FACULTAD DE INFORMATICA
UNLP

TESINA DE GRADO

TITULO: **RECONOCIMIENTO AUTOMATICO DE TEXTO BRAILLE**

AUTOR: **A.C. HECTOR P. FERRARO**
DIRECTOR: **MG. CLAUDIA RUSSO**

CARRERA: **LICENCIATURA EN INFORMATICA**

Resumen

El presente trabajo consiste en el desarrollo de una herramienta que permita escanear un texto braille, reconocer en forma automática los caracteres y obtener el texto digital correspondiente para ser almacenado y recuperado en formato textual para varios propósitos.
Este tipo de herramienta puede ser útil para reimprimir documentos Braille, como ayuda en el aprendizaje de Braille o como medio de integración para estudiantes no videntes en todos los niveles o integración en el ámbito social, de personas no videntes.

Líneas de Investigación

Segmentación de Imágenes.
Reconocimiento de Patrones en Imágenes
Tratamiento de Imágenes con Java.

Trabajos Realizados

Estudio de estado del Arte.
Estudio de tipos y control de Escáneres.
Tratamiento de Imágenes con Java.
Desarrollo de una herramienta de Reconocimiento Automático de Texto Braille implementando métodos de segmentación de imágenes.

Conclusiones

Se logró desarrollar una herramienta que permite reconocer los caracteres formados por los puntos de una hoja Braille y realizar su traducción obteniendo el texto digital correspondiente, a partir de la imagen obtenida por un escáner convencional.

La misma está totalmente desarrollada en Java y su distribución y uso es libre siendo, según lo investigado, un desarrollo único en Argentina y Latinoamérica.

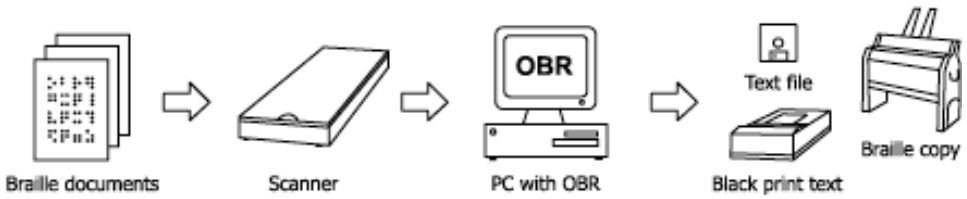
Trabajos Futuros

Procesamiento de hojas doble faz: La escritura Braille puede realizarse a doble faz. Este tipo de escritura se llama Braille interpunto. La segmentación de imágenes de documentos Braille de este tipo es compleja y costosa.

Detección Automática de tamaño y distancia entre puntos: Agregar algoritmos y procesos de detección automática de este tipo de parámetros podría ser de gran utilidad para los usuarios de esta herramienta.

Fecha de la Presentación: Noviembre de 2007

Reconocimiento Automático de Texto Braille



Alumno: **A.C. Héctor Ferraro**
Director: **Mg. Claudia Russo**
Noviembre 2007

ÍNDICE

A mis padres y hermanos, profesores y compañeros...

Capítulo 1 – Introducción

1.1 Presentación

El presente, es la Tesina de Grado para la obtención del titulo de Licenciado en Informática de Héctor Pascual Ferraro, alumno de la Facultad de Informática de la Universidad Nacional de la Plata.

Este consiste en desarrollar una herramienta que permita escanear un texto Braille con un escáner convencional, reconocer en forma automática los caracteres y obtener el texto digital correspondiente para ser almacenado y recuperado en formato textual para varios propósitos.

El proceso consta de cuatro partes:

- Digitalización de la imagen.
- Aplicación del procesamiento de segmentación.
- Aplicación del procesamiento del reconocimiento.
- Almacenamiento digital para su disponibilidad.

1.2 Definición del Problema

Actualmente la mayoría de las impresoras braille admiten una entrada ASCII. El problema radica en que si uno quiere realizar una copia de un texto en braille (del cual no se dispone en formato ASCII) debe tipear el material. Generalmente una persona lee el texto en braille y otra escribe lo que escucha en un procesador de texto, siendo este un proceso muy lento (como por ejemplo en traducciones de libros escritos por escritores no videntes). El desarrollo de una herramienta de estas características permitiría obtener de manera más eficiente el texto ASCII para poder ser impreso o reimpreso en una impresora braille, puesto que estas admiten texto digital como entrada de datos.

Por otro lado, ésta herramienta podría ser utilizada también en escuelas donde estudian alumnos videntes y no videntes, siendo de gran ayuda para maestros que no dominen completamente el lenguaje braille o inclusive para los propios compañeros videntes, pudiendo lograr una mayor integración tanto profesor-alumno como alumno-alumno.

1.3 Motivación y Expectativas

Es de gran interés contar con un desarrollo de estas características que sea de uso libre y pueda ser utilizado en institutos educacionales y ambientes públicos, a fin de integrar cada vez más a la sociedad a los discapacitados visuales.

Son varias las situaciones en donde el sistema ayudaría a la inserción social de los disminuidos visuales. Por ejemplo una persona no vidente concurre a una entidad a realizar un trámite, llevando una nota escrita en braille; el empleado que la atiende escanea la misma (con la herramienta propuesta) y la deriva a quien corresponda en formato de texto digital.

Otro caso son las empresas de servicios que remiten en braille las boletas de pago a pedido del interesado, los que podrían presentar un eventual reclamo en sistema braille.

También es interesante ver la aplicación en el ámbito estudiantil, en todos sus niveles por ejemplo un alumno universitario no vidente que debe entregar un trabajo práctico para

una materia (igual que sus compañeros videntes). Este estudiante podría escribir el texto en braille y darle una copia a su profesor de la salida obtenida por la herramienta, de esta manera no es necesario que el profesor sepa braille y el alumno se queda con su copia en braille para su posterior estudio.

Si bien existen herramientas de similares características, estas no son muchas y son de elevado costo monetario (alrededor de 1000 dólares estadounidenses).

Se propone desarrollar una herramienta multiplataforma que pueda ser usada gratuitamente. Es por la primera razón que se eligió el lenguaje JAVA para el desarrollo de la misma, además de por su capacidad para trabajar con imágenes.

