reconocimiento y tener una medición mas precisa.

Capítulo 6

Medición

Para finalizar este trabajo de tesis se presenta el último capítulo con la aplicación de la red neuronal artificial. Hasta este punto se tienen todos los elementos necesarios para dimensionar las micro piezas, desde las imágenes de entrada (base de imágenes), pasando por la construcción y entrenamiento del clasificador, hasta las imágenes generadas por el "programa *LIRA*" con los pixeles clasificados.

A continuación se muestra como se realiza la medición de los micro pistones, utilizando todos los componentes anteriormente descritos.

6.1. Programa de cálculo de dimensionamiento

Después de tener las imágenes resultantes con los pixeles clasificados del "programa *LIRA*", éstas se ingresan una por una a un programa desarrollado exclusivamente para el cálculo de la dimensión de las micro piezas, dando como resultado la medición del diámetro de los micro pistones

El programa desarrollado depende de la posición de las micro piezas, debido a que el cálculo lo realiza con un barrido de la imagen en forma vertical, contabilizando los pixeles de cada línea horizontal. Por el momento esta cualidad no es ningún inconveniente, ya que todos los micro pistones en la base de imágenes se encuentran en posición vertical, pero se ha tomado en cuenta para futuras investigaciones, las cuales se encuentran propuestas en el apartado de trabajo a futuro.

El conteo de los pixeles se realiza de dos formas, en los primeros experimentos se tomó en cuenta solo los pixeles identificados como objeto. Después

se experimentó contabilizando los pixeles identificados como objeto y borde.

6.1.1. Algoritmo del programa

El funcionamiento del programa se basa en hacer un conteo de los pixeles en cada línea horizontal y determinar la medida del pistón con un promedio de las líneas. El algoritmo se detalla en el diagrama de flujo de la figura 6.1.

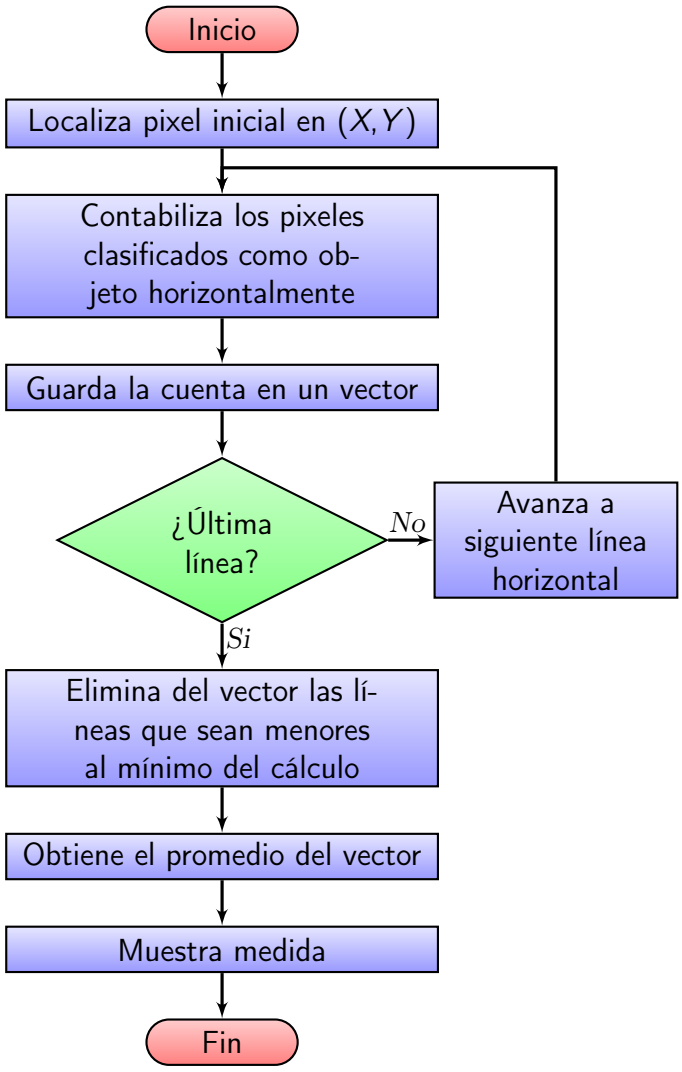


Figura 6.1: Procedimiento del programa de dimensionamiento

6.1.2. Descripción del programa

El "programa de dimensionamiento" (ver figura 6.2) es menos complejo que el "programa *LIRA*", ya que se destina a realizar la medición del objeto en la imagen creada por el "programa *LIRA*".

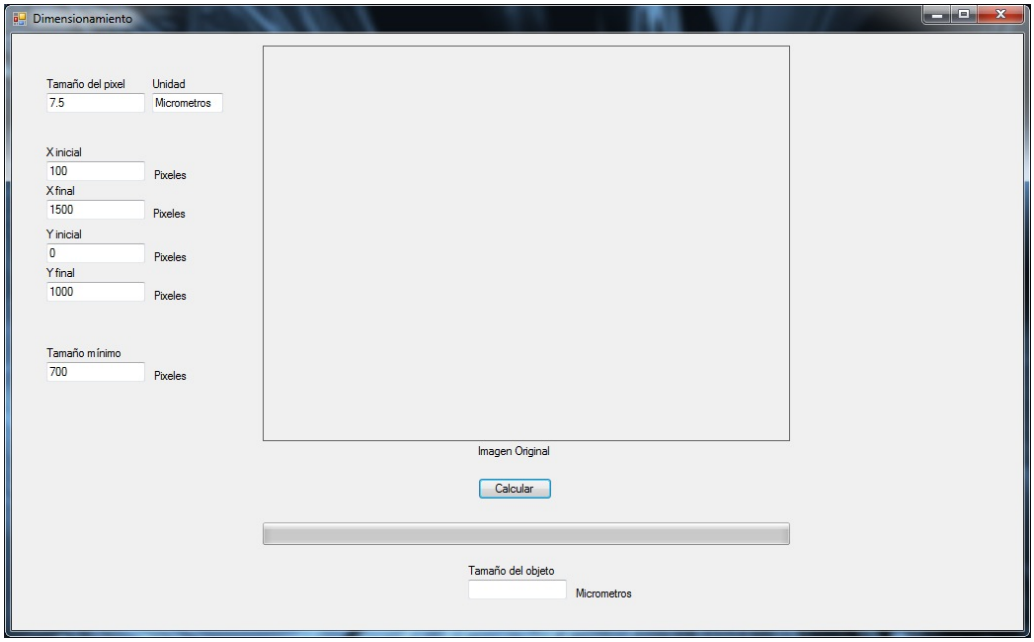
La imagen se generó en 3 colores, cada color indica como fue clasificado el pixel. Si el pixel es negro, significa que se clasificó como fondo; en cambio si el pixel es rojo, éste se ha clasificado como objeto; y para la clasificación del borde el pixel es blanco.

Los ajustes del "programa de dimensionamiento" se diseñaron de tal forma que se minimizara el tiempo de cómputo del cálculo, limitando el área donde se mide el objeto y se precisará una dimensión directa de un solo pixel, asimismo cuenta con un botón con el que se procesa la imagen, con el cual abre el cuadro de diálogo de "abrir archivo" para seleccionar la imagen.

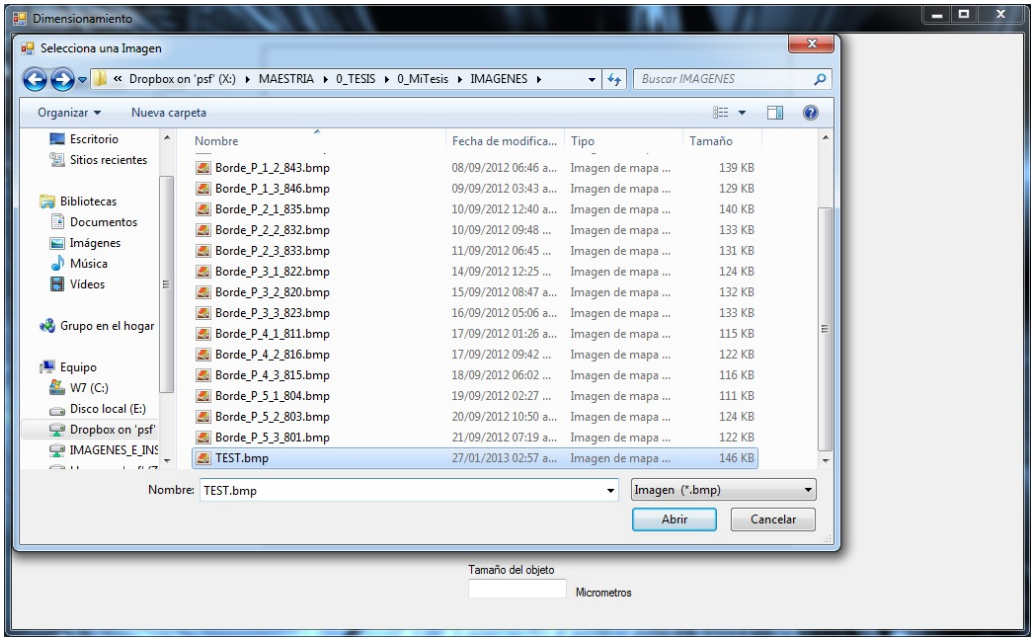
Los parámetros de ajuste del programa se explican en la tabla 6.1.

Parámetro	Descripción
Tamaño pixel	Define la medida del pixel, indicando la equivalencia de las unidades de un pixel.
Unidad del Pixel	Determina la unidad del tamaño del pixel (metros, centimetros, milímetros, etc).
X inicial	Indica la posición inicial horizontal en la imagen, donde se inicia el conteo.
X final	Indica la posición final horizontal en la imagen, donde finaliza el conteo.
Y inicial	Fija la posición inicial vertical en la imagen, donde se inicia el barrido.
Y final	Fija la posición final vertical en la imagen, donde finaliza el barrido.
Tamaño minimo	Delimita el conteo de cada línea horizontal a tener un mínimo.

