23

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ESTUDIO E IMPLEMENTACION DE HERRAMIENTAS PARA EL ANALISIS AUTOMATICO DE DOCUMENTOS GRAFICOS

FALLA DE ORIGEN

T E S I S

PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

JOSE LUIS CARBAJAL HERNANDEZ
ROCIO HERNANDEZ ELENES

DIRECTOR : DR. ROGELIO ALCANTARA SILVA

MEXICO, D. F. 1995

José Luis Carbajal Hernández.

A mis padres, Yolanda y Luis, que me han dado todo lo que un hijo puede esperar: Amor, confianza y libertad.

A Rocío, que se aventuró a compartir esta experiencia con este loco.

A Héctor, mi hermano, que tiene su manera singular de quererme.

A Juan Carlos Valadez, cuya amistad no entiende de tiempos ni distancias.

A todos mis familiares, por su cariño y su fe en mí.

A mis verdaderos amigos, que siempre han creído que soy capaz de vencer cualquier obstáculo, tanto, que me han convencido de ello.

A los "Toliros 19" por hacer de estos años de universidad una experiencia inolvidable.

A todos los novelistas y cineastas, que me han dado los vicios más maravillosos del mundo.

A esa fuerza (llámese Dios, Suerte, Amor, Destino) que siempre ha estado de mi lado.

Gracias

Rocío Hernández Elenes.

Quiero agradecer en primer lugar a Jorge Hernández y Guadalupe Elenes, mis padres, y a Ma. de Jesús Elenes, mi segunda madre, cuyo trabajo, esfuerzo y dedicación se han sembrado en mi corazón; su lucha de superación es el mejor ejemplo que ahora conservo como la mejor herencia.

Mil Gracias

Igualmente, por su invaluable compañía y ejemplo, agradezco a mis hermanos: Ma. Teresa, Marcelita, Jorge y Javier, a quienes dedico este trabajo como respuesta a su gran apoyo.

Mil Gracias

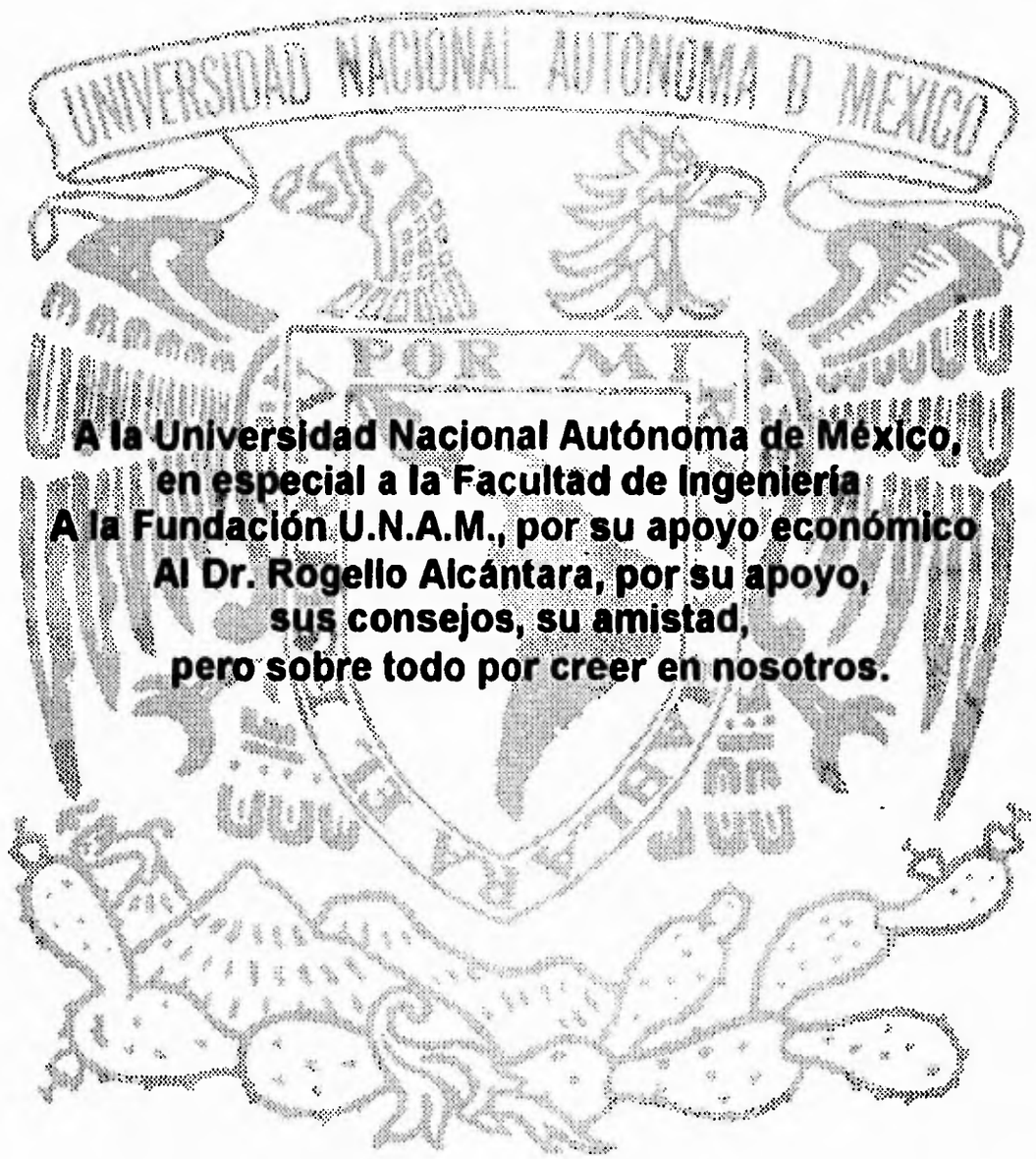
A mi mejor y gran amigo José Luis Carbajal, por su apoyo.

Mil Gracias

A mis familiares que siempre han estado conmigo en los buenos y malos momentos, y a mis familiares que a kilometros de distancia, también creen en mi. *Gracias*

Muy especialmente agradezco a mi cuñado Roger, quien es una gran persona, excelente amigo y un gran ejemplo.

Mil Gracias.



A la Universidad Nacional Autónoma de México,
en especial a la Facultad de Ingeniería
A la Fundación U.N.A.M., por su apoyo económico
Al Dr. Rogelio Alcántara, por su apoyo,
sus consejos, su amistad,
pero sobre todo por creer en nosotros.

Aquél que quiera construir torres altas, deberá permanecer largo tiempo en los fundamentos.

Anton Bruckner
Compositor austriaco

La pantalla de la computadora es una ventana por la que se ve el mundo. El desafío es hacer que parezca real.

Ivan Sutherland
Pionero de la realidad virtual, 1965

Nunca vayas siempre por el camino trazado porque sólo conduce hacia donde otros han ido ya.

Alexandre Graham Bell
Inventor estadounidense

El secreto de la felicidad no es hacer siempre lo que se quiere sino querer siempre lo que se hace.

Leo Nikolaevich
Escritor y Reformador social ruso

"... Pedro Gaviota les vio de pronto tal y como eran realmente, sólo por un momento, y más que gustarle, amó aquello que vio. ¿ No hay límites, Juan ?, pensó, y sonrió. Su carrera hacia el aprendizaje había empezado."

Richard Bach
Juan Salvador Gaviota

ESTUDIO E IMPLEMENTACION DE HERRAMIENTAS PARA EL ANALISIS AUTOMATICO DE DOCUMENTOS GRAFICOS

INDICE

1. INTRODUCCION

La necesidad en el uso de documentos por computadora surge a partir de los problemas de transmisión y almacenamiento de información. La finalidad es poder manipular los documentos y lograr mantener la información en una forma más eficiente e integrada. De esta manera, los documentos pueden ser analizados tanto por las computadoras como por las personas, y pueden ser transmitidos a una mayor velocidad. Esta propuesta se ha estado revolucionando durante las últimas dos décadas, debido principalmente a que los equipos de cómputo de que se puede disponer nos ofrecen una gran velocidad de procesamiento y un bajo costo de hardware [15].

Hoy en día muchos de estos documentos son elaborados directamente en la computadora, mediante procesadores de texto, sistemas de dibujo, bases de datos, diseño asistido por computadora, etc. Sin embargo, se siguen manejando una gran cantidad de documentos en papel, y los sistemas automáticos mencionados no pueden accesar a ellos. Si, dentro de una empresa o institución, se desea tener el control de toda la información de forma automática, se vuelve necesario que toda la documentación se almacene en un formato común al de los sistemas computarizados que la manejan, incluyendo toda la información que, por necesidad o por antigüedad, se tiene en papel.

Un documento en papel se introduce a la computadora por medios ópticos (un scanner o una cámara), y el resultado es un mapa de pixeles que conforman una copia del documento original. Sin embargo, estos pixeles sólo forman una imagen y, por si misma, no puede proporcionar a la computadora la información que necesitamos. El problema consiste en transformar ese conjunto de puntos negros y blancos en texto, diagramas, fotos, firmas, o cualquier información que contenga el documento. A este enfoque se le denomina Análisis Automático de Documentos.

Son muchos los sistemas, y las aplicaciones específicas, que se han ido desarrollando para el análisis automático de documentos. Ejemplos de estas aplicaciones los podemos encontrar en el reconocimiento y lectura de direcciones para el ordenamiento automático de sobres de correo [50], el reconocimiento de logos [51], el análisis automático de dibujos de ingeniería [1], y muchas otras aplicaciones más [15]. Para realizar la transformación antes mencionada, un sistema automático debe, primero, separar y reconocer los distintos tipos de información (en el caso de los

sobres, por ejemplo, separar las zonas de la imagen que corresponden al destinatario, el remitente y el timbre o sello), luego reconocer los conjuntos de pixeles como elementos más utiles para su interpretación (esto es, reconocer todas las letras y signos de puntuación), y después, todos estos elementos en conjunto pueden llevarnos a un nivel mayor de reconocimiento (esto es, reconocer el nombre de la persona y su dirección, para saber en que orden debe guardarse el sobre).

Aunque distintos en su objetivo final, todos los sistemas siguen una serie de pasos definidos para alcanzar el nivel de reconocimiento deseado. Esta arquitectura, más o menos común, podemos describirla de la siguiente manera:

- Eliminación de ruido o elementos no deseados
- Segmentación o separación de los elementos entre sí
- Reconocimiento de componentes básicos (letras o líneas)
- Reconocimiento de componentes mayores (texto o figuras)
- Interpretación de la información obtenida

No todos los sistemas que se están desarrollando o que se han desarrollado alcanzan todos los niveles de esta arquitectura, sin embargo, el objetivo es diseñar sistemas que analicen una mayor variedad de documentos, con una mínima cantidad de errores, y con el mayor nivel de reconocimiento posible.

El desarrollo de estos sistemas se puede dividir en dos partes: el desarrollo y mejoramiento de las herramientas que se emplean, y la creación de estructuras que permitan a estas herramientas interactuar entre sí de manera efectiva. Las herramientas son todos los procesos que se aplican a la imagen inicial para transformarla: la segmentación, la eliminación de ruido, el reconocimiento de caracteres, de líneas, de símbolos, etc. Y mientras más efectivas sean individualmente, más efectivos resultarán los sistemas que se apoyan en ellas.

Como ya se mencionó, el desarrollo de estos sistemas y herramientas es muy reciente, lo cual indica que actualmente en todo el mundo se sigue trabajando en esta área, y con la llegada de los sistemas interactivos, la multimedia, las redes de servicios integrados, etc. la necesidad de tener toda la información en formatos compatibles y la interpretación automática de esta información, alcanzarán un gran auge. El presente trabajo pretende contribuir con este desarrollo.

En primer lugar una síntesis, acerca de la arquitectura general de los sistemas que realizan análisis automático de documentos, se muestra en el capítulo 2, para

después particularizar en aquéllos que trabajan con documentos gráficos (capítulo 3). Con estas bases, presentamos un estudio de dos de las principales herramientas utilizadas en estos sistemas:

La segmentación, dentro del análisis de documentos, es el primer paso antes de cualquier otra herramienta de reconocimiento, ya que sirve para separar los diferentes tipos de información: texto, gráficos e imágenes. Proponemos un nuevo esquema de segmentación (capítulo 4.4), basado en un algoritmo original de localización de componentes conectados. Además de una síntesis de las características de los principales métodos de segmentación (capítulo 4).

La vectorización es la herramienta básica en los sistemas de análisis de documentos gráficos (e.g. dibujos de ingeniería), ya que reconoce líneas dentro de una imagen y las codifica en un formato vectorial. En el capítulo 5 se describen algunos de los métodos más comunes para la vectorización. Este formato es muy utilizado en los sistemas de diseño asistido por computadora (CAD). Se propone y desarrolla un nuevo esquema de vectorización (capítulo 5.4), que utiliza las ventajas del algoritmo de la transformada de Hough para el reconocimiento de líneas.

Las características que presentaron estas herramientas implementadas pueden resumirse en lo siguiente:

El esquema de segmentación propuesto sobresale por su sencillez, lo cual lo hace adaptable a cualquier aplicación. Retoma un tipo de segmentación que no depende de patrones de orden dentro de las imágenes, que es la segmentación por componentes conectados. Y presenta un método, diferente al que se utiliza en la literatura [29], para encontrar dichos componentes.

Este método tiene la característica principal de que no requiere de una programación compleja para implementarse, por lo que se puede utilizar para muchas de las aplicaciones que se tienen para los componentes conectados, y no difiere mucho en su desempeño de otros algoritmos.

Los resultados de la segmentación fueron sumamente satisfactorios, el proceso se realizó correctamente en todos las imágenes de prueba utilizadas, con pequeños errores propios de los métodos de segmentación por componentes conectados (capítulo 4).

El algoritmo de vectorización propuesto retoma y combina dos de los métodos más sencillos y efectivos (capítulo 5), la transformada de Hough y la transformada adaptativa de Hough, para el reconocimiento de líneas, y la idea de la clasificación de pixeles terminales (pixeles que identifican las orillas y esquinas de los segmentos de líneas) para la localización de los segmentos y su posterior codificación como vectores de línea.

La transformada de Hough es tan sencilla o compleja como se le quiera programar, y tiene la ventaja de que puede identificar patrones aún con la presencia de ruido o defectos en la captura. Por otro lado, la transformada adaptativa de Hough presenta una aproximación más exacta para la localización de los parámetros de las líneas y un ahorro considerable en la capacidad de memoria requerida. Nuestro esquema presenta una combinación de ambos métodos. Ademas proponemos una estrategia de búsqueda que comience con el reconocimiento de líneas verticales y horizontales, cuyos parámetros son más sencillos de identificar, y son más abundantes dentro de las imágenes gráficas, lo cual evita errores de identificación y deja las líneas inclinadas, que son más difíciles de identificar, aisladas. De esta forma se obtuvieron resultados muy aproximados a los originales con las imágenes de prueba.

El estudio e implementación de estas herramientas constituyen una base sólida para continuar con el desarrollo de nuevas herramientas y estructuras, que den forma a un sistema completo para el análisis automático de documentos gráficos.

2. GENERALIDADES DEL ANALISIS AUTOMATICO DE DOCUMENTOS

El uso de sistemas de manejo de documentos por computadora se ha popularizado en años recientes [1, 15, 46]. Procesadores de palabras, sistemas de publicidad electrónicos, y sistemas de diseño asistido por computadora se pueden encontrar en casi cualquier organización. La instalación y crecimiento de estos sistemas ha sido tan rápida que las empresas enfrentan ahora el problema de cómo manejar todo los documentos ya existentes en papel, y que estos sistemas electrónicos no pueden accesar.

La única forma de hacerlo es mediante la captura por medios ópticos de los documentos. El resultado obtenido es una imágen en formato raster del documento en cuestión, pero no se cuenta con información sobre los símbolos primitivos (palabras, texto, líneas, cantidades, imágenes, etc.) que lo conforman. Si se desea que estos documentos puedan accesarse por medio de los sistemas antes mencionados, todos los símbolos, ya sean líneas, curvas, caracteres, letras, números, etc., deben identificarse.

2.1 El Análisis Automático de Documentos

El análisis de documentos [15] es la interpretación automática de información en papel, introducida a la computadora por medios ópticos, con el objetivo de reconocer textos y gráficos como lo haría un ser humano.

Para el análisis de los documentos, se debe seguir una serie de pasos entre los que se encuentran: la captura de datos, un preproceso a nivel pixel, análisis de figuras, y un tratamiento de identificación de los diferentes tipos de información. Estos pasos se ilustran en la figura 2.1, y se detallan en los siguientes subcapítulos.

2.1.1 Captura

La captura de datos se efectúa desde un documento en papel, que generalmente es scaneado utilizando scanners ópticos. La imagen obtenida es almacenada en un archivo formado por elementos de imagen, llamados pixeles, que son muestreados en un patrón a través de la imagen.

Cada pixel tiene un valor propio que puede ser '0' (apagado) o '1' (encendido) para imágenes binarias, un valor de 0 a 255 para imágenes de escala de gris, o un valor de 3 canales con valores de 0 a 255 en una imagen de color. Una resolución

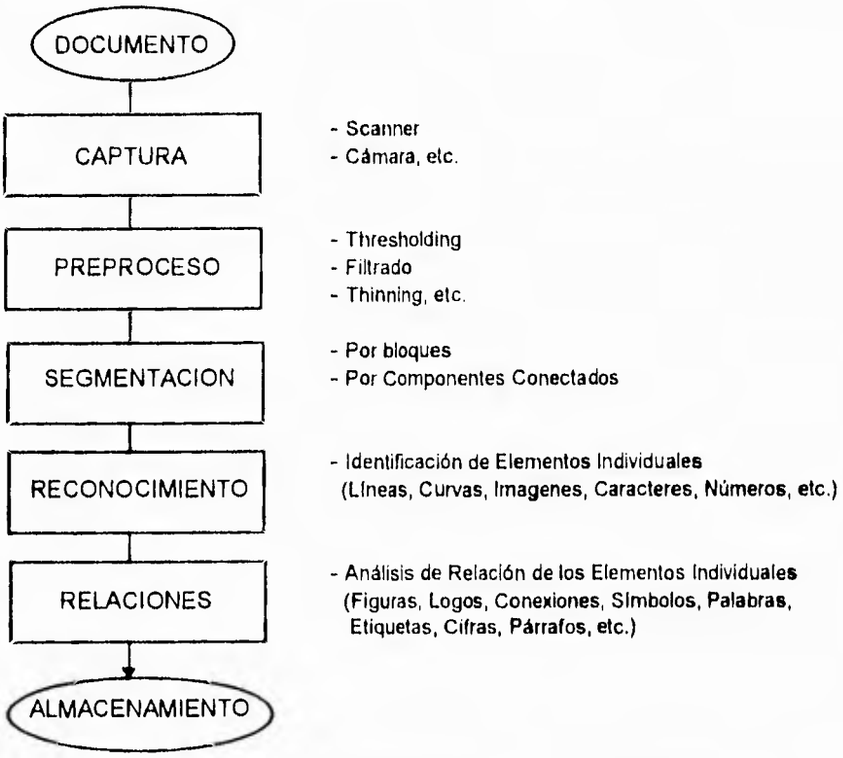


Figura 2.1 Pasos del análisis automático de documentos.

típica de muestreo es de 300 dpi (puntos por pulgada). Una página de 8.5 por 11 pulgadas puede generar una imagen de 2550 por 3300 pixeles.

2.1.2 Preproceso

El primer proceso que se le aplica a la imagen es el llamado preproceso a nivel pixel, el cual puede incluir diversas operaciones. Por ejemplo, se identifican los espacios que físicamente contienen información de aquellos espacios en blanco que no debe ser considerados como tales. El proceso separa automáticamente el fondo de la imagen, sin información, de los caracteres y gráficos que sí la contienen. Otros procesos se encargan de reducir ruido, que generalmente es producido por una mala transmisión, basuras por fotocopiado del documento, errores por degradación debido al desgaste del documento, iluminación incorrecta en el documento, ruido causado por fluctuaciones de iluminación, errores por limitaciones del ancho de banda del sistema scaneador, etc. También se pueden hacer mejoras de la imagen, reconstruyendo partes de la imágenes que hayan sido perdidas. Todos estos preprocesos preparan a la imagen para su correcto análisis y preveen errores de identificación de formas.

2.1.3 Segmentación

El siguiente paso consiste en realizar una segmentación o separación de la información, separar texto y localizar caracteres, columnas, palabras, párrafos, títulos, etc.; gráficamente separar símbolos de componentes gráficos formando rectángulos, círculos, líneas, etc.; y reconocer regiones que corresponden a imágenes, entendiendose en este sentido, fotografías, pinturas, dibujos, etc. Este procedimiento es muy importante ya que los distintos tipos de información requieren de distintos tipos de análisis, como se mencionará a continuación.

2.1.4 Reconocimiento

El proceso de mediano nivel identifica rasgos dentro de la imagen. En una imagen con texto, los rasgos globales describen a cada página y consiste en el "sesgado" (la inclinación en la que una página ha sido scaneada), líneas largas, espaciado de líneas, etc. Así como también rasgos de caracteres individuales, es decir, identificar cada uno de ellos.

En los gráficos, los rasgos globales describen el sesgo de una página, el ancho de líneas, rango de curvatura, mínimo largo de línea, etc. Los rasgos locales, que son más específicos, describen cada esquina, curva, línea recta así como figuras más comunes como son rectángulos, círculos, y otras figuras geométricas.

Para las imágenes, generalmente, se aplica un algoritmo de compresion para almacenarla, y/o algún algoritmo de reconocimiento de patrones, si se desea identificar la información que contiene.

2.1.5 Relaciones

El siguiente paso se refiere al reconocimiento de texto y de gráficos. Este está dedicado al reconocimiento y a la descripción. Los componentes son asignados a niveles semánticos y los documentos enteros son descritos como completos. En este nivel, el conocimiento es usado más extensivamente. El resultado es una descripción del documento tal y como el ojo humano quisiera verlo. Para los documentos formados por texto, no nos referimos a grupos de pixeles, ni a bloques blancos y negros, sino a títulos, subtítulos, cuerpo de texto, notas de pie de página, etc., y dependiendo de las características de estos grupos de datos se les puede clasificar como el título de una página o un papel, una tabla con contenido de un diario, una forma de negocios, una parte de una carta de correo, etc. Para las imágenes gráficas, en un diagrama eléctrico

por ejemplo, nosotros no nos referimos a líneas pegadas con círculos o rectángulos, o cualquier otra figura, sino a conexiones de compuertas AND, OR, a transistores, resistencias, etc., así, los componentes y sus conexiones describen un circuito particular, que tiene un propósito especial dentro de su dominio. Con esta descripción semántica es más eficiente el almacenamiento, y algo muy importante es que se tiene la posibilidad de añadir información así como realizar alguna modificación en particular a un componente del dibujo [15].

2.2 Captura de Documentos

Para hacer un estudio del análisis automático de documentos e imágenes, es necesario primero conocer la forma en que estos son llevados y almacenados en la computadora. A continuación se presenta un resúmen acerca de la captura y formatos de almacenamiento de documentos e imágenes.

2.2.1 Scaneo

La captura de datos a partir de los documentos realizada por scaneo óptico produce un archivo de elementos de la imagen, pixeles, que es la entrada "en bruto" para el análisis de documentos.

El documento es iluminado con una luz fluorescente, la luz reflejada pasa por un lente hacia un fotosensor llamado dispositivo de carga acoplada (CCD). El CCD es un arreglo lineal de elementos fotoeléctricos llamados celdas o elementos, y cuenta con alrededor de 2600 elementos. Cada elemento produce una señal eléctrica proporcional a la cantidad de luz que se refleja sobre él. Un punto negro en la imagen refleja menos luz y resulta en una señal de voltaje bajo.

Los voltajes de salida de los CCDs son convertidos a valores digitales y enviados a la computadora. En un scanner simple de dos tonos, una umbral de voltaje se selecciona, aproximadamente a la mitad del voltaje entre el blanco y el negro, como punto de decisión. Voltajes bajo el umbral se consideran negros (0) y aquellos arriba del umbral blancos (1). En un scanner de escalas de gris, un convertidor analógico-digital (ADC) convierte el voltaje de salida de cada elemento en un patrón de bits apropiados que representan la intensidad de la luz reflejada, dentro del número de bits por pixel soportados por el scanner. La mayoría de los scanners tienen CCDs que diferencian 256 tonalidades de gris, 8 bits por pixel.

Cada lectura de todos los elementos CCD representa un renglón completo con ancho de un pixel, sucesivos renglones son entonces leídos hasta que el documento entero o la imagen han sido capturados. El CCD puede ser avanzado al siguiente

renglón de pixeles por uno de los siguientes métodos: Moviendo manualmente el scanner (de tipo manual), automáticamente moviendo la cabeza del scanner (de tipo flatbed), automáticamente avanzando la página por la cabeza del scanner (de alimentación), o automáticamente moviendo un espejo reflejante (scanner overhead).

La resolución de un scanner es medida en pixeles por pulgada (dots per inch dpi). Este es un número fijo basado en el número de celdas en el arreglo y el total de área scaneada. Por ejemplo, un arreglo con 2590 celdas cubriendo 8.5 pulgadas, da una resolución de 300 dpi. Mientras menos sea el área, mejor la resolución.

Muchos scanners pueden producir resoluciones menores mediante el software apropiado. Algunos ofrecen incrementos en la resolución de un eje, tomando lecturas más seguido conforme la barra de luz se mueve. La interpolación es otro método de mejorar matemáticamente la resolución. Un algoritmo es usado para calcular el valor de un pixel colocado entre dos que han sido actualmente scaneados. Si el primero del par tiene una lectura de 8 y el otro una de 12, la fórmula colocará el pixel interpolado linealmente en un valor de 10 [11, 34].

2.2.2 Formatos de Imagen

Existen cuatro formas de representar imágenes, estas son:
- Gráficos Raster (llamados también bitmaps o pixel-maps).
- Gráficos vector.
- Gráficos superficie.
- Gráficos de volúmen.

Una imagen se encuentra en formato raster cuando ha sido particionada en celdas o puntos. La brillantés de cada punto (pixel), estén encendidos, apagados o sean de algún color, es registrada individualmente. La posición de cada punto, en un campo bitmap, determina cual punto representa cada pixel. En otras palabras, los puntos (bits) mapean en la imagen, de aquí el nombre bitmap. Estos formatos trabajan muy bien con imágenes con variaciones complejas de color, tales como fotografías, pinturas e imágenes digitalizadas provenientes de video. Imágenes que generan formatos raster, como son despliegues en la pantalla de la computadora, conviene almacenarlas en el mismo formato [9, 41].

En el formato vector, la imagen se describe como una serie de líneas y formas. Aunque la palabra vector se refiere solamente a líneas, dentro de esta representación se incluyen formas tales como cuadros y círculos. Cuando se examina un archivo de tipo vector, se puede encontrar una estructura semejante a un lenguaje de programación, conteniendo por ejemplo, comandos y datos en ASCII. Los formatos

vectoriales se emplean comúnmente en dibujos de arte e ingeniería formados por líneas como en CAD [9, 41].

La aproximación basada en objetos de los gráficos vector, ha sido adaptada para gráficos en tres dimensiones. Esta aproximación es llamada gráficos de superficie, que combina la tecnología raster para el despliegue y una aproximación vectorial para la representación, manipulación y construcción de escenas en tres dimensiones. Los gráficos de superficie generan únicamente las superificies de objetos sólidos tridimensionales vistos desde una dirección dada. Los gráficos de superficie recuerdan mucho a los gráficos vectoriales en muchos sentidos. Ambos representan las imágenes como un conjunto de primitivos geométricos guardados en una lista de despliegue. En las superficies, estos primitivos son transformados, mapeados a las coordenadas de la pantalla, y convertidos, mediante algoritmos de conversión scan, en un conjunto discreto de pixeles los cuales son guardados en un bufer. Este proceso de digitalización es llamado rasterización o pixelización. Cualquier cambio en un objeto, en los parámetros de visión, o en los parámetros de sombreado, requiere que el sistema repita estas operaciones y reprocese la descripción completa de la escena [39].

Los gráficos de volúmen, en lugar de una lista de objetos geométricos, usan un bufer de volúmen tridimensional, como medio para la representación y manipulación de escenas en tres dimensiones. Los datos volumétricos son un conjunto representado como una celda regular discreta tridimensional (un raster en tres dimensiones) de voxels (elementos de volúmen) y que son comúnmente almacenados en un bufer de volúmen (cubic frame buffer). Como unidad de volúmen, el voxel es la contraparte tridimensional del pixel, por lo tanto, los gráficos de volúmen son la contraparte tridimensional de los gráficos raster. Su uso, actualmente, incluye sistemas de visualización tales como simuladores y realidad virtual [39].

Los sistemas que realizan análisis de documentos, emplean los formatos raster y vector, por eso es necesario profundizar en ellos, como se verá en el capítulo 5.

3. DESCRIPCION DE LOS SISTEMAS DE ANALISIS DE DOCUMENTOS GRAFICOS

Dentro de los muchos tipos de documentos, vamos a describir la "arquitectura" de los sistemas que analizan documentos gráficos. Se conocen como documentos gráficos aquellos que están compuestos principalmente por líneas (dibujos de ingeniería, mapas, planos, diagramas, etc.).

Cabe señalar que existen otros documentos que también requieren de la vectorización y otros procedimientos que conforman los sistemas de análisis de documentos gráficos, tal es el caso del estudio de reconocimiento de logos [51] o la visión de las máquinas [11, 35] , por lo que se puede observar que el uso de estas herramientas no está limitado a un tipo de documentos o un sólo campo del procesamiento de imágenes.

3.1 Dibujos de Ingeniería

Un dibujo de ingeniería típico está formado por polígonos, círculos, y otros objetos y líneas que los interconectan. Estas entidades tienen varios atributos asociados, como son: grueso de las líneas, detalles de rellenado de los polígonos, etc. Frecuentemente estas entidades están encimadas unas con otras, lo cual oscurece algunas líneas, y los gráficos están comúnmente acompañados de cadenas de texto. Esta clase de dibujos son digitalizados con una resolución de 12 pixeles por milímetro cuadrado, generando una imagen binaria de 2048x2048 pixeles [3].

El propósito principal de los dibujos de ingeniería es el de comunicar información geométrica. Diversas organizaciones nacionales de estándares (como ANSI Y-14.2) han desarrollado estándares para estos dibujos, incluyendo anchos de línea, formatos y anotaciones. Los dibujos también cuentan con un rango de tamaños: desde tamaño A (8 y media x 11 pulgadas) hasta tamaño J (Un pie de largo por 36 o 42 pulgadas de ancho).

Un importante punto en los dibujos de ingeniería son las reglas de diseño, las cuales definen condiciones que deben ser satisfechas en la composición del dibujo. Por ejemplo, los tipos de válvulas que deben usarse para diferentes tubos y fluidos, o el espacio entre transistores en la capa de un chip electrónico. Una importante capacidad del Dibujo Asistido por Computadora es el chequeo automático de las reglas de diseño.

Además, es útil distinguir entre dos tipos de información necesariamente introducidos en los dibujos de ingeniería: Los datos del modelo, que consisten en

información isomórfica sobre el objeto que está siendo diseñado, es decir, las líneas de forma en las partes del dibujo y las relaciones topológicas entre unidades semánticas en dibujos esquemáticos de red. Y los datos de presentación, que consisten en convenciones o reglas relacionadas en la forma en que se presenta la información, es decir, líneas punteadas, líneas que indican dimensiones, flechas, puntas, etc.

Estos dos tipos de información no se distinguen cuando están en el papel, pero un sistema de cómputo que maneja información de ingeniería debe poder distinguir entre ellos [38].

Una secuencia común de pasos para el análisis de dibujos de ingeniería y documentos gráficos es, en primer lugar, el preprocesamiento de las imágenes para eliminar ruido y distorsiones. Después, un paso inicial de segmentación, generalmente aplicado a las imágenes con textos y gráficos (caso de los dibujos de ingeniería), es la separación de textos y gráficos. Una vez que el texto es segmentado, se extraen componentes típicos de la imagen, como son: líneas rectas, curvas, y regiones rellenadas. A continuación, se aplican técnicas de reconocimiento de patrones para determinar la similitud entre el componente extraído y un componente conocido, desde el punto de vista geométrico y estadístico; además de técnicas sintácticas de reconocimiento de patrones para completar esta misma tarea usando reglas (ej. gramáticas) en el contexto y secuencia de los componentes. Después de este procesamiento de mediano nivel, estos componentes son ensamblados en entidades con algún significado, el cual es dependiente del dominio particular de la operación. Las técnicas usadas para esto incluyen comparación de patrones, hipótesis y verificación, y/o métodos basados en conocimiento. La interpretación semántica de los elementos gráficos puede ser diferente, dependiendo del dominio, por lo tanto una línea puede ser un camino en un mapa, o una conexión eléctrica en un diagrama de circuitos. Métodos como estos son llamados de procesamiento de alto nivel, y en ocasiones son descritos como técnicas de inteligencia artificial [15].