1.4 Estructura del Trabajo

Esta tesis está dividida en siete capítulos:

- El capítulo 1 es una introducción a la problemática.
- El capítulo 2 habla sobre el estado del arte en el reconocimiento de texto Braille. Se exponen otros proyectos similares concluyendo con una comparación.
- En el capítulo 3 se estudia el Sistema Braille, sus características y particularidades.
- En el capítulo 4 se exponen los fundamentos teóricos del tratamiento digital de imágenes utilizados para el desarrollo de los algoritmos.
- El capítulo 5 es el eje central de esta tesis, en el se desarrollan los algoritmos para el reconocimiento automático que serán utilizados en la herramienta.
- En el capítulo 6 se exponen los aspectos de implementación de la herramienta desarrollada.
- Por último en el capítulo 7 se presentan las conclusiones del trabajo y las líneas de investigación futuras.

También se incluyen un apéndice sobre formatos de imágenes otro sobre escáneres y la presentación que se realizó en el IV Workshop de Computación Gráfica, Imágenes y Visualización del XII Congreso Argentino de Ciencias de la Computación CACIC 2006.

1.5 Sobre la Bibliografía y Referencias

Las referencias a la bibliografía aparecen como un código de la forma XXXnnn, donde:

- XXX, son las tres primeras letras del apellido del autor o empresa que realiza la publicación
- nnn, es el año de publicación.

Capítulo 2 – Estado del Arte

2.1 Introducción

Buscando en internet se han encontrado, aunque no muchas, referencias a otros proyectos similares a este.

Algunos son proyectos de universidades europeas. No se encontró referencias a proyectos de este tipo desarrollados por universidades de Argentina, ni de América.

Según la información encontrada, existe una aplicación comercial llamada OBR [WEB001]. El valor comercial de la misma es de alrededor de mil dólares.

2.2 Proyecto de la universidad de Vigo y la O.N.C.E

Este proyecto comenzó en 1994. Consistió en el desarrollo de una herramienta que permitiera escanear hojas Braille y convertirlas en texto digital. La versión final fue entregada en 1997 y validada ese mismo año por la unidad de tiflotecnología de la O.N.C.E (Organización Nacional de Ciegos Españoles) [UVO097].

La aplicación fue pensada para ejecutarse en sistemas Windows. La interfaz de usuario se desarrolló en Visual Basic, mientras que los algoritmos de reconocimiento fueron programados en C y compilados como librerías de enlace dinámico (DLL´s).

El esquema de etapas para el reconocimiento que usaron es el siguiente:

- Escaneo de la imagen
- Umbralización de la imagen
- Búsqueda de manchas blancas y negras
- Búsqueda de puntos Braille
- Detección de la inclinación de la hoja
- Detección del primer punto de la hoja
- Detección de la malla Braille (todos los posibles puntos)
- Reconocimiento
- Transformación a texto digital

Una vez escaneada la imagen en escala de grises le realizan un proceso de umbralamiento, llevándola a una imagen en tres colores, donde los puntos Braille vienen dados por parejas de manchas blancas y negras; siendo el color de fondo gris.

Luego buscan los centros de las manchas blancas y negras para después juntar parejas de manchas cercanas y de esta forma determinar algunos puntos Braille de la hoja. Estos servirán para, entre otras cosas, detectar la inclinación de la hoja sobre el escáner. La detección de la inclinación la realizan con la transformada de Hough.

Una vez hallados los puntos, buscan el primero de la hoja y crean la malla de los posibles puntos Braille del documento a partir de este. Luego inspeccionan cada posible punto para ver si este está activo o no, para después realizar la conversión a texto digital.

La herramienta que desarrollaron contiene ciertas facilidades muy útiles para sus usuarios. Se puede escanear hojas grandes por trozos. La herramienta realiza el encajado de manera que parezca que se ha escaneado en una sola pasada. Esto permite trabajar un escáner A4 y manejar hojas de tamaño 25 x 31.5 (un poco mayores que las A4) que son las más habituales en la escritura Braille.

Según publican los autores de este proyecto, el porcentaje de acierto en el reconocimiento de los puntos Braille es de alrededor del 97 %.

2.3 Proyecto de la Universidad de Manchester

Se detectan los puntos con un modelo muy parecido al planteado en el proyecto de la universidad de Vigo (basándose en las manchas blancas y obscuras).

Este sistema intenta corregir los errores provocados por las manchas unidas desarrollando un método para romperlas. Cuando aparece una pareja de manchas y una de ellas es demasiado grande, se borra parte de la mancha grande (se borra la intersección con la proyección vertical de la mancha pequeña).

No existe corrección de la inclinación. Este sistema también es capaz de leer las dos caras en un solo reconocimiento. Además dispone de un diccionario integrado para efectuar comprobaciones sobre el texto reconocido [APO096].

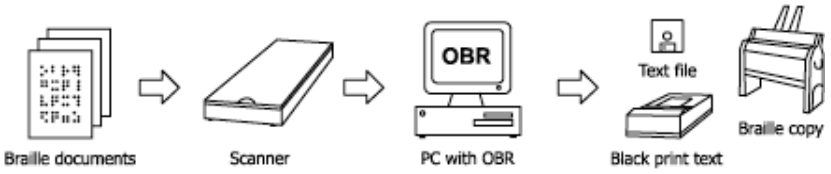
Los autores publican el porcentaje de acierto sobre el punto Braille (es decir, el porcentaje de puntos encontrados sobre el total de existentes), obteniendo un 98.5 % para los puntos del anverso (puntos hacia arriba) y un 97.6 % para los del reverso (puntos hacia abajo). También afirman que tras sus pruebas en hojas con una media de 328 caracteres solo fallan en 11 (un porcentaje de acierto del 96.74 %).

2.4 OBR – Aplicación Comercial

OBR (reconocedor óptico de Braille) es un software para sistemas Windows que permite leer documentos Braille simple y doble faz con un escáner estantard [OBR000].

Una vez traducido el documento la información es presentada como texto digital (ascii) que puede ser usado en cualquier tipo de aplicaciones Windows.