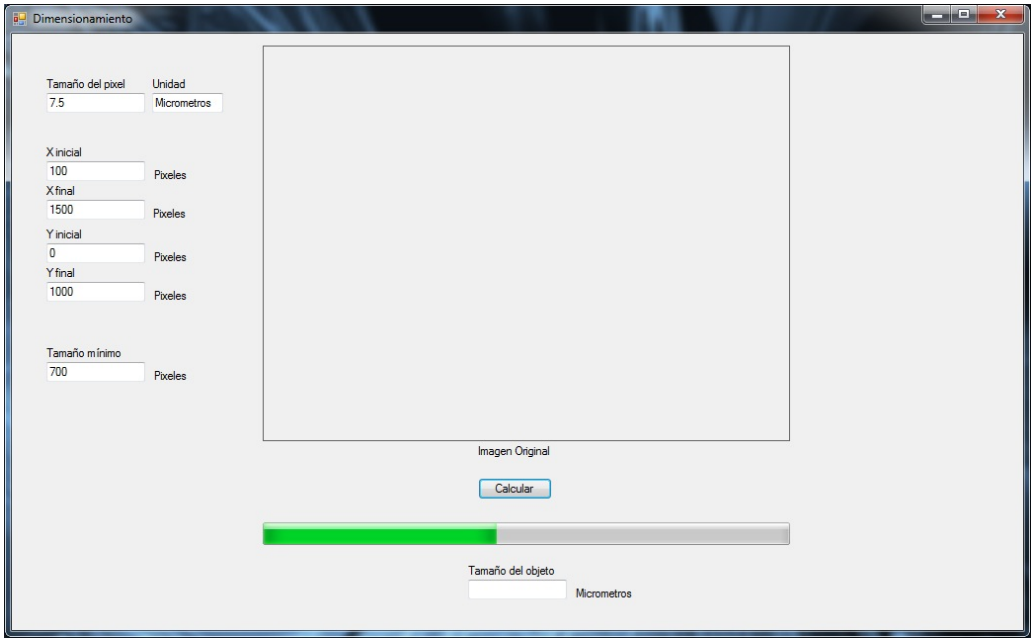
Tabla 6.1: Parámetros de ajuste del programa.



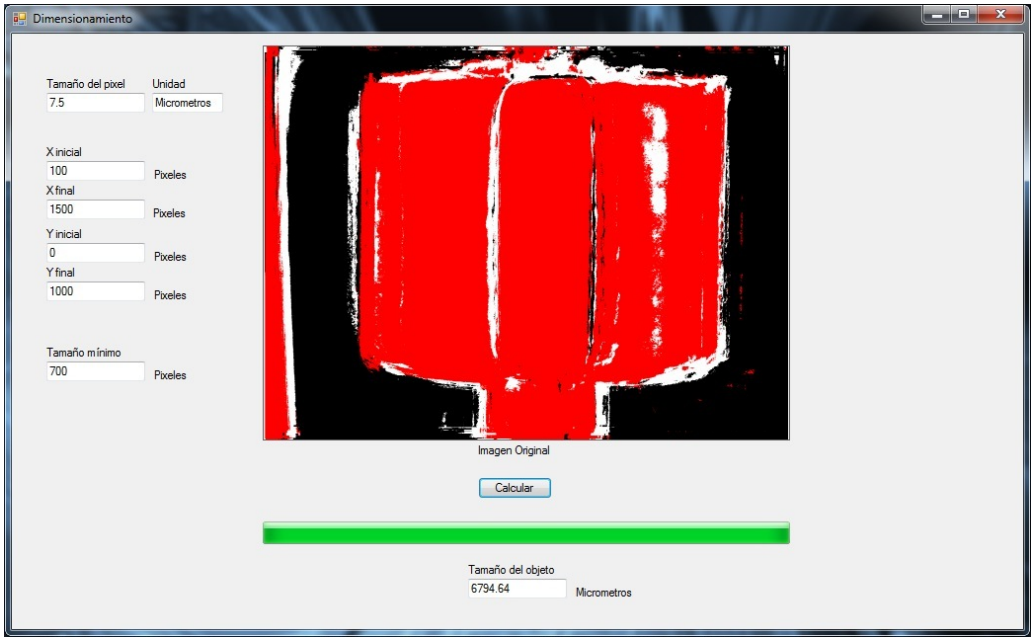
(a) Al iniciar.



(b) Abriendo una imagen.



(c) Procesando una imagen.



(d) Imagen procesada.

Figura 6.2: Programa de medición de pistones.

6.2. Resultados

Se ejecutaron varios experimentos con distintos ajustes en el programa, de los cuales se presentan a continuación como E y el número del experimento. Los ajustes se hicieron variando los parámetros de configuración, los cuales se muestran en la tabla 6.2.

Parámetro	E1	E2	E3	E4
Tamaño del pixel	9	8	7	7.3
Unidad del Pixel	μm	μm	μm	μm
X inicial	150	150	150	150
X final	1450	1450	1450	1450
Y inicial	400	400	400	400
Y final	800	800	800	800
Tamaño mínimo	700	700	700	700
Tiempo de ejecución [s]	3.5	3.5	3	3

Tabla 6.2: Parámetros y tiempo de ejecución de los experimentos.

Para clarificar el funcionamiento de los parámetros de configuración se expone de ejemplo el primer experimento (E1). Se inicia con el ajuste del tamaño del pixel, en donde se define como 9, esto quiere decir que cada pixel en la imagen tiene una dimensión de 9 unidades. El segundo parámetro es la unidad del pixel, definido en E1 como μm, el cual en combinación con el primer parámetro detalla que un solo pixel tiene una medida de 9μm. Los parámetros *X inicial*, *X final*, *Y inicial* y *Y final* acotan el área en donde se realiza el conteo, por lo que en este ejemplo, se tiene que la posición inicial en la parte superior izquierda de la imagen se encuentra en el pixel (150,400) y la posición final en la parte inferior derecha de la imagen es el pixel (1450,800). Por último se delimita el número de pixeles mínimo que se deben de contabilizar en forma horizontal, de esa forma se eliminan las líneas que contengan "artefactos" pequeños en la imagen que no corresponden al micro pistón, en este caso se fijó en 700 pixeles.

Los experimentos en los que se contabilizaron solo los pixeles identificados como objeto fueron el E1 y el E2, el resto de los experimentos (E3 y E4) se calculó la medida con los pixeles identificados como objeto y borde.

En cada uno de los experimentos se obtuvo la medida del diámetro de cada micro pistón, estas medidas se aprecian en la tabla 6.3.

Grupo	Pistón	Medición del micrómetro [mm]	E1 [mm]	E2 [mm]	E3 [mm]	E4 [mm]
1	1	8.43	9.16	8.14	7.28	8.12
1	2	8.43	9.24	8.22	8.17	8.52
1	3	8.46	9.00	8.00	7.91	8.25
2	1	8.35	9.29	8.26	8.17	8.52
2	2	8.32	9.56	8.50	8.05	8.4
2	3	8.33	9.51	8.45	8.04	8.39
3	1	8.22	9.51	8.36	7.96	8.3
3	2	8.20	8.79	7.81	7.89	8.23
3	3	8.23	8.94	7.95	7.89	8.1
4	1	8.11	9.34	8.30	7.75	8.08
4	2	8.16	8.79	7.81	7.81	8.14
4	3	8.15	9.26	8.23	7.82	8.15
5	1	8.04	9.15	8.14	7.55	7.88
5	2	8.03	8.71	7.74	7.7	8.03
5	3	8.01	9.20	8.18	7.72	8.05

Tabla 6.3: Resultados del dimensionamiento de los micro pistones.

Se compararon los resultados obtenidos por el programa y las medidas adquiridas con el micrómetro, adicionalmente se calculó su desviación estándar de las diferencias, los resultados se muestran en la tabla 6.4.

Grupo	Pistón	E1 [mm]	E2 [mm]	E3 [mm]	E4 [mm]
1	1	-0.73	0.29	0.65	0.31
1	2	-0.81	0.21	0.26	-0.09
1	3	-0.54	0.46	0.55	0.21
2	1	-0.94	0.09	0.18	-0.17
2	2	-1.24	-0.18	0.27	-0.08
2	3	-1.18	-0.12	0.29	-0.06
3	1	-1.29	-0.14	0.26	-0.08
3	2	-0.59	0.39	0.31	-0.03
3	3	-0.71	0.28	0.34	0.13
4	1	-1.23	-0.19	0.36	0.03
4	2	-0.63	0.35	0.35	0.02
4	3	-1.11	-0.08	0.33	0
5	1	-1.11	-0.1	0.49	0.16
5	2	-0.68	0.29	0.33	0
5	3	-0.17	-1.6	0.29	-0.04
σ		0.271	0.239	0.122	0.129

Tabla 6.4: Diferencias entre el valor del micrómetro y el del programa, con su desviación estándar.

La variable σ hace referencia a la efectividad de la medición, comparando los resultados del método propuesto con la obtenida con el micrómetro. El mejor resultado de σ es el más cercano a cero, esto significa que el dimensionamiento de la pieza es más exacta con respecto al valor del micrómetro.

6.3. Análisis

La medición de los micro pistones depende en gran medida de un borde muy bien definido; en las imágenes se puede apreciar un claro borde del objeto, pero internamente existe un ruido que es creado por un reflejo de luz, por lo que en esa área se reduce el número de pixeles del objeto para los experimentos 1 y 2.