Para obtener un archivo descriptivo del mayor nivel de reconocimiento posible, con la mínima interacción de operaciones, las siguientes operaciones deben realizarse sobre la imagen [3]:

1. Preprocesamiento.
2. Segmentación de cadenas de texto de los gráficos.
3. Reconocimiento y vectorización de líneas.
4. Identificación de primitivos gráficos.
5. Reconocimiento de símbolos, componentes y su interacción.

En las siguientes secciones se describe la estructura de los sistemas que realizan un análisis de este tipo de documentos, basados en el diagrama de la figura 3.1:

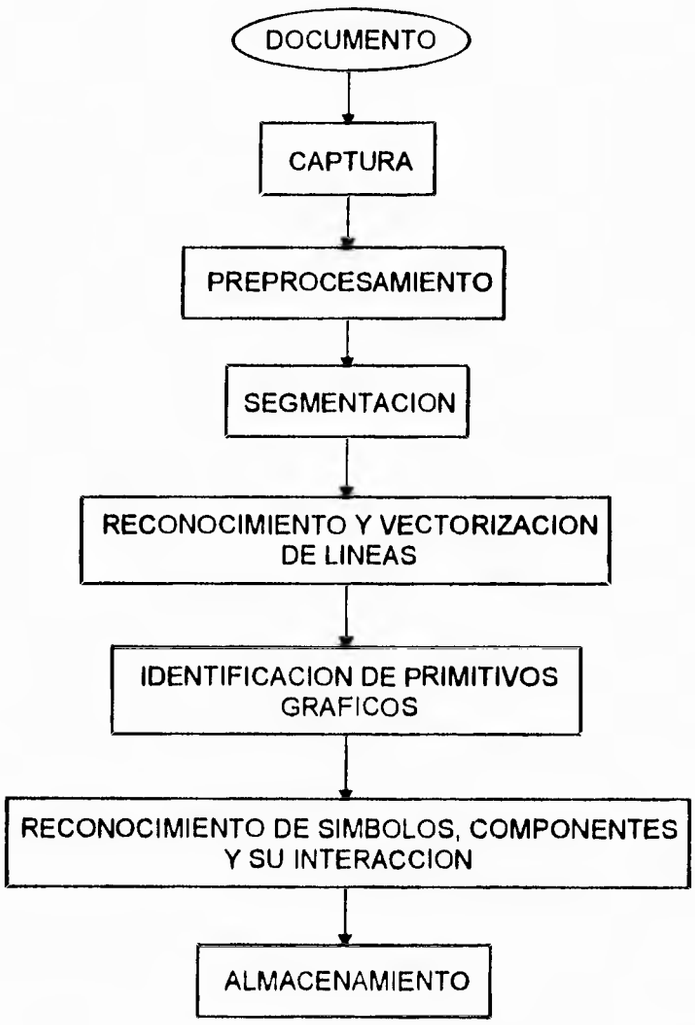


Figura 3.1 Estructura de Análisis Automático de Documento Gráficos

3.2 Preprocesamiento

El primer paso para el análisis de estos documentos es el de realizar un procesamiento sobre esta imagen, que la prepare para el análisis posterior. Algunos de los procesos más comunes para este objetivo se describen brevemente.

3.2.1 Binarización (Thresholding)

Las imágenes que se están tratando contienen gráficos y texto, que son los que nos interesa analizar (foreground), y un fondo de color uniforme con el cual contrastan (background). Sin embargo, los datos obtenidos tienden a tener más de dos niveles de intensidad, en realidad, contienen todo un rango de intensidades. El objetivo de la binarización de las imágenes es marcar aquellos pixeles que en verdad pertenecen al foreground con una intensidad '1', y todas las regiones que sean fondo con una intensidad '0' [15].

Para aquellos documentos con un buen contraste entre los componentes y el fondo, los scanners binarios combinan la digitalización con un thresholding para obtener una imagen binaria. Pero muchos de los documentos vienen en colores, fotocopias, o con zonas no muy bien definidas, por lo que el mejor umbral (threshold) para realizar la binarización no es el mismo.

La figura 3.3 muestra una imagen con un umbral adecuado. En la figura 3.2 vemos una imagen en donde el umbral es "muy oscuro", y muchos de los rasgos quedan demasiado acentuados, se pierde definición en los contornos y aparecen puntos negros de más (ruido). La figura 3.4, por el contrario, el umbral es "muy claro", y la imagen pierde detalles, aparecen huecos blancos, etc.

Por lo tanto la elección del umbral es muy importante y depende del tipo de imagen y de los dispositivos de captura.

La mejor alternativa es comenzar desde una imagen en escala de grises (0-255 valores de intensidad) para la digitalización, y después usar métodos automáticos de thresholding que determinen la mejor manera de binarizar la imagen.

El Thresholding [11, 13, 15] consiste en el establecimiento de un umbral para diferenciar componentes útiles del fondo. Esto es, si la intensidad de un punto es menor que cierto umbral se considera negro, de lo contrario blanco. Este método funciona muy bien para imágenes cuya distribución de intensidades es uniforme, y se denomina Thresholding Global.

Figura 3.2 Umbral Oscuro

Figura 3.3 Umbral Adecuado

Figura 3.4 Umbral Claro

El umbral global se puede seleccionar mediante un histograma de ocurrencias de niveles de gris de la intensidad de una imagen. Un histograma típico indica la presencia de una gran cantidad de pixeles blancos (el fondo) y una pequeña cantidad de pixeles negros (los gráficos). Pixeles con niveles de gris intermedios son el ruido del papel o pixeles grises cerca de las orillas entre pixeles blancos y negros. El umbral a seleccionarse puede ser el nivel de gris de la mínima cresta del histograma, o bien el promedio del nivel de gris entre las crestas del negro y el blanco.

Sin embargo, el hecho de utilizar un umbral fijo para todo un documento no resuelve los problemas de los documentos de baja calidad, con sombras, mal iluminados, etc. y su reproducción resulta insatisfactoria.

Para este tipo de situaciones existe el thresholding adaptativo. El umbral debe ser aplicado partiendo la imagen en ventanas que no se encimen, cada una teniendo su propio umbral. Dentro de cada ventana, se calcula la distribución de amplitudes para determinar el umbral a utilizarse. El tamaño de la ventana debe seleccionarse adecuadamente, ya que si son muy pequeñas se tendrá una distribución incierta, y si son muy grandes se reduce la sensibilidad del umbral a los cambios de intensidad pequeños. Típicamente, bloques cuadrados de 5 a 10 líneas scan y pixeles son usadas como ventanas. Uno de los problemas con esta división puede ser un cambio drástico en el umbral de una ventana a otra, lo cual podría provocar distorsiones en curvas que crucen varias ventanas. De tal forma que el umbral de una ventana debe calcularse también en base a los umbrales de las ventanas vecinas.

3.2.2 Reducción de Ruido

Después del proceso de binarización, las imágenes son usualmente pasadas por un filtro para reducir el ruido. Este ruido (llamado ruido Salt-and-pepper, impulse and speckle, o dirt) se presenta en imágenes provenientes de documentos de baja calidad, en forma de pixeles aislados, como regiones apagadas (agujeros) en regiones encendidas, como regiones encendidas (manchas) en regiones apagadas, o como deformaciones de los componentes gráficos y caracteres.

El proceso de reducción de ruido es muy importante debido a que gracias a él se eliminan rasgos extraños que impedirían el reconocimiento exacto de los gráficos y caracteres, reduce el tamaño de los archivos y con ello reduce el tiempo de procesamiento posterior.

Dos familias de procesos para la reducción del ruido son el procesamiento morfológico y el procesamiento celular. Las operaciones básicas de estos procesos son la erosión y la dilatación. Como su nombre lo indica, la erosión consiste en "pelar"

una capa de un sólo pixel de todas las regiones encendidas (ON o '1'). La dilatación, por el contrario, consiste en "agregar" una capa de un pixel a todas estas regiones.

La combinación iterada de estos procesos sirve para eliminar distorsiones en los gráficos y caracteres, esto es, eliminar pixeles de más y corregir huecos con pixeles de menos (figura 3.5). Estas combinaciones son conocidas como apertura (opening) y cerradura (closing). La primera consiste en varias erosiones seguidas de igual número de dilataciones, y la segunda lo contrario [15].

Otro de los procesos de reducción de ruido es el filtrado. Se utilizan filtros especiales que son diseñados para tomar ventaja de las características conocidas de los componentes gráficos y caracteres. En general, los filtros consisten en ventanas que scanean la imagen para encontrar ciertos patrones y corregirlos (islas de pixeles aislados, agujeros, y protuberancias). Estas correcciones se deciden dependiendo de ciertos cálculos realizados sobre los valores de intensidad de todos los vecinos de los pixeles en la ventana [15, 32].

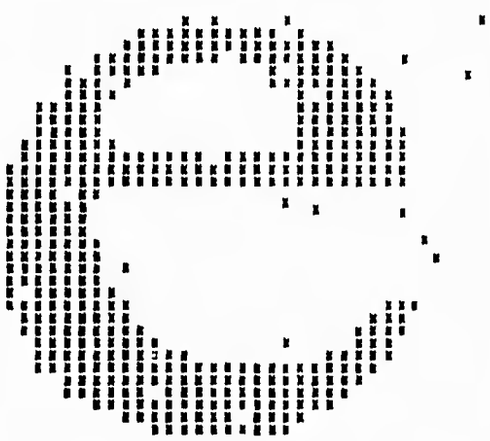


Figura 3.5 Imagen con Ruido

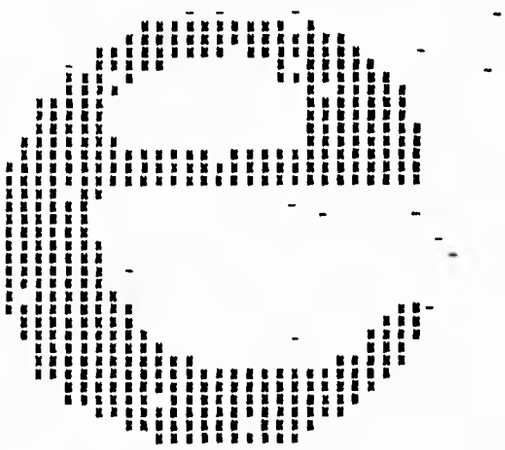


Figura 3.6 Erosión

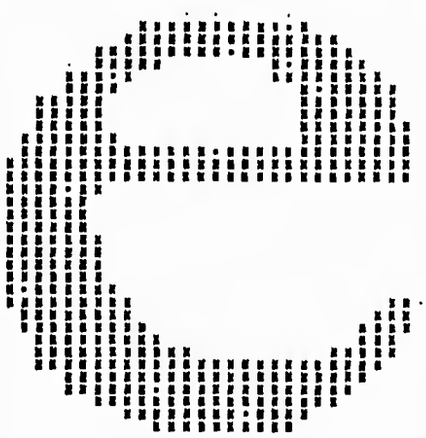


Figura 3.7 Dilatación

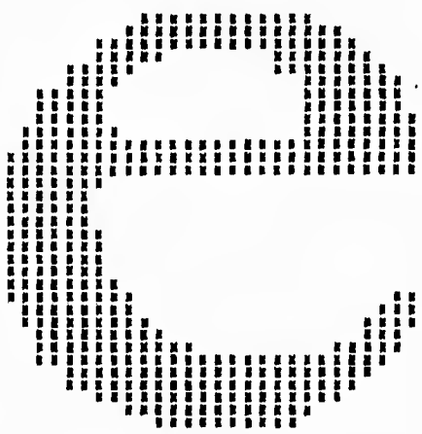


Figura 3.8 Resultado

3.3 Segmentación de Caracteres y Gráficos

En general un documento está formado por distintos tipos de información: información gráfica (imágenes y dibujos con líneas) e información escrita (texto o caracteres). Cada tipo de información debe ser analizada por diferentes métodos: los caracteres deben analizarse por reconocimiento óptico de caracteres (OCR), las imágenes deben comprimirse, y los gráficos o dibujos de líneas se vectorizan. Por lo tanto, antes de realizar cualquiera de estos análisis, se deben identificar las partes del mapa de bits que corresponden a cada tipo de información. A esto se le conoce como segmentación.

Son muy diversas las aproximaciones que se han usado para resolver este problema, y estas varían dependiendo la forma en que se represente el documento: escala de grises, blanco y negro o imagen vectorizada [46].

En los documentos representados en un mapa de bits con escalas de grises, las propiedades de las imágenes obtenidas de diferentes regiones pueden ayudar a segmentarlas. Dos propiedades que han sido consideradas son el espectro de Fourier y los histogramas de escalas de grises.

La más exitosa de estas aproximaciones ha sido el espectro de Fourier [46]. El espectro que se obtiene para regiones con texto difiere del obtenido en regiones con imágenes, incluso, el espectro es diferente para distintos tipos de letras (halftone, impresión laser, o imprenta).

Para imágenes raster binarias (blanco y negro) se han considerado dos técnicas: La segmentación por regiones y segmentación por componentes conectados (también conocidas como: análisis de arreglo de arriba-a-abajo (layout analysis top-down) y análisis de arreglo abajo-a-arriba (layout analysis bottom-up) [15]).

La segmentación por regiones consiste en agrupar toda la información en bloques, los cuales están separados entre sí en el documento por grandes espacios en blanco, y después analizar cada bloque para ver el tipo de información que contiene. La segmentación por componentes conectados agrupa en un rectángulo cada grupo de pixeles conectados entre si. Todos aquellos que sean menores que cierto umbral son considerados como texto y los demás como gráficos [46]. Estos métodos se describen más a detalle en el capítulo 4.

3.4 Reconocimiento y Vectorización de Líneas

Una vez obtenidas las regiones que contienen los gráficos se procede a reconocer y vectorizar cada segmento que los conforman.

Este paso del proceso es conocido en la literatura como: Detección de Líneas (Line Detection) o Vectorización. Aunque este último estrictamente se refiere a la codificación de los segmentos de línea en un conjunto de datos estructurados y compactos (vectores), generalmente esta codificación se realiza o comienza a realizarse inmediatamente después del reconocimiento.

La vectorización convierte ciertos patrones de pixeles de un dibujo a una entidad geométrica básica como un segmento recto de línea, arcos, polígonos, círculos, y posiblemente splines o líneas con un ancho específico, o polígonos rellenados. Mediante tablas se describe la conectividad de los vectores, y vectores que están ligeramente desconectados, pueden ser combinados en ciertos estilos de líneas, como las líneas punteadas [1].

Existen muchos métodos para reconocer líneas dentro de un mapa de bits. Algunos de ellos consisten en un "seguimiento" de los conjuntos de pixeles conectados, analizar las características de este conjunto y decidir de qué tipo de línea se trata. Ejemplos de este tipo de métodos son el Thinning (Adelgazamiento), el Seguimiento de Contornos, la Aproximación Poligonal, etc. Otros métodos tratan de obtener la descripción de las líneas en base a la distribución de los pixeles, ya que estos determinan un comportamiento especial. De esta forma algunos algoritmos obtienen parámetros que definen analíticamente una línea (Transformada de Hough), o bien se ubica a las líneas dependiendo del espacio entre sus elementos (Transformada de Distancia), etc. Estos distintos métodos se describen más a detalle en el capítulo 5.

En general, el problema más grande de la vectorización es el de las intersecciones, particularmente en entidades lineales cortas. Es mucho más fácil reconocer líneas largas y uniformes o curvas, que identificar precisamente la región donde convergen muchas líneas. Debido a las dificultades que envuelven los análisis de intersecciones, los dibujos sin muchos cruces, como mapas de contornos, tienden a ser más fáciles de convertir que los dibujos mecánicos.

Los requerimientos de la vectorización no pueden ser especificados precisamente, excepto por la referencia de un ojo humano que puede juzgar si un patrón raster dado ha sido correctamente convertido. Por esta razón, los algoritmos de vectorización, aunque están basados en principios simples, requieren un considerable refinamiento iterativo, incluyendo el uso de heurística y la sintonización (tunning) de parámetros hasta que se logre una aproximación satisfactoria [1].

3.5 Identificación de Primitivos Gráficos

Para realizar una comparación entre las líneas obtenidas de la vectorización y los modelos de objetos CAD, los segmentos de línea son de muy bajo nivel y no contienen suficiente información técnica [24].

El reconocimiento de primitivos gráficos abarca todos los procesos que tienen el objetivo de unir los segmentos de línea en entidades más complejas y de mayor utilidad, ya sean modelos geométricos (cuadrados, polígonos, curvas, etc.), o elementos dentro de una biblioteca de modelos CAD, con los cuales se van a comparar los segmentos encontrados en el dibujo.

En ocasiones, como ya se vio, este reconocimiento se hace al momento de realizar la vectorización, sin embargo, existen sistemas y algoritmos específicos para este proceso [3, 24].

3.5.1 Líneas Punteadas

Uno de los principales elementos a reconocer dentro de los dibujos de ingeniería, a partir de los segmentos de línea, son las líneas punteadas.

El sistema Celesstin [24] detecta dos tipos de línea punteada, basados en las definiciones estándar (figura 3.9) para este tipo de líneas: líneas punteadas (dot-dashed lines), que representan ejes, y líneas punteadas uniformes (dashed lines), que representan líneas escondidas. Estas líneas son muy importantes ya que simbolizan líneas de simetría sobre las cuales muchos elementos están centrados. El método trabaja en tres pasos:

1. Localización de segmentos de línea candidatos, esto es, segmentos alineados a una distancia apropiada uno del otro. Estas distancias corresponden a los estándares para los dibujos mecánicos, esto es, las distancias son uniformes.
2. Reconocimiento del tipo de línea, que puede ser, línea punteada (un segmento corto y un segmento largo) o línea punteada uniforme (únicamente segmentos cortos).
3. Reemplazo del conjunto de segmentos de línea por una entidad de línea simple con el atributo de línea punteada.

En [3] se sugiere un algoritmo para detectar líneas punteadas cuando la distancia entre los segmentos no es del todo uniforme: líneas punteadas con segmentos de la misma longitud o líneas con segmentos alternados largo y corto.

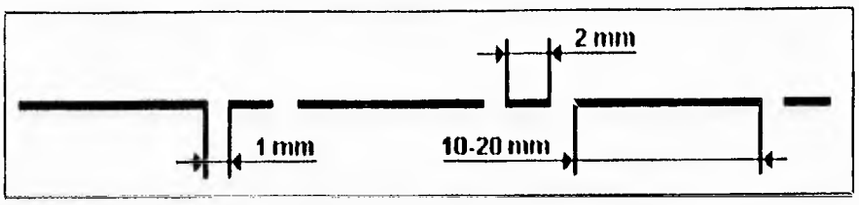


Figura 3.9 Estándar Francés de Líneas Punteadas

Líneas que tienen el mismo patrón pero diferente longitud son identificadas como líneas separadas, aunque coincidan en una misma línea en el espacio. Para que una línea sea considerada como línea punteada debe contener al menos 4 segmentos. El algoritmo identifica 4 segmentos (l_1, l_2, l_3, l_4) y sus espacios (g(1,2),g(2,3),g(3,4)) como parte de una línea punteada si cumplen con:

$$l_i \leq T_l \qquad i=1,2,3,4$$
$$T_{gl} \leq g(i,i+1) \leq T_{g2} \qquad i=1,2,3$$
$$l_i = l_{i+2} \qquad i=1,2$$
$$g(i,i+1) = g(i+1,i+2) \qquad i=1,2$$

Donde las T's son parámetros de umbral de longitud. Las longitudes y los espacios tienen permitido fluctuar ± 30% del promedio del valor de cada línea. El algoritmo consiste en dos estados: Detección y ordenamiento de segmentos vecinos y ligado de segmentos por detección de líneas punteadas.

3.5.2 Bloques

En el sistema Celesstin [24] los vectores son transformados en elementos de más alto nivel, los bloques (blocks).

En un dibujo técnico, las líneas anchas representan la proyección plana ortogonal de los contornos de un dispositivo mecánico (figura 3.10). Como las líneas delgadas no delinean el dispositivo, estas se ignoran en esta primera fase. Cualquier polígono cerrado mínimo con líneas gruesas compartiendo al menos un eje con el contorno externo del objeto tiene que encerrar materia. Si el polígono está sombreado cae en la sección del plano; si está vacío, puede representar una pieza enfrente o detrás de la sección del plano. Cualquier polígono cerrado mínimo con líneas gruesas compartiendo un eje con uno de los polígonos previamente descritos, deberá representar un cambio en geometría, una nueva pieza en contacto con la anterior, o un espacio vacío. Reglas simples como estas, permiten a Celesstin formular hipótesis acerca de estos bloques usando su contenido de líneas delgadas y el de sus vecinos.

El resultado es una descomposición completa del dibujo en un conjunto de estructuras a un nivel más alto que los vectores, esto es, al nivel de polígonos cerrados mínimos, o bloques, dibujados con líneas gruesas y con atributos de líneas delgadas (figura 3.11).

La siguiente operación sobre los bloques determina sus atributos técnicos. Antes de este análisis, el contenido de un bloque es únicamente gráfico: es un bloque vacío o un área rellenada, o un sombreado con líneas delgadas paralelas, que pueden ser paralelas a uno de los lados del bloque. El análisis cambia este contenido en un atributo. Las reglas que se usan están basadas en las reglas típicas de los dibujos de ingeniería marcadas por el estándar francés [24]. Incluso si Celesstin no puede asignar un atributo preciso a cada bloque, la descripción contiene "islas de confianza" de las cuales el proceso de interpretación puede comenzar a identificar componentes técnicos como tornillos, ejes y conexiones.

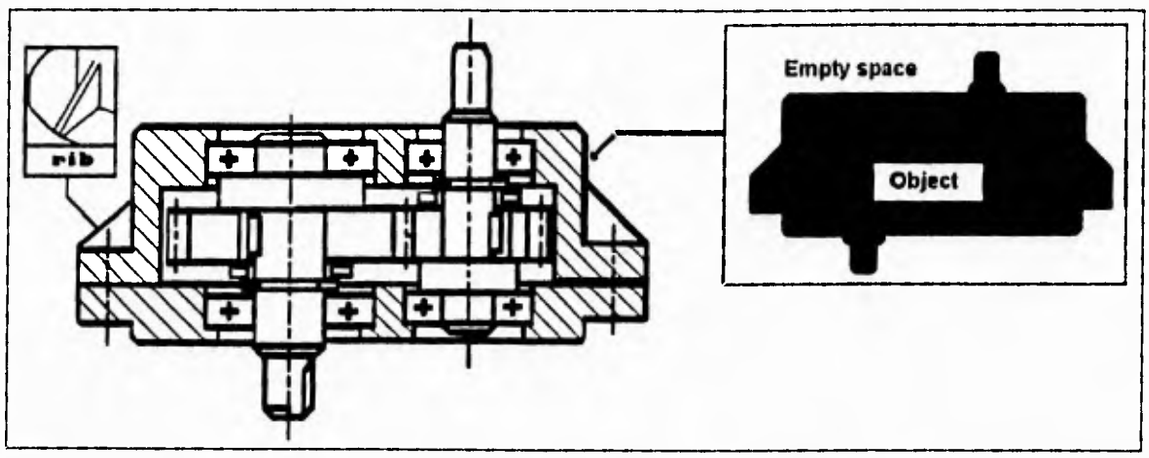


Figura 3.10 Dibujo Mecánico

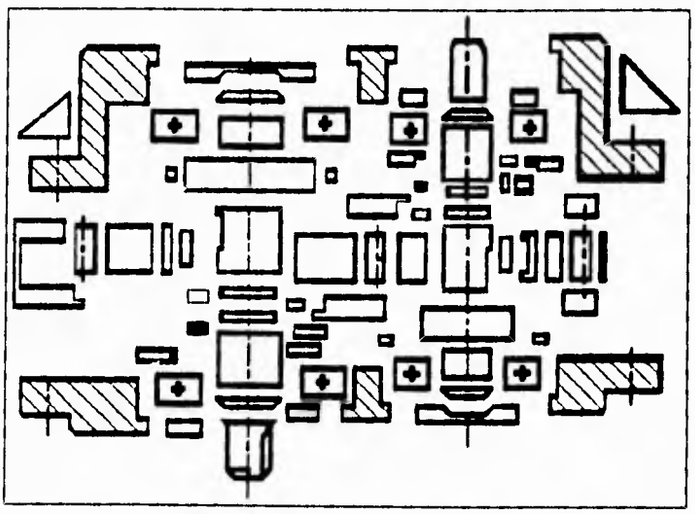


Figura 3.11 Dibujo Mecánico dividido en bloques

3.5.3 Figuras Conocidas

Existen varios métodos para el reconocimiento de primitivos gráficos [2, 3, 25]. Uno de los algoritmos más ilustrativos es el usado en [3].

El algoritmo localiza ciclos (loops) de mínima redundancia que son necesarios y suficientes para describir la imagen. Se entinde como un loop un conjunto de pixeles conectados entre sí para formar una figura cerrada. En la figura 3.12, inciso "a", se muestra la imagen de un diagrama de flujo, y los incisos "b" y "c" ilustran algunos de los loops de mínima redundancia que se pueden encontrar.

Dentro de una imagen pueden hallarse muchos loops pero únicamente se necesitan aquellos que puedan definir toda la imagen. Por ejemplo, un gran rectángulo cruzado por líneas horizontales generará muchos loops, cada uno de ellos un rectángulo, pero es más conveniente definirlo como un rectángulo cruzado por líneas. El procedimiento para identificar loops que faciliten la descripción de un gráfico consiste en 4 pasos simples.

Estos loops son comparados con un catálogo de "figuras conocidas" (figura 3.13), como son: rectángulos, cuadrados, polígonos, rombos, trapecios, etc., y aquellos que coincidan son descritos por su localización y orientación. Aquéllos que no son reconocidos como "figuras conocidas" son analizados para ver si pueden ser descritos como "figuras conocidas" que están ocultas o cortadas por otros objetos, teniendo en cuenta que el algoritmo no hace hipótesis acerca de la presencia de esquinas, por lo que los objetos no deben de obstruirlas. Aquellos loops que no puedan definirse por ninguna figura se definen como segmentos simples. El algoritmo también describe patrones de sombreado o rellenado, así como la relación espacial entre varias entidades.

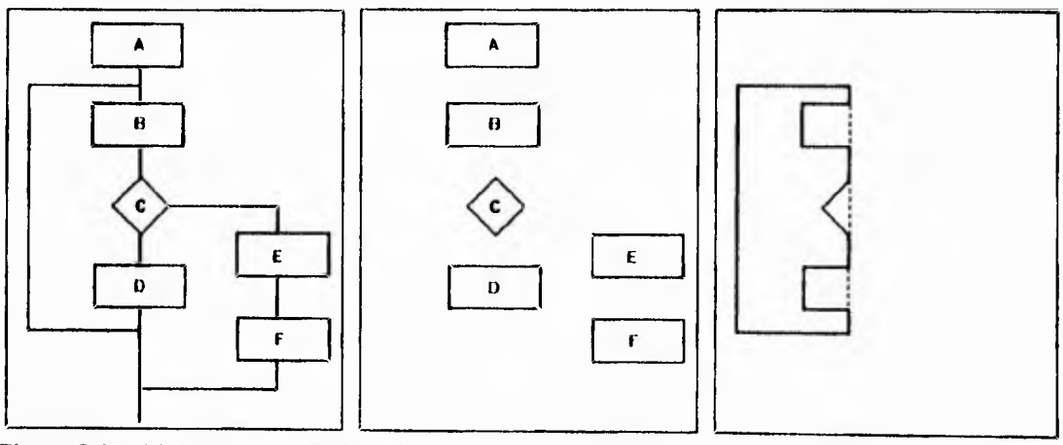


Figura 3.12 a) Diagrama de Flujo b) Loops Figuras Conocidas c) Otro Loop

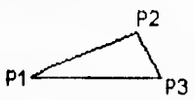
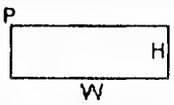
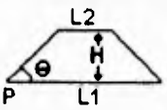
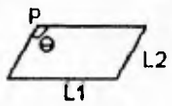
Figuras Conocidas	Atributos
Triángulo 	P1, P2, P3
Rectángulo 	P, W, H
Trapezoide 	P, L1, L2, H, Θ
Paralelogramo 	P, L1, L2, Θ

Figura 3.13 Tabla de Figuras Conocidas

3.6 Análisis de Alto Nivel (Reconocimiento de Símbolos, Componentes y su Interacción).

En algunos sistemas, la descripción de las relaciones espaciales entre los primitivos, es decir, la relación que guardan entre sí todos los segmentos de línea dentro del dibujo, sólo se define en términos de formas geométricas, bloques y/o vectores.

Otros sistemas intentan realizar un "entendimiento" mayor de los dibujos de ingeniería mediante el reconocimiento de símbolos, estructuras y componentes, es decir, una lectura de los dibujos, tal y como la haría un operador humano.

Un reto reciente es proveer un marco más flexible con herramientas de procesamiento a bajo nivel como la vectorización, separación de caracteres/gráficos, segmentación de áreas sombreadas, y extracción de primitivos, que pueden ser aplicados a todo tipo de dibujos, y unidades de reconocimiento de alto nivel que aplican un conocimiento de dominio especifico para procesar un tipo particular de dibujos. REDRAW [30], GTX 5000 [1], ANON [2], MIRABELLE [31] y CELESSTIN [24] caen en esta categoría. Todos ellos utilizan un análisis two-fold: de abajo-arriba (bottom-up) para primitivos o entidades, y arriba-abajo (top-down) para reconocer elementos en la imagen. La síntesis jerárquica es utilizada generalmente para combinar primitivos en formas simples, y formas simples en símbolos, y el reconocimiento está basado ya sea en análisis sintáctico o en reglas basadas-en-conocimiento. ANON también utiliza información del contexto, con lo que el análisis top-down se usa para modificar el análisis bottom-up. Redes neuronales y sistemas de pizarrón también han sido utilizados para manejar las condiciones no ideales o el reconocimiento inexacto [1].

Aunque estos sistemas no realizan aún un reconocimiento perfecto, y en algunos casos se asume la intervención de un operador, los sistemas de "comprensión" de dibujos se están volviendo más flexibles y versátiles [20].

3.6.1 GTX 5000. El Pizarrón y la Máquina de Inferencias

La mayor parte del procesamiento de contexto del GTX 5000 [1] recae en una estructura de datos común, llamada "pizarrón" (blackboard), y su asociado mecanismo de procesamiento, llamado "la máquina de inferencias" (The inference engine). El pizarrón es una estructura de datos global accesible, que contiene los elementos del dibujo en varios estados de reconocimiento, además de información de control. La máquina de inferencias interpreta las secuencias de instrucciones lógicas, llamadas

reglas base, y las aplica a los datos contenidos en el pizarrón para construir grupos abstractos crecientes de geometría y texto.

El pizarrón está dividido en dos niveles, cada uno representando un plano bidimensional, conteniendo componentes de la descripción del dibujo aplicable a ese nivel. Esto es, un nivel puede contener cadenas de texto, otros vectores sin usar, y otros símbolos parcial o totalmente reconocidos.

Cada nivel realiza un tipo particular de reconocimiento o transformación de los elementos del dibujo. Cada acto de reconocimiento generalmente agrega varias formas "menores" (e.g. un conjunto de pixeles conectados) de elementos del dibujo a formas "mayores" (e.g. una letra), las cuales suben a un diferente nivel de reconocimiento. Los niveles mayores o más altos del pizarrón tienen que trabajar con elementos más abstractos del dibujo representando porciones más grandes del mismo. En la figura 3.14a se maneja un nivel con compuertas. Conforme el reconocimiento progresa, los elementos del dibujo se mueven entre los niveles del pizarrón, agregándose a unidades más largas y abstractas, hasta que ya no es posible combinarlas.

Ya que la topología de un dibujo es siempre en dos dimensiones, una estructura típica de pizarrón utilizada son listas multiplemente ligadas, o algún otro sistema de codificación de datos gráficos.

El lenguaje de las reglas base está basado en la sintaxis de LISP, pero sustituye las funciones primitivas por unas más adecuadas para la tarea de reconocimiento.

Asociado con cada nivel está un conjunto de reglas base implementando tres operaciones básicas: distribución, síntesis y recolección (harvesting). Algunos ejemplos de niveles son el nivel de "cabezas de flecha", el nivel "flechas", el nivel de contexto de caracteres, y el nivel de asociatividad. El primero reconoce áreas rellenas de forma triangular como cabezas de flecha, y el segundo combina estas cabezas con líneas que se tocan para formar directamente flechas. Simultáneamente, el nivel de contexto de caracter ensambla caracteres en cadenas. Finalmente, el nivel de asociatividad relaciona la cadena cerca de la cola de una flecha con el objeto cerca de su cabeza.