El proceso de escaneo es simple y rápido. Con solo unos pocos comandos y en menos de 30 segundos, se obtiene la representación textual de una página Braille de ambos lados en un solo escaneo.



Este gráfico muestra la operatoria del OBR

La aplicación también permite trabajar con hojas más grandes que el formato A4. El documento se escanea por partes y luego se hace un merge. Esta característica es muy interesante ya que los escáneres A4 son mucho más económicos que los de formato mas grande.

Según afirman sus creadores, el reconocimiento de un documento Braille de alta calidad es excelente; mientras que en el escaneo de documentos Braille viejos (gastados), la tasa de fallos es bastante baja. Se afirma una tasa de aciertos de alrededor del 99.98 %.

No se encontró información sobre como están desarrollados los algoritmos que intervienen en el reconocimiento.

2.5 Documentos de la IEEE

En el sitio de la IEEE [IEE05] se encontraron algunos papers que tratan este tema. No hablan de implementaciones comerciales concretas, pero si de los métodos y modelos utilizados.

Los más interesantes son los siguientes:

2.5.1 Optical Recognition of Braille Writing Using Standard Equipment

Este paper es de diciembre de 1994 [MTF094]. Los investigadores realizan el reconocimiento de esta manera:

Crean una máscara para un punto Braille estándar. Convolucionan esta máscara con la imagen, resultando picos en la posición de los puntos Braille. Estos picos los agrupan en celdas para convertirlos en caracteres. Esto significa que también calculan la malla sobre la cual los puntos Braille están posicionados.

Según afirman, el mayor problema de esta técnica es que los puntos Braille varían un poco en tamaño y shape, inclusive en la misma página. Además se generan algunos picos falsos en los lugares donde los puntos de una cara y la otra están muy cerca (casi juntos).

En vista de estos problemas se decidieron adoptar un enfoque diferente. Usando una máscara muy simple se intentan encontrar la posición de las líneas de la grilla.

Para detectar la inclinación de la imagen sobre el escáner trabajaron con la desviación de las suma de las filas. Cada vez la imagen es girada un píxel en dirección vertical y se calcula la desviación de la suma de las filas. Se obtiene un máximo cuando los puntos están alineados horizontalmente.

Con respecto a los aspectos técnicos y de implementación, utilizaron un escáner A3 para el escaneo de la imagen mientras que los algoritmos fueron desarrollados en MATLAB y luego la versión final fue programada en C++. El ambiente de desarrollo fue el MacApp 3.01 de una computadora Apple.

Los resultados los dividen en tres grupos. En el primer grupo (cuatro sets) fueron convertidos sin ningún error. En el segundo grupo (siete sets) obtuvieron un promedio de error de 0,25 % sobre los caracteres básicos. En este grupo los errores se debieron a

defectos en la superficie del papel. En el último grupo (un set) no se pudo realizar el reconocimiento con un error aceptable, debido a una deformación particular de los caracteres de la matriz con los cuales el programa no pudo operar.

2.5.2 Regular Feature Extraction for Recognition of Braille

Este paper fue escrito por investigadores del Departamento de Computación y Matemática del HK Technical College (Chai wan) de Hong Kong, China [VNG097].

En su modelo el reconocimiento consiste de seis etapas:

- Restricciones de escena: explotar e imponer restricciones de ambiente para reducir la complejidad de las etapas siguientes, a fin de llevarlas a un nivel manejable.
- Adquisición de la imagen: captura de la pagina Braille y conversión de esta en un array con los datos de cada píxel de la imagen.
- Preprocesamiento: modificar y preparar los valores de los píxeles de la imagen digitalizada para las operaciones siguientes. Esto incluye realce y ajuste del contraste, reducción de ruido y detección de los bordes de los puntos Braille.
- Segmentación: separar la imagen adquirida en regiones de puntos salientes y hoyos. Esta operación separa los puntos de cada cara de la hoja.
- Extracción de características: Extrae los puntos Braille agrupándolos en celdas.
- Interpretación: convierte las celdas Braille en el texto correspondiente.

Para la adquisición utilizaron una cámara digital con una resolución de 512 dpi en escala de grises iluminando la hoja a 45 grados con luz amarilla polarizada.

La etapa de preprocesamiento consiste de dos operaciones: filtrado del ruido de adquisición y realce de bordes. Para reducir el ruido de adquisición aplican un filtro Gaussiano pasa-bajos. Para el realce de bordes aplican dos operaciones de filtrado independiente usando máscaras de Sobel (denotadas por X e Y), de la siguiente forma:

$$\text{output} = |\text{convolute(input, X)}| + |\text{convolute(input, Y)}|$$

En la etapa de segmentación binarizan la imagen. Para ello consideran un histograma hecho solo con los puntos de la imagen cercanos a los bordes de los puntos Braille. Los histogramas resultantes contienen picos puntiagudos y valles largos.

Para determinar los píxeles cercanos a los bordes primero aplican el operador Laplaciano a toda la imagen para identificar los puntos de borde. El promedio de los valores de nivel de gris de los píxeles de borde se usa como valor global de umbral (threshold). Con este valor de umbral binarizan la imagen.

La función principal de la etapa de extracción de características es extraer los puntos Braille de la imagen binarizada. Como la forma de un punto Braille es conocida, los píxeles de frontera de los puntos Braille pueden ser obtenidos por el boundary based Chain Code algorithm. Este algoritmo detecta las coordenadas de frontera de los puntos Braille.

Como las coordenadas de los puntos son detectadas, se puede deducir el diámetro de los mismos. Como los puntos son embosados en ambas caras de la página, los que estan en el frente deben ser separados de los que están en el dorso antes de seguir el procesamiento.

Las características de convexidad y concavidad de los puntos reflejan la iluminación de la hoja en dos ángulos diferentes. Estas características pueden ser usadas para distinguir los puntos del frente y del dorso de la página.

Basándose en la información de coordenadas de frontera y las características de reflección de la iluminación construyen dos templates estandards para representar los puntos del frente y del dorso. Estos templates se aplican en cada posición de la imagen evaluando la correlación con cada posición de píxel. Dependiendo de los valores de correlación, se extraen los puntos del frente y del dorso.