Los mejores resultados se encuentran en el experimento 3, con una configuración de un área muy reducida y que contiene en varias líneas pixeles continuos del objeto y borde, desde el lado izquierdo del borde hasta el lado derecho del borde. En el experimento 3 se calculó el diámetro del micro pistón contando los pixeles del borde y del objeto, de esa forma se obtuvo una σ de 0.122.

Se destaca que en el experimento 4 se tienen 2 medidas con una diferencia de 0 mm, 8 medidas con una diferencia menor a 0.1 mm y 3 menores a 0.2 mm con respecto a las medidas del micrómetro, además de una σ de 0.129.

Es de suma importancia tomar en cuenta el tiempo de ejecución del programa, ya que se empleará en manufactura de micro piezas, por lo que el experimento 4 no solo tiene una aproximación de la medición del micro pistón, si no que también es el más rápido.

El método propuesto para el dimensionamiento de micro piezas por medio de la detección del borde de un objeto en una imagen a través de la red neuronal artificial *LIRA_grayscale* presenta diversas ventajas con respecto a los métodos tanto convencionales como los desarrollados con redes neuronales artificiales. Una de estas ventajas es el procesamiento con imágenes en escala de grises, las cuales permiten extraer más características del micro pistón bajo análisis. Otra ventaja que presenta, principalmente en relación a los métodos convencionales, es la identificación y segmentación [1] de varios objetos o clases dentro de la imagen, por lo que es posible no solo distinguir el borde del micro pistón del resto de la imagen, si no que también el micro pistón y el fondo. Asimismo, la ventaja primordial es la reducción de los contornos que no representan el borde del micro pistón, por lo que se generan mediciones más exactas y se disminuye la variabilidad del sistema al entrenarse con una base de imágenes.

[1] Proceso que divide una imagen digital en varias partes (grupos de pixeles) u objetos. [41]

Además el método desarrollado es heurístico con un algoritmo avanzado, por lo que se adapta a las necesidades cambiantes de la manufactura de micro piezas, como por ejemplo las distintas intensidades de iluminación, el tipo de objeto a manufacturar o las modificaciones que se realizan a la micro fábrica.

Conclusiones

En este trabajo se consideró el problema de la medición de micro piezas, el cual se abordó desde el punto de vista de los paradigmas de visión por computadora, para poder ser implementado en la automatización de una micro fábrica.

El principal problema detectado durante el desarrollo del trabajo fue que para medir las micro piezas por medio de sistemas de visión computacional es necesario definir el borde del objeto en la imagen, el cual se determinó a través del reconocimiento de contornos.

En el reconocimiento de los contornos se utilizaron métodos convencionales y métodos heurísticos con algoritmos avanzados. Los métodos convencionales principalmente empleados fueron el operador Sobel y el algoritmo de Schwartz, ya que el operador Roberts y el operador Prewitt eran muy parecidos al operador Sobel y que los resultados en otras investigaciones no eran muy significativos. Éstos métodos convencionales extrajeron contornos "falsos", que no coincidían con el borde.

Para el caso del reconocimiento de los contornos con métodos heurísticos, se propuso el uso del clasificador neuronal artificial *LIRA_grayscale*, debido a que las redes neuronales artificiales han demostrado ser muy versátiles en tareas donde la frontera es difusa. Asimismo, *LIRA_grayscale* ha sido probado con mucho éxito en otras tareas de manufactura de micro piezas.

Los resultados de los experimentos fueron convincentes, concluyendo que sí es posible utilizar sistemas de visión computacional para la medición automática y que las redes neuronales artificiales sirven para resolver problemas de alta complejidad.

Adicionalmente se concluye que las redes neuronales son superiores a los métodos convencionales; en aspectos como la configuración, separación de regiones no lineales e incluso en la construcción de sistemas de control mas flexibles y adaptables en la automatización de procesos.

Trabajo a futuro

Como ya se mencionó, este trabajo contribuye a un proyecto superior (manufacturar micro piezas en una micro fábrica), por lo cual el proyecto tiene una gran gama de oportunidades de mejora, trabajo adicional e investigación.

En primer lugar se propone robustecer el método de dimensionamiento de micro piezas para mejorar la precisión del método, esto se logra modificando el software y su configuración, además de implementarlo en un hardware que sea dedicado a la tarea.

El robustecimiento del software se consigue mejorando el desarrollo del "programa *LIRA*" y del "programa de dimensionamiento". Los dos programas están diseñados en POO (facilitando mucho su mantenimiento e innovación), pero el manejo de los recursos computacionales no es el óptimo; ya que la imagen es cargada en un cuadro de figura de las ventanas de los programas y desde ese recurso visual se manipulan los pixeles y se aplican los algoritmos, esto genera que el tiempo en el reconocimiento de los pixeles se incremente; se recomienda que la imagen se cargue en memoria y ahí se manipulen los pixeles y se apliquen los algoritmos.

Asimismo se propone agregar más protecciones a los programas, para reducir lo más posible los errores del usuario al trabajar con el clasificador (crear su estructura y entrenarlo) y medir las micro piezas.

Se desarrollaron 2 programas independientes, uno especialmente para el clasificador y otro para la medición del objeto, se sugiere unirlos en un programa general, en donde permita seleccionar la opción de trabajar con el clasificador ó medir micro piezas.

También se propone mejorar el método a través de la configuración del clasificador, investigando la influencia del tamaño de las ventanas, el número de neuronas ON, el número de neuronas OFF, el número de neuronas en la capa A, el parámetro de decremento (TDS) y el número de iteraciones de

entrenamiento. Inicialmente lo más recomendable es que se incremente el número de neuronas de la capa A, ya que los experimentos en otras aplicaciones demuestran un mejor porcentaje de reconocimiento.

Otra forma de mejorarlo es cambiando el hardware por uno que sea más eficiente, eficaz y dedicado a la tarea; como lo es el FPGA[2], las tarjetas computadora (SBC[3]), las unidades de procesamiento gráfico (GPUs[4]) ó las tarjetas DSP[5].

Adicionalmente se propone incrementar la base de imágenes, con la que se obtienen mas características de las imágenes de los micro pistones. Es necesario que esta nueva base de imágenes contenga distintas iluminaciones de diferentes lámparas, para así reducir la variabilidad del reconocimiento.

Para consolidar este método se requiere que primeramente se reconozca la micro pieza y la posicione en cierta forma que pueda ser medida. Existen investigaciones de reconocimiento y posicionamiento de micro piezas con el clasificador neuronal *LIRA*, con lo que se pueden combinar el presente trabajo y el anterior, para la automatización de la micro fábrica.

Por otro lado, es necesario continuar con las investigaciones de las aplicaciones basadas en redes neuronales artificiales, en especial con el clasificador neuronal *LIRA*.

[2]Siglas en Inglés de *Field Programmable Gate Array*
[3]Siglas en Inglés de *Single-Board Computer*
[4]Siglas en Inglés de *Graphics Processing Unit*
[5]Siglas en Inglés de *Digital Signal Processor*

Apéndices

Apéndice A

Software

A.1. Código fuente del procedimiento de la "Obtención coordenadas de pixeles"

```
public void GetEdgePosition(int iHeight, int iWidth, String URL_Nombre_Imagen)
{
  int i, j, iNPixels;
  int offset;

  offset = 52;

  StreamWriter file = File.AppendText(URL_Nombre_Imagen + ".CXY");
  file.Close();

  iNPixels = 0;

  //Separa en Vector en tres matrices (RGB)
  for (i = iHeight - offset; i > offset; i--)
    for (j = iWidth - offset; j > offset; j--)
    {
      if ((bRojo[i, j] == 255) && (bVerde[i, j] == 0) && (bAzul[i, j] == 0))
        {
          SavePosition(Convert.ToString(iWidth - j - 1) + " , " + Convert.ToString(iHeight - i - ↩
              1), URL_Nombre_Imagen + ".CXY");
           iNPixels++;
        }
     }
  SavePosition(URL_Nombre_Imagen + "  " + Convert.ToString(iNPixels), txt_Path.Text + "\\PIXELES.↩
       txt");

  Random rand = new Random();
  StreamWriter fondo = File.AppendText(URL_Nombre_Imagen + "_fondo" + ".CXY");
  fondo.Close();

  for (int pos = 0; pos < iNPixels; pos++)
  {
    StreamReader cont = File.OpenText(URL_Nombre_Imagen + ".CXY");
    StreamReader fondoF = File.OpenText(URL_Nombre_Imagen + ".CXY");
    bool res, res_f;
    string coord;

    do
    {
      int iH = rand.Next(offset, iHeight - offset);
```

```
        int iW = rand.Next(offset, iWidth - offset);

        coord = Convert.ToString(iW) + " , " + Convert.ToString(iH);

        string texto = cont.ReadToEnd();
        string txt_f = fondoF.ReadToEnd();

        res = texto.Contains(coord);
        res_f = txt_f.Contains(coord);
    }
    while((res)||(res_f));

    fondoF.Close();
    cont.Close();
    SavePosition(coord, URL_Nombre_Imagen + "_fondo" + ".CXY");
  }
}
```