La programación requerida para el desarrollo de reglas base, está a un nivel más elevado que el software de reconocimiento (escrito en C, [40]) y se enfoca más al domino del problema que en los detalles de implementación.

El reconocimiento de símbolos es el proceso mediante el cual los "racimos" (clusters) de vectores presentes en el dibujo son ensamblados en agregados jerárquicos, y son identificados como iconos estándar tales como bombas, válvulas, compuertas, y resistencias (figura 3.14b). Este proceso también envuelve la

identificación de conectores, como sería la representación esquemática de tubos, líneas y cables.

El proceso de reconocimiento de símbolos en el GTX 5000 comienza cuando las entidades de bajo nivel producidas por la vectorización son colocadas en la cola de objetos y son ofrecidas a los distintos niveles del pizarrón. Entonces, basándose en su posición y conectividad, el objeto es colocado en un racimo asociado con un nivel del pizarrón. Una buena separación y clasificación de los racimos (spatial clustering) es crítica para la precisión y eficiencia del proceso de reconocimiento de símbolos. Si la partición es muy fina, los componentes de un mismo símbolo pueden separarse entre racimos. Si la partición es muy ruda, la complejidad del tiempo del algoritmo crece geométricamente. Los algoritmos de agrupamiento en racimos (clustering) están basados en heurística apropiada para cada nivel.

Al igual que la selección de características para el reconocimiento de caracteres, escribir reglas para identificar símbolos es más un arte que una ciencia. Los datos de la mayoría de los dibujos son ruidosos y pueden resultar en una incorrecta vectorización. Las reglas deben reconocer símbolos tanto en perfectas condiciones como cuando están pobremente formados.

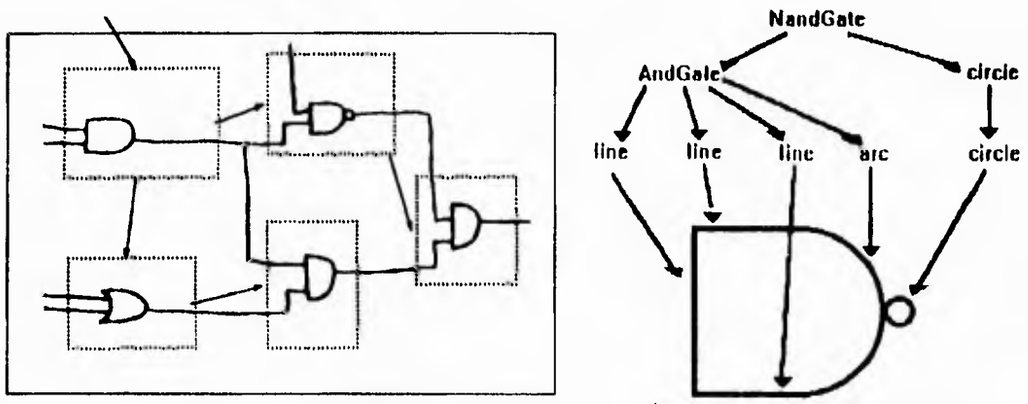


Figura 3.14 a) Pizarrón, b) Racimos y niveles para una compuerta NAND

3.6.2 Librerías

En el sistema propuesto por Yu y otros [20] se describen la reglas básicas del uso de librerías para el reconocimiento de símbolos. En el sistema diseñado para cualquier dibujo de flujo (aquellos dibujos formados por elementos, todos conectados entre sí, como son: diagramas de circuitos lógicos y electrónicos), es necesario diseñar una libreria de símbolos para cada dominio. La mejor forma de implementarla parece ser la comparación estructural, ya que los símbolos pueden ser dibujados en diferentes

tamaños u orientaciones, pero su estructura es la misma. El reconocimiento de símbolos puede ser separado en dos estados: Primero, se deben reconocer, mediante una comparación con figuras genéricas, formas simples como triángulos, rectángulos, medios círculos, medias elipses, etc. Después, los símbolos utilizados pueden ser reconocidos por un comparador que utilice las relaciones específicas del dominio para las figuras genéricas. Este comparador puede tener acceso a la base de datos formada por la librería de símbolos, de tal forma que, cambiando la base de datos de la librería de símbolos, el sistema se puede adaptar a cada dominio y cada aplicación.

Además, el sistema debe corregir todos los errores que provienen de la segmentación. Para ello puede hacer uso de la semántica que tiene la aplicación de los dibujos, por ejemplo, el estricto alternado entre los símbolos y las líneas de conexión, esto es, una línea de conexión solo puede estar conectada a símbolos, y los símbolos solo pueden estar conectados a líneas de conexión.

Inicialmente, el sistema de entendimiento realiza la hipótesis de que cada componente conectado en el dibujo segmentado es un símbolo. Para verificar la hipótesis, se ayuda con el comparador. Si el comparador verifica la hipótesis, entonces el componente conectado es identificado como un símbolo. Una vez que un símbolo es identificado, es posible seguir sus líneas de conexión para identificar otros símbolos potenciales. El dibujo entonces puede ser interpretado de manera recursiva.

Otros sistemas que utilizan librerías son Celesstin [24], y el sistema propuesto por Kasturi y otros [3], que, como ya se describió en el capitulo anterior, una vez que reconoce loops de mínima redundancia, los compara con una librería de figuras conocidas.

3.6.3 Reglas basadas en Conocimiento

El sistema Celesstin [24] es un ejemplo de la aplicación de reglas basadas en conocimiento previo de los dibujos para el reconocimiento de símbolos y componentes, en este caso, de dibujos de ingeniería mecánica.

El trabajo de interpretación se inicia como lo haría un ingeniero mirando un dibujo que no tiene ni dimensionamiento ni nomenclatura. Los ingenieros generalmente siguen las líneas punteadas principales que cruzan todo el dibujo. Entonces, después de reconocer los diferentes componentes colocados en estos ejes, tratan de entender el mecanismo de transmisión-movimiento. De aquí, la principal estrategia de interpretación es enfocar primero los bloques localizados sobre las líneas punteadas.

La idea principal es comenzar desde islas de confianza encontradas por las etiquetas de los bloques que indican sus atributos técnicos (bloques que contengan materia), y propagar la materia a lo largo de la línea eje.

Algunos atributos de los bloques son indicaciones fuertes de la presencia de materia. Un buen punto de inicio para el reconocimiento, por ejemplo, de un eje (shaft), es:

- Un bloque "suavizado" (threaded), o
- Un bloque que contiene una sección parcial y está localizado sobre sólo una línea punteada.

Si Celesstin no encuentra indicaciones fuertes para iniciar el reconocimiento, puede usar indicaciones más débiles, como la presencia de un bloque de encierro (enclosing block) representando el plano de fondo de un eje.

Cuando se encuentra un punto de inicio, se aplican las siguientes reglas de propagación:

- Cualquier bloque rectangular o trapezoidal, simétrico con respecto a la línea punteada y vecino de un bloque que ya ha sido reconocido como materia, es también un bloque de materia.
- Cualquier bloque conteniendo una sección parcial (no necesariamente rectangular o simétrica) y extendiendo el eje, representa materia.
- Cualquier trapezoide cerrado en el eje, detiene la propagación de materia.
- Cualquier bloque encerrando un bloque de materia se supone vacío, ya que usualmente representa un espacio vacío o una caja en el fondo de la sección del plano.
- Un bloque agujereado (tapped) está vacío.

Cuando se reconoce un eje el conjunto de bloques ligados es descrito con una notación simbólica "shaft".

Tales reglas no son siempre absolutas, pero globalmente el método de propagación permanece robusto. Para fallas en el reconocimiento con reglas débiles, un último recurso es la intervención del usuario. Además, las fases de interpretación de un nivel más alto pueden cuestionar la etiqueta propuesta y proponer otras hipótesis.

Celesstin puede analizar un dibujo en un nivel más alto mediante la visión del desensamblado y la cinemática de todo el conjunto. Pero incluso cuando se aplican solo procedimientos de razonamiento básico y comparación simple a los bloques que reconoce, puede analizar el dibujo en términos de elementos técnicos tomados de una librería CAD.

Celesstin incluye otro estado de reconocimiento de alto nivel, el cual se basa en el conocimiento de reglas de ensamblado y cinemática, para determinar si los elementos reconocidos son correctos.

En [30] se describe REDRAW, un sistema para la interpretación de diferentes clases de documentos técnicos. Usa un conocimiento *a priori* para lograr la interpretación a nivel semántico. El objetivo es construir un sistema general de reconocimiento de modelos-manejados que pueden ser completamente parametrizados. El modelo contiene conocimiento específico de cada clase de documento. El sistema es entonces llevado por este modelo usando una caja general de herramientas para operaciones de bajo nivel. En él se describen dos aplicaciones: mapas de ciudades y dibujos mecánicos.

En el sistema propuesto por Fahn y otros [25], se utiliza un método de reconocimiento basado en gramáticas que definen caracteres y símbolos dentro de diagramas de circuitos.

El extractor de componentes atraviesa un diagrama de circuitos siguiendo los esqueletos de los objetos. Mediante una aproximación lineal inteligente (piecewise) cada segmento es descrito como una secuencia de vectores para el reconocimiento de primitivos. Durante el recorrido en forma transversal del diagrama, los constituyentes de cada símbolo o caracter son combinados en un racimo mediante el análisis del contexto de los segmentos. Con las bases de las cualidades, que se encuentran en los modelos ejemplo de componentes esenciales, cada racimo que relaciona primitivos se identifica como un símbolo o un caracter. Si los segmentos están fuera de los racimos, son clasificados como líneas de conexión.

En este método la técnica de búsqueda profunda, constituída por un conjunto de reglas específicas, es llamada la búsqueda de la mejor relación (relational best search RBS). En una imagen binaria, cada componente esencial de un circuito, que satisface un conjunto de relaciones, es colocado en un grupo de grafos conectados.

En el inicio de la extracción de componentes esenciales, la búsqueda de objetos, con excepción de puntos aislados, se realiza con una ventana de 3 x 3, de derecha a izquierda y de arriba a abajo. Cada grafo conectado es atravesado desde el primer punto característico detectado o el primer punto inspeccionado durante la detección de puntos característicos. Al mismo tiempo que los grafos conectados son atravesados, se realiza la búsqueda de patrones de arcos y loops. Esto es debido a que los símbolos de los circuitos poseen varias cualidades, entre ellas los arcos y los loops. Estas cualidades están disponibles para categorizar los componentes esenciales. Todos estos patrones de arcos y loops son combinados en un conjunto de segmentos complejos para facilitar el reconocimiento de componentes esenciales.

Para el reconocimiento y extracción de estos componentes se utilizan gramáticas de contexto libre, ya que los segmentos están formados por grafos conectados.

3.6.4 ANON. El Schemata

En [2] se describe un sistema manejado por schemas (estructuras de datos con información referente a un elemento de la imagen), llamado ANON, y su aproximación está basada en la combinación de schemata (conjunto de schemas), describiendo construcciones de dibujos prototípicos con una librería de rutinas de análisis de imágenes de bajo nivel, y un conjunto de reglas de control explícitas. El sistema opera directamente en la imagen sin ningun preprocesamiento previo o vectorización, combinando la extracción de primitivos (bajo nivel) con su interpretación (alto nivel). ANON integra estrategias bottom-up y top-down en un marco simple.

Cada entidad que es localizada por el sistema es representada por una instancia de una clase particular de schema. Cada schema contiene una descripción geométrica de la construcción que representa y un conjunto de variables de estado que dan cuenta de la condición actual de representación. El schemata forma una red de nodos cuyos arcos corresponden a subpartes y subclases de relaciones. Los schemas están provistos de una interfase para comunicarse con la librería de rutinas de análisis de imagen, rutinas de búsqueda, rutinas de seguimiento, etc. Esta colección de funciones y procedimientos puede subdividirse en aquéllas que rastrean la imagen para clasificarla apropiadamente, y otras que se usan para el desarrollo de la descripción de primitivos de bajo nivel. Por lo tanto, cada schema contiene funciones de análisis de bajo nivel, funciones que examinan el resultado de estas operaciones, y reportan de sus hallazgos. Así, el control del schema se realiza en ambas direcciones, análisis de la imagen e interpretación de resultados.

El sistema de control de ANON consiste en reglas de estrategia escritas en la forma de una gramática LR(1) y aplicadas por un parser. Las reglas gramaticales no intentan definir un dibujo de ingeniería, solo especifican estrategias para reconocer varios componentes de estos dibujos.

4. SEGMENTACION DE DOCUMENTOS

Uno de los problemas que se tienen que afrontar antes de iniciar cualquier análisis dentro de un documento, es la identificación de la información que contiene. El proceso que realiza tal reconocimiento se llama segmentación, y es uno de los más importantes dentro de cualquier sistema.

4.1 Conceptos

Un documento en general consiste en varias áreas de imagen que pueden ser: texto, gráficos o diagramas, y fotografías o pinturas (conocidas en general como imágenes). Antes de poder identificar lineas o texto es necesario identificar las zonas donde cada tipo de información se encuentra.

El propósito del proceso de segmentación es segmentar (dividir) un documento en varios bloques separados, donde cada bloque representa un tipo de medio (texto, gráfico, imagen) [47].

La mayoría de las técnicas desarrolladas para la segmentación de texto [46] están basadas en la información contextual del tipo de imagen. Esa información puede ser: que el texto forma líneas horizontales en la imagen, o bien, que todos los caracteres son de un tamaño similar y pueden ser discriminados de la información de los dibujos.

Para imágenes de dos niveles se han desarrollado dos importantes técnicas: Los algoritmos que trabajan por regiones, y los algoritmos que segmentan caracteres individuales.

4.2 Segmentación por Regiones

La segmentación por regiones consiste en dividir el mapa de bits correspondiente a un documento en diversos bloques, sin importar el tipo de información que contengan. El tipo de información: texto, gráficos o imágenes, se determina una vez que se han localizado dichos bloques. A continuación se describen dos técnicas representativas de esta forma de segmentación.

Uno de los algoritmos más utilizados es el llamado Run-Lenght. El uso de estos algoritmos fue propuesto por primera vez en [52], y estos se han ido mejorando [46, 53]. Dichos algoritmos están basados en la idea de que, en la mayoría de los documentos, el texto está acomodado de tal manera que forma líneas horizontales, y

estas líneas horizontales forman bloques entre si (lo que vendrían a ser los párrafos y columnas). La técnica consiste en hacer crecer los espacios negros (formados por el texto) para hacer desaparecer las "corridas" de espacios en blanco (de aquí el nombre run-lenght) que separan las líneas de texto y las letras.

Una de las técnicas más actuales de este tipo de segmentación se presenta en [47]. El método propuesto es una combinación de un algoritmo de "embarrado" (smearing) run-lenght y un procedimiento de unión de bandas.

Smearing (embarrado) es una operación para conectar dos "corridas" en una sola, si la distancia entre estas dos corridas es menor a cierto umbral. Por ejemplo, si el umbral es 4 tenemos:

Corridas antes del smearing:

11111110000011111111111100011

Corridas después del smearing:

11111110000011111111111111111

Esta operación se aplica primero en dirección horizontal, renglón por renglón, y después se realiza en forma vertical, de columna en columna. Cada uno de estos pasos son combinados para ejecutar una operación AND que genera el resultado final. Si el bloque es una imagen o gráfico, el resultado es el mínimo rectángulo que encierra esta imagen. Si el bloque es un texto, entonces el resultado es una serie de bandas (figura 4.1b), cada una representando una línea de texto (en el supuesto de que el umbral del smearing vertical sea menor que la distancia entre dos líneas consecutivas de texto).

Después se realiza una operación de unión de bandas (figura 4.1c). Dos bandas deben unirse si pertenecen al mismo bloque de texto, es decir, al mismo párrafo desde el punto de vista gramatical. Existen tres posibles formas de delimitar dos bloques de texto, lo cual sirve para decidir si dos bandas deben unirse.

1. Dos bandas vecinas no deben unirse en el mismo bloque de texto si la distancia vertical entre ellas es mayor que cierto umbral. Esto es debido a que los párrafos suelen separarse por un espacio en blanco más ancho que la separación entre líneas.
2. Dos bandas vecinas no deben unirse si la banda inferior tiene un espacio en blanco a la izquierda, el cual la hace de menor longitud que la banda superior. Este espacio en blanco corresponderá a una identación, que se utiliza comúnmente al inicio de un párrafo.
3. Dos bandas vecinas no deben unirse si la banda superior tiene un espacio en blanco a la derecha, el cual la hace de menor longitud que la banda inferior. Este espacio en blanco corresponderá a una indentación provocada por un "punto y aparte".

Por lo tanto, dos bandas vecinas se unen si la proyección de ellas en dirección vertical se traslapa, y si ninguna de las tres condiciones anteriores ocurre.

Figura 4.1 a) Documento Original, b) Resultado después del "Agrandamiento", c) Resultado después de Unión de Bandas.

Una vez realizada esta operación, el documento queda segmentado en bloques. La última operación que se necesita es identificar el tipo de bloque que cada uno representa. Bloques de texto deben pasar a un procedimiento de reconocimiento óptico de caracteres (OCR), bloques gráficos a un proceso de vectorización, y bloques de imágenes a un proceso de compresión.

Para la clasificación de bloques de texto se utiliza un criterio basado en la proyección de rasgos (feature projection). Los bloques de texto presentan una proyección-y (vertical) que exhibe un comportamiento periódico (un espacio en blanco, una línea obscura de texto, un espacio en blanco, una línea obscura de texto, y así consecutivamente). Por lo tanto, si la proyección-y de un bloque presenta un comportamiento periódico se le clasifica como bloque de texto.

Para distinguir entre un bloque gráfico y uno de imágenes se utiliza el criterio de la densidad de puntos. Un gráfico formado por líneas presenta una gran distribución de puntos blancos (los puntos negros sólo forman las líneas y algunos caracteres), mientras que una imagen presenta una gran distribución de puntos negros (para simular y formar las distintas escalas de gris de la imagen original), como se puede observar en el ejemplo de la figura 4.2. Por lo tanto, se calcula el valor de conectividad de cada pixel dentro de un bloque. Este valor de conectividad no es otra cosa que el número de pixeles negros que rodean al pixel actual. Con estos valores se hace un histograma de conectividad.

En base a este histograma se determina:

1. Si el histograma de conectividad de un bloque no-texto tiene una distribución hacia la izquierda (hacia los valores más pequeños) se le considera un bloque gráfico.
2. Si el histograma de conectividad de un bloque no-texto tiene una distribución hacia la derecha (hacia los valores más grandes) se le considera un bloque imagen.

Para determinar si el histograma se distribuye hacia la izquierda o hacia la derecha, se calcula un valor que caracterice a la distribución dentro del histograma, a este valor se le llama valor de conectividad homogéneo (connectivity homogeneous value):

$$CHV = \frac{\sum N(i) * i}{\sum N(i)}$$

donde N(i) denota el número de pixeles cuyo valor de conectividad es igual a i, y el rango de i , lógicamente, es de 0 a 8.

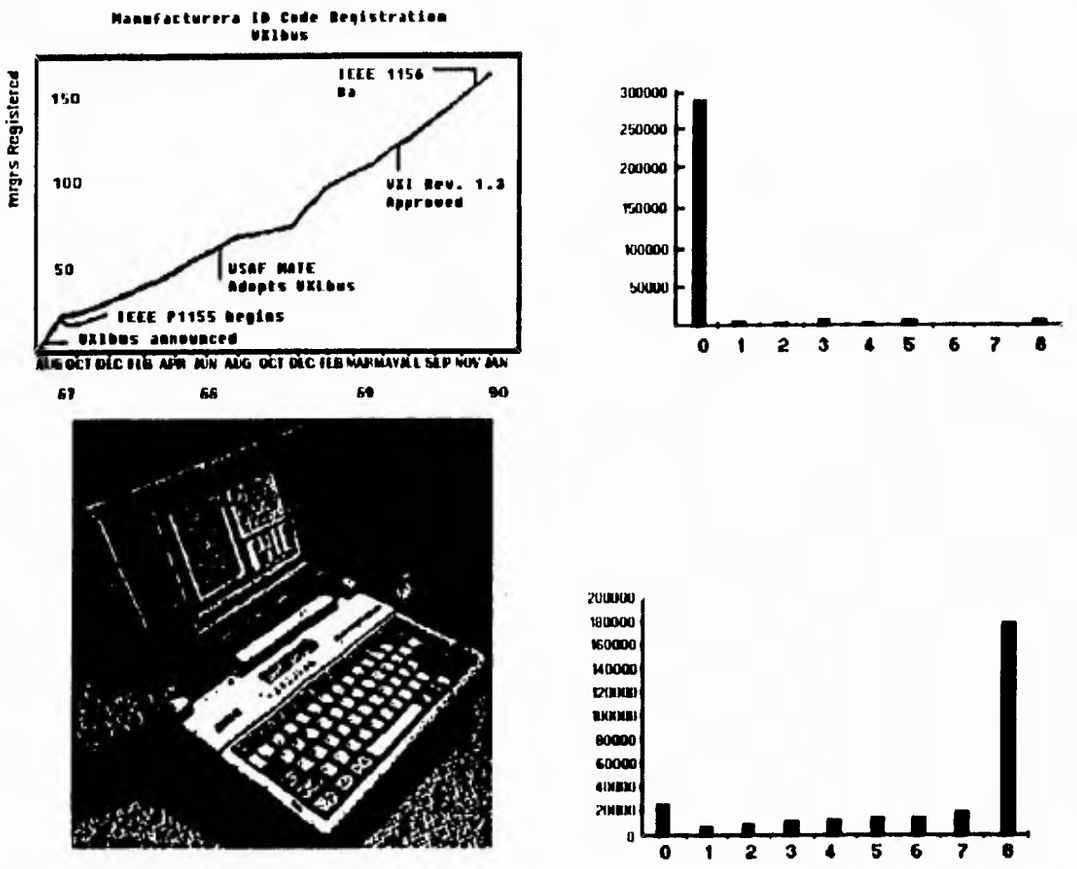


Figura 4.2 Histogramas de Conectividad.

Otro de los enfoques más recientes [48] realiza una operación de "agrandamiento" (growing) en lugar del smearing run-lenght.

En primer lugar, se buscan áreas llenas de objetos alineados que parezcan letras. La determinación de estos objetos se hace por su tamaño y su elongación. Con estos objetos detectados se sigue cada uno de los siguientes pasos:

1. Crecimiento (growing) de los segmentos de línea. En esta etapa se buscan secuencias de objetos alineados, y que dichas secuencias estén separadas entre si por espacios largos. Se comienza con el objeto más a la izquierda, y se busca el objeto a su derecha que tenga la posición vertical más cercana (esto es, alineado a él). Si este objeto está lo suficientemente cerca se unen, de lo contrario se inicia un nuevo segmento de línea. De esta etapa se toman aquellos segmentos que contienen cientos de unidades, es decir, los más largos.
2. Estimación de la inclinación. La orientación de estas líneas largas se utiliza para calcular la inclinación del texto. Todos los parámetros son recalculados para obtener una imagen que no tenga ninguna inclinación.

3. Detección de vecinos de los segmentos de línea. Cada segmento puede tener un solo vecino a la izquierda y uno a la derecha, pero puede tener varios arriba y abajo. En posición horizontal se buscan aquéllos más cercanos y que no estén muy desplazados en su posición vertical con respecto al segmento actual. En la posición vertical se buscan aquéllos que traslapen al segmento actual en esa posición (es decir, que tengan la mismo o menor longitud), y que no estén "muy lejos".
4. Unión de segmentos de línea. Se unen los segmentos de línea entre si. Aquellos segmentos "pequeños" se unen a los más cercanos verticalmente (si la distancia no es muy "grande"). De forma horizontal solo se unen aquéllos que aparecen cerca y tienen una región grande blanca a un lado.
5. Crecimiento y Unión de bloques de texto. Un bloque de texto es un fragmento de texto rodeado por áreas en blanco. Cada bloque puede incluir varios párrafos, si estos no están separados por un espacio en blanco. El crecimiento de los bloques se realiza para aquellos segmentos que estén lo suficientemente cerca para tener los mismos rasgos (ancho, distancia interlineal, etc.). La unión de bloques sólo se efectúa para aquellos bloques que tienen rasgos idénticos y que pueden agruparse en un solo bloque.

La ventaja de segmentar por regiones es que se utiliza la estructura global de las páginas para realizar un análisis más rápido, y donde en ocasiones no es necesario hacer un análisis de todos los pixeles. Sin embargo, cuando el texto no está distribuido en líneas o párrafos, o cuando el texto está mezclado con los gráficos o las imágenes, estos métodos resultan inapropiados [15].

4.3 Segmentación por Componentes Conectados

Existen dos enfoques [46] a la segmentación de texto basada en caracteres: aquellas basadas en el tamaño de los caracteres y aquellas basadas en el reconocimiento de caracteres. Ambas parten de la premisa de que los caracteres están formados por regiones conectadas. Para la primera aproximación, una región conectada es segmentada como texto si el área de la mínima caja que la rodea es menor que cierto umbral.

Uno de los algoritmos característicos de esta forma de segmentación se describe en [8]. Es un algoritmo diseñado específicamente para la tarea de separar componentes de texto de las regiones gráficas, independientemente de su orientación. Sus pasos principales son:

1. Generación de componentes conectados.

Los componentes conectados en la imagen digitalizada son aislados mediante un scaneo raster (raster-scanning) de la imagen y "agrandando" (growing) los componentes conforme son encontrados. El algoritmo lleva cuenta de los pixeles más arriba, más abajo, más a la derecha y más a la izquierda correspondientes al rectángulo más pequeño que encierre cada componente, y el porcentaje de pixeles dentro de estos rectángulos del tipo fondo (background).

Con esto, cada componente conectado es rodeado y marcado por un rectángulo (figuras 4.3 y 4.4).

2. Filtros de Area y Radio.

El filtro de área es diseñado para identificar componentes que son muy grandes, comparados con el tamaño promedio de los componentes conectados de la imagen. En los documentos estos componentes tienden a ser gráficos. Obteniendo un histograma de la frecuencia relativa de ocurrencia de componentes conectados, en función de su área, es decir, obtener el área más frecuente y el promedio general de área, se puede determinar un umbral para separar gráficos de texto.

3. Agrupamiento de componentes colineales.

Una vez determinados todos aquellos componentes conectados que son caracteres se procede a agruparlos en palabras. Se define una cadena de texto como un grupo de al menos 3 caracteres que son colineales y que satisfacen cierto criterio de proximidad. Esta parte del algoritmo identifica caracteres colineales aplicando la transformada de Hough (capítulo 5) a los centroides de los componentes conectados. De la transformada se obtendrán los parámetros que definen la línea que cruza el centro de cada palabra. Todos los caracteres cuyo centroide esté sobre esa línea formarán parte de la palabra.

4. Separación de componentes en Palabras y Frases.

Una vez identificadas las cadenas colineales, se vuelven a filtrar para eliminar componentes colineales que no sean caracteres. Después se agrupan usando las siguientes reglas:

- Espacio entre caracteres menor o igual a Ah
- Espacio entre palabras menor o igual a 2.5 x Ah

Donde Ah es la altura promedio local, que se calcula usando los 4 componentes que están a cada lado del espacio.

Posteriormente se puede hacer un análisis de reconocimiento de caracteres sobre estas cadenas, que muchos sistemas incluyen [1, 3].

5. Preprocesamiento para refinar la segmentación.

El algoritmo anterior describe líneas rotas y caracteres repetidos (e.g. ********) como cadenas de texto ya que satisfacen la heurística de las cadenas, pero no lo son. Además, todos aquellos caracteres o cadenas que tocan componentes gráficos no son segmentados. Algunos de los procesos para mejorar estos defectos son:

- Filtro de caracteres repetidos. Las cadenas hechas con caracteres repetidos muchas veces, son separados de las cadenas de texto checando su consistencia en varianza de densidad de pixeles, área, y el radio de la longitud de los lados de el rectángulo. Estas varianzas se normalizan dividiéndolas entre sus umbrales. Cuando el producto de la normalización de varianzas es menor que 1 se clasifica como gráfico.
- Filtro de líneas punteadas. Como las líneas punteadas no tienen una separación regular, su única característica es que su espesor es pequeño. Por lo tanto la información de espesor se obtiene de la imagen y se compara con un umbral.
- Texto conectado a gráficos. Un caso típico de esto son aquellos textos que aparecen subrayados. Para localizar estos caracteres se procede a encontrar una línea central de la cadena de texto, y una línea entre los dos componentes alrededor de un espacio es revisada para ver si tiene contacto con componentes conectados. Si uno de los componentes contactados no pertenece a la cadena, entonces se investiga. Una caja de tres lados orientada en el ángulo de la cadena de texto comienza a crecer hasta que encierra el componente por los tres lados. Se calcula la densidad de pixeles en cada línea, y si al final se crea una caja que satisface el tamaño y características de los componentes de la cadena de texto, el caracter se incluye [8, 3, 1].

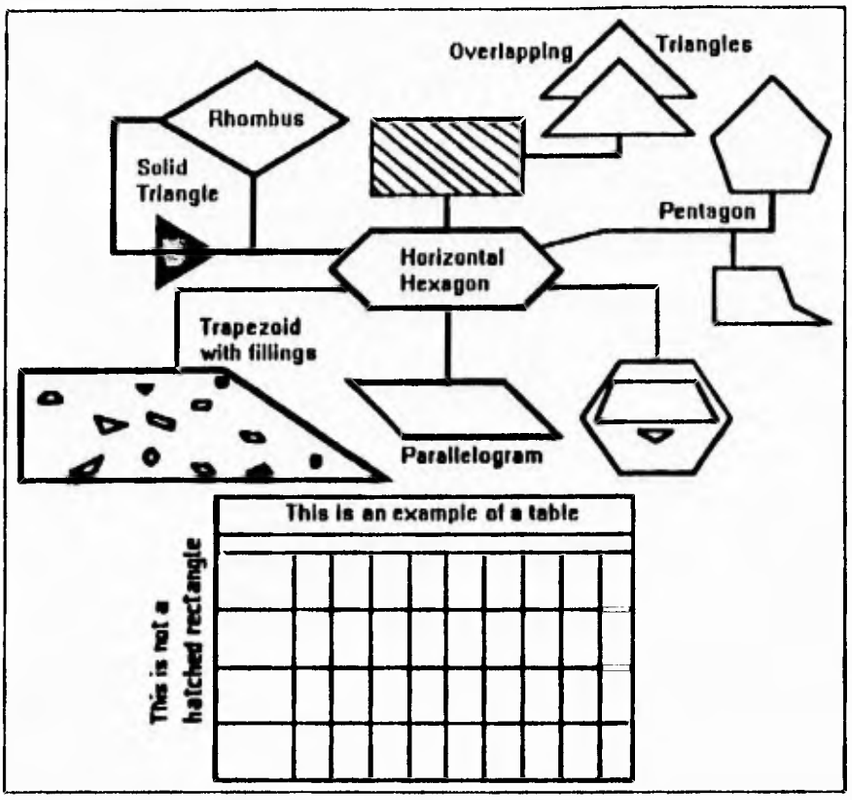


Figura 4.3 Documento Original.

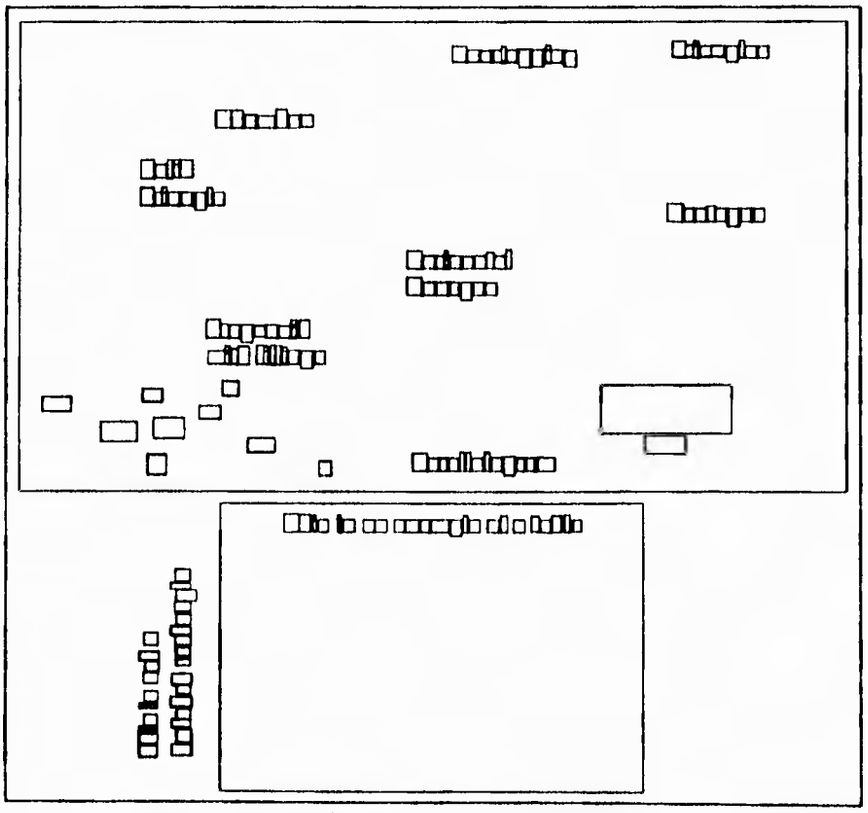


Figura 4.4 Localización de Componentes Conectados.