La función de la etapa de interpretación es agrupar los puntos Braille hallados, en celdas (caracteres Braille) para convertirlas en el texto digital correspondiente. Para cada punto, se mide la distancia entre su centro y el de los cuatro puntos vecinos.

Con esta información, los puntos se agrupan en celdas. Para cada celda se determina el patrón de puntos y se lo representa como una cadena de bits. La cadena de bits se busca en un diccionario (alfabeto) Braille y se obtiene el carácter textual correspondiente. Estos se agrupan en palabras. Luego cada palabra se chequea contra un diccionario ingles. Si la palabra no se encuentra, se selecciona y se resalta en pantalla la palabra con mayor porcentaje de similitud, para posterior edición.

Con respecto a los resultados, los autores de este paper afirman un margen de acierto de 100 % para páginas simple faz y 97 % para páginas doble faz.

2.6 Conclusiones

Casi todos los proyectos expuestos siguen un esquema de solución similar: segmentan la imagen utilizando técnicas de umbralización de imágenes para convertir la información de puntos Braille en manchas blancas y negras, hallan los puntos y luego fabrican la malla de los posibles puntos de la hoja para ver cuales de esos puntos están activos y de ésta forma determinar de que carácter de texto se trata.

En general, las implementaciones son solo para sistemas windows; imposibilitando el uso de este tipo de herramientas en sistemas linux, mac, etc.

Por otro lado, parece no haber disponibles versiones libres de estas implementaciones, que puedan ser bajadas desde la web.

Capítulo 3 – Sistema Braille

3.1 Introducción

El braille es un medio táctil de lectura y escritura, a través de la yema de los dedos, consistente en unos puntos en relieve organizados de forma parecida a los del dominó.

Mediante este sistema las personas que no ven nada, o aquellas cuyo resto visual no les permite la lectura a través de los medios y soportes convencionales (impresión en tinta, lápiz, etc.), pueden leer e intercambiar información tanto con otras personas con ceguera como con personas que ven.

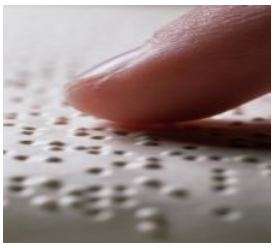


Figura 1. Leyendo Braille

Precisamente, y dadas su sencillez y manejabilidad, también las personas que ven pueden aprenderlo sin gran esfuerzo pero leyéndolo visualmente.

La difusión y conocimiento del sistema braille entre las personas que ven supone, en consecuencia, un elemento facilitador de la comunicación con las personas con ceguera y, en última instancia, una contribución inestimable a su integración y participación social.

3.2 El Sistema Braille

El sistema Braille no es un idioma, ni una lengua secreta o esotérica. El braille es, simplemente, un alfabeto. Lo único que lo diferencia de las letras comunes o "en tinta" es que cada uno de los grafemas se representa por medio de puntos en relieve.

El llamado "signo generador", del cual parten todas las letras, tiene seis puntos, ordenados y numerados de la siguiente forma:

1 • • 4
2 • • 5
3 • • 6

Combinando estos seis puntos por presencia o ausencia de los mismos se obtienen 63 signos o figuras diferentes. Algunos autores hablan de 64 combinaciones tomando también al cero (el espacio en blanco entre palabras) como un signo más.

De más está decir que, al tratarse de un alfabeto, con él se pueden representar letras y palabras en cualquier idioma, ya que la "a" (punto uno) tanto sirve para el francés, el alemán o el castellano.

Hablamos de sistema y no exclusivamente de alfabeto braille porque, utilizando sus 63 combinaciones, se han desarrollado distintos códigos, en unos casos usando signos que no tienen valor de letra y, en otros, asignándole nuevos valores a las propias letras, ya sea por el contexto en el que se encuentran o anteponiendo unos "marcadores" que alteran su significado original.

Los principales códigos desarrollados son: código matemático y científico; la musicografía (escritura musical) y la "estenografía" o escritura abreviada en sus grados II y III.

El hecho de utilizar solo 63 combinaciones obligó a la invención de los llamados símbolos dobles o también llamados determinadores, para representar números, mayúsculas y expresiones matemáticas.

El tamaño y distribución de los seis puntos que forman el llamado Signo Generador, no es un capricho sino el fruto de la experiencia de Louis Braille. Las terminaciones nerviosas de la yema del dedo distinguen con facilidad este tamaño en particular.

Es de hacer notar que el propio Luis Braille, además de inventar el sistema, desarrolló los primeros códigos matemáticos, musicales y estenográficos, la mayor parte de los cuales se usan aún como él los diseñara.

3.3 ¿Quién fue Louis Braille?

Louis Braille, el inventor del sistema de lectura y escritura que lleva su nombre, no nació ciego. A la edad de tres años pierde la vista en un accidente que ha sido contado de distinta manera por quienes se han ocupado de su biografía, dado que no existen documentos que nos informen fehacientemente del mismo.

Su padre, Simón-René, era talabartero en el pueblo de Coupvray (distante unos 40 kilómetros al este de París), oficio que la familia Braille venía desarrollando desde varias generaciones atrás en ese pueblo.

Se cuenta que un día, jugando en el taller familiar, Louis se clava una lezna en un ojo. Parece que la herida se infectó y, por simpatía, pierde también el otro ojo quedándose totalmente ciego, como ya se dijo, a la edad de tres años.

Se trataba de un chico vivaz e inteligente, que contó con un gran apoyo familiar (sorprendente para la época), lo que le permitió seguir desarrollando sus habilidades.

Cuando Louis cuenta con ocho años, su padre consigue que el maestro del pueblo lo acepte en sus clases y allí demuestra sus dotes como alumno aunque sólo podía seguir las clases de forma oral.

Más tarde, el maestro tiene noticias de que existe una escuela para ciegos en París. Como la familia no disponía de recursos, le consiguen una beca y es así como el 15 de febrero

de 1819, a la edad de diez años, Louis parte de su pueblo natal para residir en el colegio como interno.