A.2. Código fuente del procedimiento del "operador Sobel"

```
public static void Sobel (int iHeight, int iWidth)
{
  int m, n, i, j;
      int SUMxRojo, SUMyRojo;
      int SUMxVerde, SUMyVerde;
      int SUMxAzul, SUMyAzul;
      double SUMRojo, SUMVerde, SUMAzul;

    //Mascaras para obtener los gradientes.
    int[,] Gx = new int [,] {{-1, 0, 1},
                             {-1, 0, 1},
                             {-1, 0, 1}};
      int[,] Gy = new int [,] {{1, 1, 1},
                             {0, 0, 0},
                             {-1, -1, -1}};

      for (m = 1; m < iHeight - 1; m++)
          for (n = 1; n < iWidth-1; n++)
          {
            // Inicializacion
              SUMxRojo = 0;
              SUMyRojo = 0;
            SUMxVerde = 0;
              SUMyVerde = 0;
              SUMxAzul = 0;
              SUMyAzul = 0;
              for (i = 0; i < 3; i++ )
                for (j = 0; j < 3; j++)
                  {
                    //Obtencion del gradiente en direccion X y Y
                      SUMxRojo = SUMxRojo + bRojo [i+m-1,j+n-1] * Gx [i,j];
                      SUMyRojo = SUMyRojo + bRojo [i+m-1,j+n-1] * Gy [i,j];
                    SUMxVerde = SUMxVerde + bVerde[i+m-1,j+n-1] * Gx[i, j];
                      SUMyVerde = SUMyVerde + bVerde[i+m-1,j+n-1] * Gy[i, j];
                      SUMxAzul = SUMxAzul + bAzul[i+m-1,j+n-1] * Gx[i, j];
                      SUMyAzul = SUMyAzul + bAzul[i+m-1,j+n-1] * Gy[i, j];
                  }

            //Generacion del gradiente en X y Y
            SUMRojo = Math.Sqrt((SUMxRojo * SUMxRojo) + (SUMyRojo * SUMyRojo));
              SUMVerde = Math.Sqrt((SUMxVerde * SUMxVerde) + (SUMyVerde * SUMyVerde));
              SUMAzul = Math.Sqrt((SUMxAzul * SUMxAzul) + (SUMyAzul * SUMyAzul));

            //Delimitacion del gradiente maximo.
              if (SUMRojo > 255)
```

```
                SUMRojo = 255;
              if (SUMVerde > 255)
                  SUMVerde = 255;
              if (SUMAzul > 255)
                  SUMAzul = 255;

            //Asignacion del gradiente en X y Y a la Imagen Sobel resultante.
              SobelRojo [m, n] = Convert.ToByte(SUMRojo);
              SobelVerde [m, n] = Convert.ToByte(SUMVerde);
            SobelAzul [m, n] = Convert.ToByte(SUMAzul);
          }
}
```

A.3. Código fuente del procedimiento del "Algoritmo de Schwartz"

```
public int[] Histo_Orientacion_Schwartz(int iHeight, int iWidth)
{
  int Y1, Y2;
    int m, n;
    int[] histograma = new int[512];
    float Delta1, Delta2, Deltamaximo, Deltaminimo;
    double Y;

    bSchwartz = new byte[iHeight, iWidth];

    for (m = 0; m < iHeight; m++)
      for (n = 0; n < iWidth; n++)
          bSchwartz[m, n] = 0;
    for (m = 0; m < iHeight - 1; m++)
      for (n = 0; n < iWidth - 1; n++)
        {
          Y1 = bRojo[m, n] + bRojo[m + 1, n + 1];
            Y2 = bRojo[m, n + 1] + bRojo[m + 1, n];
            if (Math.Abs(Y1 - Y2) > Convert.ToDouble(txtC.Text))
            {
              Y = (Y1 + Y2) / 4;
                Delta1 = bRojo[m, n] - bRojo[m + 1, n + 1];
                Delta2 = bRojo[m + 1, n] - bRojo[m, n + 1];
                Deltamaximo = Math.Max(Math.Abs(Delta1), Math.Abs(Delta2));
                Deltaminimo = Math.Min(Math.Abs(Delta1), Math.Abs(Delta2));
                bSchwartz[m, n] = 255;
            }

         }
    return (histograma);
}
```

A.4. Código fuente de *LIRA*

A.4.1. Programa principal, desde donde se ejecuta.

Procedimiento "Conjunto de entrenamiento y prueba"

```
private void cmd_TrainTestSets_Click(object sender, EventArgs e)
{
  Class_ImageProcessing ImageProcessing = new Class_ImageProcessing ();
    byte iOpcion = 1;
    byte[] SetFormation;

    //Define la opcion para el conjunto de Imagenes (Ajedres, las primeras imagenes o ↩
        Aleatorias)
    if (r_Primeras.Checked == true)
      iOpcion = 1;
    if (r_Ajedrez.Checked == true)
        iOpcion = 2;
    if (r_Aleatorio.Checked == true)
        iOpcion = 3;

    SetFormation = ImageProcessing.ConjuntosTrainTest(txt_DataFilesPath.Text + ↩
         szTrainTestSet_FileName, Convert.ToInt32(txt_DBImNum.Text), Convert.ToInt32(↩
         txt_TrainNum.Text), iOpcion);

    for (int i = 0; i < Convert.ToInt16(txt_DBImNum.Text); i++)
    {
      //Obtiene el nombre de la imagen del directorio.
        FileInfo ImageInfo = ImageProcessing.GetFileInfo(txt_ImagePath.Text, i, "*.bmp");
        txt_ProsIma.Text = ImageInfo.Name;

        //Muestra la imagen Original en el PictureBox.
        pB_imagenORG.Load(ImageInfo.FullName);

          //Convierte y Muestra la imagen Original a escala de Grises.
          Bitmap img_ORG = new Bitmap(ImageInfo.FullName);
          pB_EscalaGris.Image = ImageProcessing.EscalaGris(img_ORG, (float)(Convert.ToDouble(↩
              txt_Ecualizador.Text)));

          //Obtiene los datos de la imagen en Escala de Gris.
          Bitmap img_EG = new Bitmap(pB_EscalaGris.Image);
          byte[] img_Datos = ImageProcessing.ObtieneDatosdelaImagen(img_EG);

          ImageProcessing.GuardaImagenAArchivo(txt_DataFilesPath.Text + "\\" + ImageInfo.Name↩
               .Substring(0, ImageInfo.Name.Length - 3) + "idt", img_Datos, img_EG.Height, ↩
               img_EG.Width);

          //Actualiza la Ventana.
          Update();
      }

      ImageProcessing = null;
  }
```

Procedimiento "Mascara"

```
    private void cmd_Masks_Click(object sender, EventArgs e)
    {
        //Class_ImageProcessing ImageProcessing = new Class_ImageProcessing();
        Class_LiRA LiRA = new Class_LiRA(Convert.ToInt32(txt_PositivPoint.Text), Convert.↩
             ToInt32(txt_NegativPoint.Text), Convert.ToInt32(txt_NetSize.Text), Convert.↩
             ToInt32(txt_ClassSize.Text));
```

```
            long MaskSize = (Convert.ToInt32(txt_PositivPoint.Text) + Convert.ToInt32(↩
                txt_NegativPoint.Text)) * Convert.ToInt64(txt_NetSize.Text);

            //Limpia las imagenes.
            pB_imagenORG.Image = null;
            pB_EscalaGris.Image = null;

            //Creacion de la Mascara
            LiRA.MaskCreation(txt_DataFilesPath.Text + szMask_PositionX_FileName, ↩
                txt_DataFilesPath.Text + szMask_PositionY_FileName, txt_DataFilesPath.Text + ↩
                szMask_Threshold_FileName, Convert.ToInt32(txt_PositivPoint.Text), Convert.↩
                ToInt32(txt_NegativPoint.Text), Convert.ToInt64(txt_NetSize.Text), Convert.↩
                ToInt32(txt_Analysis_WindowWidth.Text), Convert.ToInt32(↩
                txt_Analysis_WindowHeight.Text), Convert.ToInt32(txt_WindowWidth.Text), Convert.↩
                ToInt32(txt_WindowHeight.Text));

            //Prueba de la mascara.
            MaskTest(MaskSize);

            //ImageProcessing = null;
        }
```