Otros autores sugieren que los rasgos globales, tales como el tamaño de los caracteres, no dan una buena segmentación, y proponen que cada región conectada pase por un reconocimiento de caracteres, y sólo aquellas regiones que sean reconocidas como caracteres se segmenten como texto [46].

Estas técnicas permiten identificar texto no importando su número, localización u orientación, por lo que resultan apropiadas para documentos que no presentan (y también para los que presentan) una estructura de bloques. Entre las desventajas podemos mencionar: En ocasiones requieren más tiempo de cómputo que la segmentación por regiones, si estamos hablando de documentos con muchas letras o una estructura de bloques; se pueden presentar fallas cuando existen distintos tamaños de letras, en el caso de no reconocer caracteres antes de segmentar [15].

4.4 Propuesta e Implementación de un Nuevo Esquema de Segmentación

Los documentos scaneados (documentos de oficina o dibujos de ingeniería), contienen información gráfica, imágenes y texto, por lo tanto es necesario realizar una segmentación antes de cualquier otro análisis (OCR, vectorización, etc.).

La idea del esquema que se presenta está basada en el desarrollado por Kasturi & Fletcher en [8], pero la implementación es diferente, sobre todo en la localización de componentes conectados. Este algoritmo realiza la segmentación por componentes conectados, lo cual resulta más conveniente, ya que los dibujos de ingeniería y los documentos de oficina presentan caracteres intercalados con gráficos, que los métodos de segmentación por regiones no separan.

El primer paso para realizar la segmentación es encontrar todos los componentes conectados en el mapa de bits, y con ello las coordenadas del mínimo rectángulo que los contenga. Un componente conectado es un conjunto de puntos negros ('1') dentro de la imagen, en el que cada uno de ellos es vecino de, al menos, otro punto negro del grupo.

Dentro de la literatura, el algoritmo para encontrar componentes conectados más utilizado se describe en [29]. Dicho algoritmo es muy sencillo, y rápido, ya que solo requiere dos pasadas: una renglón por renglón y otra columna por columna. Cada componente conectado (es decir, cada conjunto de pixeles conectados entre si) se clasifica con un número o una letra, y conforme se van encontrando nuevos o diferentes grupos se asignan nuevas o diferentes letras. De esta forma, al final, cada elemento de un componente conectado tiene el valor que identifica a ese componente (esta operación se conoce como etiquetado (labeling)).

Sin embargo, consideramos que este tipo de aproximación no resulta muy adecuada para nuestros objetivos. No sólo necesitamos identificar cada componente conectado, sino el mínimo rectángulo que lo contenga, es decir, no nos preocupa mucho (por el momento) cuales son los pixeles que pertenecen a cada componente, sino la posición que ocupan. Utilizando un algoritmo como el anterior, necesitaríamos dos pasadas para etiquetar cada componente (se debe observar también, que si tenemos una imagen con 80 letras, necesitamos 80 etiquetas diferentes), y después hacer una búsqueda, para cada componente, de todos los pixeles que tengan la etiqueta del componente actual, para poder conocer el rectángulo que los rodea.

El método que proponemos nos permite conocer todos los elementos de un componente conectado en el momento que se encuentra uno de sus elementos, con lo cual se tiene también el rectángulo que los rodea. Además, no necesita etiquetas (aunque se podrían poner si se requiriese) y su tiempo de ejecución es muy pequeño. Otra característica importante es su sencillez, la cual permite adaptarlo fácilmente a cualquier aplicación (etiquetar componentes conectados, borrarlos, marcarlos, agruparlos, etc.)

Para encontrar los componentes conectados se desarrolló una función recursiva (pacman(x,y)). Dicha función comienza con un punto negro en la imagen. Este punto se considera el más a la izquierda, más a la derecha, más arriba y más abajo del conjunto (las coordenadas del rectángulo). Este punto se marca como parte del conjunto (cambiando su valor de '1' a '2'), y después se procede a revisar todos sus vecinos, en búsqueda de puntos negros. Este es el listado de la función:

```
void pacman(x,y)
int x;
int y;
{
/* Revisamos si el punto queda dentro del rectángulo, de lo contrario, se modifican las
coordenadas del rectángulo para contenerlo */
  if ( x > der )
    der = x;
  if ( x < izq )
    izq = x;
  if ( y > arriba)
    arriba = y;
  if ( y < abajo)
    abajo = y;
/* Se clasifica el punto para que no se revise más de una vez */
```

```
  imgmap[x][y] = 2;
/* Se analizan los vecinos del punto, pero sólo aquellos con valor '1', lo cual indica que
están conectados al conjunto, pero no han sido analizados */
  if (imgmap[x+1][y] == 1)
    pacman(x+1,y);
  if (imgmap[x+1][y-1] == 1)
    pacman(x+1,y-1);
  if (imgmap[x][y-1] == 1)
    pacman(x,y-1);
  if (imgmap[x-1][y-1] == 1)
    pacman(x-1,y-1);
  if (imgmap[x-1][y] == 1)
    pacman(x-1,y);
  if (imgmap[x-1][y+1] == 1)
    pacman(x-1,y+1);
  if (imgmap[x][y+1] == 1)
    pacman(x,y+1);
  if (imgmap[x+1][y+1] == 1)
    pacman(x+1,y+1);
  return;
}
```

Como se puede apreciar, dicha búsqueda se ejecuta mediante la misma función, por lo tanto, el algoritmo se vuelve recursivo, y va marcando y buscando los vecinos de cada punto hasta encontrar alguno que quede sin vecinos. Entonces la función comienza a "regresar" de cada una de sus llamadas para seguir analizando los vecinos de los puntos anteriores.

Antes de clasificar cada punto como parte del conjunto se comparan sus coordenadas con las del rectángulo. Si el punto queda afuera, entonces las coordenadas del rectángulo se modifican para incluir dicho punto dentro del mismo.

Por lo tanto, a la salida de la primera llamada a la función se habrán agrupado y clasificado todos los puntos conectados al punto inicial y se tendrán las coordenadas del rectángulo mínimo que los contenga. Como la función cambia el valor de todo el conjunto de puntos, este procedimiento se debe realizar únicamente una vez para cada componente conectado, es decir, si recorremos la imagen y cada que encontremos un '1' llamamos a la función, encontraremos todos los componentes conectados de la imagen. Las coordenadas de los rectángulos se guardan en una lista ligada.

Al mismo tiempo que se van encontrando cada uno de los rectángulos, se realiza el cálculo del promedio del área (Ap) y del área más frecuente (Amp).

Si consideramos que los caracteres son más numerosos que los gráficos, y que los gráficos son mucho más grandes que el texto, el área más frecuente (Amp) representará los componentes de texto y, quizá, algunos gráficos pequeños. Si se asegura que el umbral es mayor que Amp, la posibilidad de descartar componentes de texto se elimina. Esto solo no es adecuado para procesar imágenes que contienen cadenas de texto de distintos tamaños, de ahí que se calcule también el área promedio. Aquellos rectángulos cuya área sea "muy grande" no se toman en cuenta en el cálculo del promedio, puesto que es seguro que se trata de elementos gráficos o imágenes. El valor para un área "muy grande" que elegimos fue 15,000. El umbral de área es colocado a 5 veces el valor más grande entre Ap y Amp [8].

Con este umbral se revisa la lista, y se separan todos aquellos rectángulos cuya área sea menor que él.

Además, se utiliza un último filtro. Existen segmentos de línea aislados, que obviamente no son caracteres, y sus rectángulos tienen un área muy pequeña, que podría ser segmentada por el criterio anterior. Por lo tanto, se calcula el radio de los rectángulos cuya área este debajo del umbral, si dicho radio es mayor de 10 pixeles, se considera una línea y no un caracter.

4.4.1 Resultados de la Implementación del Algoritmo de Segmentación

El algoritmo fue implementado en ANSI C [40] y probado en una computadora Sun SPARC station 2. Las imágenes de prueba fueron de tamaño 200 x 300 pixeles:

El algoritmo de segmentación presentó las siguientes características:

- Es muy sencillo, lo cual aumenta su eficiencia.
- Los resultados obtenidos son muy positivos, la segmentación se realiza correctamente, para distintos tipos de letras y combinación de gráficos y caracteres, y a pesar de la sencillez del criterio para definir el umbral. Los errores que se presentaron son los comúnes en esta clase de algoritmos: gráficos de tamaño aproximado al de los caracteres y viceversa; caracteres conectados entre si y cuyo tamaño juntos rebasa el umbral; y, caracteres parecidos a líneas (como letras "l" grandes o paréntesis).
- El tiempo de ejecución fue muy pequeño y, obviamente, se va incrementado conforme se tienen más puntos negros.

A continuación se muestran algunos ejemplos de la aplicación del algoritmo, en ellos se incluyeron distintos tipos de tamaños y formas de caracteres, símbolos especiales, puntos, etc. Se combinaron estos con distintos tipos de gráficos, como son subrayados, cajas, ilustraciones, y formas complejas.

En la figura 4.5 se tiene una imágen original, y los resultados de la segmentación en las figuras 4.6, con los componentes considerados gráficos, y 4.7, con los componentes considerados caracteres. Podemos observar que se segmentan correctamente casi todos los caracteres, aún teniendo distintos tamaños y características. Unicamente, las letras "l" se consideran como líneas (su longitud es mayor al radio establecido) y el círculo negro se considera caracter (su tamaño queda dentro del promedio de los caracteres).

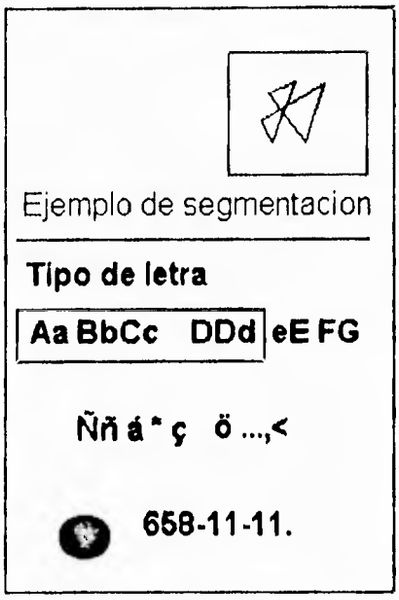


Figura 4.5 Imagen Original

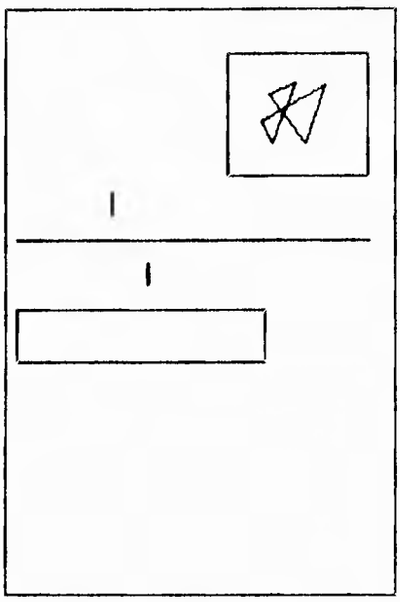


Figura 4.6 Gráficos

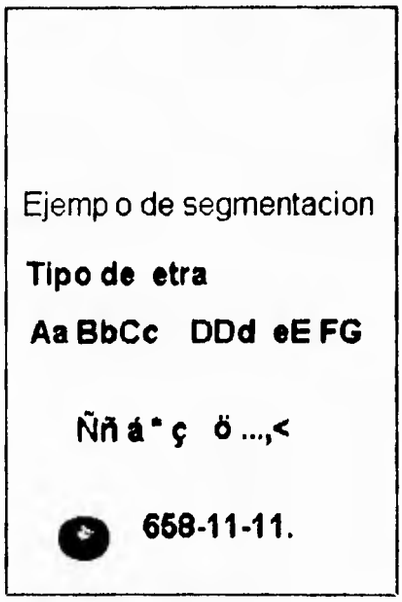


Figura 4.7 Caracteres

En la figura 4.8 colocamos puntos de ruido dentro de la imagen. La cantidad de puntos es mayor que la cantidad de letras. Si sólo calculásemos el área más frecuente Amp, esta sería la de los puntos, y los caracteres no se segmentarían. Con la ayuda del promedio Ap, el umbral incluye a los caracteres y la segmentación es correcta, como se aprecia en las figuras 4.9 y 4.10.

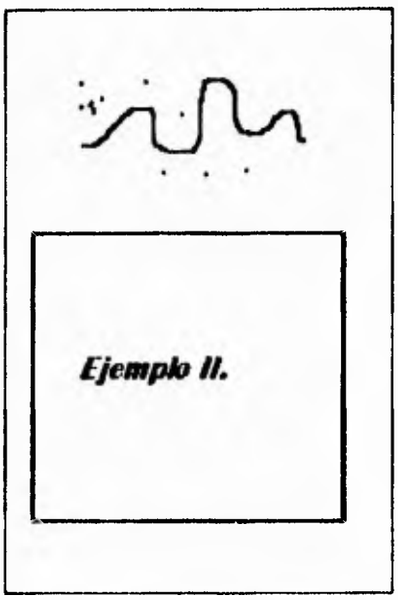


Figura 4.8 Imagen Original

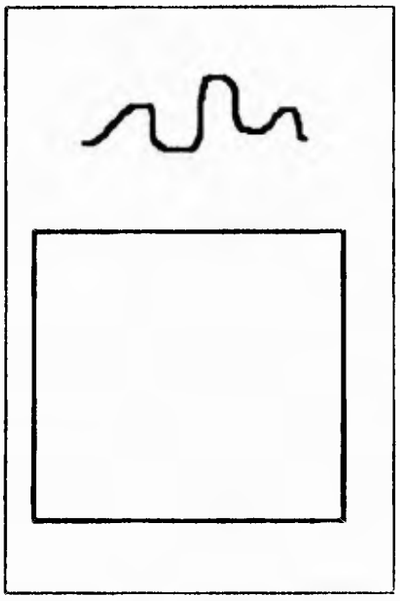


Figura 4.9 Gráficos

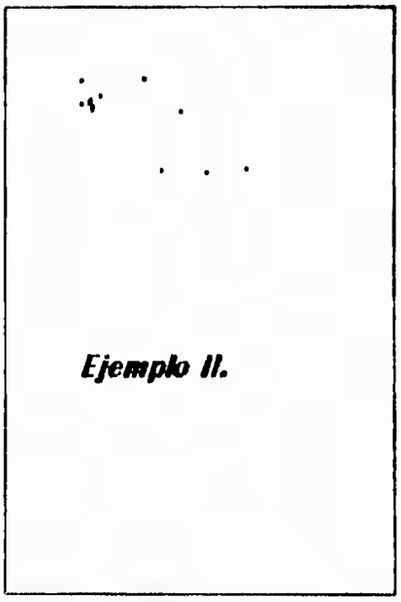


Figura 4.10 Caracteres

La figura 4.11 es un ejemplo similar al anterior. El área más frecuente es la misma (tenemos igual número de caracteres que de gráficos), por lo que todos los componentes quedarían debajo del umbral, ya que se escoge el Amp mayor, y el mayor entre Ap y Amp. Por lo tanto, resalta la importancia de poner un límite al tamaño de los caracteres, es decir, contemplar un área máxima, y todos aquellos componentes que rebasen esa área son considerados gráficos. De tal forma quedan segmentados los cuadros como gráficos (figura 4.12) y los ceros como caracteres (figura 4.13).

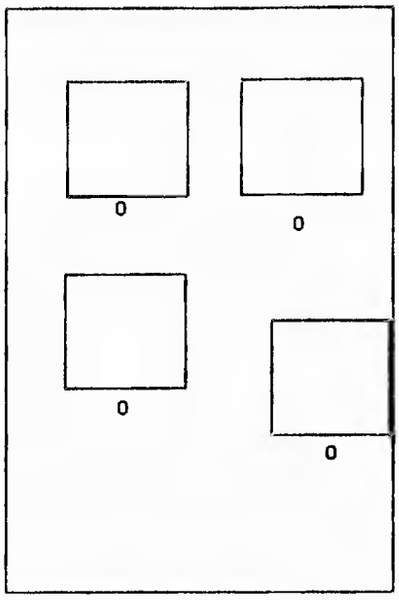


Figura 4.11 Imagen Original

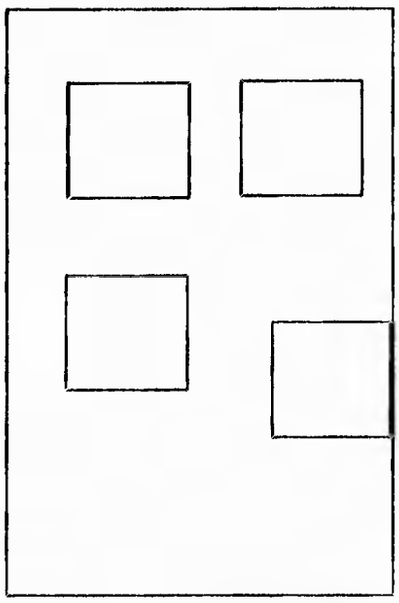


Figura 4.12 Gráficos

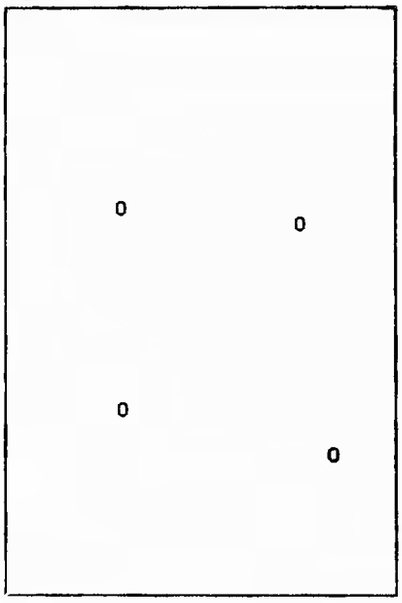


Figura 4.13 Caracteres

En el ejemplo de la figura 4.14 se presenta un documento con una distribución uniforme de los caracteres, en los que comúnmente se aplica la segmentación por regiones. Aquí podemos observar que nuestro algoritmo realizó una correcta segmentación, sin importar la presencia de mayúsculas, minúsculas y gráficos. Todos los caracteres quedan segmentados en la figura 4.16, y los gráficos presentes en la figura 4.15.

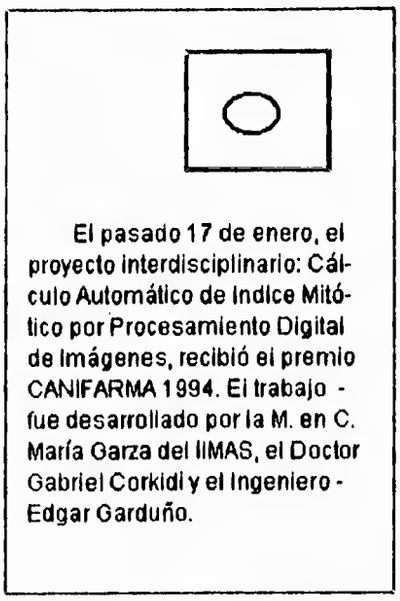


Figura 4.14 Imagen Original

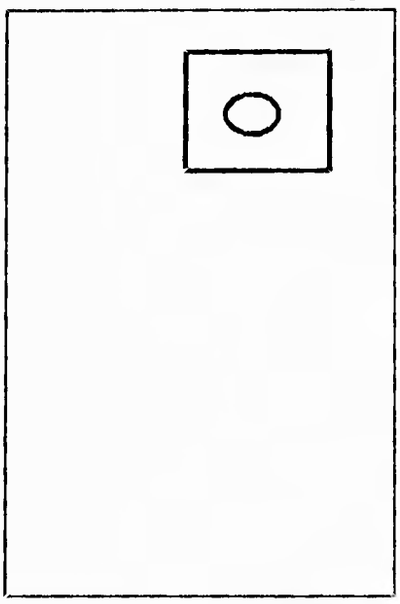


Figura 4.15 Gráficos

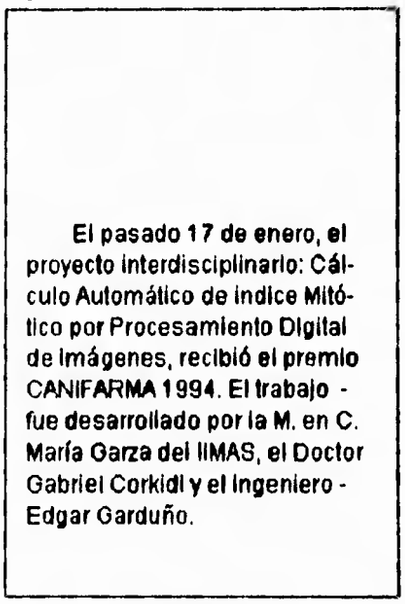


Figura 4.16 Caracteres

Al ejemplo de la figura 4.14 se le agregaron puntos para apreciar la efectividad del algoritmo en presencia de ruido, resultando la figura 4.17. Debido a esto quedan conectados unos caracteres con otros (en las palabras "proyecto", "CANIFARMA", "Automático", etc.). En los resultados de las figuras 4.18 y 4.19 se observa que únicamente aquellos caracteres que estaban unidos de manera evidente se segmentaron como gráficos, ya que el área del rectángulo que los rodea resultó mayor al umbral.

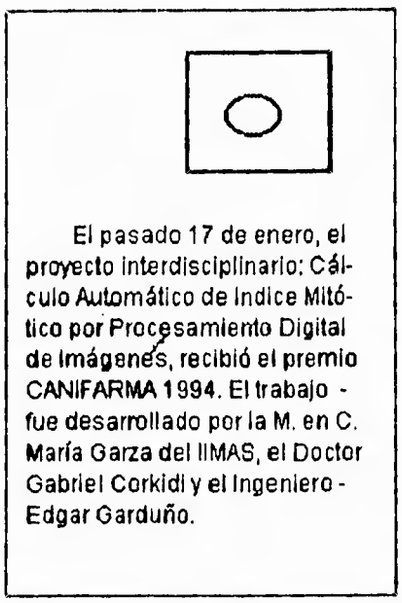


Figura 4.17 Imagen Original

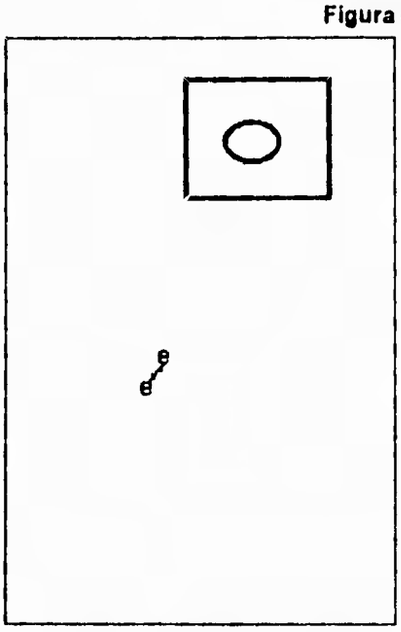


Figura 4.18 Gráficos

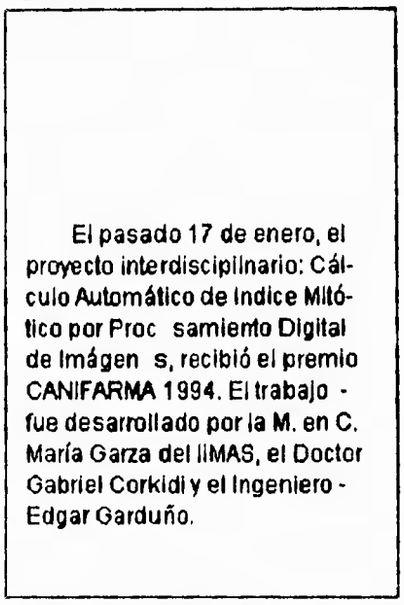


Figura 4.19 Caracteres

En la tabla 4.1 podemos ver que el algoritmo no consume una gran cantidad de tiempo de procesamiento (este tiempo incluye la creación de dos archivos para guardar los resultados que se muestran en cada figura). Regularmente durante este tipo de segmentación, se consume más tiempo en encontrar los componentes conectados y sus rectángulos [5]. En nuestro caso, el algoritmo recursivo para encontrar componentes conectados resulta muy rápido, y por su sencillez, puede adaptarse a cualquier requerimiento.

Figura	Tiempo (seg)
Figura 4.5	3.84
Figura 4.8	3.88
Figura 4.11	3.91
Figura 4.14	3.91
Figura 4.17	3.84

Tabla 4.1. Tiempos de Ejecución

La eficiencia del algoritmo podría aumentarse si, para elegir el umbral de área, se tuviera conocimiento previo de las características del documento. Esto es, si se pudiese incluir información (o la manera de obtenerla) sobre la cantidad de caracteres, la diversidad del tamaño de estos, y su diferencia en tamaño con respecto de los gráficos, el umbral podría ajustarse de manera más "inteligente" o podrían incluirse otros criterios, que evitarían los errores que el actual algoritmo presenta.

Los sistemas de análisis de documentos que han sido desarrollados sufren de esta clase de errores (incluir texto en gráficos y viceversa). Dichos errores son corregidos en las etapas de reconocimiento de caracteres y vectorización [1 - 3]. Si un componente conectado no puede identificarse como caracter, se le considera un gráfico, y si un gráfico es muy pequeño para reconocerse o vectorizarse, se contempla la posibilidad de que sea un caracter.

5. RECONOCIMIENTO Y VECTORIZACION DE PRIMITIVOS GRAFICOS

Una de las herramientas indispensables para el análisis de documentos gráficos es un vectorizador, es decir, un algoritmo que encuentre los primitivos gráficos dentro del mapa de bits (líneas rectas, curvas o incluso figuras geométricas) y las codifique en un formato vectorial.

5.1 La Conversión Raster a Vector. Ventajas y Desventajas

5.1.1 Almacenamiento y Comunicación

La representación vectorial varía significativamente en su compactación. Los formatos vectoriales se benefician de compartir las definiciones de símbolos y otras estructuras, con lo cual pueden almacenar curvas complejas con sólo unos cuantos parámetros. El alto costo de almacenamiento proviene del espacio requerido para almacenar la estructura del dibujo, ya que el almacenamiento de valores reales para las coordenadas ocupa un espacio significativo de memoria.

Los bitmaps raster ocupan una gran cantidad de memoria. Esto no sólo afecta almacenamiento, sino también paginación y comunicaciones de red. Para ellos se han desarrollado métodos de compresión que reducen el tamaño del almacenamiento. Sin embargo, la edición requiere de una descompresión completa de la imagen, lo cual demanda grandes cantidades de memoria y tiempo de conversión. Una de las soluciones propuestas es la de dividir las imágenes en secciones cuadradas, que sean comprimidas separadamente, lo cual reduce la cantidad de memoria requerida y tiempo.

La conversión entre formatos vectoriales depende de la forma en que se definan las representaciones paramétricas, y las estructuras en las que los parámetros se almacenen. En contraste, la conversión entre dibujos raster es simple y directa. En el peor de los casos se requiere de descomprimir completamente una imagen y recomprimirla en el nuevo formato.

La conversión de un formato vectorial a un formato raster es simple también. La mayoría de los sistemas CAD soportan el uso de plotters, los cuales usan una entrada raster, la cual es también una entrada aceptable para los sistemas de dibujo raster.

La mayor distinción entre ambos es la facilidad de convertir documentos en papel a estos formatos. La captura por medios ópticos de dibujos de ingeniería se realiza por medio de scanners u otros dispositivos ópticos, los cuales realizan la captura en formato raster. Así, en segundos, el documento en papel es capturado para

su edición, archivado o transmisión raster. La conversión a formatos vectoriales de tipo CAD, sin ayuda de métodos automáticos, toma horas de edición para separar modelos de los datos de representación y definir las estructuras de alto nivel [11, 29, 38].

5.1.2 Ventajas y Desventajas

Cualquier imagen puede ser representada en el formato raster puesto que siempre es posible descomponer dicha imagen en pequeñas celdas o puntos, sin embargo surge un gran problema con el almacenamiento de la imagen, ya que generalmente su tamaño es muy grande. Una imagen con alta resolución requiere muchos megabytes de espacio para almacenamiento, y otros tantos para su despliegue, por esta razón se usan técnicas de compresión para almacenarlas [9].

El manejo de las imágenes raster implica gran demanda de procesador y del bus de la computadora, es por ello que para lograrlo en tiempos razonables es necesario utilizar estaciones de trabajo capaces, o bien, computadoras personales veloces.

Las imágenes raster no pueden expandirse ni reducirse con facilidad ya que los bits de la imagen no guardan una relación intrínseca entre sí, si es necesario hacerlo, los programas que lo realizan tienen que hacer un proceso difícil, por ejemplo, encontrar todos los puntos adyacentes con características similares, clasificarlos y operar sobre todos ellos.

Otro problema de flexibilidad de los formatos raster está en la resolución, cuando se requiere agrandar la imagen cada punto se extiende en un entorno rectangular, lo que ocasiona que la resolución no sea tan buena como antes. En muchos casos el graficador inserta líneas entre los puntos, lo que causa también más tiempo de procesamiento. Cuando una imagen raster se almacena en un espacio menor al que ocupa originalmente, también se pierde su resolución al momento de regresarla a su tamaño original [9, 38].

Los formatos de tipo vector tienen limitaciones en cuanto lo que pueden representar, pero tienen las ventajas de eficiencia y flexibilidad sobre el formato raster. Aquí tenemos, por ejemplo, que una línea recta puede ser representada solamente por dos puntos finales, una línea curva puede ser aproximada por líneas rectas ligadas, un círculo puede ser codificado con una distancia radial y dos puntos finales. Así mismo, las imágenes tipo vector pueden ser escaladas positiva o negativamente sin ningún problema de resolución.

Entre las desventajas de los formatos vectoriales, se pueden mencionar: La dificultad para realizar operaciones por bloques o booleanas, las mediciones analíticas

(distancia, área, etc.) que en ocasiones se vuelven muy complejas, y su falta de adecuación a imágenes digitalmente muestreadas, es decir, la difícil conversión de una imagen raster a un formato vectorial [9, 38, 39]. En la tabla 5.1 se muestra una comparación de las ventajas que presentan cada uno de los formatos de imagen.

Ventajas de las tecnologías Vector y Raster	
Vector	Raster
Entradas	**Entradas**
Entrada de símbolos y ensambles definidos paramétricamente	Entrada de primitivos no paramétricos, como son: a mano, brocha de aire y texturas
Precisión geométrica de alto nivel	Patrones definidos por el usuario y rellenado rápido de regiones
Dimensionamiento automático de alta precisión	Rápida entrada y composición de líneas y símbolos
Revisión de reglas durante la entrada de símbolos	
Edición	**Edición**
Edición de líneas, texto y símbolos paramétricamente definidos	Edición y borrado a mano
Rápida y precisa rotación y escalamiento	Operaciones rápidas de cortado y pegado
Mínima pérdida de precisión en rotación y escalamiento	Mayor interacción con el usuario y más rápida
Interfases	**Interfases**
Geometría lo suficientemente exácta para entrada de otras aplicaciones	No se necesita conversión para scaneo o ploting
Capacidades de ambas tecnologías	
Entrada de primitivos y símbolos geométricos paramétricamente definidos	
Baja precisión en dimensionamiento automático	
Uso de capas, para parte de dibujos sombreadas y para acceso y control de versión	
Capacidades de edición de textos	
Asociación de atributos no geométricos	

Tabla 5.1 Ventajas de los formatos raster y vector.

5.2 Métodos para la Conversión Raster-Vector de Primitivos Gráficos

Son muchos los métodos para reconocer líneas, y son muchas las variaciones que cada uno de los métodos tienen. Hoy en día no existe un método que sea el más apropiado para realizar esta tarea, y cada día surgen nuevos algoritmos y nuevas aproximaciones [15, 45]. Generalmente, aquellos algoritmos que realizan un mejor reconocimiento resultan ser complicados, lentos o consumen muchos recursos de cómputo, mientras que aquellos que son más sencillos, rápidos y que no consumen recursos, realizan un reconocimiento regular (incluso malo) de las líneas [1, 15].