Allí también se destaca rápidamente por su inteligencia y vivacidad. Tanto es así que, contando con apenas doce años, el director del colegio le encarga una tarea sorprendente, si se tiene en cuenta su edad: evaluar la llamada "sonografía", o sistema Barbier.

En ese colegio se enseñaba a leer con lo que se conoce como método Haüy, que consistía en imprimir en alto relieve y sobre un papel resistente las letras comunes que usan las personas que ven.

Aunque Louis aprendió rápidamente a leer con ese sistema, el mismo presentaba dos graves inconvenientes: en primer lugar, con él no se podía escribir y, en segundo lugar, la lectura resultaba muy trabajosa, dado que era necesario utilizar tipos grandes para ser percibidos por el tacto y su tamaño, entonces, requería el lento reconocimiento de cada letra antes de pasar a la siguiente.

Un día del año 1821 se presenta en la escuela Charles Barbier de la Serre. Se trataba de un capitán de artillería del ejército de Luis XVIII quien sostiene haber creado un sistema que permite a los ciegos leer. El director del colegio, de entre todos los profesores y alumnos, convoca a Louis para que valore las posibilidades del invento de Barbier.

La sorpresa y hasta el "mosqueo" del militar fueron, según parece, mayúsculos. Louis contaba con doce años de edad y el capitán no estaba dispuesto a que su gran invento fuera juzgado y analizado por quien él consideraba un "mocoso".

Louis, en cambio, se sintió maravillado. Sus dedos podían percibir perfectamente esos signos y, además, con ellos ¡se podía escribir!.

El sistema de Barbier, que él denominaba con dos nombres (escritura nocturna o sonografía), consistía en unos signos formados por la combinación de doce puntos, distribuidos en dos filas verticales de seis cada una. La presencia o ausencia de puntos generaba cada una de las grafías.

Barbier lo desarrolló para que los soldados pudieran comunicarse en la oscuridad y de ahí el nombre de "escritura nocturna". Se podía escribir con una pauta y un punzón sobre un papel resistente y se leía con las yemas de los dedos.

Presentaba dos graves inconvenientes rápidamente detectados por el joven Louis: los signos resultaban demasiado grandes, con lo cual no se podían percibir, en su totalidad, de una vez, con la yema de los dedos y, por otra parte, no constituía un alfabeto sino una "sonografía". Es decir, representaba los sonidos, pero no la ortografía de cada palabra.

Louis aporta a ese mecanismo dos modificaciones esenciales: por un lado reduce su tamaño (de 12 a 6 puntos como máximo para cada signo, colocados en dos filas verticales de tres puntos cada una) y lo transforma o, mejor dicho, inventa un alfabeto.

El capitán Barbier, sólo al final de su vida, y a regañadientes, acepta dichas modificaciones.

El propio Braille, al publicar el método, en el cual expone su sistema, en 1827, señala que se ha limitado a adaptar la sonografía de Barbier.

Y no sólo inventa el alfabeto: lo adapta a las matemáticas y a las ciencias, desarrolla un sistema de abreviaturas y, lo que resulta más interesante, lo adecúa también para la música. La llamada "musicografía" braille es, realmente, muy inteligente, ya que transforma la escritura musical, de vertical, en otra horizontal y consecutiva. Hasta entonces las personas ciegas debían aprender las partituras de memoria y exclusivamente "de oído", ya que los intentos de Haüy por ponerla en relieve resultaron (igual que hoy) infructuosos.

El sistema braille y su inventor chocaron con múltiples actitudes de rechazo. En primer lugar del capitán Barbier, que se negó durante años a aceptar modificaciones; en segundo lugar de los seguidores de Haüy que no estaban dispuestos a cederle terreno; y en tercer lugar, de las personas con vista que consideraban que el braille aislaba a los ciegos dado que ellos no podían leerlo (cosa que no es cierta, ya que cualquier persona que ve, a poco que se lo proponga, puede leerlo con la vista).

Durante varios años, el propio Louis Braille y sus compañeros ciegos lo utilizaron a escondidas dentro de los muros del Instituto, dado que llegó a estar prohibido.

Braille, además, inventó una pauta para escribir con su sistema y un aparato llamado Rafígrafo (este último en colaboración con Foucault, otro ciego ilustre) por medio del cual se podían escribir las letras comunes con puntos en relieve y que les permitía comunicarse con las personas que ven.

En el año 1840 se acepta oficialmente el sistema braille y, en 1878, un congreso internacional decide promoverlo en el mundo entero al considerarlo el mejor sistema para el tacto.

Louis Braille, que había nacido el 4 de enero de 1809, murió de tuberculosis en París a la edad de 43 años, y fue enterrado en Coupvray, su pueblo natal. Hoy día sus restos descansan en el Panteón de Hombres Ilustres no muy lejos del edificio (inaugurado en 1844) que aún ocupa el Instituto de Jóvenes Ciegos donde falleciera el 6 de enero de 1852.

Su casa natal es un museo. También en su pueblo se ha erigido un monumento a su memoria y sus manos se conservan en una urna en el cementerio de Coupvray.

3.4 Estructura del código Braille

Los caracteres braille tienen un tamaño (tipo y cuerpo) estándar. No deben ser agrandados ni achicados, porque ello dificulta su percepción táctil, ya que ha de ser percibida de forma completa cuando la yema del dedo se posa en ella.

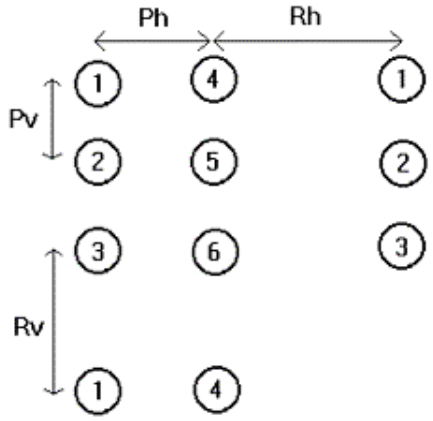
La escritura en braille tampoco se "justifica" dado que la justificación implica alterar la distancia entre caracteres, con lo cual se modifican las referencias espaciales entre los signos. Esto es muy importante, ya que el valor de cada signo depende, para su mejor y más rápida comprensión, tanto de los puntos que en sí mismo contenga como de su relación con los demás.