Procedimientos de "Codificación"

```
        private void cmd_BegCoding_Click(object sender, EventArgs e)
        {
            Class_ImageProcessing ImageProcessing = new Class_ImageProcessing();

            long MaskSize = (Convert.ToInt32(txt_PositivPoint.Text) + Convert.ToInt32(↩
                txt_NegativPoint.Text)) * Convert.ToInt64(txt_NetSize.Text);
            int imageNumber = 0;

            //Limpia las imagenes.
            pB_imagenORG.Image = null;
            pB_EscalaGris.Image = null;

            //Obtiene la estructura de la Red Neuronal, las posiciones y los umbrales de la ↩
                mascara (capa I).
            int[] MascaraPosicionX = ImageProcessing.MaskPositionGet(txt_DataFilesPath.Text + ↩
                szMask_PositionX_FileName, (Convert.ToInt32(txt_PositivPoint.Text) + Convert.↩
                ToInt32(txt_NegativPoint.Text)) * Convert.ToInt64(txt_NetSize.Text));
            int[] MascaraPosicionY = ImageProcessing.MaskPositionGet(txt_DataFilesPath.Text + ↩
                szMask_PositionY_FileName, (Convert.ToInt32(txt_PositivPoint.Text) + Convert.↩
                ToInt32(txt_NegativPoint.Text)) * Convert.ToInt64(txt_NetSize.Text));
            byte[] MascaraUmbrales = ImageProcessing.ReadFile(txt_DataFilesPath.Text + ↩
                szMask_Threshold_FileName, (Convert.ToInt32(txt_PositivPoint.Text) + Convert.↩
                ToInt32(txt_NegativPoint.Text)) * Convert.ToInt64(txt_NetSize.Text));

            //Muestra la mascara.
            MaskTest(MaskSize);

            CodeGeneration(MascaraPosicionX, MascaraPosicionY, MascaraUmbrales, imageNumber);

            ImageProcessing = null;
        }

        private void CodeGeneration(int[] MascaraPosicionX, int[] MascaraPosicionY, byte[] ↩
             MascaraUmbrales, int imageNumber)
        {
            Class_ImageProcessing ImageProcessing = new Class_ImageProcessing();

            int[] iPixelsNumber = GetPixelsNumber();

            for (; imageNumber < Convert.ToInt32(txt_DBImNum.Text); imageNumber++)
            {
                if (StopFlag == false)
                {
                    String[,] EdgeCoor = GetPixelsCoor(txt_DataFilesPath.Text + ↩
                        szCoorBorde_FileName, imageNumber, iPixelsNumber[imageNumber]);
```

```
                    String[,] BkgCoor = GetPixelsCoor(txt_DataFilesPath.Text + ↩
                        szCoorFondo_FileName, imageNumber, iPixelsNumber[imageNumber]);
                    String[,] ObjCoor = GetPixelsCoor(txt_DataFilesPath.Text + ↩
                        szCoorObjeto_FileName, imageNumber, iPixelsNumber[imageNumber]);

                    //Lee la imagen desde el archivo.
                    FileInfo ImageInfo = ImageProcessing.GetFileInfo(txt_DataFilesPath.Text, ↩
                        imageNumber, "*.idt");
                    txt_ProsIma.Text = ImageInfo.Name;
                    byte[] ImRead = ImageProcessing.ReadFile(txt_DataFilesPath.Text + "\\" + ↩
                        ImageInfo.Name, ImageInfo.Length);
                    //Muestra la imagen desde el archivo.
                    pB_imagenORG.Image = MuestraImagen(1600, 1200, ImRead);

                    EnlargeImage(pB_imagenORG.Image.Width, pB_imagenORG.Image.Height);
                    //EnlargeImage(1600, 1200);
                    Update();

                    FileStream ArchivoCoImagen = new FileStream(txt_DataFilesPath.Text + "\\" + ↩
                        ImageInfo.Name.Substring(0, ImageInfo.Name.Length - 3) + "icd", FileMode↩
                        .Create);
                    ArchivoCoImagen.Close();

                        for (int iPixelN = 0; iPixelN < iPixelsNumber[imageNumber]; iPixelN++)
                        {
                            //Borde
                            Code(Convert.ToInt32(EdgeCoor[iPixelN, 0]), Convert.ToInt32(EdgeCoor[↩
                                iPixelN, 1]), ImageInfo, MascaraPosicionX, MascaraPosicionY, ↩
                                MascaraUmbrales, 1);
                            //Fondo
                            Code(Convert.ToInt32(BkgCoor[iPixelN, 0]), Convert.ToInt32(BkgCoor[↩
                                iPixelN, 1]), ImageInfo, MascaraPosicionX, MascaraPosicionY, ↩
                                MascaraUmbrales, 2);
                            //Objeto
                            Code(Convert.ToInt32(ObjCoor[iPixelN, 0]), Convert.ToInt32(ObjCoor[↩
                                iPixelN, 1]), ImageInfo, MascaraPosicionX, MascaraPosicionY, ↩
                                MascaraUmbrales, 3);
                            txt_ImageNumber.Text = "Imagen Numero: " + Convert.ToString(↩
                                imageNumber + 1);
                            txt_Pixel.Text = "Pixel: " + Convert.ToString(iPixelN + 1);
                            Update();
                        }
                    }
                }
                Update();
            }
            ImageProcessing = null;
        }

        private void Code(int X, int Y, FileInfo ImageInfo, int[] MascaraPosicionX, int[] ↩
             MascaraPosicionY, byte[] MascaraUmbrales, int clase)
        {
            Class_LiRA LiRA = new Class_LiRA(Convert.ToInt32(txt_PositivPoint.Text), Convert.↩
                ToInt32(txt_NegativPoint.Text), Convert.ToInt32(txt_NetSize.Text), Convert.↩
                ToInt32(txt_ClassSize.Text));

            int ActivNeuronNumber = LiRA.Coding((Bitmap)pB_EscalaGris.Image, X + 1600, Y + 1600, ↩
                MascaraPosicionX, MascaraPosicionY, MascaraUmbrales, Convert.ToInt32(↩
                txt_PositivPoint.Text), Convert.ToInt32(txt_NegativPoint.Text), Convert.ToInt32(↩
                txt_NetSize.Text), Convert.ToInt32(txt_Analysis_WindowWidth.Text), Convert.↩
                ToInt32(txt_Analysis_WindowHeight.Text));

            LiRA.Save_Code(txt_DataFilesPath.Text + "\\" + ImageInfo.Name.Substring(0, ImageInfo.↩
                Name.Length - 3) + "icd");
            txt_Suma.Text = "Suma Neuronas Activas: " + Convert.ToString(ActivNeuronNumber);
            if (clase == 1)
                pB_CodTest.Image = Prueba_Codificacion(LiRA.i_A);
            if (clase == 2)
                pB_CodTest_BK.Image = Prueba_Codificacion(LiRA.i_A);
            if (clase == 3)
                pB_CodTest_OBJ.Image = Prueba_Codificacion(LiRA.i_A);

            Update();
            LiRA = null;
        }

        private Image Prueba_Codificacion(byte[] codigo)
```

```
        {
            Class_ImageProcessing ImageProcessing = new Class_ImageProcessing();
            int constante = Convert.ToInt32(Math.Sqrt(Convert.ToInt64(txt_NetSize.Text)));

            Bitmap imgBitmap = new Bitmap(constante, constante);

            for (int i = 0; i < codigo.Length; i++)
                if (codigo[i] == 1)
                    codigo[i] = 255;

            byte[] dat_imgBitmap = ImageProcessing.CreaImagenDeArchivo(codigo, ImageProcessing.↩
                 ObtieneDatosdelaImagen(imgBitmap));
            imgBitmap.Dispose();
            return (ImageProcessing.ObtieneImagendelosDatos(dat_imgBitmap));
        }
```