Algunas de los métodos utilizados para la conversión raster-vector son: Thinning o adelgazamiento, Seguimiento de contornos, Seguimiento de líneas, Transformada de la distancia, Aproximación poligonal, Segmentación poligonal y Transformada de Hough. A continuación describiremos brevemente estos métodos.

5.2.1 Thinning

El Thinning o 'Adelgazamiento", también es conocido como Skeletonizing (Esqueletonización) o Medial Axis Transform (Transformada del Eje Medio). Estos algoritmos generan una descripción precisa de las líneas centrales o figura central de los objetos.

La operación básica iterativa de "adelgazamiento" (Thinning) es examinar cada pixel de una imagen junto con el contexto de la región vecina, de al menos 3 x 3 pixeles, y borrar o "pelar" las orillas de la región, una capa de pixeles a la vez, hasta que las regiones queden reducidas a líneas de un pixel. Este proceso es iterativo, en cada operación cada pixel de la imagen es inspeccionado, y aquellas orillas con ancho de un pixel que no se requieren se borran y las que presentan conectividad se mantienen [15].

Existen infinidad de algoritmos Thinning. La teoría básica puede revisarse en [13]. Una revisión muy completa de ellos se puede encontrar en [7] y uno de los desarrollos más recientes en [21].

Sin embargo, estos algoritmos son en ocasiones lentos, y no hacen un buen trabajo reconociendo el ancho verdadero de una línea o reconociendo áreas rellenadas [1]. Sufren además de sensitividad al ruido, pérdida de continuidad y distorsión [19].

La propuesta más reciente para la esqueletonización [19] consiste en comenzar en el dominio de los vectores, en lugar de en el dominio de la imagen, como lo hacen los algoritmos thinning tradicionales. De tal forma que el resultado instantáneo

es la representación vectorial de los "esqueletos" en lugar de una imagen raster de esqueletos, que después se vectoriza.

El thinning es la única aproximación implementada para la vectorización dentro de los sistema comerciales, y en particular aquéllos desarrollados para el uso de código de calidad, seguido de una aproximación poligonal de los esqueletos [24]. En la literatura existen varios sistemas que utilizan el thinning para el reconocimiento de líneas [1, 3, 5, 20, 25, 26].

Por otro lado el ancho de los dibujos de ingeniería, es decir, el ancho de sus líneas, es normalmente de más de un pixel, lo cual crea serios problemas en la vectorización ya que un segmento de línea puede generar dos o más vectores lineales paralelos. Para evitar estos problemas, la esqueletonización es un proceso de pre-requisito (normalmente) para reducir el ancho de los dibujos a un pixel, lo cual facilita la tarea de vectorización [19, 5]. En la figura 5.1a podemos ver un ejemplo de un dibujo de ingeniería, al cual se le aplica un "adelgazamiento" (figura 5.1b), lo cual facilita su vectorización (figura 5.1c).

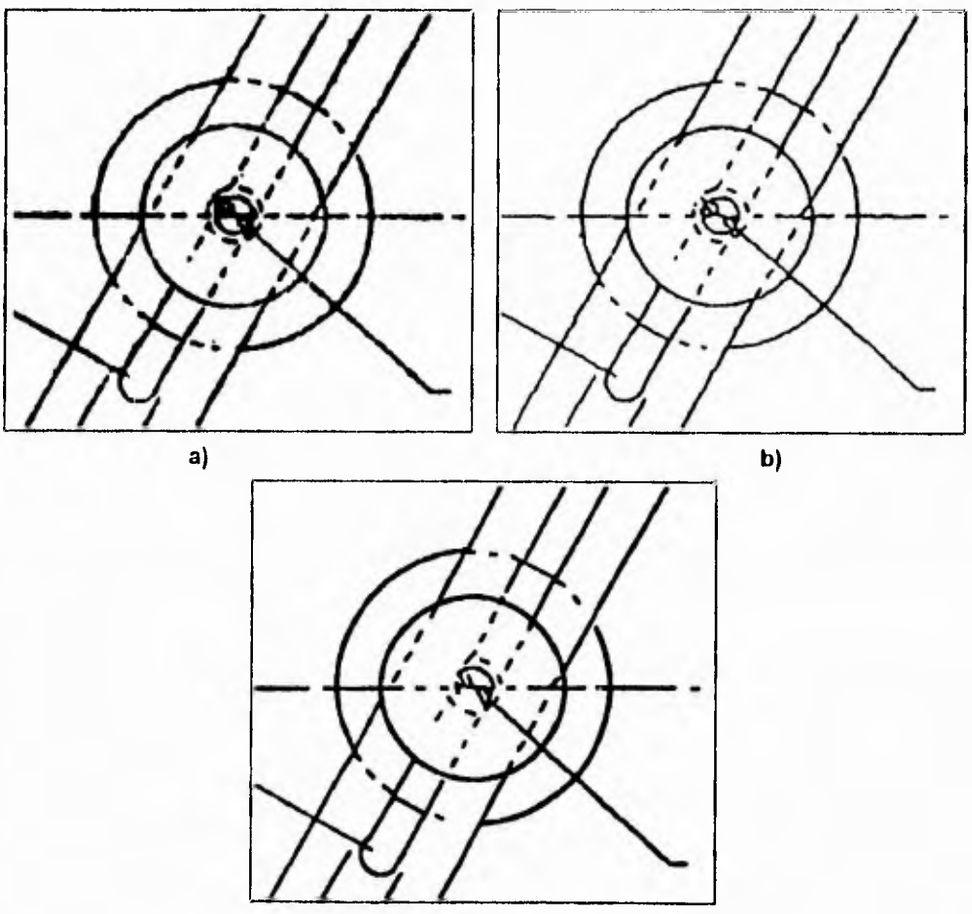


Figura 5.1 a) Dibujo Original, b) Thinning, c) Vectorización.

5.2.2 Seguimiento de Contornos (Countour Tracking)

Este método "sigue" los límites del contorno de un objeto generando un código de cadena (chain code) para ese límite. Por si mismo, este método no produce una descripción útil para dibujos de ingeniería, pero puede ser bueno para otras aplicaciones como la conversión de logos, o trabajos de arte a una representación electrónica para hacer cortes en vinil, etc [6].

El contorno de un conjunto conectado de pixeles R, puede definirse como el conjunto de todos los pixeles en R que tienen al menos un pixel vecino que no está en el conjunto R [13].

En la matriz binaria, cada segmento de línea del dibujo original resulta en un área (un conjunto de elementos visibles conectados o pixeles conectados). Un segmento es transformado por un contorno casi lineal, cuyos ejes son líneas a menudo interrumpidas por escalones generados normalmente por ruido; un arco es transformado en un contorno muy similar a una parte de un anillo, que se encuentra entre dos ejes curvos cercanamente concéntricos, y que se aproximan a dos círculos ruidosos.

Estas observaciones sugieren el estudio de diferentes contornos en la imagen binaria, y la construcción de una estructura de datos asociada y apropiada, la cual guarde la información relacionada con cada contorno [6].

Los algoritmos de seguimiento de contornos buscan encontrar los límites de los gráficos en la imagen, con el fin de distinguirlos después como elementos primitivos (por ejemplo, segmentos de línea) y regularmente está apoyado por alguno de los otros métodos mencionados de vectorización [6, 22].

5.2.3 Seguimiento de Líneas (Line Tracking)

En este método, un dibujo es primeramente scaneado detalladamente para estimar un promedio del ancho de línea y el mínimo espacio entre líneas. De estos valores, se determinan las dimensiones de una ventana de búsqueda (tracking window), que no es otra cosa mas que una caja conceptual para "deslizar" sobre los objetos. Cuando una parte de un objeto puede ser guardada dentro de la ventana, se considera que esa parte es lineal. Cuando las partes de un objeto van más allá de la ventana, se considera un objeto complejo, y se pasa a revisar con otro método [1, 2, 24].

Uno de los enfoques más recientes de este concepto se puede encontrar en el sistema Celesstin [24], que es una adaptación a ingeniería mecánica del algoritmo de procesamiento de diagramas propuesto por [27].

El método está basado en un conocimiento *a priori* del ancho máximo de una línea y la distancia mínima entre dos líneas. Ese tipo de conocimiento está disponible fácilmente en ingeniería mecánica, donde se aplican estándares para las líneas.

Por ejemplo, las reglas aplicadas en Francia (French Standard NF E04-103) son:

1. El ancho de una línea delgada debe ser un cuarto de una línea gruesa.
2. La distancia entre dos líneas gruesas debe ser al menos 0.8 mm.
3. La distancia entre dos líneas "sombreadas" (hatching) varía entre 1.5 y 2 mm.

El método consiste en dividir la imagen binaria en n x n mallas, donde n es más grande que el máximo ancho de línea y más pequeño que la mínima distancia entre dos líneas. Con esto, cuando mas una línea en una dirección dada debe cruzar cada malla, y cualquier línea no traspasará más de dos mallas. El algoritmo de vectorización busca sólo en los bordes de las mallas y aproxima las líneas con segmentos rectos que cruzan cada malla.

En [27], se tiene una librería compilada de 51 mallas características que se aplican en la mayoría de los casos. La malla se parte ortogonalmente en el primer borde que contiene al menos una secuencia negro-blanco-negro. El particionamiento se hace en el intervalo blanco entre los dos intervalos negros. Se aplica la operación de particionamiento recursivamente, hasta que no queden secuencias desconocidas.

Desde esta forma se va progresivamente desde un particionamiento "geográfico" de la imagen de un conjunto de mallas regulares hasta un particionamiento "geométrico" donde cada malla contiene rasgos geométricamente característicos.

Para localizar de manera precisa los ángulos entre dos líneas se utilizan mallas dinámicas, las cuales se arreglan ellas mismas en el modo más conveniente para el seguimiento de líneas. Primero, aplicando offsets a la estructura de la malla se recentran las líneas que traslapan dos mallas vecinas en una sola malla. Después, para reducir el número total de mallas usadas para describir un dibujo, se deja que las mallas dinámicas "crezcan" mediante el pegado de dos mallas adyacentes que tengan el mismo código como mallas simples.

Finalmente, a partir de la estructura de datos de mallas, Celesstin crea una estructura de líneas mediante el seguimiento de líneas de una malla a la otra. El resultado de este proceso de vectorización, es una estructura de datos que describe la

imagen como segmentos de diferentes tipos (líneas delgadas, líneas gruesas y contornos de regiones negras), y uniones entre estos segmentos.

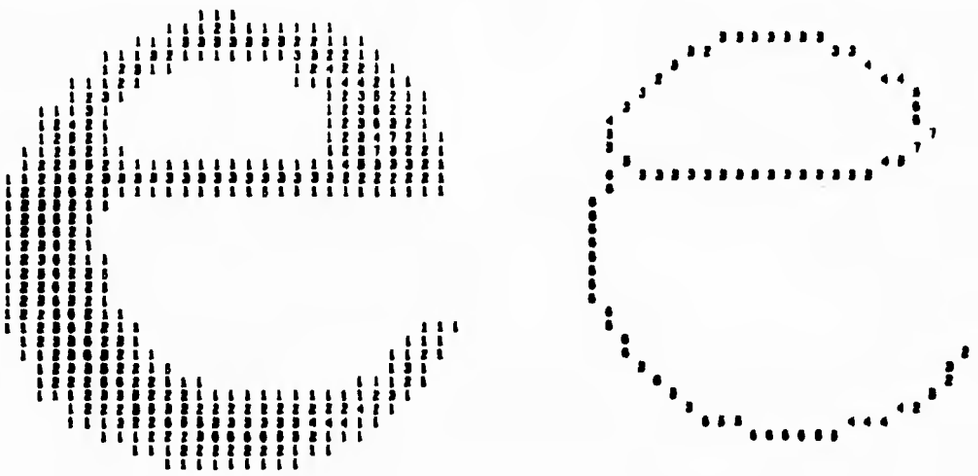
5.2.4 Transformadas de Distancia (Distance Transforms)

Se comienza por asociar cada pixel "encendido" ('1') con la distancia que lo separa del más cercano pixel "apagado" ('0'). Un proceso posterior puede realizarse en estas distancias para inferir líneas y su ancho asociado. Este método determina con exactitud líneas centradas (centerlines) pero ocupa mucha memoria y tiempo de procesamiento, y requiere muchas pasadas sobre un mapa de bits de valores enteros.

Existen dos métodos generales para obtener la transformada de distancia. Una de ellas es similar al método de thinning. En cada iteración, los límites son "borrados" de las regiones, pero, en lugar de cambiar el valor de cada pixel borrado" a '0', son cambiados al valor de la distancia que tienen con respecto al límite original. Por lo tanto, en la primera iteración (examinando máscaras de tamaño 3 x 3), los límites borrados son aquellos con valor '0'. En la segunda iteración, todos los pixeles a ser borrados son los que tienen valor '1' de distancia vertical u horizontal, o '2' para la distancia diagonal, al límite original. Este proceso continúa hasta tener una figura de un pixel de ancho.

El segundo método requiere un número fijo de pasadas por la imagen. Para obtener la aproximación entera de la distancia Euclideana a los límites, son necesarias dos pasadas. La primera pasada procede de arriba a abajo, de izquierda a derecha. Las distancias son propagadas de forma similar al método anterior, pero debido a la dirección de scaneo, estos valores son sólo temporales, pues únicamente contienen información de distancia desde arriba y a la izquierda. La segunda pasada procede en orden contrario, para tomar en cuenta ahora la información de distancia desde abajo y a la derecha [15, 33]. El resultado es una imagen con los valores de las distancias (figura 5.2a), para después eliminar todos aquellos límites cuya distancia sea "muy corta" (figura 5.2b).

Un método similar es la Propagación de Distancias Direccional (DDP) [4]. Es una técnica para determinar la longitud de un patrón medido en 4 direcciones específicas. Esto se realiza mediante un scaneo hacia adelante y hacia atrás de la imagen. La distancia desde el punto de inicio del patrón es incrementada conforme avanza el scaneo hacia adelante, y la distancia extrema del punto final es propagada al primer punto en el scaneo hacia atrás.



a) b)

Figura 5.2 a) Valor de la Distancia al límite, b) Resultado.

La técnica está basada en una idea que recuerda la transformada de distancia, pero difiere de esta en que, mientras la última determina la distancia interna para extraer un "esqueleto", la actual determina la longitud contigua máxima de un patrón en una dirección específica.

5.2.5 Aproximación Poligonal

La aproximación poligonal es un método comúnmente usado para la vectorización donde los primitivos básicos lineales son restringidos a segmentos rectos de línea [5, 13].

La aproximación poligonal es la aproximación más común para obtener rasgos a partir de curvas. El objetivo es, dada una curva, aproximarla con líneas rectas conectadas de tal manera que el resultado es muy cercano al original, pero la descripción es más simple. Se puede elegir el grado de aproximación especificando una medida del máximo error a partir del original, mientras menor sea el error, es mayor la aproximación, pero también es mayor el número de líneas requeridas [15].

El método más simple comienza con la conexión de un segmento recto de línea entre los puntos finales de los datos (figura 5.3a). Se mide la distancia perpendicular desde el segmento a cada punto de la curva (figura 5.3b). Si una distancia es mayor a cierto umbral elegido, el segmento es reemplazado por dos segmentos a partir de los puntos finales del segmento original y el punto de la curva donde la distancia a este segmento es mayor (figura 5.3c). Este proceso se repite de la misma forma para cada porción de la curva que se encuentre entre dos puntos finales del polígono, y las

iteraciones terminan cuando todos los segmentos están dentro del umbral de error de la curva (figura 5.3d).

Existen métodos más sofisticados que producen mejores aproximaciones, pero también necesitan más tiempo de cómputo y resultan más complejos [15].

Otra forma de aproximar segmentos curvos es la aproximación mediante arcos de circunferencia [3]. En general se seleccionan tres puntos de cada curva, dos cerca de las puntas y uno aproximadamente en el centro. Con esta información, se calculan el centro y el radio de un círculo que pase por esos puntos. Después el segmento es dividido en mitades y se calcula el radio para cada segmento. Si los valores de los radios concuerdan con cierta tolerancia, entonces el segmento se considera un segmento circular.

Además de los métodos de aproximación poligonal y arcos de circunferencia existen otros métodos de aproximación a curvas como son los splines, los b-splines, etc. [13, 15, 16, 17].

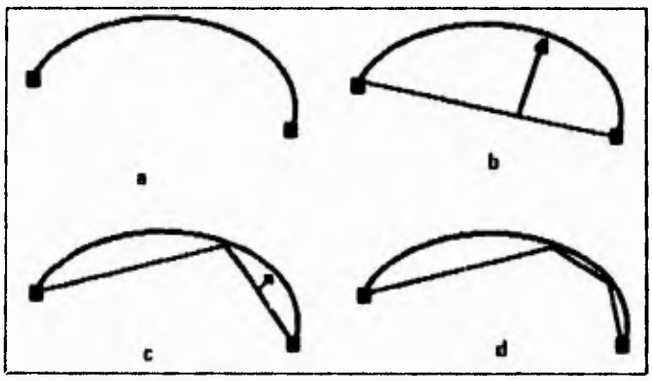


Figura 5.3 a) Puntos terminales de la curva, b) Normal a la distancia entre los puntos terminales, c) Creación de dos rectas y nueva normal, d) Nueva aproximación.

5.2.6 Segmentación Poligonal (Polygonal Segmentation)

Intenta descomponer objetos raster directamente en polígonos rellenados (filled polygons) o posiblemente otras figuras rellenadas. Esta descomposición es guiada por la detección de esquinas en los límites del objeto. Los polígonos rellenados son entonces ensamblados en líneas, uniones, etc. Este método es muy rápido, pero mantener una descripción precisa de las conectividad de las entidades es usualmente problemático [1].

5.2.7 Transformada de Hough

La transformada de Hough es un método que sirve para obtener los parámetros que caracterizan un primitivo gráfico (línea recta, curva, círculo, arco, etc.). Esto lo hace mediante el mapeo de todos los puntos que conforman dicho primitivo a un espacio paramétrico multidimensional.

Cada punto del primitivo al mapearse dentro de dicho espacio generará un primitivo paramétrico particular (e.g. un punto en el espacio cartesiano generará una parábola en el espacio polar), y todos estos primitivos paramétricos se intersectarán en un punto del espacio paramétrico, correspondiente al valor de los parámetros característicos del primitivo original.

Existen en la literatura muchas implementaciones de la transformada de Hough para el reconocimiento de líneas, curvas y otros patrones [14 - 17, 42 - 45].

Este método se decribirá con mayor detalle en el siguiente capítulo.

Existen muchos otros algoritmos para la detección de líneas [18, 29, 15, 45], pero los anteriores son los más representativos dentro del reconocimiento de segmentos de línea para el análisis de documentos y de dibujos de ingeniería. Existen además otro tipo de aproximaciones como son: las redes neuronales, la lógica difusa, etc.

5.3 Características de la Transformada de Hough

La transformada de Hough provee un método para detectar líneas rectas en imágenes digitales, y ha sido extendido para detectar muchas formas definidas analíticamente, como son círculos y elipses [12-17, 36-37, 42-45].

Se ha probado que la transformada de Hough trabaja en presencia de ruido o incluso cuando las partes de un objeto están obscurecidas o ausentes. En consecuencia, ha encontrado mucho éxito en muchos campos [12].

La transformada de Hough (HT) fue introducida como un método para detectar patrones complejos de puntos en una imagen binaria. Esto lo logra determinando valores específicos de parámetros que caracterizan estos patrones. Patrones extendidos espacialmente (dibujos formados por líneas) son transformados de tal forma que producen rasgos espacialmente compactos en un espacio de valores paramétricos posibles (es decir, producen rectas o curvas). La HT convierte el difícil problema de detección global en el espacio de la imagen en uno más fácil de resolver, la detección local de máximos (peaks) en un espacio paramétrico [14].

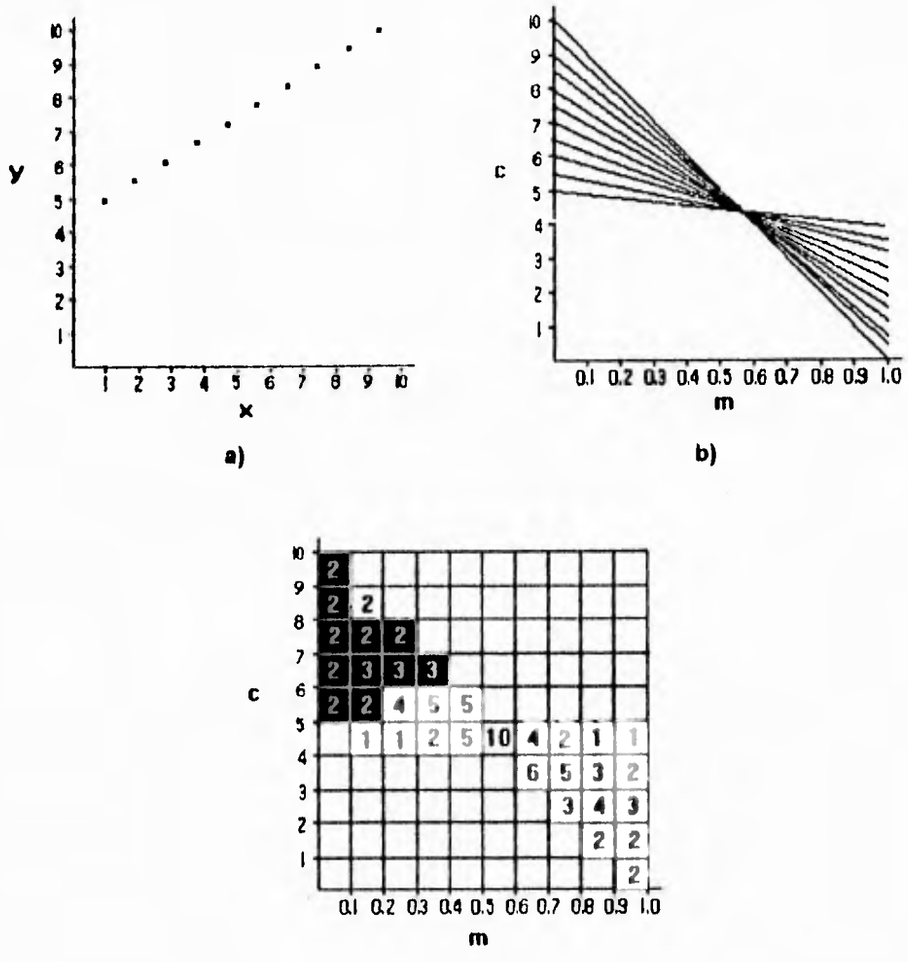


Figura 5.4 a) Puntos colineales, b) Transformada de Hough, c) Arreglo acumulatorio.

Las ideas clave del método pueden ilustrarse considerando la identificación de conjuntos de puntos colineales en una imagen (figura 5.4a). Un conjunto de puntos de la imagen (x,y) que forman una línea recta pueden ser definidos por la relación f tal que:

$$f((\hat{m},\hat{c}),(x,y)) = y - \hat{m}x - \hat{c} = 0 \qquad (5.1)$$

donde m y c son los parámetros, pendiente y ordenada al origen, que caracterizan la línea. La ecuación 5.1 mapea cada valor de la combinación paramétrica (^m,^c) a un conjunto de puntos de la imagen. Se usa el símbolo de "sombrero" (^) para denotar cantidades en el dominio del mapeo. La HT usa la idea de que la ecuación 5.1 puede ser vista como una constante entre los puntos de la imagen y puntos paramétricos, y

por lo tanto, puede interpretarse como la definición de uno de muchos mapeos entre los puntos de la imagen a un conjunto de posibles valores paramétricos. Esto corresponde a calcular los parámetros de todas la líneas rectas que pertenecen al conjunto y que pasan por un punto de la imagen (^x,^y). Esta operación es llamada proyección inversa (backprojection) del punto de la imagen y la relación que la define es:

$$g((\hat{x},\hat{y}),(m,c)) = \hat{y} - \hat{x}m - c = 0 \qquad (5.2)$$

En el caso de una línea recta cada punto de la imagen (^x,^y) se proyecta inversamente o define una línea recta en (m,c) del espacio paramétrico. Los puntos que son colineales en el espacio de la imagen se intersectan en un punto en común del espacio paramétrico, y las coordenadas de este punto paramétrico caracterizan la línea recta que conecta a estos puntos de la imagen (figura 5.4b). La HT identifica estos puntos de intersección en el espacio paramétrico. La determinación del punto de intersección en el espacio paramétrico es una operación local y debe ser considerablemente más fácil que detectar patrones de puntos extendidos en el espacio de la imagen.

La extensión del método para detectar curvas paramétricamente definidas en la imagen que no sean líneas rectas implica: Puntos de la imagen en una curva caracterizada por n parámetros, $a_1,...a_n$, pueden ser definidos por una ecuación de la forma:

$$f((\hat{a}_1,...,\hat{a}_n),(x,y)) = 0 \qquad (5.3)$$

Intercambiando los roles de parámetros y variables, la ecuación 5.3 puede ser usada para dar la definición de la proyección inversa que mapea puntos de la imagen al espacio paramétrico:

$$g((\hat{x},\hat{y}),(a_1,...,a_n) = 0 \qquad (5.4)$$

La ecuación 5.4 mapea una hypersuperficie en el espacio paramétrico n-dimensional. Los parámetros más probables para imágenes curvas son indicados por la intersección de varias de estas hypersuperficies.

Este método puede extenderse también a la detección de formas arbitrarias [14].

Con el propósito de calcular la HT en una computadora digital, el espacio paramétrico continuo es usualmente considerado como la unión de un número de

regiones de tamaño finito. En implementaciones estándar, el espacio se particiona en rectángulos multidimensionales de tamaño adecuado, y cada rectángulo es asociado con un elemento de un arreglo multidimensional. Los elementos del arreglo actúan como contadores y son incrementados cuando una hypersuperficie, desde la proyección inversa a un punto de la imagen, pasa a través de la región de espacio paramétrico asociado con el elemento.

El arreglo es llamado arreglo acumulatorio (accumulator array) (figura 5.4c). Un máximo en la cuenta es el lugar obvio donde se intersectan varias superficies. La detección de la intersección de hypersuperficies es entonces aproximada por la detección de máximos locales en el número de cuentas del acumulador. Esta detección es mucho más fácil que el cálculo de la intersección de patrones extendidos de puntos en el espacio de la imagen. El tamaño del máximo es una función del número de puntos que componen la línea, el número de otros puntos en la imagen y la elección del tamaño de la celda paramétrica. Este tamaño es elegido de acuerdo a la precisión requerida para la estimación de los parámetros.

El método HT tiene muchas características deseables. Primero, cada punto de la imagen es tratado independientemente y por lo tanto, el método puede ser implementado usando más de una unidad de procesamiento, esto es, el procesamiento en paralelo de todos los puntos es posible. Segundo, su independiente combinación de evidencia indica que puede identificar figuras parciales o ligeramente deformadas, ya que en primer lugar el tamaño del máximo paramétrico es directamente proporcional al número de puntos que concuerdan en él. Tercero, el método es muy efectivo a la adición de datos aleatorios producidos por una segmentación pobre de la imagen. Y finalmente, la HT puede acumular simultáneamente evidencia de varios ejemplos de una clase de figura particular que ocurra en la misma imagen. Generalmente cada instancia de la figura produce un máximo distinto en el arreglo acumulador.

La principal desventaja de la implementación estándar de la HT son los altos requerimientos computacionales y de almacenamiento. La determinación de q parámetros a resolverse en A intervalos requiere un acumulador de A^q elementos. Esto puede ser prohibitivo si alguno de los dos es grande. El mayor costo computacional del algoritmo es el cálculo de las intersecciones. En el caso más simple las superficies paramétricas se expanden (q-1) veces de las dimensiones paramétricas q [14].

5.4 Propuesta e Implementación de un Nuevo Algoritmo de Reconocimiento y Vectorización

El algoritmo que se implementó realiza una transformada de Hough para el reconocimiento de líneas, y una clasificación de pixeles de acuerdo a sus vecinos para localizar los segmentos de línea dentro de la imagen.

La transformada de Hough [14] consiste en la transformación de una línea en el espacio de coordenadas cartesianas a un punto en el espacio de coordenadas paramétricas. La representación paramétrica de la línea recta presentada en la ecuación 5.1 tiene varias desventajas, sobre todo cuando la pendiente de la línea se aproxima a los 90°. Por lo tanto, se utiliza una representación paramétrica polar.

Una línea recta se decribe polarmente de la siguiente manera:

$$\rho = x \cos\theta + y \sin\theta \qquad 5.5$$

donde ρ es la distancia normal desde el origen hasta la línea, y θ es el ángulo de esta normal con respecto al eje x. La transformada de Hough de la línea es simplemente un punto en las coordenadas (ρ, θ) en el dominio polar, mientras que un punto describe una curva en este dominio.

Un conjunto de puntos colineales en el dominio cartesiano generará una familia de curvas en el dominio polar. Estas curvas se intersectarán en un solo punto, aquel cuyos parámetros correspondan a la línea recta que forman en el dominio cartesiano. De esta manera, para encontrar los parámetros que describen una línea, basta encontrar el punto de intersección de las curvas que generan los puntos de dicha línea en el dominio polar.

Con el propósito de encontrar las intersecciones dentro del espacio paramétrico, se crea un arreglo de celdas correspondientes al sistema de coordenadas ρ-θ. Cada punto negro en el dominio de la imagen es transformado en una curva en el dominio polar. Si un elemento de la curva cae en el valor de una celda, esa celda en particular se incrementa en 1. Una vez que todos los puntos son transformados, las celdas ρ-θ son examinadas. Valores muy grandes en las celdas corresponden a puntos colineales, que probablemente corresponden a una línea recta, descrita por los parámetros ρ-θ correspondientes a esa celda. Valores pequeños generalmente indican puntos aislados que deben de ser borrados.

Las características que debe tener la imagen, para poder aplicar el algoritmo son:

- Debe de tener la menor cantidad de ruido (Imagen Filtrada).
- Unicamente debe haber segmentos de línea (Imagen segmentada).
- El ancho de los segmentos de línea debe ser de un pixel (Procesamiento de "Adelgazamiento" (Thinning)).
- El algoritmo no identifica líneas curvas.

Este algoritmo fue desarrollado originalmente por Duda & Hart [35, 36]. El origen de la imagen se establece en la esquina inferior izquierda. Las cooordenadas discretas del punto de la imagen (j, k) son:

$x_k = k - ½$

$y_j = J + ½ - j$

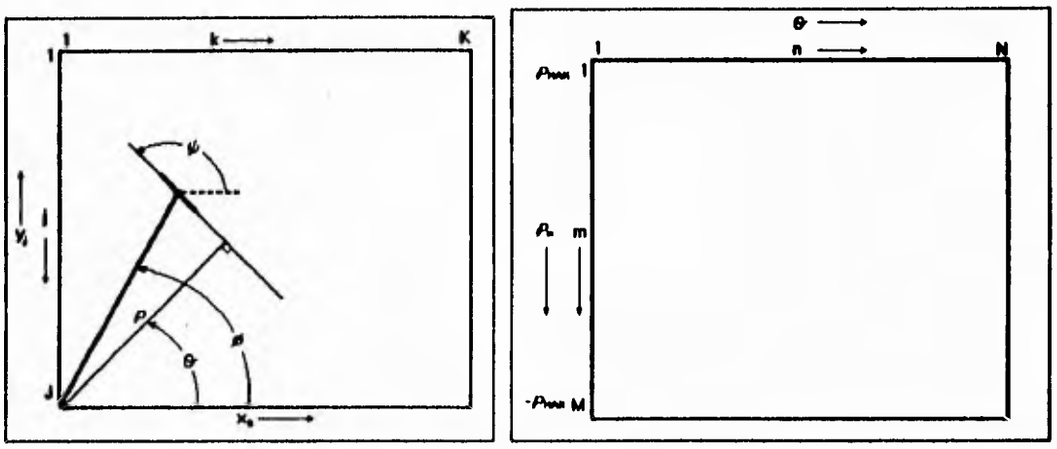


Figura 5.5 a) Definición de parámetros, b) Arreglo de Hough.