El Braille, al tacto, se percibe como pequeños relieves punzantes en positivo, practicados sobre la superficie en que se imprime, generalmente papel, pero puede tratarse de otro material laminado (plástico, cartulina, cinta adhesiva, etc.) del grosor apropiado, que son unos 0,15 milímetros como mínimo. Las hendiduras, al ser de una profundidad y grosor muy específicos, no llegan a perforar el papel (siempre que éste no sea demasiado fino o sea de poca calidad) sino que generan unas pequeñas depresiones que son cómodas al tacto y muy fáciles de reconocer una vez se tiene aquél bien adiestrado.

Existe, sin embargo, la desventaja de que al oprimir los puntos con los dedos mientras se lee, como también durante el transporte de los escritos, éstos pueden perder su firmeza y borrarse a veces casi completamente.

Cada carácter del código se compone de seis puntos. Los puntos de cada carácter están colocados en dos filas verticales contiguas de tres cada una, ligeramente separados pero todos a igual distancia del adyacente en ambas direcciones, y se disponen numerados del uno al seis, constituyendo lo que llamamos una celda: a la izquierda figuran el uno (arriba), el dos (en medio) y el tres (abajo), y a la derecha hay el cuatro (arriba), el cinco (en medio) y el seis (abajo).

Una celda de Braille se separa de sus compañeras por la izquierda y por la derecha con una distancia equivalente a medio punto, y de las inferiores y superiores por el espacio correspondiente a un punto entero (ver Figura 2).



Los valores de Ph, Rh, Pv y Rv son un estandard y sus valores son:

Ph = 2.5 mm
Pv = 2.5 mm
Rh = 5 mm
Rv = 5 mm

Algunas veces se usan valores mayores para facilitar la lectura a personas con capacidad táctil reducida o que están aprendiendo a leer Braille.

Figura 2. Estructura del código Braille

Los valores de Ph, Rh, Pv y Rv son un estándar. Algunas veces se usan valores mayores a los estandarizados, para facilitar la lectura a personas con capacidad táctil reducida o que están aprendiendo a leer Braille.

En una hoja que tenga las dimensiones de un folio convencional (21×30 centímetros) caben unas 28 líneas de 32 celdillas cada una. Si se dispone de los instrumentos adecuados, se puede escribir una página de Braille por las dos caras, lo que se denomina "interpunto", pues las distancias a que nos referíamos al principio del párrafo anterior

fueron ideadas con tal objetivo.

La composición de un carácter Braille se efectúa combinando la aparición o no de cada uno de los seis puntos disponibles en él, lográndose por tanto un máximo de 64 símbolos posibles. Este número es muy reducido dada la gran cantidad de caracteres que posee la escritura convencional. Este hecho ocasiona que algunos caracteres precisen más de una celda para ser representados, o equivalgan a un mismo símbolo Braille según el contexto en que se lean.

Lo primero que debe saber una persona interesada en aprender la lectura en Braille, antes que la formación de las letras y demás símbolos, es calibrar el espacio ocupado por cada carácter, cosa muy sencilla a la vista pero ciertamente laboriosa al tacto, a fin de que más adelante no mezcle las líneas y columnas que vaya a leer.

3.5 Alfabeto Braille

3.5.1 Letras

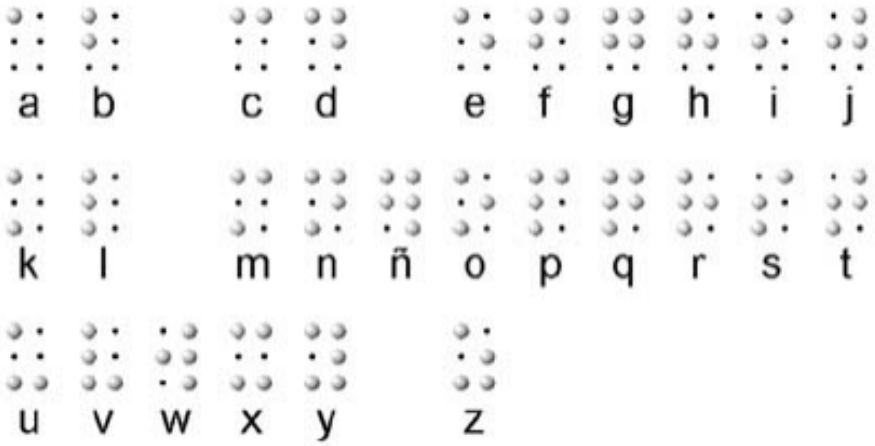


Figura 3. Letras del alfabeto Braille

Si se observa con atención la Figura 3, se nota que los símbolos correspondientes a la primera fila ocupan sólo los cuatro puntos superiores del signo generador. Los que corresponden a la segunda fila son iguales a los de la primera, pero se le agrega el punto inferior izquierdo (salvo la ñ que es propia del idioma español), y en los de la tercera se agregan los dos inferiores.

3.5.2 Vocales acentuadas

Ya que no es posible colocar un tilde encima de los puntos correspondientes a las vocales se tuvo que inventar un nuevo símbolo para cada una.

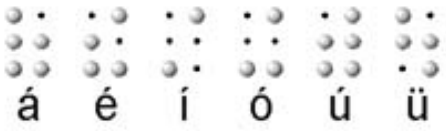


Figura 4. Vocales acentuadas

3.5.3 Signos de puntuación

En los signos de puntuación correspondientes a la admiración, interrogación y comillas no se diferencian los símbolos de abrir y cerrar.

Figura 5. Signos de puntuación

3.5.4 Mayúsculas

Las letras mayúsculas se representan anteponiendo el determinador de mayúscula a la letra en cuestión.

Figura 6-A. Determinador de mayúscula

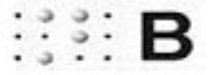


Figura 6-B. Ejemplo: letra B mayúscula

Para representar una palabra completa en mayúsculas se antepone dos veces el determinador de mayúscula a la palabra.