Procedimientos de "Entrenamiento"

```
        private void cmd_BegTrain_Click(object sender, EventArgs e)
        {
            Class_ImageProcessing ImageProcessing = new Class_ImageProcessing();
            Class_LiRA LiRA = new Class_LiRA(Convert.ToInt32(txt_PositivPoint.Text), Convert.↩
                 ToInt32(txt_NegativPoint.Text), Convert.ToInt32(txt_NetSize.Text), Convert.↩
                 ToInt32(txt_ClassSize.Text));

            int ErrorNumber;
            int index;

            //Limpia las imagenes.
            pB_imagenORG.Image = null;
            pB_EscalaGris.Image = null;

            byte[] SetFormation = ImageProcessing.ReadFile(txt_DataFilesPath.Text + ↩
                 szTrainTestSet_FileName, Convert.ToInt32(txt_DBImNum.Text));

            FileStream errors = File.Open(txt_DataFilesPath.Text + szErrors_FileName, FileMode.↩
                 Create);
            errors.Close();

            int[] iPixelsNumber = GetPixelsNumber();

            for (int iter = 0; iter < Convert.ToInt32(txt_PassN.Text); iter++)
            {
                ErrorNumber = 0;

                for (int imageNumber = 0; (imageNumber < Convert.ToInt32(txt_DBImNum.Text)); ↩
                     imageNumber++)
                {
                    if (SetFormation[imageNumber] == 1)
                    {
                        //Lee la imagen desde el archivo.
                        FileInfo ImageDataInfo = ImageProcessing.GetFileInfo(txt_DataFilesPath.↩
                             Text, imageNumber, "*.idt");
                        txt_ProsIma.Text = ImageDataInfo.Name;
                        byte[] ImRead = ImageProcessing.ReadFile(txt_DataFilesPath.Text + "\\" + ↩
                             ImageDataInfo.Name, ImageDataInfo.Length);
                        //Muestra la imagen desde el archivo.
                        pB_imagenORG.Image = MuestraImagen(1600, 1200, ImRead);
                        Update();

                        FileInfo ImageInfo = ImageProcessing.GetFileInfo(txt_DataFilesPath.Text, ↩
                             imageNumber, "*.icd");
                        byte[] FileCode = LiRA.ReadFile(txt_DataFilesPath.Text + "\\" + ImageInfo↩
                             .Name.Substring(0, ImageInfo.Name.Length - 3) + "icd", ImageInfo.↩
                             Length);
                        index = 0;


                        for (int Pixel = 0; Pixel < iPixelsNumber[imageNumber] * Convert.ToInt32(↩
                             txt_ClassSize.Text); Pixel++)
```

```
                        {
                            for (int i = 0; i < Convert.ToInt32(txt_NetSize.Text); i++)
                                LiRA.i_A[i] = FileCode[index++];

                            //Define supervisadamente a que clase pertenece el pixel.
                            int true1 = LiRA.ClName(Pixel, Convert.ToInt32(txt_ClassSize.Text));

                            int rec = LiRA.recognition(Convert.ToInt32(txt_ClassSize.Text), ↩
                                 Convert.ToInt32(txt_NetSize.Text), Convert.ToDouble(txt_TDS.Text↩
                                 ), true1);

                            if (rec != true1)
                            {
                                ErrorNumber++;
                                LiRA.training(Convert.ToInt32(txt_NetSize.Text), true1, rec);
                                Bitmap ImageERROR = new Bitmap(pB_imagenORG.Image);
                                pB_EscalaGris.Image = ImageERROR;
                            }

                            txt_Errores.Text = "Errorres: " + Convert.ToString(ErrorNumber);
                            txt_Pixel.Text = "Pixel: " + Convert.ToString(Pixel + 1);
                            txt_ImageNumber.Text = "Imagen Numero: " + Convert.ToString(↩
                                 imageNumber + 1);
                            txt_Iteracion.Text = "Iteracion: " + Convert.ToString(iter + 1);
                            Update();
                        }
                    }
                }

                StreamWriter Errorsfile = File.AppendText(txt_DataFilesPath.Text + ↩
                     szErrors_FileName);
                Errorsfile.WriteLine(Convert.ToString(ErrorNumber));
                Errorsfile.Close();
            }

            //Escribe los pesos al archivo.
            FileStream Weights = new FileStream(txt_DataFilesPath.Text + szWeights_FileName, ↩
                 FileMode.Create);
            BinaryWriter Weightsfile = new BinaryWriter(Weights);

            for (int i = 0; i < Convert.ToInt64(txt_NetSize.Text); i++)
                for (int j = 0; j < Convert.ToInt32(txt_ClassSize.Text); j++)
                {
                    Weightsfile.Write(LiRA.i_Weights[j, i]);
                }
            Weightsfile.Close();
        }

        private void cmd_Recognize_Click(object sender, EventArgs e)
        {
            Class_ImageProcessing ImageProcessing = new Class_ImageProcessing();
            Class_LiRA LiRA = new Class_LiRA(Convert.ToInt32(txt_PositivPoint.Text), Convert.↩
                 ToInt32(txt_NegativPoint.Text), Convert.ToInt32(txt_NetSize.Text), Convert.↩
                 ToInt32(txt_ClassSize.Text));

            int ErrorNumber;
            int index;

            //Limpia las imagenes.
            pB_imagenORG.Image = null;
            pB_EscalaGris.Image = null;

            byte[] SetFormation = ImageProcessing.ReadFile(txt_DataFilesPath.Text + ↩
                 szTrainTestSet_FileName, Convert.ToInt32(txt_DBImNum.Text));

            int[] iPixelsNumber = GetPixelsNumber();

            ErrorNumber = 0;

            //Lee los pesos del archivo.
            FileStream Weights = File.Open(txt_DataFilesPath.Text + szWeights_FileName, FileMode.↩
                 Open);
            BinaryReader Weightsfile = new BinaryReader(Weights);

            for (int i = 0; i < Convert.ToInt64(txt_NetSize.Text); i++)
                for (int j = 0; j < Convert.ToInt32(txt_ClassSize.Text); j++)
                    LiRA.i_Weights[j, i] = Weightsfile.ReadInt32();
```

```
            Weightsfile.Close();

            for (int imageNumber = 0; (imageNumber < Convert.ToInt32(txt_DBImNum.Text)); ↩
                 imageNumber++)
            {
                if (SetFormation[imageNumber] != 1)
                {
                    //Lee la imagen desde el archivo.
                    FileInfo ImageDataInfo = ImageProcessing.GetFileInfo(txt_DataFilesPath.Text, ↩
                         imageNumber, "*.idt");
                    txt_ProsIma.Text = ImageDataInfo.Name;
                    byte[] ImRead = ImageProcessing.ReadFile(txt_DataFilesPath.Text + "\\" + ↩
                         ImageDataInfo.Name, ImageDataInfo.Length);
                    //Muestra la imagen desde el archivo.
                    pB_imagenORG.Image = MuestraImagen(1600, 1200, ImRead);
                    Update();

                    FileInfo ImageInfo = ImageProcessing.GetFileInfo(txt_DataFilesPath.Text, ↩
                         imageNumber, "*.icd");
                    byte[] FileCode = LiRA.ReadFile(txt_DataFilesPath.Text + "\\" + ImageInfo.↩
                         Name.Substring(0, ImageInfo.Name.Length - 3) + "icd", ImageInfo.Length);
                    index = 0;


                    for (int Pixel = 0; Pixel < iPixelsNumber[imageNumber] * Convert.ToInt32(↩
                         txt_ClassSize.Text); Pixel++)
                    {
                        for (int i = 0; i < Convert.ToInt32(txt_NetSize.Text); i++)
                            LiRA.i_A[i] = FileCode[index++];

                        //Define supervisadamente a que clase pertenece el pixel.
                        int true1 = LiRA.ClName(Pixel, Convert.ToInt32(txt_ClassSize.Text));

                        int rec = LiRA.recognition(Convert.ToInt32(txt_ClassSize.Text), Convert.↩
                             ToInt32(txt_NetSize.Text), Convert.ToDouble(txt_TDS.Text), true1);

                        if (rec != true1)
                        {
                            ErrorNumber++;
                            Bitmap ImageERROR = new Bitmap(pB_imagenORG.Image);
                            pB_EscalaGris.Image = ImageERROR;
                        }

                        txt_Errores.Text = "Errorres: " + Convert.ToString(ErrorNumber);
                        txt_Pixel.Text = "Pixel: " + Convert.ToString(Pixel + 1);
                        txt_ImageNumber.Text = "Imagen Numero: " + Convert.ToString(imageNumber +↩
                              1);
                        Update();
                    }
                }
            }

            StreamWriter Errorsfile = File.AppendText(txt_DataFilesPath.Text + szErrors_FileName)↩
                ;
            Errorsfile.WriteLine(Convert.ToString(ErrorNumber));
            Errorsfile.Close();
        }
```

A.4.2. Clase *LIRA*.

```
    public class Class_LiRA
    {
        public bool[,] b_ON;
        public bool[,] b_OFF;
        public byte[] i_A;
        public int[] i_R;
        public int[,] i_Weights;

        public Class_LiRA(int PositivPoint, int NegativPoint, int NetSize, int ClassSize)
```

```
        {
            b_ON = new bool[PositivPoint, NetSize];
            b_OFF = new bool[NegativPoint, NetSize];
            i_A = new byte[NetSize];
            i_R = new int[ClassSize];
            i_Weights = new int[ClassSize, NetSize];

            for (int i = 0; i<NetSize; i++)
            {
                for (int j = 0; j < PositivPoint; j++)
                    b_ON[j, i] = false;
                for (int j = 0; j < NegativPoint; j++)
                    b_OFF[j, i] = false;

                i_A[i] = 0;

                for (int j = 0; j < ClassSize; j++)
                {
                    i_R [j] = 0;
                    i_Weights[j, i] = 0;
                }
            }
        }

        public void MaskCreation(String FilePathPositionX, String FilePathPositionY, String ↩
             FilePathThreshold, int PositivPoint, int NegativPoint, long NetSize, long ↩
             AnalysisWidth, long AnalysisHeight, int WindowWidth, int WindowHeight)
        {
            Random rand = new Random();
            long MaskSize = (PositivPoint + NegativPoint) * NetSize;
            int[] lPositionX = new int[MaskSize];
            int[] lPositionY = new int[MaskSize];
            byte[] bUmbral = new byte[MaskSize];

            for (int i = 0; i < MaskSize; i++)
                bUmbral[i] = 0;

            for (int j = 0; j < NetSize; j++)
            {
                // Genera los valores aleatorios para las coordenadas en X & Y de la ventana LiRA

                int u = rand.Next(0, Convert.ToInt32(AnalysisWidth - WindowWidth));
                int v = rand.Next(0, Convert.ToInt32(AnalysisHeight - WindowHeight));

                for (int i = 0; i < (PositivPoint + NegativPoint); i++)
                {
                    // Genera los valores aleatorios para las coordenadas en X & Y para la ventana ↩
                         de las neuronas.
                    int x = rand.Next(0, WindowWidth) + u;
                    int y = rand.Next(0, WindowHeight) + v;
                    lPositionX[j * (PositivPoint + NegativPoint) + i] = x;
                    lPositionY[j * (PositivPoint + NegativPoint) + i] = y;
                    //Umbral Aleatorio.
                    bUmbral[j * (PositivPoint + NegativPoint) + i] = Convert.ToByte(rand.Next(0, ↩
                         255));
                }
            }

            //Guarda la mascara (capa I)
            FileStream ArchivoPosicionX = new FileStream(FilePathPositionX, FileMode.Create);
            BinaryWriter MaskXPositionfile = new BinaryWriter(ArchivoPosicionX);
            FileStream ArchivoPosicionY = new FileStream(FilePathPositionY, FileMode.Create);
            BinaryWriter MaskYPositionfile = new BinaryWriter(ArchivoPosicionY);

            for (int i = 0; i < MaskSize; i++)
            {
                MaskXPositionfile.Write(lPositionX[i]);
                MaskYPositionfile.Write(lPositionY[i]);
            }

            MaskXPositionfile.Close();
            MaskYPositionfile.Close();

            FileStream ArchivoUmbral = File.Open(FilePathThreshold, FileMode.Create);
            ArchivoUmbral.Write(bUmbral, 0, bUmbral.Length);
            ArchivoUmbral.Close();
        }
```

```
        public int Coding(Bitmap Imagen, int CoorX, int CoorY, int[] MascaraPosicionX, int[] ↩
             MascaraPosicionY, byte[] MascaraUmbrales, int PositivPoint, int NegativPoint, int ↩
             NetSize, int AnalysisWidth, int AnalysisHeight)
        {
            int index = 0;
            int ON, OFF;
            int iCont = 0;

            for (int iNeuronA = 0; iNeuronA < NetSize; iNeuronA++)
            {
                for (int k = 0; k < PositivPoint; k++, index++)
                {
                    int X = CoorX - ((AnalysisWidth - 1) / 2) + MascaraPosicionX[index];
                    int Y = CoorY - ((AnalysisHeight - 1) / 2) + MascaraPosicionY[index];
                    Color cPixel = Imagen.GetPixel(X, Y);
                    if (cPixel.R > MascaraUmbrales[index])
                        b_ON[k, iNeuronA] = true;
                }

                for (int k = 0; k < NegativPoint; k++, index++)
                {
                    int X = CoorX - ((AnalysisWidth - 1) / 2) + MascaraPosicionX[index];
                    int Y = CoorY - ((AnalysisHeight - 1) / 2) + MascaraPosicionY[index];
                    Color cPixel = Imagen.GetPixel(X, Y);
                    if (cPixel.R < MascaraUmbrales[index])
                        b_OFF[k, iNeuronA] = true;
                }

                for (ON = 0; (ON < PositivPoint) && (b_ON[ON, iNeuronA] == true); ON++) ;
                for (OFF = 0; (OFF < NegativPoint) && (b_OFF[OFF, iNeuronA] == true); OFF++) ;

                if ((ON == PositivPoint) && (OFF == NegativPoint))
                {
                    i_A[iNeuronA] = 1;
                    iCont++;
                }
            }

            return (iCont);
        }

        public int recognition(int ClassSize, int NetSize, double TDS, int true1)
        {
            int rec;
            int[] sumaN = new int[ClassSize];
            double f1, f2;
            int max;

            //Inicializacion de la suma
            for (int i = 0; i < ClassSize; i++)
                sumaN[i] = 0;

            for (int k = 0; k < ClassSize; k++)
                for (int j = 0; j < NetSize; j++)
                    sumaN[k] += i_Weights[k, j] * Convert.ToInt32(i_A[j]);

            f1 = sumaN[true1];
            f2 = f1 * (1 - TDS);
            sumaN[true1] = (int)f2;
            max = 0;
            rec = 0;
            for (int i = 0; i < ClassSize; i++)
                if (sumaN[i] > max)
                {
                    max = sumaN[i];
                    rec = i;
                }

            return (rec);
        }

        public void training(int NetSize, int true1, int rec)
        {
            for (int j = 0; j < NetSize; j++)
            {
                if (i_A[j] == 1)
```

```
                {
                    i_Weights[true1, j]++;
                    i_Weights[rec, j]--;
                }
            }
        }
    }
```