Considerese el segmento de línea en la imagen binaria F(j, k), figura 5.5a, el cual contiene un punto de coordenadas (j, k) con un ángulo ϕ con respecto al eje horizontal. Cuando este segmento de línea es proyectado, intersecta una línea normal de longitud ρ que sale del origen con un ángulo θ con respecto al eje horizontal. El arreglo de Hough, H(m, n), consiste en celdas de valores correspondientes a las variables ρ_m y θ_n. Donde:

$-\rho_{max}/2 \leq \rho_m \leq \rho_{max}$

$-\pi/2 \leq \theta_n \leq \pi$

donde

$$\rho_{max} = [(x_K)^2 + (y_1)^2]^{1/2}$$

Para la interpretación es conveniente adoptar los límites simétricos de la figura 5.5b, y hacer M y N enteros impares para que el centro del arreglo de Hough represente $\rho_m = 0$ y $\theta_n = 0$.

El algoritmo se desarrolla como sigue:

1. Inicializar el arreglo de Hough a cero.
2. Para cada punto (j, k) que sea igual a 1, calcular:

$$\rho(n) = x_k \cos \{ \theta_n \} + y_j \sin \{ \theta_n \}$$

donde

$$\theta_n = \pi - (2\pi (N - n)) / (N - 1)$$

que es incrementado en el rango $1 \leq n \leq N$ bajo la restricción:

$$\phi - \pi/2 \leq \theta_n \leq \phi + \pi/2$$

donde

$$\phi = \tan^{-1} \{ y_j / x_k \}$$

3. Determinar el índice m del correspondiente valor

$$m = [M - ((\rho_{max} - \rho(n))(M - 1)) / (2\rho_{max})]_N$$

donde $[.]_N$ denota el entero más cercano a este argumento.

4. Incrementar el arreglo de Hough

$$H(m, n) = H(m, n) + 1$$

En el algoritmo original [35, 36] M y N son de tamaño 127, para imágenes de 512 x 512. Sin embargo, para tener un mejor aprovechamiento de la memoria, realizamos una modificación de este algoritmo en base a la Transformada Adaptativa de Hough, desarrollada por J. Illingworth y J. Kittler. [37].

En la transformada Adaptativa de Hough (Adaptative Hough Transform AHT), se utiliza un arreglo pequeño, por ejemplo, de 9 x 9. Una vez llenado el arreglo, se busca el máximo valor, el cual será usado para reajustar los límites de los parámetros que definen el arreglo. El centro del nuevo rango de parámetros es colocado en el centroide de este valor máximo. Esto es, el arreglo se utiliza como una ventana que va adaptando sus parámetros de acuerdo a la localización del máximo más significativo.

Con esto, el número de operaciones a realizar y la capacidad de memoria requerida se reducen, además de que se consigue una mayor precisión en el cálculo de los valores de los parámetros.

En [37], se utiliza la descripción cartesiana de la línea y un algoritmo de componentes conectados para ajustar los parámetros del arreglo. En el presente algoritmo continuamos con la descripción polar de la línea y un redondeo para el ajuste de parámetros.

Este ajuste de parámetros se realiza una vez localizado el máximo valor. Este valor queda como centro del nuevo arreglo y los límites se ajustan cierto intervalo antes y después de este valor central, tanto para ρ como para θ. El intervalo se calcula de acuerdo a la última división de los límites de los parámetros, esto es:

i = (límite superior - límite inferior) / Número de celdas del arreglo

por lo que los nuevos valores de los límites parámetricos son:

límite inferior = centro - i
límite superior = centro + i

La transformada se aplica para cada línea que se presente en el dibujo. El hecho de ajustar los parámetros del arreglo de Hough alrededor del valor máximo, hace que el número de puntos a evaluar y localizar en el arreglo se reduzca considerablemente, y que la precisión en el valor de los parámetros encontrados sea mayor en cada iteración. El número de iteraciones en el ajuste de parámetros del arreglo fue 4, el mínimo con el cual se obtuvo una precisión óptima. La aplicación del algoritmo sobre la imagen se presenta en el siguiente diagrama de flujo:

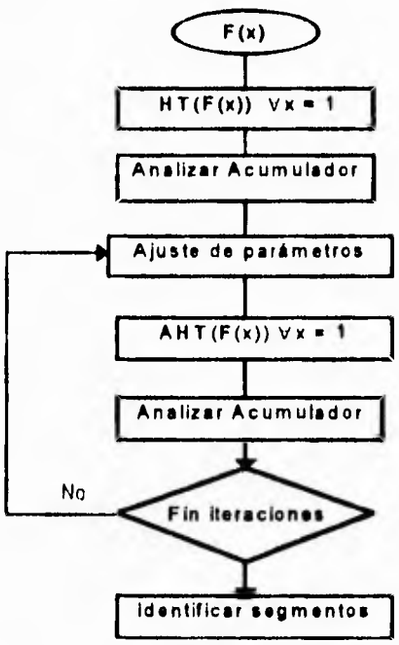


Figura 5.6 Diagrama de Flujo de la Aplicación de la Transformada de Hough

Donde F(x) es la imagen, HT( ) es la transformada de Hough de Duda & Hart, y AHT( ) es la transformada adaptativa de Hough.

Para la localización del(los) segmento(s) de línea dentro de la imagen se utiliza una clasificación de pixeles, similar a la propuesta en [3]. Mediante una ventana de 3 x 3 se recorre la imagen completa, clasificando los pixeles de acuerdo a sus vecinos. Esto es:

0 - Pixeles de fondo (blancos).

1 - Pixeles intermedios.

2 - Pixeles finales.

3 - Pixeles esquinas o puntas.

4 - Pixeles de unión de líneas.

El análisis de los pixeles, para poder clasificarlos dentro de alguno de estos tipos, se realiza mediante su número código decimal [17, 6].

Los vecinos de un pixel P encendido ('1') se representan de la siguiente manera:

3	2	1
4	P	0
5	6	7

Figura 5.7 Vecinos del Pixel P

El número código decimal (NCD) para el pixel P se calcula de la siguiente manera:

$$\sum_{i=0}^{7} 2^i n_i \qquad (5.6)$$

Por lo tanto, cada combinación de pixeles encendidos tiene un NCD particular. Se crearon listas con los NCD de los tipos de pixeles terminales a clasificar (finales, esquinas o uniones), por lo que, si el NCD del pixel que se está analizando pertenece a alguna lista, se le cambia su valor de '1' por el correspondiente ('2', '3' o '4').

En la figura 5.8 se muestran algunos ejemplos de pixeles terminales. Aquellos cuadros vacíos representan pixeles blancos, los cuadros con números representan pixeles negros, y el cuadro P es el pixel terminal. El inciso (a) es un punto final. Los incisos (b) al (e) son ejemplos de puntos esquina. Los incisos (g) al (l) son puntos de unión.

Esta clasificación facilita la extracción de los segmentos. Una vez localizada una línea mediante el algoritmo, se procede a encontrar aquellos puntos terminales ('2', '3' o '4') que satisfagan la ecuación de dicha línea, con lo cual se tendrán el(los) punto(s) de inicio y fin del(los) segmento(s) de línea de la imagen. Estos segmentos son eliminados para que no influyan en la búsqueda de la siguiente línea.

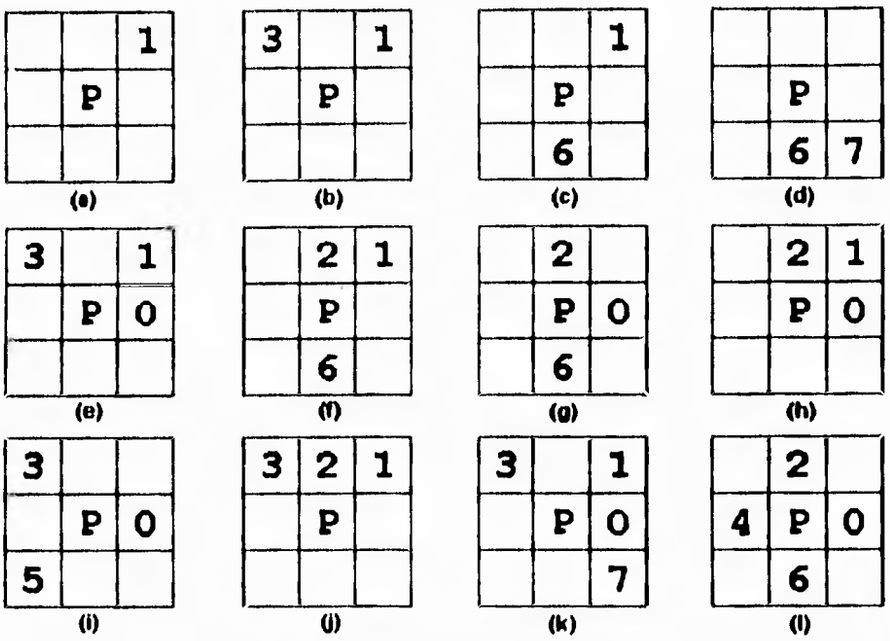


Figura 5.8 Ejemplos de Pixeles Terminales

Debido a que el borrado de los segmentos en ocasiones puede borrar puntos de otras líneas o dejar puntos que debieron borrarse, el análisis de esquinas se realiza despúes de cada extracción de líneas de la imagen.

La estrategia de búsqueda consiste en tres etapas: búsqueda de líneas verticales, búsqueda de líneas horizontales y búsqueda de líneas inclinadas.

Esto se realiza debido a que las líneas verticales y horizontales son más fáciles de localizar y más abundantes en los dibujos y diseños. Para su localización se mantiene fijo el parámetro central del arreglo de Hough en teta = 0° (para líneas verticales, puesto que teta es el ángulo de la normal a la recta) y en teta = 90° (para líneas horizontales), y se localizan los valores máximos ubicados en esas posiciones.

Esto contribuye, además, a que las líneas inclinadas del dibujo queden segmentadas del resto, lo cual facilita su extracción.

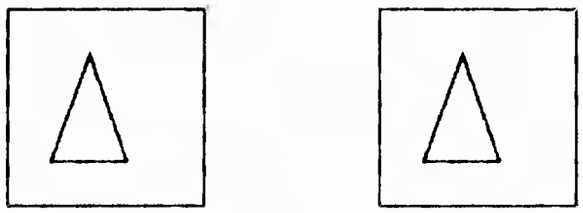
La figura 5.9 muestra un diagrama de flujo de la estrategia de búsqueda del algoritmo. En cada uno de los cuadros denominados búsqueda" se realiza el ciclo del diagrama de flujo anteriormente descrito en la figura 5.6.

Figura 5.9 Estrategia de Búsqueda

5.4.1 Resultados de la Implementación del Algoritmo de Vectorización

El algoritmo fue implementado en ANSI C [40] y probado en una computadora Sun SPARC station 2. Las imágenes son de tamaño 100 x 100 pixeles.

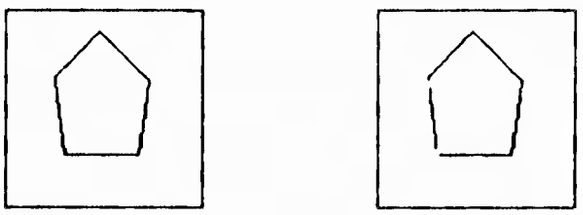
A continuación veremos algunos ejemplos de la aplicación del algoritmo. En la figura 5.9a se presenta la imagen de un triángulo. Podemos ver que su vectorización es exacta.



a) b)

Figura 5.9 a) Original, b) Vectorización

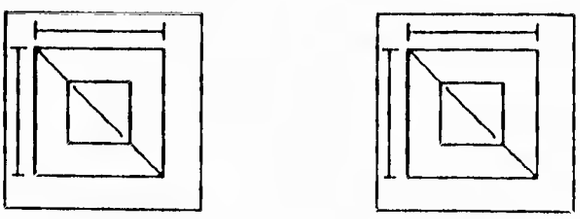
El ejemplo de la figura 5.10a presenta un pentágono. La vectorización (figura 5.10b) es casi correcta de manera perceptual, y su corrección en un editor sería mínima. La línea que aparece incompleta se debe a que el método trata de aproximarla con líneas verticales antes que con una línea inclinada, dado su ángulo de inclinación.



a) b)

Figura 5.10 a) Original, b) Vectorización

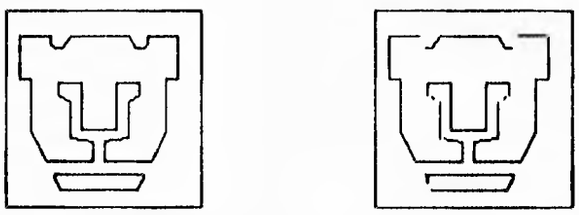
La figura 5.11a muestra una serie de cuadros, de lo que pudiese corresponder a una imagen del tipo de los dibujos de ingeniería.El resultado de la vectorización es 100% correcto, a pesar de contar con segmentos muy pequeños y con líneas muy cercanas a la intersección entre sí.



a) b)

Figura 5.11 a) Original, b) Vectorización

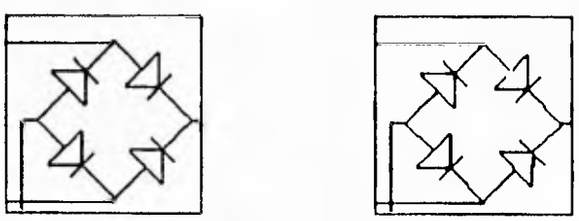
La figura 5.12 muestra el escudo de los Pumas de la Universidad. Este ejemplo es sumamente complicado ya que cuenta con muchos segmentos pequeños (menores de los 6 pixeles que se establecieron como límite) que no estan alineados entre sí. Sin embargo, podemos ver que el algoritmo efectúa una vectorización muy correcta, presentandose errores perceptuales en algunos de los segmentos pequeños. Esta clase de imágenes ejemplifican la complejidad de encontrar los puntos de intersección dentro del arreglo de Hough, y la ventaja de utilizar una estrategia como la que planteamos en el algoritmo. Al buscar primero las líneas verticales y horizontales, cubrimos una buena parte de los segmentos que conforman la imagen sin posibilidad de error (esto es, identificar un segmento inclinado como una línea vertical u horizontal), y dejando libres los segmentos inclinados para su posterior identificación.



a) b)

Figura 5.12 a) Original, b) Vectorización

La figura 5.13 presenta un ejemplo de un diagrama propio de los dibujos de ingeniería, un puente de diodos. Este ejemplo resulta complejo también, por la cantidad de intersecciones, que como ya se mencionó en el capítulo 5.2 son los casos más complicados de identificar, y por la presencia de muchas líneas inclinadas de tamaño similar. El algoritmo identifica absolutamente todos los segmentos, y sólo cambian de manera perceptual algunos puntos de interección, es decir, los segmentos de línea comienzan y/o terminan en un punto vecino al punto del que parten en la imagen original.



a) b)

Figura 5.13 a) Original, b) Vectorización

Como se puede apreciar, los resultados son muy aproximados a los originales de manera perceptual, algunos de ellos exactamente iguales (figuras 5.9 y 5.11). La estrategia

de búsqueda nos garantiza que los segmentos horizontales y verticales se identificarán correctamente, lo cual es importante porque son en los que, de manera perceptual, son más obvios los errores. El uso de la transformada adaptativa de Hough nos permite ser más exactos en los valores de los parámetros, mientras que con la transformada de Hough todos los errores de aproximación de los parámetros tienen que ser corregidos por un método de búsqueda de segmentos de línea, más complicado e "inteligente". La clasificación de pixeles terminales para la identificación de los segmentos resulta un método sencillo y efectivo para este propósito, en comparación con métodos más complicados, como podrían ser el seguimiento de contornos o la detección de orillas [13, 35].

Los principales errores en las otras figuras corresponden a segmentos de 2 a 5 pixeles, que están fuera del rango mínimo que se estableció (6 pixeles para el segmento de línea más pequeño), como es el caso de la figura 5.12, donde aparecen huecos que corresponden a líneas muy pequeñas. Por otro lado, detalles de diferencia entre las vectorizaciones y las imágenes originales, son consecuencia de ciertas consideraciones del método de búsqueda de segmentos en la imagen, el cual es muy sencillo y realiza decisiones que para el ojo humano pudiesen ser errores de interpretación. Teniendo un método más inteligente de búsqueda de segmentos estos errores serían fácilmente corregidos.

El tiempo de ejecución (Tabla 5.2) es bastante bueno, tomando en cuenta la complejidad de algunos de los dibujos, siendo el más largo y complicado el de la figura 4 (Puma) con menos de 2.5 minutos. Uno de los problemas que se presentan con respecto al tiempo de ejecución, es el hecho de que se pierde mucho tiempo en la búsqueda de un máximo real dentro del arreglo de Hough, es decir, de entre todos los máximos que se presentan en el arreglo, sólo unos corresponden a segmentos "reales", mientras que otros corresponden a puntos colineales que no forman un segmento. Esta búsqueda de máximos dentro del arreglo de Hough puede hacerse más eficiente aplicando ciertos algoritmos de componentes conectados y búsquedas piramidales [14].

Imagen	Tiempo (segundos)
Figura 1 (Triángulo)	18.68
Figura 2 (Pentágono)	26.88
Figura 3 (Cuadros)	48.01
Figura 4 (Puma)	147.81
Figura 5 (Puente de Diodos)	109.71

Tabla 5.2 Tiempos de Ejecución

6. RESULTADOS, CONCLUSIONES Y PERSPECTIVAS

El análisis automático de documentos se ha ido convirtiendo en una herramienta necesaria para poder hacer compatible el uso de documentos en papel (como comúnmente los utiliza el hombre) con el uso de sistemas de manejo de documentos por computadora (procesadores de palabras, hojas de cálculo, multimedia, CD-ROM, etc.).

El desarrollo de sistemas que realicen esta clase de análisis, y de las herramientas que utilizan estos sistemas, es muy reciente, lo cual implica que todavía existen muchas cosas por hacer y desarrollar.

Los sistemas de análisis de documentos, como se explica en los capítulos 2 y 3, utilizan muchas herramientas y procesos, cada uno encaminado a presentar la información en un nivel "más elevado", es decir, más cercana al tipo de información que manejamos las personas. Desde una imagen formada por puntos negros y blancos, hasta información escrita, visual o numérica. Dichas herramientas tienen propósitos específicos, y estan separadas unas de otras con el fin de poder adaptarlas a diversas necesidades. Estas herramientas realizan tareas que van desde la eliminación de puntos sobrantes dentro de una imagen (ruido), hasta la identificación de una tuerca, el logotipo de una compañía o el rostro de una persona, pasando por la separación de los elementos del documento, la identificación de caracteres, de líneas, de palabras, de figuras, etc.

En este trabajo presentamos una propuesta de dos herramientas fundamentales: la segmentación y la vectorización.

La segmentación se encarga de separar toda la información contenida en un documento para que pueda ser procesada por separado y por el método que le corresponda. Así, los caracteres pasaran a reconocimiento de caracteres, los gráficos a la vectorización, y las imágenes a la compresión o algún otro método. El esquema propuesto en el capítulo 4, sobresale por su sencillez, lo cual lo hace adaptable a cualquier aplicación. Retoma un tipo de segmentación que no depende de patrones de orden dentro de las imágenes, que es la segmentación por componentes conectados. Y presenta un método original para encontrar dichos componentes.

Este método tiene la característica principal de que no requiere de una programación compleja para implementarse, por lo que se puede utilizar para muchas de las aplicaciones que se tienen para los componentes conectados, y no difiere mucho en su desempeño de lo que hace el algoritmo que se utiliza en la literatura.

El algoritmo de vectorización propuesto en el capítulo 5 retoma y combina dos de los métodos más sencillos y efectivos, la transformada de Hough y la transformada

adaptativa de Hough, para el reconocimiento de líneas, y la idea de la clasificación de pixeles terminales (pixeles que identifican las orillas y esquinas de los segmentos de líneas) para la identificación de los segmentos y su posterior codificación como vectores de línea.

La transformada de Hough es tan sencilla o compleja como se le quiera programar, y tiene la ventaja de que puede identificar patrones aún con la presencia de ruido o defectos en la captura. Por otro lado, la transformada adaptativa de Hough presenta una aproximación más exacta para la localización de los parámetros de las líneas y un ahorro considerable en la capacidad de memoria requerida. Nuestro esquema presenta una combinación de ambos métodos. Ademas proponemos una estrategia de búsqueda que comience con el reconocimiento de líneas verticales y horizontales, cuyos parámetros son más sencillos de identificar, y son más abundantes dentro de las imágenes gráficas, lo cual evita errores de identificación y deja las líneas inclinadas, que son más difíciles de identificar, aisladas.

A continuación podemos ver dos ejemplos de la aplicación de los esquemas de segmentación y vectorización propuestos, trabajando conjuntamente como lo harían dentro de un sistema de análsis de documentos.

En la figura 6.1 a pesar de la variedad de tipos de letra y su tamaño, la segmentación resultó correcta, resaltando en hecho de la segmentación de las etiquetas que acompañan al gráfico, lo cual era importante y deseable para su posterior vectorización. Unicamente los paréntesis se consideraron como posibles líneas.

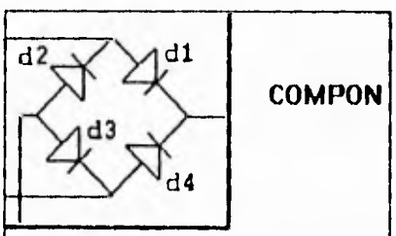
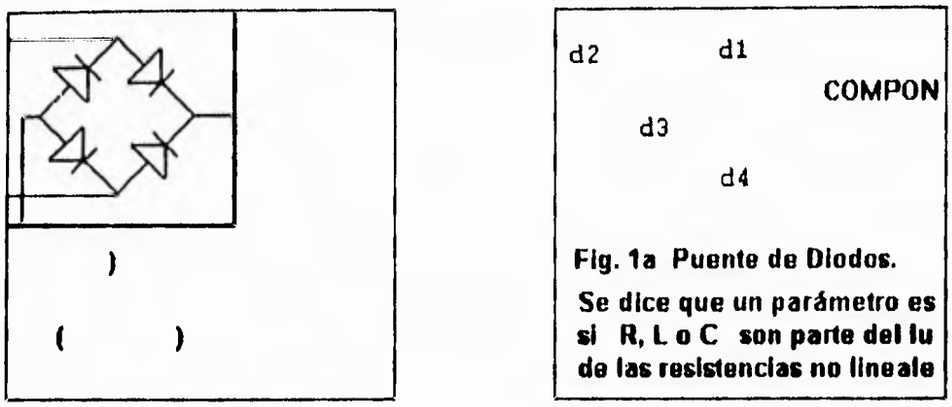


Fig. 1a) Puente de Diodos.

Se dice que un parámetro es si (R, L o C) son parte del lu de las resistencias no lineale

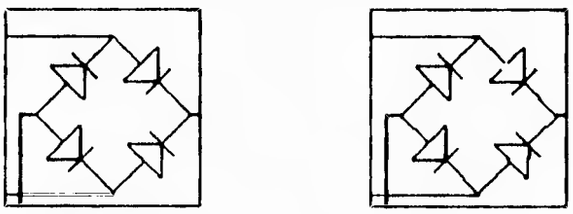
a)



b) c)

Figura 6.1 a) Documento Original, b) Gráficos segmentados, c) Caracteres segmentados

Tomando el gráfico de la figura 6.1b procedemos a vectorizarlo. La vectorización (figura 6.2) es sumamente aproximada, y las modificaciones que se tendrían que efectuar en un sistema de edición (eg. AutoCAD) son mínimas, ya que sólo consisten en la localización del punto de inicio y/o fin de algunos de los segmentos.

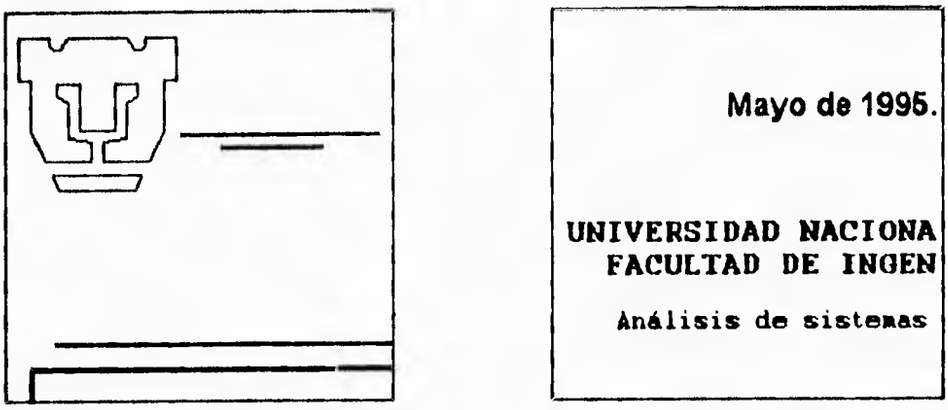


a) b)

Figura 6.2 a) Gráfico Original, b) Vectorización

La figura 6.3 sirve perfectamente para ejemplificar el uso de nuestros algoritmos. Como se aprecia, no se presentaron errores dentro de la segmentación, incluso con la variedad de tipos de letra.

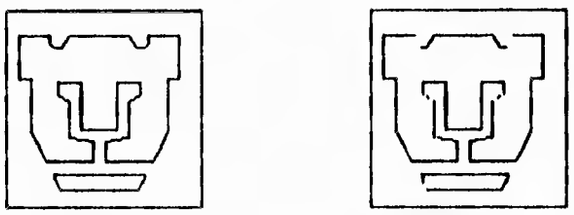
a)



b) c)

Figura 6.3 a) Documento Original, b) Gráficos segmentados, c) Caracteres segmentados

A pesar de la complejidad del gráfico (figura 6.4), el algoritmo obtuvo una excelente aproximación. Como se señala en los resultados del algoritmo de vectorización (Sección 5.1.1), es muy difícil encontrar todos los segmentos que son menores de 6 pixeles de longitud.



a) b)

Figura 6.4 a) Gráfico Original, b) Vectorización

Los resultados son muy buenos. La segmentación es correcta y rápida, tomando en cuenta que la localización de componentes conectados es la operación que más tiempo consume en estos métodos [8], y la vectorización es muy aproximada a los gráficos originales. Con estos resultados se tiene un paso importante para el análisis de métodos para optimizar el desempeño de estos algoritmos, y el desarrollo de nuevos elementos que juntos integren la estructura de un sistema de análisis de documentos.

Entre las posibles modificaciones que optimicen el desempeño de los esquemas propuestos podemos mencionar: dentro de la segmentación, el desarrollo o implementación de criterios más "inteligentes" para la decisión de cuales componentes conectados son caracteres y cuales son gráficos, así como un método para segmentar aquellos gráficos y caracteres que se encuentren conectados entre sí [3]. Dentro de la vectorización, la implementación de un método para optimizar la búsqueda de máximos dentro del arreglo de Hough [14], que es el proceso que toma más tiempo, y

alguna herramienta que pueda corregir errores dentro de la identificación de los segmentos de línea [2]. Con la implementación de herramientas de preprocesamiento y de reconocimiento de caracteres, podríamos hablar de que es factible el desarrollo de sistemas de manejo y análisis de documentos gráficos, incluso para aquellos otros tipos de documentos, por ejemplo: documentos de oficina (cartas, oficios, currículums, historiales, etc.), que tengan elementos gráficos, como logos y/o diagramas.

Cada uno de los métodos y algoritmos que se usan actualmente no son definitivos, y hoy en día ningún sistema obtiene resultados 100% exactos. En todo el mundo se sigue trabajando sobre estos problemas, y en nuestro país el interés es grande, pero poco el trabajo al respecto. Aquí comprobamos la factibilidad del desarrollo de herramientas para esta clase de aplicaciones, con un algoritmo recursivo para encontrar componentes conectados, sumamente efectivo y muy diferente del que se utiliza comúnmente.

La síntesis aquí presentada representa una base sólida para un posterior estudio de todos los aspectos que rodean al análisis de documentos. Existen en la universidad muchos estudiantes interesados en el procesamiento de imágenes, el reconocimiento de patrones, las aplicaciones multimedia, etc. , y que buscan una aplicación concreta, real y actual. Nosotros creemos que el análisis de documentos es una buena opción.

BIBLIOGRAFIA

FILIPSKI92

1) Filipski, Alan J. & Flandrena, Robert.
Automated Conversion of Engineering Drawings to CAD Form.
Proceedings od the IEEE. Vol. 80, No. 7, July 1992.

JOSEPH92

2) S.H. Joseph and T.P. Pridmore.
Knowledge-Directed Interpretation of Mechanical Engineering Drawings.
IEEE Trans. on Pattern Ana. & Machine Int. Vol 14, No. 9, Sept 1992.

KASTURI90

3) Kasturi,Rangachar & otros.
A System for Interpretation of Line Drawings.
IEEE Trans. on Patt. Ana. & Machine Int. Vol. 12, No. 10, Oct 1990.

KANEKO92

4) Kaneko, Toru.
Line Structure Extraction From Line-Drawing Images.
Pattern Recognition. Vol. 25, No. 9 , 1992.

NAGASAMY90

5) Nagasamy,Vijay and Noshir A. Langrana.
Engeneering Drawing Processing and Vectorization System.
Computer Vision, Graphics & Image Proccesing. No. 49, 1990.

CUGINI84

6) U. Cugini & otros.
Pattern Directed Restoration and Vectorization of Digitized
Engineering Drawings. Computer & Graphics. Vol. 8, No. 4, 1984.

SMITH87

7) Smith, Raymond W.
Computer Proccesing of Line Images: A Survey.
Pattern Recognition. Vol. 20, No. 1, 1987.

FLETCHER88

8) Lloyd Alan Fletcher and Rangachar Kasturi.
A Robust Algorithm for Text String Separation from mixed
Text/Graphics Images.
IEEE Trans. on Patt. Ana. & Machine Int. Vol. 10, No. 6, Nov 1988.

KAY92

9) Kay David C. & Levine Jhon L.
Graphics File Formats.
McGraw Hill, 1992.

FOLEY94

10) Foley, James , Van Dam Andries & otros.
Introduction to Computer Graphics.
Addison Wesley, 1994.

NETRAVAL88

11) Netravali Arun M. & Haskell Barry G.
Digital Pictures: Representation and Compression.
Plenum, New York, 1988.

ZAHID91

12) Zahid Hussain.
Digital Image Processing.
Ellis Horwood, England, 1991.

PAVLIDIS82

13) Pavlidis Theo.
Algoritms for Graphics and Image Processing.
Computer Science Press, 1982

ILLIN88

14) Illingworth J. & Kittler J.
A survey of the Houg Transform.
Computer Vision, Graphics, and Image Processing. Vol. 44,p. 87-116, 1988.

OGORMAN93

15) O'Gorman, Lawrence & Kasturi, Rangachar.
Document Image Analysis: A Tutorial.
Computer Engineering Technical Report, TR-93-126, June 1993.
The Pennsylvania State University.

JIAFENG92

16) Y. Jiafeng, B. CaoQing, T. Agui & T. Nagao.
The Use of Complex Transform for Extraction Circular Arcs
and Straight Lines in Engineering Drawings.
Proceedings of 11th International Conference on Pattern Recognition
Deng Haag (Netherlands), Vol. 3, No. 1, 1992.

KAMEI88

17) K. Kamei, Y. Nakamura & S. Abe.
Fast Shape Detection Using Hough Transform by Raster Operation.
Proceedings of 9th International Conference on Pattern Recognition
Rome (Italy), p. 1109-1112, 1988.

CHEN88

18) Chen, Ling-Hwei & Tsai, Wen-Hsiang.
Moment-Preserving Line Detection.
Pattern Recognition, Vol. 21, No. 1, 1988.

HAN94

19) Han,Chin-Chuan & Fan,Kuo-Chin.
Skeleton Generation Of Engineering Drawings Via Contour Matching.
Pattern Recognition, Vol. 27, No. 2, 1994.

YU94

20) Yu, Yuhong, Smal, Ashok & Seth, Sharad.
Isolating Symbols from Connection Lines in a Class of Engineering Drawings.
Pattern Recognition, Vol. 27, No. 3, 1994.

OGORMAN90

21) O'Gorman, Lawrence.
kxk Thinning.
Computer Vision Graphics & Image Processing,Vol. 51, p. 195-215, 1990.

GIBSON82

22) Gibson,L. & Lucas,D.
Vectorization of Raster Images Using Hierarchical Methods.
Computer Graphics and Image Processing, Vol. 20, 1982.

LYSAK90

23) D. Lysak & R. Kasturi.
Interpretation of Line Drawings with Multiple Views.
Proceedings of 10th International Conference on Pattern Recognition, Vol. 1, 1990.

VAXIVI92

24) Vaxivière, Pascal & Tombre, Karl.
Celesstin: CAD Conversion of Mechanical Drawings.
IEEE COMPUTER Magazine, Vol. 25, No. 7, July 1992.

FAHN88

25) C.-S. Fahn, J.-F. Wang & J.-Y. Lee.
A Topology-Based Component Extractor for Understanding Electronic Circuit Diagrams.
Computer Vision, Graphics and Image Processing, Vol. 44, 1988.