3.5.5 Números

Los números se forman utilizando las primeras letras del alfabeto precedidas por el determinador de número. Se utilizan las letras de la "a" a la "j" correspondiendo con los números del 1 al 9 y luego el 0.

Figura 7-A. Determinador de Número

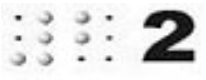


Figura 7-B. Ejemplo: número dos

Los números fraccionarios tienen la particularidad de que el numerador se representa con el mismo conjunto de puntos que el número normal, pero utilizando los cuatro puntos de abajo.

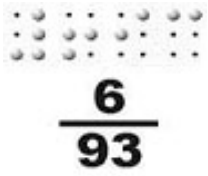


Figura 8. Ejemplo: número fraccionario

3.6 ¿Como se escribe en Braille?

3.6.1 Punzón y Regleta

El mismo Louis Braille inventó los dos utensilios fundamentales que se requieren para escribir. Estos son, una pauta y un punzón.

La pauta o regleta (Figura 9-B) consiste en una plancha de plástico o metal en la cual se graban surcos paralelos o líneas de puntos cóncavos en sentido horizontal. A esta plancha se superpone otra (o una guía) formada por filas de rectángulos (cajetines), cada uno de los cuales abarca tres surcos o seis puntos cóncavos y que constituyen los generadores o cajetines de los signos braille.

Figura 9-A Punzones para escritura Braille

Figura 9-B Regleta o Pauta Braille

Entre estas dos planchas, o la plancha y su guía, se coloca el papel y, mediante el punzón (Figura 9-A), se graba un signo braille en cada cajetín.

Existen muchos "estilos" de punzones, generalmente formados por una "cabeza" de plástico o madera por medio de la cual se sujeta, y una punta metálica con la cual se graba el papel. La característica fundamental que deben presentar los punzones es que su punta sea roma, para no perforar el papel.

3.6.2 Máquinas de Escribir Mecánicas

También se han desarrollado varios modelos de máquinas de escribir mecánicas. Una de las más difundidas es la máquina Perkins (ver figura 10), desarrollada en EEUU por David Abraham y comercializada por la Howe Press, Perkins School for the Blind (Massachussets).

Estas máquinas constan sólo de 9 teclas. Siete centrales, que representan los 6 puntos braille más el espaciador y otras dos separadas que son: la tecla de retroceso (retrocede un espacio cada vez) y la de avance de línea (sería como el "intro" de un teclado de ordenador).

Figura 10 Maquina Perkins – Perkins Brailler

Las 6 teclas correspondientes al generador braille accionan, mediante un sistema de palancas, 6 punzones que se encuentran en la "cabeza estampadora" de la máquina. Esta cabeza estampadora se desplaza en forma horizontal, de izquierda a derecha, cubriendo el ancho del renglón.

A diferencia de las máquinas de escribir corrientes, el "carro" permanece fijo y lo que se desplaza es la cabeza grabadora o estampadora.

Prácticamente todas las personas que se enfrentan por primera vez a una máquina de escribir en braille se sorprenden de las pocas teclas que posee (generalmente, además, dudan de que efectivamente se pueda escribir con ella).

Pero es que, siendo el braille un invento tan sencillo como genial, las máquinas diseñadas para escribir con él no podían permanecer al margen de estas dos características.

3.6.3 Aparatos Informáticos

Con el advenimiento y creciente desarrollo de la informática, el uso tanto de las regletas como de las máquinas mecánicas, está siendo relegado por la utilización de los aparatos informáticos.

Uno de los más populares, el "Braille Hablado", comparte con las máquinas Perkins, las dos características fundamentales de las mismas: es tan sencillo como genial.

Se trata de un miniordenador personal (o notebook) para ciegos que tiene un teclado similar al de las Perkins, pero que, en lugar de grabar en papel, lo hace en forma electrónica; ya sea en su memoria RAM o sobre un diskette. Luego lee lo escrito a través de un dispositivo de voz sintética.

3.7 Como se Lee en Braille

No es totalmente cierto, que el acto de leer en braille sea totalmente secuencial y necesariamente lento. Lo es en medida más o menos similar que lo es para una persona que ve la lectura de las letras en tinta.

Como lo demostraron los múltiples estudios realizados para elaborar los sistemas estenográficos (escritura abreviada), con frecuencia las personas ciegas, al leer, saltan parte de la palabra por haberla reconocido ya.

No es necesario, y no debiera fomentarse nunca, el "contar" los puntos que conforman cada signo. Lo que se percibe es un figura sobre la yema del dedo, y no la cantidad exacta de puntos que lo constituyen. Por otra parte, en la aprehensión de las formas gráficas del braille, tienen tanta importancia los puntos que están como los que no están. Los "huecos" que deja la ausencia de puntos tienen tanto valor semántico como los puntos mismos.

3.8 Otros Conceptos Asociados

3.8.1 Esteganografía

El código obtenido por la sustitución de grupos de letras e incluso palabras completas por unos pocos caracteres Braille, similarmente a las conocidas contracciones de la lengua castellana "al" y "del" se denomina estenografía o Braille grado 2.

De esta forma se puede escribir texto Braille utilizando menos caracteres y por lo tanto ocupando menos espacio.

3.8.2 Braille de 8 puntos

La llegada de la era de la informática, lejos de suponer el inicio de la decadencia del sistema Braille, ha supuesto un beneficio pues se ha visto reforzado por las posibilidades que ofrecen las nuevas tecnologías. Sin embargo, sí ha supuesto un nuevo cambio, pues a pesar de la versatilidad de los 6 puntos y los 64 símbolos diferentes, las necesidades informáticas exigían una tabla de combinaciones mayores, que ha hecho necesario añadir dos puntos más a los seis originales con el fin de cubrir los nuevos requerimientos.

En este caso, la celda Braille esta formada por 8 puntos distribuidos de la siguiente forma: cuatro a la izquierda y cuatro a la derecha. Se la puede ver como una celda Braille normal a la que se le ha agregado una fila de puntos.

1 . . 4

2 . . 5

3 . . 6

7 . . 8

Esta nueva configuración permite codificar 256 símbolos distintos y se lo usa solo en informática.