A.5. Código fuente del procedimiento de "Dimensionamiento del objeto"

```
private void Calcular_Click(object sender, EventArgs e)
{
  int iHeight, iWidth, X, Y, iCont;
    int[] iLine;
    int[] iLineReduce;
    double dMean;

    progressBar.Value = 0;

  //Crea el objento de cuadro de dialogo de "Abrir Archivo", con los parametros dados
    OpenFileDialog AbrirArchivo = new OpenFileDialog();
    AbrirArchivo.Filter = "Imagen |*.bmp";
    AbrirArchivo.Title = "Selecciona una Imagen";

    //Muestra el cuadro de dialogo y obtiene su respuesta.
    if (AbrirArchivo.ShowDialog() == DialogResult.OK)
    {
      //Obtiene el nombre del archivo del cuadro de dialogo "Abrir Archivo"
        String URL_Nombre_Imagen = AbrirArchivo.FileName;

        //Muestra la imagen Original en el PictureBox
        pB_imagenORG.Load(URL_Nombre_Imagen);

        Bitmap BMPImage = new Bitmap(URL_Nombre_Imagen);

        //Obtiene el tamao de la imagen.
        iHeight = pB_imagenORG.Image.Height;
        iWidth = pB_imagenORG.Image.Width;
        iLine = new int[iHeight];
        iLineReduce = new int[Convert.ToInt32(txtYEnd.Text) - Convert.ToInt32(txtYStart.Text)];
        progressBar.Minimum = Convert.ToInt32(txtYStart.Text);
        progressBar.Maximum = Convert.ToInt32(txtYEnd.Text);

        //Inicializacion del contador
        for(iCont=0;iCont<iHeight;iCont++)
          iLine[iCont]=0;

        for (Y = Convert.ToInt32(txtYStart.Text); Y < Convert.ToInt32(txtYEnd.Text); Y++)
        {
          for (X = Convert.ToInt32(txtXStart.Text); X < Convert.ToInt32(txtXEnd.Text); X++)
            {
              Color pixelColor = BMPImage.GetPixel(X, Y);

                if((pixelColor.R==255)&&(pixelColor.G==0)&&(pixelColor.B==0))
                  iLine[Y]++;
            }
            progressBar.Increment(1);
        }

        for (Y = iCont = 0; Y < iHeight; Y++)
        {
          if (iLine[Y] > Convert.ToInt32(txtMinimumSize.Text))
            {
              iLineReduce[iCont] = iLine[Y];
                iCont++;
```

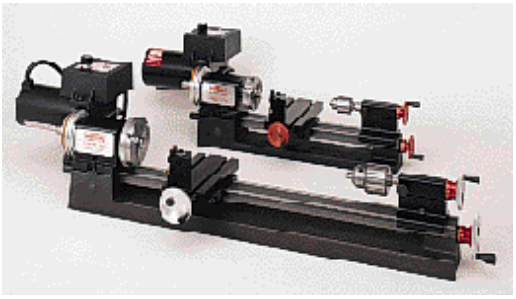
```
            }
        }

        dMean = iLineReduce.Average();
        txtObjectSize.Text=Convert.ToString(dMean*Convert.ToDouble(txtPixelSize.Text));
        lblUnidad.Text = txtUnidad.Text;
    }
}
```

Apéndice B

Especificaciones del torno.

SHERLINE PRODUCTS
INCORPORATED 1974

Sherline Lathes



The Model 4000 lathe (rear) is shown with the longer Model 4400 lathe (front). The chucks shown on the lathes are included in the "A" package.

When the Sherline lathe first came on the market over thirty years ago, its use of rigid, extruded components meant miniature machine tools were no longer just toys for producing simple hobby projects. They could now be considered serious machine tools built specially to produce accurate, small parts. Since then, we have not only added a vertical milling machine and extensive accessory line, we have also found ways to improve the accuracy and utility of the tools themselves. The introduction of CNC machines into our production facility has greatly improved the accuracy of Sherline tools. The lathes feature precision rolled leadscrews and handwheels graduated in thousandths of an inch (.001") or hundredths of a millimeter (.01 mm).

When used with its various accessories, Sherline lathes will perform a host of tasks. They will turn, face, bore, drill, ream, polish, cut tapers, and cut both inch and metric threads. When used with its vertical milling column attachment it can be used for milling, fly cutting, drilling, and boring operations. The machines

Sherline now offers several lathes to fit every budget and need, and they are available with either inch or millimeter calibrations. A high-torque DC motor with variable speed control is standard on each machine. This speed control is internally equipped with a converter that automatically adjusts to incoming AC current from 100 to 240 volts, 50 or 60 cycles/sec without loss of torque.

Choose a Lathe...

Click on the product number below for a photo, complete description and price for each machine. (Part numbers for metric machines are given in parenthesis. Price is the same for either.)

4000 (4100) 3.5" x 8" lathe

4500 (4530) 3.5" x 8" lathe w/ adjustable zero handwheels

4400 (4410) 3.5" x 17" lathe

Lathes with digital readouts factory installed

Lathes with CNC-ready package factory installed

Lathe Specifications

FEATURE	4000(4100)	4400(4410)
Swing over bed	3.50" (90 mm)	3.50" (90 mm)
Swing over carriage	1.75" (45 mm)	1.75" (45 mm)
Distance between centers	8.00" (200 mm)	17.00" (430 mm)
Hole through spindle	.405" (10 mm)	.405" (10 mm)
Spindle nose thread	3/4"-16 T.P.I.	3/4"-16 T.P.I .
Spindle nose taper	#1 Morse	#1 Morse
Travel of crosslide	4.25" (110 mm)	4.25" (110 mm)
Tailstock spindle taper	#0 Morse	#0 Morse.
Protractor graduations	0° to 45° by 5°	0° to 45° by 5°
Handwheel graduations	.001" (.01 mm)	.001" (.01 mm)
Length overall	24" (610 mm)	32.25" (820 mm)
Width overall	7.5" (190 mm)	8.75" (220 mm)
Height overall	6" (150 mm)	8" (200 mm)
Shipping weight	24 lb. (10.9 kg)	30 lb. (13.6 kg)
Motor	90 volt DC with electronic speed control that accepts any incoming current from 100VAC to 240 VAC, 50 Hz or 60 Hz. Click here for motor specifications.	
Spindle speed range	70-2800 RPM continuously variable by electronic speed control	

Save money with package deals

Lathes can now be purchased packaged with chucks and other preselected groups of accessories. See the page on PACKAGE DEALS. Save time and money by buying a package.

RETURN TO SHERLINE'S HOME PAGE

Apéndice C

Especificaciones de la cámara.

2.0MP Low Lux Microscope Camera Digital Image System

Model MD700

You are bidding on an AmScope 2.0MP USB2.0 low lux color digital image system that is specially designed for microscopes. It comes with an extra chip to handle the image resolution and color in low light applications, providing extremely high quality microscope images. It is the first choice for all of the microscopy applications in brightfield, darkfield, phase contrast, fluorescence, and metallurgy. This color digital image system comes with a 2.0 MP low lux digital camera, user-friendly software & manual, detailed user's operation instructions, and adapters. It captures microscope images and displays live videos on your PC screen. It offers full-screen display and provides 2.0 megapixel high resolution images. The digital camera has a built-in reduction lens and offers the same field of view images on your PC screen as those seen through eyepieces. The software is compatible with Windows 2000/XP/Vista. It can edit microscope images on your computer in the same way as PhotoShop dose. It can make real time videos or capture still images and save them in BMP, TIFF, JPG, PICT, PTL and other formats. It can also make measurement across microscope images for distance, angle, area, and etc. Beside user-friendly manual, this system also includes detailed user's operation instructions to run the system smoothly and upgrade the software easily. Furthermore, this color image system is designed to be parfocal (in focus when changing from binocular viewing to trinocular viewing) with AmScope trinocular microscopes. In other words, you will see a clear image on your computer screen while you get the one in your microscope eyepiece(s) in focus. This color image system fits all microscopes including monocular, binocular, trinocular, stereo, dissecting and any other specialty microscopes. Unbeatable low price is guaranteed!