PAVLIDIS86

26) Pavlidis, Theo.
A Vectorizer and Feature Extractor for Document Recognition.
Computer Vision, Graphics and Image Processing, Vol. 35, 1986.

LIN85

27) X. Lin et al.
Efficient Diagram Understanding with Characeristic Pattern Detection.
Computer Vision, Graphics and Image Processing, Vol 30, No. 1,
Abril 1985.

WOLFE88

28) L.S. Wolfe & J. de Wyze.
An Update on Drawings Conversion.
Computer Aided Design Report, Vol. 8, No. 6, Junio 1988.

ROSENFELD82

29) A. Rosenfeld & A. C. Kak.
Digital Image Proccesing.
Vols. 1 y 2. Academic Press, New York. 1982.

ANTOINE92

30) D. Antoine, S. Collin & K. Tombre.
Analysis of technical documents: the redraw system.
Structured Document Image Analysis.
H. Baird, H. Bunke & K. Yamamoto eds., Springer-Verlag. 1992.

MASINI83

31) G. Masini & R. Mohr.
Mirabelle, a system for structural analysis of drawings.
Pattern Recognition, Vol. 16, No. 4, 1983.

STORY92

32) G. Story, L. O'Gorman, D. Fox, L.L. Schaper, y H.V. Jagadish.
The RightPages image-based electronic library for alerting and
browsing.
IEEE Computer, Vol. 25, No. 9, Sept. 1992.

BORGEFOR86

33) G. Borgefors.
Distance Transformations in digital images.
Computer Vision, Graphics, and Image Processing, Vol. 34, 1986.

KARNEY92

34) Karney, James.
Inside a color scanner.
PC Magazine, April 14, 1992.

PRATT91

35) Pratt, William K.
Digital Image Processing.
Edit. John Wiley & Sons, Inc. 1991.

DUDA72

36) R.O. Duda & P.E. Hart.
Use of the Hough Transform to detect lines and curves in pictures.
Communications of the ACM, Vol. 15, No. 1, 1972.

ILLIN87

37) J. Illingworth & J. Kittler.
The Adaptative Hough Transform.
IEEE Trans. on Pattern Analysis and Machine Intelligence
Vol. 9, No. 5, 1987.

EASTMAN90

38) C.M. Eastman.
Vector versus Raster: A Functional Comparison of Drawing Technologies.
IEEE Computer Graphics and Applications, Vol. 10, No.5 , 1990.

KAUFMAN93

39) Kaufman, Arie. Cohen, Daniel, Yagel, Roni.
Volume Graphics.
IEEE COMPUTER, Julio 1993.

KERNIGHAN91

40) Kernighan, Brian W.
El lenguaje de programacion C.
Prentice-Hall, 1991.

RIMMER92

41) Rimmer, Steve.
The Graphic File Toolkit.
Addison Wesley, 1992.

THRIFT83

42) Thrift, Philip R. & Dunn, Stanley M.
Approximating Point-Set Images by Line Segments Using a Variation of the Hough Transform.
Computer Vision, Graphics and Image Processing, Vol. 21, p. 383-394, 1983.

CASASE87

43) Casasent, David & Krishnapuram, Raghuram.
Curved Object location by Hough Transformation and Inversions.
Pattern Recognition, Vol. 20, No. 2, 1987.

MCKENZIE90

44) D.S. McKenzie & S.R. Protheroe.
Curve Description using the Inverse Hough Transform.
Pattern Recognition, Vol. 23, No. 3/4, 1990.

HAN94

45) Joon H. Han, Laszlo T. Koczy, Timothy Poston.
Fuzzy Hough Transform.
Pattern Recognition Letters,Vol. 15, p. 694-658, 1994.

ELLIMAN90

46) D.G. Elliman & I.T. Lancaster.
A Review of Segmentation and Contextual Analysis Techniques for Text Recognition.
Pattern Recognition, Vol.23, No.3/4, 1990.

FAN94

47) K.C. Fan, C.H. Liu, Y.K. Wang.
Segmentation and Classification of mixed text/graphics/image documents.
Pattern Recognition Letters, Vol. 15, p. 1201-1209, 1994.

ZLATOPO94

48) A.A. Zlatopolsky.
Automated document segmentation.
Pattern Recognition Letters, Vol. 15, p. 699-704, 1994.

SER95

49) Pui-Kin Ser & Wan-Chi Siu.
Memory compression for straight line recognition using the Hough Transform.
Pattern Recognition Letters, Vol. 16, p. 133-145, 1995.

DOWNTON90

50) A.C. Downton & C. G. Leedham.
Preprocessing and Presorting of Envelope Images for automatic sorting using OCR.
Pattern Recognition, Vol. 23, No. 3/4,1990.

DOERMANN93

51) David Doermann, Ehud Rivlin, Isaac Weiss.
Logo Recognition.
Tech. Report, Documenting Processing Group, University of Maryland,1993.

JOHNSTON74

52) E.G. Johnston.
Printed Text Discrimination.
Computer Graphics and Image Processing,No. 3, 1974.

WONG82

53) K.Y. Wong, F.M. Wahl, R.G. Casey.
Block segmentation and image extraction in mixed text/image documents.
Computer Graphics and Image Processing, Vol. 20, p. 375-390, 1982.

Apéndice A. Referencias de Ilustraciones

F. 3.5 - 3.8 Fig 3, Cap. 2.3, pag. 23.
O'Gorman, Lawrence & Kasturi, Rangachar.
Document Image Analysis: A Tutorial.
Computer Engineering Technical Report, TR-93-126, June 1993.
The Pennsylvania State University.

F. 3.9 Fig. 4, pag. 50.
Vaxivière, Pascal & Tombre, Karl.
Celesstin: CAD Conversion of Mechanical Drawings.
IEEE COMPUTER Magazine, Vol. 25, No. 7, July 1992.

F. 3.10 - 3.11 Fig. 5 y Fig. 6, pag. 50.
Vaxivière, Pascal & Tombre, Karl.
Celesstin: CAD Conversion of Mechanical Drawings.
IEEE COMPUTER Magazine, Vol. 25, No. 7, July 1992.

F. 3.12 Fig. 10, pag. 985.
Kasturi, Rangachar & otros.
A System for Interpretation of Line Drawings.
IEEE Trans. on Patt. Ana. & Machine Int. Vol. 12, No. 10, Oct 1990.

F. 3.13 Tabla II, pag. 985.
Kasturi, Rangachar & otros.
A System for Interpretation of Line Drawings.
IEEE Trans. on Patt. Ana. & Machine Int. Vol. 12, No. 10, Oct 1990.

F. 3.14 Fig. 9 , pag. 1205 y Fig. 12, pag. 1207.
Filipski, Alan J. & Flandrena, Robert.
Automated Conversion of Engineering Drawings to CAD Form.
Proceedings od the IEEE. Vol. 80, No. 7, July 1992.

F. 4.1 Fig. 2, pag. 1205.
K.C. Fan, C.H. Liu, Y.K. Wang.
Segmentation and Classification of mixed text/graphics/image documents.
Pattern Recognition Letters, Vol. 15, p. 1201-1209, 1994.

F. 4.2 Fig. 5, pag. 1206.
K.C. Fan, C.H. Liu, Y.K. Wang.
Segmentation and Classification of mixed text/graphics/image documents.
Pattern Recognition Letters, Vol. 15, p. 1201-1209, 1994.

F. 4.3 Fig. 1, pag. 979.
Kasturi, Rangachar & otros.
A System for Interpretation of Line Drawings.
IEEE Trans. on Patt. Ana. & Machine Int. Vol. 12, No. 10, Oct 1990.

F. 4.4 Fig. 2, pag. 979.
Kasturi, Rangachar & otros.
A System for Interpretation of Line Drawings.
IEEE Trans. on Patt. Ana. & Machine Int. Vol. 12, No. 10, Oct 1990.

T. 5.1 Tabla 1, pag. 76.
C.M. Eastman.
Vector versus Raster: A Functional Comparison of Drawing Technologies.
IEEE Computer Graphics and Applications, Vol. 10, No.5 , 1990.

F. 5.1 Fig.11, Fig. 12 y Fig. 13, pags. 390-391.
Nagasamy, Vijay and Noshir A. Langrana.
Engeneering Drawing Processing and Vectorization System.
Computer Vision, Graphics & Image Proccesing. No. 49, 1990.

F. 5.2 Fig. 3, Cap. 2.4, pag. 31.
O'Gorman, Lawrence & Kasturi, Rangachar.
Document Image Analysis: A Tutorial.
Computer Engineering Technical Report, TR-93-126, June 1993.
The Pennsylvania State University.

F. 5.3 Fig. 2, Cap. 3.2, pag. 46.
O'Gorman, Lawrence & Kasturi, Rangachar.
Document Image Analysis: A Tutorial.
Computer Engineering Technical Report, TR-93-126, June 1993.
The Pennsylvania State University.

F. 5.4 Fig. 1, pag. 88.
Illingworth J. & Kittler J.
A survey of the Houg Transform.
Computer Vision, Graphics, and Image Processing. Vol. 44,p. 87-116, 1988.

F. 5.5 Fig. 18.4-7, pag. 616.
Pratt, William K.
Digital Image Processing.
Edit. John Wiley & Sons, Inc. 1991.

Apéndice B. Programa de Segmentación

```
/*
Programa de Segmentacion de Documentos
desarrollado por:
                    Josó Luis Carbajal Hdz.
                    Rocío Hernández Elenes
Lee una imagen de 200x300, en formato raster y
segmenta los caracteres de los elementos
gráficos
*/
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<unistd.h>

/* CONSTANTES */
#define K 200  /* Numero de renglones */
#define J 300  /* Numero de columnas */
#define PERMS 0666  /*Permisos del archivo dxf
                      Unix */

/* ESTRUCTURAS */
typedef struct rectangulo{
    int der;
    int izq;
    int ar;
    int ab;
    int area;
    struct rectangulo* sig;
};
typedef struct moda{
    int valor;
    int cuenta;
    struct moda* ant;
};

/* FUNCIONES */
void segmenta();
void pacman();
void elimina();
void redibuja();
void lee_imagen();

/* VARIABLES */
FILE *asc,*nuevo;  /* Archivos */
int imgmap[K][J];  /* Imagen */
int arriba,abajo,izq,der; /* Rectangulo */

/* PROGRAMA PRINCIPAL */

int main(argc,argv)
int argc;
char *argv[];
{
   long u,total=0.0;
   int arch;

/* Se inicializa el reloj */
      u = clock();

/* Se abre el archivo con la imagen */
      asc = fopen(argv[1],"r");
      if (asc == NULL)
      {
        printf("\n Error. Archivo no existe");
        return(0);
      }
      lee_imagen();

/* Se obtienen todos los segmentos de lineas */
       segmenta();

/* Se crea y se lee el archivo para guardar los
graficos */
       arch = creat(argv[2],PERMS);
       close(arch);
       nuevo = fopen(argv[2],"w");
       redibuja();

 /* Se reporta el tiempo de ejecucion */
      total = clock();
      printf("\nTiempo: %f \n",(double)
              (total/1000000.0));
      return(1);
}

/*Funcion que lee la imagen del archivo raster */
void lee_imagen()
{
  char b;
  int i,j;

  for (i = 0; i <= J-1 ; i++)
  {
   for ( j = 0; j<= K-1; j++)
   {
     fscanf(asc,"%c",&b);
     if ( b == '0')
       imgmap[j][i] = 0;
     else
       if ( b == '1')
         imgmap[j][i] = 1;
   )
  }
  fclose(asc);
}
```

```
 /* Funcion que hace la busqueda de un
componente conectado */
void pacman(x,y)
int x;
int y;
{
  if ( x > der )
    der = x;
  if ( x < izq )
    izq = x;
  if ( y > arriba)
    arriba = y;
  if ( y < abajo)
    abajo = y;

  imgmap[x][y] = 2;

  if (imgmap[x+1][y] == 1)
    pacman(x+1,y);
  if (imgmap[x+1][y-1] == 1)
    pacman(x+1,y-1);
  if (imgmap[x][y-1] == 1)
    pacman(x,y-1);
  if (imgmap[x-1][y-1] == 1)
    pacman(x-1,y-1);
  if (imgmap[x-1][y] == 1)
    pacman(x-1,y);
  if (imgmap[x-1][y+1] == 1)
    pacman(x-1,y+1);
  if (imgmap[x][y+1] == 1)
    pacman(x,y+1);
  if (imgmap[x+1][y+1] == 1)
    pacman(x+1,y+1);
  return;
}

/* Funcion que realiza la segmentacion */
void segmenta()
{
  int x,y;
  struct rectangulo *p,*q,*aux;
  struct moda *r,*primero,*s;
  unsigned int
area,total_area,cantidad,amp,cuenta_amp;
  float average,umbral;

/* Se crea la lista ligada para los rectangulos, y
para el histograma de ocurrencias
con el que se encuentra el area mas frecuente
(amp) */
  p=(struct rectangulo*) malloc(sizeof(struct
      rectangulo));
  p->der = p->izq = p->ar = p->ab = p->area = 0;
  p->sig = NULL;
  r=(struct moda*) malloc(sizeof(struct moda));
  r->valor = r->cuenta = 0;
  r->ant = NULL;

  cantidad =0;
  total_area = 0;

/* Se hace un ciclo para encontrar todos los
componentes conectados de la imagen */
  for(y =0; y<= J-1; y++)
   for (x =0; x<= K -1; x++)
   {
     if (imgmap[x][y] == 1)
     {
       der = izq = x;
       arriba = abajo = y;

 /* Se busca el conjunto de pixeles conectados al
pixel actual */
       pacman(x,y);

/* Se guardan las coordenadas del rectangulo y
se calcula su area */
       p->der = der+1;
       p->izq = izq-1;
       p->ar = arriba+1;
       p->ab = abajo-1;
       area = (p->der - p->izq) * (p->ar - p->ab);
       p->area = area;

/* Se incluye el area dentro del histograma de
ocurrencias */
       primero = r->ant;
       if (primero != NULL)
         do
         {
           if (primero->valor == area)
           {
             primero->cuenta++;
             break;
           }
           else
           {
             primero = primero->ant;
           }
         }
         while (primero != NULL);
       if (primero == NULL)
       {
         r->valor = area;
         r->cuenta = 1;
         s=(struct moda*) malloc(sizeof(struct
             moda));
         s->valor = s->cuenta = 0;
         s->ant = r;
         r = s;
         s = NULL;
       }

/* Se agrega el area actual al calculo del
promedio de area */
       cantidad++;
        if ( area < 15,000)
          total_area = total_area + area;
```

```
/* Se agrega el rectangulo a la lista */
      q=(struct rectangulo*) malloc(sizeof(struct
          rectangulo));
      q->der = q->izq = q->ar = q->ab =
      q->area = 0;
      q->sig = p;
      p = q;
      q = NULL;
    }
  }

/* Se encuentra dentro del histograma el area
mas frecuente */
  cuenta_amp = 0;
  amp = 0;
  primero = r->ant;
  do
  {
    if ( primero->cuenta == cuenta_amp )
    {
      if ( (amp < primero->valor) && (primero-
          >valor < 400) )
        amp = primero->valor;
    }
    if (primero->cuenta > cuenta_amp)
    {
      cuenta_amp = primero->cuenta;
      amp = primero->valor;
    }
    primero = primero->ant;
  }
  while (primero != NULL);

/* Se calcula el promedio de area */
  average =(float) ( (float)total_area /
              (float)cantidad );

/* En base al mayor entre amp y average se
determina al umbral */
  if ( (average > amp) && (average < 400 ))
   umbral = average;
  else
   umbral = (float) amp;
   umbral = umbral * 5.0;

/* Se segmentan todos los rectangulos menores
al umbral */
  q = p;
  aux = p;
  p = p->sig;
  do
  {
    if ( p->area > umbral )
    {
      aux->sig = p->sig;
      free(p);
      p = aux->sig;
    }
    else
/* Se determina si el rectangulo corresponde a
una linea recta */
    if ( abs( (p->der - p->izq) - (p->ar - p->ab) )
        >= 10 )
    {
      aux->sig = p->sig;
      free(p);
      p = aux->sig;
    }
    else
    {
      elimina(p->der,p->izq,p->ar,p->ab);
      aux = p;
      p = p->sig;
    }
  }
  while ( p != NULL) ;
}

/* Esta función segmenta los componentes
conectados dentro del rectangulo */
void elimina(a,b,c,d)
int a;int b;int c;int d;
{
  int i=0,j=0;

 for(i=b+1;i<=a-1;i++)
 {
  for(j=d+1;j<=c-1;j++)
  {
    imgmap[i][j] = 0;
  }
 }
}

/* Almacena los gráficos en un archivo */
void redibuja()
{
 int x,y;

 for(y =0; y <= J-1; y++)
  for (x =0; x<= K -1; x++)
  {
    if (imgmap[x][y] != 0)
      imgmap[x][y] = 1;
    fprintf(nuevo,"%d",imgmap[x][y]);
  }
 fclose(nuevo);
}
```

Apéndice C. Programa de Vectorización

```
/*
Programa de Vectorización de líneas rectas
desarrollado por:
                    Jose Luis Carbajal Hernandez.
                    Rocio Hernandez Elenes.

Lee una imagen de 100x100, sin ruido, con
ancho de un pixel y sin líneas curvas.
Realiza una Transformada Adaptativa de Hough
con parámetros polares, y va reconociendo y
eliminando segmento por segmento.
Almacena los resultados en un archivo DXF.

*/

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<unistd.h>

/* CONSTANTES */
#define K 100 /* Numero de renglones */
#define J 100  /* Numero de columnas */
#define M 15  /* Limite vertical rho del arreglo de
                 Hough */
#define N 15  /* Limite horizontal teta del arreglo
                 de Hough */
#define ITERA 4 /*Numero de iteraciones para la
                   AHT */
#define ITERA1 3 /* Numero de iteraciones
                    opcional */
#define X 2 /*Factor de peso de la variable
               intervalo*/
#define FIN 8 /*Numero de puntos finales */
#define ESQ 48 /* Numero de puntos esquina */
#define UNI 54 /* Numero de puntos union */
#define PERMS 0666 /*Permisos del archivo dxf
                     Unix */

/* ESTRUCTURAS */

/*  Parametros de linea */
typedef struct parametros{
        double rho;
        double teta;
        int valor;
        struct parametros* ant;
};
/* Puntos que delimitan el segmento de linea */
typedef struct limite{
    int j;
    int k;
    int tipo;
    struct limite* ant;
};
*Nuevos limites del arreglo de Hough */
typedef struct aht_limi{ /
    double rho1;
    double rho2;
    double teta1;
    double teta2;
};

/* FUNCIONES */
void lee();
void limpiah();
void ht();
struct parametros* mayor();
double calcularho();
double calculateta();
int lineas();
void peaks();
struct parametros* analiza();
struct parametros* analiza_rectas();
struct aht_limi* nuevos_limites();
void esquinas();
int borra();
int revisa();
int fail();
double fraccion();
void encabezado();
void archiva_linea();

 /* VARIABLES */

 /* Puntos finales*/
int terminal[FIN]={ 1,2,4,8,16,32,64,128};

/*Puntos esquina*/
int uniones[ESQ]={ 3,5,6,7,9,10,11,12,13,14,18,
                   20,22,24,26,28,33,36,40,44,
                   48,52,56,65,66,67,72,80,88,
                   96,97,104,112,129,130,131,
                   132,133,134,144,160,161,
                   176,192,193,194,208,224};

/*Puntos de union*/
int junctions[UNI]={ 19,21,25,37,41,42,43,49,51,
                     53,69,70,73,74,76,77,81,82,
                     83,84,85,86,87,89,93,100,
                     101,102,106,117,138,141,
                     145,146,148,153,154,162,
                     164,166,168,169,170,171,
                     172,174,178,186,196,202,
                     204,212,213,234};
int imgmap[J][K];  /* Mapa de bits de la imagen
*/
unsigned int ahough[M][N];  /* Arreglo de Hough
*/
FILE *dxf,*asc; /* Archivo CAD */
struct tm *tempo1, *tempo2;
int falsos[25]; /*Referencias falsas del arreglo de
                  Hough*/
int f,        /*Contador de referencias falsas*/
   iter,      /* Variable de iteraciones */
   no_esq=0,  /* Bandera que indica que no se
encontraron segmentos */
```

```
    termina,
    re,         /* Bandera que indica el tipo de lineas
                   buscadas */
    boundary;   /* Bandera para delimitar el
                   arreglo de Hough */

/* PROGRAMA PRINCIPAL */

int main(argc,argv)
int argc;
char *argv[];
{
    long u,total=0.0;
    int arch;

    /* Se inicializa el reloj */
       u = clock();

    /* Se abre el archivo con la imagen */
       asc = fopen(argv[1],"r");
       if (asc == NULL)
       {
         printf("\n Error. Archivo no existe");
         return(0);
       }
       lee();

   /* Se crea el archivo CAD */
       arch = creat(argv[2],PERMS);
       close(arch);

   /* Se abre el archivo CAD */
      dxf = fopen(argv[2],"w");
      encabezado();

    /* Se obtienen todos los segmentos de linea */
        peaks();

    /* Se reporta el tiempo de ejecucion */
        total = clock();
        printf("\nTiempo: %f \n",(double)
              (total/1000000.0));
        return(1);
}

/* Funcion que realiza la busqueda de lineas y
devuelve el archivo tipo CAD */
void peaks()
{
  int i,arch;
  struct parametros* p;

/* Se inicializan las variables para el primer
analisis. Para facilitar la busqueda, e ir
limpiando la imagen, en primer lugar se buscan
las lineas mas faciles de localizar (lineas
verticales y horizontales) y despues las lineas
inclinadas. De esta forma las lineas inclinadas,
que tienden a ser mas dificiles, quedan aisladas
*/
   boundary = 0;
   printf("\nBusco verticales");
   iter = ITERA1;
   termina = no_esq = 0;
   re = 0;

/* Se hace un analisis de esquinas a la imagen
*/
   esquinas();

/* Se buscan lineas verticales */
   do
   {
     despliega();
     f=0;
     for(i=0;i<30;i++)
       falsos[i] = 0;

     /* Se buscan los parametros de una linea */
     p = analiza_rectas(f,1);

     /* Si se encontraron parametros, entonces se
localizan los segmentos */
     if (p != NULL)
       arch = lineas(p,0);

     /* Si no se encontraron segmentos "arch" es
0, por lo que terminamos */
     if (arch == 0)
       break;

     /* Se vuelven a analizar esquinas, por si
algun segmento quedo truncado */
      esquinas();
   }
   while( p != NULL); /* Se termina el ciclo
cuando ya no hay segmentos verticales */

   /* Se revisa si todavia existen puntos en la
imagen */
   if ( revisa() == 0)
   {
    return;
   }

   /* Se buscan lineas horizontales */
   esquinas();
   printf("\nBusco horizontales");
   termina = no_esq = 0;
   re = 0;
   iter = ITERA;
   do
   {
     despliega();
     f=0;
     for(i=0;i<30;i++)
       falsos[i] = 0;
     p = analiza_rectas(f,2);
     if (p != NULL)
       arch = lineas(p,0);
     if (arch == 0)
       break;
```

```
    esquinas();
   }
   while( p != NULL); /* Se termina el ciclo
cuando ya no hay segmentos horizontales */

   if ( revisa() == 0)
   {
    return;
   }

   /* Se buscan lineas inclinadas */
   printf("\nBusco inclinadas");
   iter = ITERA;
   re = 1;
   termina = no_esq = 0;
   esquinas();
   do
   {
    despliega();
    f=0;
    for(i=0;i<30;i++)
      falsos[i] = 0;
    p = analiza(f);
    if (p != NULL)
      arch = lineas(p,0);
    if (arch == 0)
      break;
    esquinas();
   }
   while( p != NULL); /* Se termina el ciclo
cuando ya no hay segmentos */

   /* Se cierra el archivo CAD */
   fprintf(dxf,"%d\n",0);
   fprintf(dxf,"ENDSEC\n");
   fclose(dxf);
   return;
}

/* Llena los encabezados del archivo CAD, que
van antes de la definicion de las lineas */
void encabezado()
{
    fprintf(dxf,"%d\n",0);
    fprintf(dxf,"SECTION\n");
    fprintf(dxf,"%d\n",2);
    fprintf(dxf,"TABLES\n");
    fprintf(dxf,"%d\n",0);
    fprintf(dxf,"SECTION\n");
    fprintf(dxf,"%d\n",2);
    fprintf(dxf,"ENTITiES\n");
}

/* Lee la imagen del archivo */
void lee()
{
  char b;
  int i,j;

  for (i = J-1; i>= 0; i--)
  {
   for ( j = 0; j<= K-1; j++)
   {
    fscanf(asc,"%c",&b);
    if ( b == '0')
      imgmap[j][i] = 0;
    else
      if ( b == '1')
        imgmap[j][i] = 1;
   }
   fscanf(asc,"%c",&b);
  }
  fclose(asc);
}

 /* Realiza la transformada de Hough de Duda &
Hart y llena el arreglo de Hough */
void ht(rhomax,rhomin)
double rhomax;
double rhomin;
{
  int j,k,n,m,flag;
  double rho,teta,phi,m1;
  char w;

  double h,dos,tres,x,y;

  /* Se limpia el arreglo de Hough*/
  limpiah();

  /* Se hace un ciclo que revise todos los puntos
de la imagen*/
  for (j=0;j<=J-1;j++)
   for (k=0;k<=K-1;k++)
    if (imgmap[j][k] >= 1) /* Solo se toman los
                              puntos imagen */
    {

    /*Se establecen las coordenadas cartesianas
de x y y */
      x = k - 0.5;
      y = J + 0.5 - j;
      phi = atan2(y,x);
      flag = 0;

    /*Se calcula, para cada teta, el valor de rho, y
se mapea en el arreglo*/
      for (n=0;n<=N-1;n++)
      {
          teta= M_PI - ((2*M_PI * (N - n))/(N - 1));
          if ( (teta >= (phi - M_PI_2)) && (teta <=
            (phi + M_PI_2)))
          {
            flag++;
            rho = x * cos(teta) + y * sin(teta);
            m1 =  M - ( ((rhomax - rho) * (M-1)) /
                  (rhomax - rhomin) );
            if ( h=fraccion((double)m1) >= 0.5)
              m = ceil(m1);
            else
              m = floor(m1);
```

```
 /* Si la bandera "re" es cero se estan buscando
verticales u horizontales, por lo que se suman
unos al arreglo. De lo contrario se suma el valor
del punto en la imagen, de tal manera que los
puntos terminales actuan como factor de peso
para encontrar los parametros de las lineas */
        if (re == 0)
          ahough[m][n] = ahough[m][n] + 1;
        else
          ahough[m][n] = ahough[m][n] +
                         imgmap[j][k];
        }
    }
  }
}

/* Realiza la transformada adaptativa de Hough
y llena el arreglo de Hough, la principal
diferencia con la funcion ht() es que esta funcion
recibe los nuevos limites de los parametros teta
y rho, mientras que la otra parte de los limites
originales */
void aht(rhomax,rhomin,teta1,teta2)

double rhomax;
double rhomin;
double teta1;
double teta2;
{
  int j,k,n,m,flag;
  double rho,teta,phi,x,y,m1;
  char w;
  double dos,h,tres;

  /*Se limpia el arreglo de Hough */
  limpiah();

  /*Se hace un ciclo para todos los puntos de la
imagen */
  for (j=0;j<=J-1;j++)
   for (k=0;k<=K-1;k++)
    if (imgmap[j][k] >= 1)  /* Solo se toman los
                               puntos negros */
   {

/*Se inicializan las coordenadas catersianas x y
y */
     x = k - 0.5;
     y = J + 0.5 - j;
     phi = atan2(y,x);
     flag = 0;

   /*Se calcula rho para cada teta, y se mapea
en el arreglo */
     for (n=1;n<=N;n++)
     {
     teta= teta2 - (((teta2-teta1)* (N - n))/(N - 1));
     if ( (teta >= (phi - M_PI_2)) && (teta <= (phi
         + M_PI_2))(
          flag++;
          rho = x * cos(teta) + y * sin(teta);
          m1 =  M - (((rhomax - rho) * (M-
                1))/(rhomax- rhomin))-1;
          if ( h=fraccion((double)m1) >= 0.5)
             m = ceil(m1);
          else
             m = floor(m1);
          if ( ( m >=0) && (m<=M-1))
          if (re == 0)
            ahough[m][n] = ahough[m][n] + 1;
          else
            ahough[m][n] = ahough[m][n] +
                           imgmap[j][k];
          }
     }
   }
}

/* Esta funcion realiza el ajuste de los limites del
arreglo de Hough alrededor del valor maximo
que se va localizando y devuelve los parametros
de la linea encontrada */
struct parametros* analiza(f)
int f;
{
  struct aht_limi* limit;
  double h;
  double r,t,t1,rhomax,rhomin;
  struct parametros* p;
  int bandera,ref;
  double centro,intervalo;
  int e=0;

  /* Se revisa que existan puntos en la imagen */
  if ( revisa() == 0)
  {
   return(NULL);
  }

 /* Se crea la estructura que contiene los limites
del arreglo de Hough */
 limit=(struct aht_limi*) malloc(sizeof(struct
        aht_limi));
 limit->rho1 = limit->rho2 = limit->teta1 = limit-
               >teta2 = 0.0;

 /* Se establecen los limites de rho de acuerdo
a la bandera "boundary". Debido a que el arreglo
de Hough es muy peque&o, se analiza primero
del valor maximo al minimo, despues en la parte
positiva de los valores y despues en la negativa,
de tal forma que podemos tener una mejor
precision y una mayor rapidez en la localizacion
de las lineas */
   rhomax = sqrt( pow((double)(K-0.5), (double)
            (2))+pow((double)(J+0.5),(double)
            (2)));
   rhomin = - rhomax;
 if (boundary == 0)   /* Limites maximos */
 {
  limit->rho1 = rhomax;
  limit->rho2 = rhomin;
```

```
 /* Si la bandera "re" es cero se estan buscando
verticales u horizontales, por lo que se suman
unos al arreglo. De lo contrario se suma el valor
del punto en la imagen, de tal manera que los
puntos terminales actuan como factor de peso
para encontrar los parametros de las lineas */
          if (re == 0)
            ahough[m][n] = ahough[m][n] + 1;
          else
            ahough[m][n] = ahough[m][n] +
                           imgmap[j][k];
          }
      }
    }
}

/* Realiza la transformada adaptativa de Hough
y llena el arreglo de Hough, la principal
diferencia con la funcion ht() es que esta funcion
recibe los nuevos limites de los parametros teta
y rho, mientras que la otra parte de los limites
originales */
void aht(rhomax,rhomin,teta1,teta2)

double rhomax;
double rhomin;
double teta1;
double teta2;
{
  int j,k,n,m,flag;
  double rho,teta,phi,x,y,m1;
  char w;
  double dos,h,tres;

  /*Se limpia el arreglo de Hough */
  limpiah();

  /*Se hace un ciclo para todos los puntos de la
imagen */
  for (j=0;j<=J-1;j++)
   for (k=0;k<=K-1;k++)
    if (imgmap[j][k] >= 1)  /* Solo se toman los
                               puntos negros */
    {

/*Se inicializan las coordenadas catersianas x y
y */
      x = k - 0.5;
      y = J + 0.5 - j;
      phi = atan2(y,x);
      flag = 0;

    /*Se calcula rho para cada teta, y se mapea
en el arreglo */
      for (n=1;n<=N;n++)
      {
      teta= teta2 - (((teta2-teta1)* (N - n))/(N - 1));
      if ( (teta >= (phi - M_PI_2)) && (teta <= (phi
          + M_PI_2)){
           flag++;
           rho = x * cos(teta) + y * sin(teta);
           m1 =  M - (((rhomax - rho) * (M-
                 1))/(rhomax- rhomin))-1;
           if ( h=fraccion((double)m1) >= 0.5)
              m = ceil(m1);
           else
              m = floor(m1);
           if ( ( m >=0) && (m<=M-1))
           if (re == 0)
             ahough[m][n] = ahough[m][n] + 1;
           else
             ahough[m][n] = ahough[m][n] +
                            imgmap[j][k];
           }
      }
    }
}

/* Esta funcion realiza el ajuste de los limites del
arreglo de Hough alrededor del valor maximo
que se va localizando y devuelve los parametros
de la linea encontrada */
struct parametros* analiza(f)
int f;
{
  struct aht_limi* limit;
  double h;
  double r,t,t1,rhomax,rhomin;
  struct parametros* p;
  int bandera,ref;
  double centro,intervalo;
  int e=0;

  /* Se revisa que existan puntos en la imagen */
  if ( revisa() == 0)
  {
   return(NULL);
  }

 /* Se crea la estructura que contiene los limites
del arreglo de Hough */
 limit=(struct aht_limi*) malloc(sizeof(struct
        aht_limi));
 limit->rho1 = limit->rho2 = limit->teta1 = limit-
               >teta2 = 0.0;

 /* Se establecen los limites de rho de acuerdo
a la bandera "boundary". Debido a que el arreglo
de Hough es muy peque&o, se analiza primero
del valor maximo al minimo, despues en la parte
positiva de los valores y despues en la negativa,
de tal forma que podemos tener una mejor
precision y una mayor rapidez en la localizacion
de las lineas */
   rhomax = sqrt( pow((double)(K-0.5), (double)
            (2))+pow((double)(J+0.5),(double)
            (2)));
  rhomin = - rhomax;
 if (boundary == 0)    /* Limites maximos */
 {
  limit->rho1 = rhomax;
  limit->rho2 = rhomin;
```

```
 }
 else if (boundary == 1)  /* Limites positivos */
 {
  limit->rho1 = rhomax;
  limit->rho2 =  -20; /* - limit->rho1; */
 }
 else if (boundary == 2)  /* Limites negativos */
 {
  limit->rho2 = rhomin;
  limit->rho1 = 20;
 }
 else if (boundary == 3)
 {
  limit->rho1 = rhomax;
  limit->rho2 = rhomin;
 }
 else
   return(NULL);

 /* Los limites del angulo teta permanecen
intactos -pi a pi */
  limit->teta1 = -M_PI;
  limit->teta2 = M_PI;

 /* "bandera" cuenta el numero de iteraciones
actual */
  bandera = 0;

 /*Primero se calcula la transformada HT */
  ht(rhomax,rhomin);

/*Se realiza un ciclo que realice la AHT el
numero de iteraciones necesarias */
  do
  {
   bandera++;

/* Se localiza el valor maximo del arreglo de
Hough */
    p = mayor(f);

/* Se calculan los parametros de acuerdo a los
valores encontrados en el arreglo */

   if (bandera == 1)
   {

/* Si se encuentran menos de 6 puntos como
maximo en el arreglo y el numero de maximos
analizados es diferente de cero indican que los
segmentos no se encuntran dentro de los limites
actuales, entonces se cambian las condiciones
de los limites y las iteraciones */
      if ((p->valor <= 6) && (f != 0))
      {
       iter--;
       if (iter == 2)
       {
        if (boundary == 3)
          return(NULL);
        else
        {
          boundary++;
          iter = ITERA;
        }
       }
       f = 0;
       p = analiza(f);
          return(p);
      }
      else
      {

/* Si el numero de puntos en el maximo es 6 y
ya analizamos todos los posibles limites del
arreglo, entonces consideramos que los puntos
que quedan en la imagen son ruido y no forman
ninguna linea recta */
       if( (p->valor <= 6) && (boundary == 3) )
         return(NULL);
       if ( (p->valor <= 6) && (boundary != 3) )
       {
         boundary++;
       }
          r =calcularho(p->rho);
          t =calculateta(p->teta,0);
          ref = p->valor;
    }
    else
    {

 /* Si el maximo encontrado despues de varias
iteraciones es menor que 8, entonces estamos
buscando un segmento formado por puntos
colineales no vecinos. Por lo tanto analizamos
otro de los valores del arreglo */
      if ( p ->valor < 8)
      {
       falsos[f] = ref;
       f++;
       if(f == 30)
       {
         boundary++;
         f = 0;
       }
          p = analiza(f);
          return(p);
       }

/* "r" y "t" contienen los valores de los
parametros encontrados hasta el momento */
          r = p->rho; t = p->teta;
          r = limit->rho1 + ((p->rho-M+1)*(limit-
              >rho1-limit->rho2))/(M-1);
          t = limit->teta2 - ((limit->teta2-limit-
              >teta1)*(N - p->teta))/(N-1);
          if ( h=fraccion((double)r) >= 0.499)
            r = r - h + 1;
          if ( (t > -0.001) && (t < 0.001))
            t = 0;
    }
```

```
    /* Se calculan los nuevos limites del arreglo
de Hough */
    limit = nuevos_limites(limit,p,0,r,t);

    /*Una vez establecidos los limites, se calcula
la AHT con ellos */
    aht(limit->rho1,limit->rho2,limit->teta1,limit-
        >teta2);
  }

  /*Se realiza un numero "iter" de veces la AHT
para obtener la mejor precision */
  while(bandera <= iter);

  /* Encontramos los valores de rho y teta
despues de terminar la AHT */
  p = mayor(f);
  p->rho = limit->rho1 + ((p->rho-M+1)*(limit-
           >rho1-limit->rho2))/(M-1);
  p ->tete = limit->teta2 - ((limit->teta2-limit-
             >teta1)*(N - p->teta))/(N-1);

  /*Retornamos la estructura que contiene los
valores rho y teta */
  return(p);
}

/* Esta funcion realiza el ajuste de los limites del
arreglo de Hough alrededor del valor maximo
que se va localizando */
struct parametros* analiza_rectas(f,pendiente)
int f;
int pendiente;
{
  struct aht_limi* limit;
  double h;
  double r,t,t1;
  struct parametros* p;
  int bandera,ref;
  double centro,intervalo;
  int e=0;

/* Se revisa que existan puntos en la imagen */
  if ( revisa() == 0)
   {
    return(NULL);
   }

 /* Se crea la estructura que contiene los limites
del arreglo de Hough */
  limit=(struct aht_limi*) malloc(sizeof(struct
        aht_limi));
  limit->rho1 = limit->rho2 = limit->teta1 = limit-
                >teta2 = 0.0;

 /*Primero se calcula la transformada HT */
 limit->rho1 = sqrt( pow((double)(K-0.5),
               (double)(2))+ pow((double)(J+0.5),
               (double)(2)));
  limit->rho2 = - limit->rho1;
  ht(limit->rho1,limit->rho2);

 /* Para lineas verticales y horizontales los
limites iniciales del arreglo de Hough son
constantes, ya que el centro es el mismo (0 o 90
grados) y rho siempre es positivo */
  limit->rho1 = 110;
  limit->rho2 = -20;
  limit->teta1 = -M_PI;
  limit->teta2 = M_PI;
  bandera = 0;

/*Se realiza un ciclo que realice la AHT el
numero de iteraciones necesarias */
  do
  {
   bandera++;

/* Se localiza el valor maximo del arreglo de
Hough */
    p = mayor(f);

    /* Se calculan los parametros de acuerdo a
los valores encontrados en el arreglo */
    if (bandera == 1)
    {
        if ((p->valor <= 6) && (f != 0))
     {
       f = 0;
       iter--;
       if (iter == 1)
         return(NULL);
       p = analiza_rectas(f,pendiente);
       return(p);
      }
      else
       if (p->valor <= 6)
        return(NULL);
         r =calcularho(p->rho);
         t =calculateta(p->teta,0);
         ref = p->valor;
    }
    else
    {
      if ( p ->valor < 8)
      {
          falsos[f] = ref;
          f++;
          if (f == 30)
          {
            iter--;
            f = 0;
            if (iter == 1)
            return(NULL);
      }
          p = analiza_rectas(f,pendiente);
          return(p);
     }
     r = p->rho; t = p->teta;
     r = limit->rho1 + ((p->rho-M+1)*(limit->rho1-
        limit->rho2))/(M-1);
```

```
        t = limit->teta2 - ((limit->teta2-limit-
            >teta1)*(N - p->teta))/(N-1);
        if ( h=fraccion((double)r) >= 0.499)
          r = r - h + 1;
        if ( (t > -0.001) && (t < 0.001))
          t = 0;
    }

/* Se calculan los nuevos limites del arreglo de
Hough */
    limit = nuevos_limites(limit,p,pendiente,r,t);

/*Una vez establecidos los limites, se calcula la
AHT con ellos */
    aht(limit->rho1,limit->rho2,limit->teta1,limit-
        >teta2);
  }

 /*Se realiza un numero iter de veces la AHT
para obtener la mejor precision */
  while(bandera <= iter);

/* Encontramos los valores de rho y teta
despues de terminar la AHT */
  p = mayor(f);
  p->rho = limit->rho1 + ((p->rho-M+1)*(limit
            ->rho1-limit->rho2))/(M-1);
  p ->teta = limit->teta2 - ((limit->teta2-limit
            ->teta1)*(N - p->teta))/(N-1);

  /*Retornamos la estructura que contiene los
valores rho y teta */
  return(p);
}

/*Se establecen los nuevos limites parametricos.
De acuerdo a la ubicacion del valor maximo, se
coloca como centro del nuevo arreglo, y se
toman valores alrededor de el. Es decir, un
intervalo a la izquierda del centro sera el limite
inferior de rho, un intervalo a la derecha del
centro el limite superior, etc.*/

struct aht_limi*
nuevos_limites(x,p,bandera,rho,teta)
struct aht_limi* x;
struct parametros* p;
int bandera;
double rho;
double teta;
{
  double centro,intervalo;

        intervalo = (x->rho1-x->rho2)/M;
        intervalo = intervalo*X;
        centro = rho;

/*Si el valor de rho encontrado esta en la
esquina del arreglo (M-1), el limite maximo se
conserva y se amplia el limite inferior */
    if (p->rho == M-1)
        x->rho2 = centro - 2*intervalo;
    else
    {
/* Si el valor de rho esta en la otra esquina (0),
se conserva el limite inferior y se amplia el
mayor */
        if (p->rho == 0)
          x->rho1 = centro + intervalo;
        else
        {
/*Si el valor de rho no esta en ninguna esquina,
los limites se amplian por igual */
          x->rho1 = centro + intervalo;
          x->rho2 = centro - intervalo;
        }
    }

/* Al igual que con rho, se determina si el valor
de teta se encuentra en alguna esquina del
arreglo */
        intervalo = (x->teta2 - x->teta1)/N;
        intervalo = intervalo*X;
        centro = teta;
    if (p->teta == N-1)
        x->teta1 = centro - 2*intervalo;
    else
    {
        if (p->teta == 0)
          x->teta2 = centro + intervalo;
        else
        {
/* Si estamos analizando horizontales, el valor
de teta (el angulo de la normal a la linea) es 90
grados, por lo que se mantiene constante en el
centro. Para lineas verticales, teta es cero */
          if (bandera == 1)
            centro = M_PI_2;
          else
            if (bandera == 2)
              centro = 0;
            else
              centro = teta;
            x->teta1 = centro - intervalo;
            x->teta2 = centro + intervalo;
        }
    }

  /* Devolvemos los nuevos limites */
  return(x);
}

/* Devuelve el valor maximo optimo del arreglo
de Hough */
struct parametros* mayor(f)
int f;
{
 struct parametros* centinela,*guia;
 int i,j,d,d1,e,valor;
 /*Se inicializa la estructura que contendra las
coordenadas del arreglo correspondientes al
valor maximo */
```

```
  centinela=(struct parametros *) malloc(sizeof(
            struct parametros));
  centinela->ant = NULL;
  centinela->rho = centinela->teta = 0;
  centinela ->valor=0;
  guia = centinela;
  valor = 1;

  /* Se revisa todo el arreglo de Hough */
  for (i=0;i<=M;i++)
   for (j=0;j<=N;j++)

/*Si se encuentra un valor mayor al que
actualmente se considera mayor (valor), se
guarda como el nuevo valor maximo */
    if ( (ahough[i][j] >= valor) &&
         (fail(ahough[i][j],f)) )
    {
/*Si el valor es igual, se revisa cual de los dos
esta mas cercano al centro del arreglo, ya que la
AHT localiza el valor optimo en el centro del
arreglo */
          if (ahough[i][j] == valor)
          {
            if ((i == (M-1)/2) && (j == (N-1)/2))
            {
              guia->rho = i;
              guia->teta = j;
              guia->valor = valor = ahough[i][j];
            }
            else
            {
              d = abs((M-1)/2 - i);
              d1 = abs((N-1)/2 - j);
              if (d < d1)
                d = d1;
              e = abs((M-1)/2 - guia->rho);
              d1 = abs((N-1)/2 - guia->teta);
              if (e < d1)
                e = d1;
              if ( d <= e)
              {
               guia->rho = i;
               guia->teta = j;
               guia->valor = valor = ahough[i][j];
              }
            }
          }
          else
          {
            guia->rho = i;
            guia->teta = j;
            guia->valor = valor = ahough[i][j];
          }
    }
    return(guia);
}

/* Revisa si el valor "ah" se encuentra o no en la
lista de maximos analizados previamente */
int fail(ah,f)
int ah;
int f;
{
  int i;
  if ( f == 0 )
   return(1);
  for(i=0;i<=f;i++)
    if(ah == falsos[i])
      return(0);
  return(1);
}

/* Revisa que existan puntos en la imagen, ya
que los segmentos se van borrando conforme se
les localiza */
int revisa()
{
   int j,k,c=0,d=0;

    for(j=0;j<=J-1;j++)
     for(k=0;k<=K-1;k++)
      if (imgmap[j][k] >= 1)
            c++;
    if((c >= 8))
     return(1);
    else
    return(0);
}

/* Se realiza una clasificacion de los pixeles que
se encuentran en la imagen de acuerdo a su
supuesta posicion dentro de las lineas.
      0 - Fondo
      1 - Punto intermedio
      2 - Punto final
      3 - Punto esquina de dos o mas lineas
      4 - Punto union de dos o mas lineas
de tal forma que al buscar los segmentos de
linea, solamente se revisaran estos puntos */
void esquinas()
{
  int i,x,x1,y,y1,nc,f,flag=0,punto;
  double eleva;
  int juntos;

/*Se realiza un primer ciclo, solo para todos
aquellos puntos, diferentes de cero, que no se
encuentren en las orillas de la imagen */
   for(x=0;x<=K-1;x++)
     for(y=0;y<=J-1;y++)
     {
       if (imgmap[x][y] < 0)
         imgmap[x][y] = 0;
       if (imgmap[x][y] >= 1)
       {
         imgmap[x][y] = 1;
          nc = 0;
          flag = 0;
          juntos = 0;
 /*Se calcula el numero codigo decimal del pixel
actual */
```

```
        for(i=0;i<=7;i++)
        {
          switch (i) {
            case 0: x1 = x+1; y1 = y; break;
            case 1: x1 = x+1; y1 = y-1; break;
            case 2: x1 = x; y1 = y-1; break;
            case 3: x1 = x-1; y1 = y-1; break;
            case 4: x1 = x-1; y1 = y; break;
            case 5: x1 = x-1; y1 = y+1; break;
            case 6:  x1 = x; y1 = y+1; break;
            case 7: x1 = x+1; y1 = y+1; break;
          }
        if (imgmap[x1][y1] == 2)
          juntos = 1;

  /* Se realiza una busqueda especial en la
orillas del arreglo, ya que para estas orillas, y en
especial para las esquinas, no existen ciertos
pixeles vecinos.Ejemplo, para la esquina
superior izquierda (0,0), no existen los vecinos
arriba ni a la izquierda de el, por lo tanto no se
deben tomar en cuenta al calcular su numero
codigo decimal */
    if( (x1 < 0) || (x1 > K-1) || (y1 < 0) || (y1 > J-1))
       punto = 0;
    else
    {
      if(imgmap[x1][y1] <= 0)
         punto = 0;
      else
         punto = 1;
    }
    eleva = pow(2.0,(double)(i));
    nc =(int)( nc + eleva*punto);
 }

/*Si el numero codigo decimal del pixel se
encuentra dentro de alguna de las listas, se le
clasifica, dependiendo de la lista como punto
final, esquina o union */
       if (nc != 0)
       {
          for(i=0;i<=FIN-1;i++)
           if( nc == terminal[i])
           {
            if (juntos == 1)
              imgmap[x][y] = 0;
            else
              imgmap[x][y] = 2;
            flag++;
            break;
           }
/* La bandera nos indica que el pixel no ha sido
clasificado aun */
           if(flag == 0)
           {
            for(i=0;i<=ESQ-1;i++)
              if( nc == uniones[i])
              {
                imgmap[x][y] = 3;
                flag++;
                break;
              }
           }
           if(flag == 0)
           {
            for(i=0;i<=UNI-1;i++)
              if( nc == junctions[i])
              {
                imgmap[x][y] = 4;
                flag++;
                break;
              }
           }
       }
       else
         imgmap[x][y] = 0;
     }
   }
}

/* Busca los segmentos de linea de una linea
encontrada por la AHT La transformada nos
devuelve los parametros de una linea en la
imagen, pero no indica de donde a donde va la
linea.Por lo tanto,puede tratarse de una sola
linea, o de segmentos colineales */
int lineas(guia)
struct parametros* guia;
{
  double x,y,z,c,h;
  struct limite *p,*q,*ini,*fin,*inip;
  int j,k,y1,x1,resultado,flag;

 /*Se inicializa el primer elemento de una lista
ligada, la cual contendra todos aquellos puntos
que puedan ser inicio y fin de segmentos de
linea colineales a la linea encontrada */
  q=(struct limite *) malloc(sizeof(struct limite));
  p = q;
  p->j = p->k = p->tipo = 0;
  p->ant = NULL; q = NULL;
  if ((guia->teta == -0.0) || ((guia->teta <= 0.01)
      && (guia->teta >= -0.01)))
    guia->teta = 0.0;
  if (guia->rho == -0)
    guia->rho = 0;

 /* Se revisa toda la imagen, pero solo por
aquellos puntos que esten clasificados como
puntos terminales de lineas.Si la linea es casi
vertical (el angulo teta de su normal esta  entre
-45 y 45 grados) se deben seguir los puntos en
el eje de las ordenadas, obteniendo sus valores
en el eje de las absisas. En caso contrario, se
deben proponer las absisas y calcular las
ordenadas */
 if ( !((guia->teta < M_PI/4) && (guia->teta > -
       M_PI/4)) )
 {
  for(k=0;k<=K-1;k++)
    for(j=0;j<=J-1;j++)
```

```
    if(imgmap[j][k] >= 2)
    {
          if ( k != 0)
            x = k - 0.5;
          else
            x = 0;
          z = (guia->rho - x * cos(guia
               ->teta))/(sin(guia->teta));
          y = J - z + 0.5;
          if ( y < 0)
            y = -1 * y;
          if ( (h=fraccion((double)y)) >= 0.5)
            y1 = ceil(y);
          else
            y1 = floor(y);

/*Se propone x, se calcula y, y se revisa que
coincida con la coordenada actual. Esto indica
que el pixel corresponde a la linea y se agrega a
la lista, de lo contrario, no pertenece a la linea */
          if ((y1 == j) )
          {
           p->j = j; p->k = k; p->tipo =
                  imgmap[j][k];
          q=(struct limite *) malloc(sizeof(struct
              limite));
          q->j = q->k = 0;
          q->ant = p;
          p = q;
          q = NULL;
        }
        else
        {  if (re == 1)
           {
           if( (y1+1 == j) || (y1-1 == j) )
           {
            p->j = j; p->k = k; p->tipo =
                   imgmap[j][k];
            q =(struct limite *) malloc(sizeof
                (struct limite));
            q->j = q->k = 0;
            q->ant = p;
            p = q;
            q = NULL;
           }
          }
        }
    }
 }
 else
 {
  for(j=0;j<=J-1;j++)
    for(k=0;k<=K-1;k++)
      if(imgmap[j][k] >= 2)
      {
/*Debido a la forma polar de la ecuacion de la
linea, cuando  teta es cero, el calculo de y se
indetermina, por lo tanto en ese caso, se
propone y y se calcula x */
          if ( guia->teta != 0)
          {
          if ( k != 0)
            x = k - 0.5;
          else
            x = 0;
          z = (guia->rho - x * cos(gula
              ->teta))/(sin(guia->teta));
          y = J - z + 0.5;
          if ( y < 0)
            y = -1 * y;
          if ( (h=fraccion((double)y)) >= 0.5)
            y1 = ceil(y);
          else
            y1 = floor(y);

/* Se propone x, se calcula y, y se revisa que
coincida con la coordenada actual. Esto indica
que el pixel corresponde a la linea y se agrega a
la lista, de lo contrario, no pertenece a la linea */
          if ((y1 == j) )
          {
              p->j = j; p->k = k; p->tipo =
                   imgmap[j][k];
              q=(struct limite *) malloc(sizeof
               (struct limite));
              q->j = q->k = 0;
              q->ant = p;
              p = q;
              q = NULL;
          }
          else
          {
            if (re == 1)
            {
            if( (y1+1 == j) || (y1-1 == j) )
             {
                  p->j = j; p->k = k; p->tipo =
                      imgmap[j][k];
                  q =(struct limite *) malloc(sizeof
                     (struct limite));
                  q->j = q->k = 0;
                  q->ant = p;
                  p = q;
                  q = NULL;
             }
            }
  }
 }
 else
 {

/*Caso para teta igual a cero, para evitar la
indeterminacion */
          y = J - j + 0.5;
          z = (guia->rho - y * sin(guia-
              >teta))/(cos(guia->teta));
          x = z + 0.5;
          if ( x < 0)
            x = -1 * x;
          if ( h=fraccion((double)x) >= 0.5)
            x1 = ceil(x);
          else
```

```
            x1 = floor(x);
            if ((x1 == k)  )
            {
              p->j = j; p->k = k; p->tipo =
                    imgmap[j][k];
              q=(struct limite *) malloc(sizeof
                 (struct limite));
              q->j = q->k = 0;
              q->ant = p;
              p = q;
              q = NULL;
        }
        else
        {
          if (re == 1)
          {
          if( (x1+1 == j) || (x1-1 == j) )
           {
            p->j = j; p->k = k; p->tipo =
                  imgmap[j][k];
                q =(struct limite *)
                    malloc(sizeof(struct limite));
                q->j = q->k = 0;
                q->ant = p;
                p = q;
                q = NULL;
           }
          }
     }
    }
   }
  }
```

/*Cuando se han encontrado todas los pixeles terminales que cumplen con la ecuacion de la recta, se procede a revisar si se trata de uno o mas segmentos. Tambien se borran estos, para que no influyan en la busqueda de las siguientes lineas */

```
  ini = p->ant;
```

/* Si no se encontraron esquinas para los parametros, quiere decir que la linea fue mal localizada, por lo que se cambian las condiciones de los limites del arreglo y de las iteraciones para la exactitud del calculo */

```
  if (ini == NULL)
  {
   if ( re == 0 )
   {
    iter--;
    if (iter <= 2)
      return(0);
    return(1);
   }
   boundary++;
  if ( no_esq == 1 )
   return(0);
  else
   no_esq++;
  return(1);
  }
  free(p);
  fin = ini;
```

/*Se cuentan el numero de pixeles terminales */

```
  c=0;
  do
  {
   c++;
   fin=fin->ant;
  }
  while(fin != NULL);
```

/* Si solo se tiene una esquina de la linea, entonces sus parametros estan mal calculados, por lo que se cambian las condiciones de calculo */

```
  if ( c == 1 )
  {
   iter--;
   if (iter <= 2)
   {
    if (re == 1)
     boundary++;
    else
     return(0);
   }
   return(1);
  }
  else
  {
```

/* Si el numero de pixeles terminales es dos, solo existe una linea */

```
    if ( c == 2 )
    {
     fin=ini->ant;
```

/* Se borra el segmento entre los puntos terminales "ini" y "fin" */

```
     resultado=borra(ini,fin,guia);
```

/* Si existia un segmento se guarda en el archivo CAD, de lo contrario se trataba de un par de puntos pertenecientes a otras lineas y se vuelven a dibujar en la imagen */

```
     if (resultado != 0 )
     {
      if( !((ini->j == fin->j) && (ini->k == fin->k)) )
       {
       printf("\n Linea desde %d,%d hasta %d,
              %d",ini->j,ini->k,fin->j,fin->k);
       archiva_linea((float)fin->j,(float)fin->k,
                     (float)ini->j,(float)ini->k);
        no_esq = 0;
       }
       return(1);
     }
     else
     {
       imgmap[ini->j][ini->k] = 1;
       imgmap[fin->j][fin->k] = 1;
```

```
        if ( re == 0 )
          return(0);
        else
        {
          if (boundary <= 1)
            boundary++;
          else
            boundary = 0;
        }
         return(1);
      }
    }
    else
    {

/*Si el numero de pixeles terminales es mayor
que dos, se deben analizar por parejas. inip
apunta al inicio del segmento. ini y fin van
apuntando a cada par de pixeles, investigando si
se trata de un espacio en blanco entre dos
segmentos, o si se trata de una porcion del
mismo segmento de recta al que apunta inip */
          inip = ini;
          fin = ini->ant;
          flag = 0;
          do
          {
            resultado = borra(ini,fin,guia);

/*Si resultado es cero se encontro un espacio en
blanco, quiere decir que se trata de, quiza, dos
segmentos de recta. Se indica el segmento
reconocido actualmente y se inicia la busqueda
del siguiente */
            if (resultado == 0)
            {
              imgmap[ini->j][ini->k] = 1;
              imgmap[fin->j][fin->k] = 1;
              if (flag != 0)
              {
                if( !((inip->j == ini->j) && (inip->k ==
                    ini->k)) )
                {
                  printf("\n Linea desde %d,%d
                      hasta %d,%d",inip->j,inip-
                      >k,ini->j,ini->k);
                  archiva_linea((float)ini->j,(float)ini-
                      >k,(float)inip->j,(float)inip->k);
                  no_esq = 0;
          }
              ini = fin;
              inip = ini;
              fin = ini->ant;
              flag = 0;
            }
            else
            {
              ini = fin;
              inip = ini;
              fin = ini->ant;
            }
          }
/*Si resultado es diferente de cero, entonces
seguimos analizando el mismo segmento de
recta. */
          else
          {
            ini = fin;
            fin = fin ->ant;
            flag++;
          }
        }
        while(fin != NULL);

/* Se almacena el segmento de recta final
localizado, excepto si se trata de un punto */
        if( !((inip->j == ini->j) && (inip->k == ini->k)) )
        {
          printf("\n Linea desde %d,%d hasta %d,
                %d",inip->j,inip->k,ini->j,ini->k);
          archiva_linea((float)ini->j,(float)ini->k,
                (float)inip->j,(float)inip->k);
          no_esq = 0;
        }
        return(1);
      }
    }
}

/*Esta funcion borra el segmento de recta entre
el pixel ini y el pixel fin con la finalidad de que no
interfiera con la busqueda posterior de otras
lineas. Ademas, indica la presencia del
segmento, o si se trata de un espacio entre
segmentos. */
int borra(ini,fin,guia)
struct limite* ini;
struct limite* fin;
struct parametros* guia;
{
  int j,h;
  double basura;
  double x,y,z;
  int lij,lsj,lik,lsk,d,cuenta;
  int k, x1,y1;

  cuenta = d = 0;

/*Se establecen los limites entre los cuales se
van a evaluar los segmentos */
  if (ini->j >= fin->j)
  {
    lsj = ini->j;
    lij = fin->j;
  }
  else
  {
    lsj = fin->j;
    lij = ini->j;
  }
  if (ini->k >= fin->k)
  {
```

```
    lsk = ini->k;
    lik = fin->k;
  }
  else
  {
    lsk = fin->k;
    lik = ini->k;
  }

 /*Si el angulo parametrico de la linea esta entre
45 y 90, o -45 y -90, la linea es casi horizontal,
por lo que evaluamos en esa direccion */
   if ( !((guia->teta < M_PI/4) && (guia->teta > -
       M_PI/4)) )
   {
     for(k=lik;k<=lsk;k++)
     {
           if ( k != 0)
             x = k - 0.5;
           else
             x = 0;
           z = (guia->rho - x * cos(guia-
                >teta))/(sin(guia->teta));
           y = J - z + 0.5;
           if ( y < 0)
             y = -1 * y;
           if ( h=fraccion((double)y) >= 0.5)
             y1 = ceil(y);
           else
             y1 = floor(y);

 /*Se borran todos los puntos del segmento.
Para borrarlos se les coloca un -1, para evitar
que se cuenten como puntos en blanco. Este -1
se cambia por cero en el siguiente analisis de
esquinas. Cuando se analizan lineas inclinadas,
es posible que los puntos  no coincidan
exactamente con el valor encontrado para la
linea parametrica, por lo que se les da un
umbral de error de un punto. */
       if((imgmap[y1][k] >= 1))
            imgmap[y1][k] = -1;
       else
         if((imgmap[y1+1][k] >= 1) &&
           (imgmap[y1][k] == 0) && (re != 0))
           imgmap[y1+1][k] = -1;
         else
           if((imgmap[y1-1][k] >= 1) &&
             (imgmap[y1][k] == 0) && (re != 0))
           imgmap[y1-1][k] = -1;
           else
/*Se lleva una cuenta de los pixeles blancos */
             if (imgmap[y1][k] == 0)
                  cuenta++;
     d++;
/* Y se lleva una cuenta de los pixeles
examinados (d)*/
     }

/*Si la cantidad de puntos blancos sobrepasa la
mitad de la cantidad de puntos analizados,
entonces consideramos que se trata de un
espacio en blanco, que tiene ruido. De lo
contrario, consideramos que se trata de una
linea con algunos defectos. */
     basura=fraccion( (double) ( (double)d/2.0) );
     if( (cuenta >= 3) && (cuenta >= (int)(
        (double)d/2.0 - basura)) && ((int)(
        (double)d/2.0 - basura) != 0))
          return(0);
     else
     {

/*Si el numero de pixeles evaluados es menor
que 3, entonces se trata de pixeles terminales
que se encuentran juntos, y pertenecen a la
misma linea, pero no forman un segmento
aparte, por lo tanto lo indicamos devolviendo un
valor 2. */
           if (d <= 3)
             return(2);
           else
             return(1);
     }
   }
   else

/*Si el angulo parametrico de la linea esta entre
45 y -45 grados, la linea tiende a una posicion
vertical, por lo tanto, conviene evaluar los
puntos en esta direccion. */
   {
     for(j=lij;j<=lsj;j++)
     {
           y = J - j + 0.5;
           z = (guia->rho - y * sin(guia
              ->teta))/(cos(guia->teta));
           x = z + 0.5;
           if ( x < 0)
             x = -1 * x;
           if ( h=fraccion((double)x) >= 0.5)
             x1 = ceil(x);
           else
             x1 = floor(x);
           if((imgmap[j][x1] >= 1))
             imgmap[j][x1] = -1;
           else
          if((imgmap[j][x1+1] >= 1) &&
           (imgmap[j][x1] == 0) && (re != 0))
             imgmap[j][x1+1] = -1;
           else
           if((imgmap[j][x1-1] >= 1) &&
             (imgmap[j][x1] == 0) && (re != 0))
             imgmap[j][x1-1] = -1;
           else
             if (imgmap[j][x1] == 0)
                 cuenta
           d++;
     }
     basura=fraccion( (double) ((double)d/2.0) );
     if( (cuenta >= 3) && (cuenta >= (int)(
        (double)d/2.0 - basura)) && ((int)(
```

```
        (double)d/2.0 - basura) != 0))
      return(0);
     else
         if (d <= 3)
           return(2);
         else
           return(1);
  }
}

/* Coloca en el archivo CAD los valores del
punto inicial y el punto final de segmento de
linea indicado */
void archiva_linea(x1,y1,x2,y2)
float x1;
float y1;
float x2;
float y2;
{
      fprintf(dxf,"%d\n",10);
      fprintf(dxf,"%f\n",x1);
      fprintf(dxf,"%d\n",20);
      fprintf(dxf,"%f\n",y1);
      fprintf(dxf,"%d\n",30);
      fprintf(dxf,"%f\n",0);
      fprintf(dxf,"%d\n",11);
      fprintf(dxf,"%f\n",x2);
      fprintf(dxf,"%d\n",21);
      fprintf(dxf,"%f\n",y2);
      fprintf(dxf,"%d\n",31);
      fprintf(dxf,"%f\n",0);
}

/* Devuelve la parte entera del numero doble "m"
*/
double fraccion(m)
double m;
{
  int z=0;
  double d;

  do
  {
   if(z <= m)
     z++;
   else
   {
     z--;
     d = m - z;
     return(d);
   }
  }
  while (z > 0);
}

/* Calculo de rho a partir del valor m del arreglo
de Hough */
double calcularho(m)
double m;
{
  double rhomax,rho;
  rhomax = sqrt(pow((double)(K-0.5),2.0)+
               pow((double)(J+0.5),2.0));
  rho = ((2 * rhomax * (m - M))/(M - 1)) +
         rhomax;
  return(rho);
}

/* Calculo de teta a partir del valor n del arreglo
de Hough */
double calculateta(n,tipo)
double n;
int tipo;
{
  double teta;
  teta = M_PI - ((2 * M_PI * (N - n))/(N - 1));
  if (tipo == 1)
    teta = (teta * 180)/M_PI;
  return(teta);
}
```