Features & Specifications:

- A 2.0MP high resolution, high quality low lux color USB2.0 PC digital image system for all microscopes
- The first choice for all of the microscopy applications in brightfield, darkfield, phase contrast, fluorescence, and metallurgy
- Captures microscope images and shows live video on your PC screen
- Displays 2.0 MegaPixel full-screen clear images with high resolution
- Saves still images in BMP, TIFF, JPG, PICT, PTL or other formats

Apéndice D

Especificaciones del microscopio.

Metallurgical Trinocular Microscope NJF-120A

Short review XTX-series NF-120A. *Suitable for inspection and analysis with incident and transmitted illumination.*

Metallurgical microscope NJF-120A is suitable for **inspection** and **analysis** of the structure of various metals, alloys and non-metal materials. Its infinitive optical system provides excellent optical functions. The microscope is equipped with the **incident** and **transmitted illumination**. **NJF-120A** is widely used in **electronic industry** and **chemical engineering** with instruments and meters to observe surfaces of opaque materials, PCB chips, LCDs, wires, fibers, plating etc. The instrument can also be used in **biological study**.

Specifications

Compensation viewing head	trinocular
Eyepiece	WF10 × / 18 mm
Objective: Infinitive Plan Achromatic	4 × / 0.1∞ / - WD25.4 mm 10 × / 0.25 ∞ / -WD11.0 mm
Objective	20 × / 0.40 ∞ / -WD6.0 mm 40 × / 0.60∞ / -WD3.7 mm
Nosepiece	quadruple nosepiece
Stage	double layers mechanical stage 150 mm × 140 mm; moving range 75 mm × 50 mm
Focusing	coaxial coarse & fine focus adjustment system
Ullumination	halogen lamp 6v20 V,adjustable brightness
Polarization	polarizer
Filter	blue, yellow, green and frosted glass
Digital camera port	"C" standard port

Bibliografía

[1] William S. Trimmer and Institute of Electrical and Electronics Engineers. *Micromechanics and MEMS: Classic and Seminal Papers to 1990*. Wiley-IEEE Press, 1 edition, Jan 1997.

[2] Yuichi Okazaki, Nozomu Mishima, and Kiwamu Ashida. Microfactory: Concept, history, and developments. *Journal of Manufacturing Science and Engineering*, 126(4):837 – 844, 2004.

[3] Ernst M. Kussul, Dmitri A. Rachkovskij, Tatyana M. Baidyk, and Semion A. Talayev. Micromechanical engineering: a basis for the low-cost manufacturing of mechanical microdevices using microequipment. *Journal of Micromechanics and Microengineering*, 6(4):410 – 425, 1996.

[4] F. Chollet and HB. Liu. A (not so) short introduction to mems. At the website `http://memscyclopedia.org/introMEMS.html`, Oct. 2012. [Online; accessed 25-November-2012].

[5] Ernst Kussul, Tatiana Baidyk, and Donald C. Wunsch. *Neural Networks and Micromechanics*. Springer Berlin Heidelberg, 1 edition, 2010.

[6] E. Kussul, T. Baidyk, L. Ruiz-Huerta, A. Caballero-Ruiz, G. Velasco, and O. Makeyev. Techniques in the Development of Micromachine Tool Prototypes and Their Applications in Microfactories MET Technology. In CorneliusT. Leondes, editor, *MEMS/NEMS*, pages 616 – 677. Springer US, 2006.

[7] Tanaka Makoto. Development of desktop machining microfactory. *RIKEN*, 34:46 – 49, April 2001.

[8] E. Kussul, T. Baidyk, L. Ruiz-Huerta, A. Caballero-Ruiz, G. Velasco, and L. Kasatkina. Development of micromachine tool prototypes

for microfactories. *Journal of Micromechanics and Microengineering*, 12(6):795, 2002.

[9] Microlution Inc. Microlution CNC. At the website `http://www.microlution-inc.com/products/363.php`, 2013. [Online; accessed 20-April-2013].

[10] MCC Inc. MCC CNC. At the website `http://www.marucit.com/products/L20.html`, 2013. [Online; accessed 20-April-2013].

[11] MCC Inc. MCC CNC. At the website `http://www.marucit.com/products/K16.html`, 2013. [Online; accessed 20-April-2013].

[12] Nakamura tome Precision Industry co. WT100 CNC. At the website `http://www.nakamura-tome.co.jp/e/products/turret/wt-series/wt-100.html`, 2013. [Online; accessed 20-April-2013].

[13] Ernst Kussul, Oleksandr Makeyev, Tatiana Baidyk, and Omar Olvera. Design of Ericsson Heat Engine with Micro Channel Recuperator. *ISRN Renewable Energy*, Vol. 2012(4):8, 2012. doi:10.5402/2012/613642.

[14] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley-Interscience, New York; Chichester, 2 edition, October 2000.

[15] Gengis K. Toledo, Ernst Kussul, and Tatiana Baidyk. Neural classifier for micro work piece recognition. *Image and Vision Computing*, 24(8):827 – 836, 2006.

[16] T. Baydyk and E. Kussul. Application of neural classifier for flat image recognition in the process of microdevice assembly. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 1, pages 160 – 164, 2002.

[17] P.J. Terry and D. Vu. Edge detection using neural networks. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 391 – 395 vol.1, nov 1993.

[18] G. Grassi, P. Vecchio, E. Di Sciascio, L.A. Grieco, and D. Cafagna. Neural networks for image processing: New edge detection algorithm. In

Electro/Information Technology, 2007 IEEE International Conference on, pages 498 – 502, may 2007.

[19] A. Damak, M. Krid, and D.S. Masmoudi. Neural network based edge detection with pulse mode operations and floating point format precision. In *Design and Technology of Integrated Systems in Nanoscale Era, 2008. DTIS 2008. 3rd International Conference on*, pages 1 – 5, march 2008.

[20] H. Mehrara, M. Zahedinejad, and A. Pourmohammad. Novel edge detection using bp neural network based on threshold binarization. In *Computer and Electrical Engineering, 2009. ICCEE '09. Second International Conference on*, volume 2, pages 408 – 412, dec. 2009.

[21] Zhengquan He and M.Y. Siyal. Edge detection with bp neural networks. In *Signal Processing Proceedings, 1998. ICSP '98. 1998 Fourth International Conference on*, volume 2, pages 1382 – 1384 vol.2, 1998.

[22] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550 –1560, oct 1990.

[23] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: foundations of research. chapter Learning representations by back-propagating errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[24] Emiliano Aldabas-Rubira. Introducción al reconocimiento de patrones mediante redes neuronales. pages 1 –3, 2002. UPC-Campus Terrassa-DEE-EUETIT Colom, 1 08222 Terrassa Barcelona, España. `http://www.jcee.upc.es/JCEE2002/Aldabas.pdf`.

[25] Cognex. Sistemas de visión artificial: Perspectiva general de insight. At the website `http://www.cognex.com/ProductsServices/VisionSystems/default.aspx?id=46&langtype=1034&locale=mx`, 2012. [Online; accessed 25-November-2012].

[26] Labview. Labview add-on: Vision/image processing. At the website `http://www.ni.com/analysis/lvaddon_vision.htm`, 2012. [Online; accessed 25-November-2012].

[27] Cognex. Visionpro, software de visión de cognex. At the website `http://www.cognex.com/ProductsServices/VisionSoftware/default.aspx?id=2394&langtype=1034&locale=mx`, 2012. [Online; accessed 25-November-2012].

[28] Wikipedia. C (programming language) — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/C_(programming_language`, November 2012. [Online; accessed 25-November-2012].

[29] Wikipedia. Lenna — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/Lenna`, November 2012. [Online; accessed 24-November-2012].

[30] Wikipedia. Canny edge detector — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/Canny_edge_detector`, December 2012. [Online; accessed 20-December-2012].

[31] Wikipedia. Prewitt operator — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/Prewitt`, November 2012. [Online; accessed 20-December-2012].

[32] Wikipedia. Roberts cross — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/Roberts_Cross`, September 2011. [Online; accessed 20-December-2012].

[33] R.M.O. Cruz, G.D.C. Cavalcanti, and T.I. Ren. Handwritten digit recognition using multiple feature extraction techniques and classifier ensemble. *17th International Conference on Systems, Signals and Image Processing*, 2010.

[34] Wikipedia. Sobel operator — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/Sobel_operator`, December 2012. [Online; accessed 20-December-2012].

[35] Wikipedia. Gradient — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/wiki/Gradient`, December 2012. [Online; accessed 22-December-2012].

[36] Wikipedia. Roberts cross — wikipedia, the free encyclopedia. At the website `http://en.wikipedia.org/w/index.php?title=`

Roberts_Cross&oldid=541152523, February 2013. [Online; accessed 23-April-2013].

[37] Robert Schwartz. Method and a circuit for determining a contour in an image. Patent US4433912. At the website http://www.google.com/patents/US4433912, February 1984. [Online; accessed 17-January-2012].

[38] T. Baidyk, E. Kussul, O. Makeyev, A. Caballero, L. Ruiz, G. Carrera, and G. Velasco. Flat image recognition in the process of microdevice assembly. *Pattern Recognition Letters*, 25(1):107 – 118, jan 2004.

[39] Ernst Kussul and Tatiana Baidyk. Improved method of handwritten digit recognition tested on mnist database. *Image and Vision Computing*, 22(12):971 – 981, 2004. Proceedings from the 15th International Conference on Vision Interface.

[40] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

[41] Wikipedia. Segmentación (procesamiento de imágenes) — wikipedia, la enciclopedia libre. At the website http://es.wikipedia.org/w/index.php?title=Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)&oldid=66275803, May 2013. [Online; accessed 12-May-2013].