El llamado Braille Informático ha permitido aumentar las posibilidades de imprimir libros, revistas y toda clase de documentos en tal proporción que hubiese sido imposible hace sólo dos décadas atrás, permitiendo, incluso, la fabricación de impresoras braille de uso personal y que muchos centros dispongan de impresoras braille industriales para uso colectivo.

3.8.3 Tiflotecnologia

Por tiflotecnología (del griego "tiflon", ciego) se entiende el uso de la tecnología aplicada a aquellos aparatos y dispositivos que le permiten a las personas ciegas y deficientes visuales acceder a muchas tareas y conocimientos como son, por ejemplo la lectura y la utilización de ordenadores.

En el campo del braille se ha aplicado a la producción de libros, la lectura, el acceso a los ordenadores con todas sus posibilidades.

La lista de programas y dispositivos que hacen esto posible es larga y necesariamente temporal ya que, prácticamente día a día, aparecen nuevos productos. Pero la mayoría de ellos consisten en:

- Programas:

  Pueden dividirse en dos clases: los destinados a usuarios ciegos y los destinados a usuarios deficientes visuales. Los primeros "transforman" la información de pantalla por medio de dos recursos: el braille y la voz sintética. Los destinados a deficientes visuales lo que hacen es agrandar los tipos de letra de pantalla ofreciendo también otras múltiples posibilidades como agrandar sólo una línea, un sector de la pantalla, ofrecer distintos tipos de fondo (fondo oscuro, letras claras o viceversa), según las necesidades visuales del usuario.

- Periféricos de ordenadores:

  Los programas que transforman la información al braille lo hacen a través de las "líneas braille" (visores) que utilizan el llamado "braille efímero". Consisten en una línea de "signos generadores" (los 6 puntos) formados por pequeños punzones móviles que van tomando la forma de cada letra por impulsos eléctricos.

Para la impresión de texto se han desarrollado impresoras braille. Las hay de muchos tipos y para distintos fines, desde las personales hasta las destinadas a realizar grandes tiradas y que utilizan el papel por ambas caras (interpunto).

- Máquinas de leer:

  Consisten en aparatos independientes de los ordenadores (aunque generalmente compatibles con ellos) que llevan un escaner para capturar el texto en tinta y la salida se produce ya sea en braille o a través del uso de la voz sintética.

  Las personas deficientes visuales pueden servirse también de una gran ayuda, como son las "lupas televisión", para acceder a los textos escritos. Consiste en un circuito cerrado de televisión que amplía en su pantalla el texto y que permite varias posibilidades de tamaño y fondo según las necesidades del usuario.

Nota:

Para el desarrollo de este capítulo se utilizaron las referencias bibliográficas DEV099, BBL005, AFB005, FMC005, FBU005

Capítulo 4 – Fundamentos para el reconocimiento de texto Braille

4.1 Introducción

4.1.1 Representación digital de imágenes

Una imagen digital es una imagen f(x, y) que se ha discretizado en sus coordenadas espaciales como en su brillo. Una imagen digital puede considerarse como una matriz cuyos índices de fila y columna identifican un punto de la imagen y el valor del correspondiente elemento para esa fila y columna indica el nivel de gris en ese punto.

Los elementos que componen una imagen digital se denominan píxels (de su abreviación inglesa picture element).

4.1.2 Etapas fundamentales del procesamiento de imágenes

La primera etapa del procesamiento de imágenes es la adquisición de la imagen digital. Para ello se necesita un sensor de imágenes y una manera de digitalizar la señal producida por ese sensor. El sensor puede ser una cámara, un escáner, o algún otro dispositivo de adquisición.

Una vez que se ha obtenido la imagen digital, la siguiente etapa consiste en el preprocesamiento de esa imagen. La función básica del preprocesamiento es la de mejorar la misma con el fin de aumentar las posibilidades de éxito en los procesos posteriores.

La siguiente etapa es la segmentación. A grandes rasgos, la segmentación consiste en partir una imagen de entrada en sus partes u objetos constituyentes. En general, la segmentación autónoma es uno de los procesos más difíciles del tratamiento digital de imágenes. A la salida del proceso de segmentación habitualmente se tienen los datos de píxel en bruto. Estos pueden ser el contorno de una región, o bien todos los puntos de una región determinada.

En cada caso es necesario convertir los datos a una forma adecuada para el tratamiento por computadora. La primera decisión que hay que tomar es si los datos se han de representar como un contorno o como una región completa. La representación como un contorno es la adecuada cuando el interés radica en las características de la forma exterior, como esquinas e inflexiones. La representación regional es adecuada cuando el interés se centra en propiedades internas, como la textura o la estructuración. Sin embargo, en algunas aplicaciones, ambas representaciones coexisten.

También debe especificarse un método para describir los datos de forma que se resalten los rasgos de interés. La descripción, también denominada selección de rasgos, consiste en extraer rasgos con alguna información cuantitativa de interés o que sean fundamentales para diferenciar una clase de objetos de otra.

La última etapa incluye el reconocimiento e interpretación. El reconocimiento es el proceso que asigna una etiqueta a un objeto basándose en la información proporcionada por sus descriptores. La interpretación implica asignar significado a un conjunto de objetos reconocidos.

En un sistema de procesamiento de imágenes el conocimiento sobre un dominio especifico está codificado como una base de conocimiento. Este conocimiento puede ser tan simple como detallar las regiones de una imagen donde se sabe que se ubica información de interés, limitando así la búsqueda que ha de realizarse para hallar tal información. La base de conocimiento también puede ser muy compleja, como una lista interrelacionada de todos los posibles defectos en un problema de inspección de materiales o una base de datos que contenga imágenes de satélite de alta resolución de una región, en conexión con aplicaciones de detección de cambios [GON002].

4.2 Fundamentos de la imagen digital

4.2.1 Muestreo y cuantificación

Para usar una función de imagen *f*(x, y) en una computadora, ésta debe ser digitalizada tanto espacialmente como en su amplitud. La digitalización de las coordenadas espaciales (x, y) se denomina muestreo de la imagen mientras que la digitalización de la amplitud se conoce como cuantificación del nivel de gris.

4.2.1.1 Muestreo uniforme y cuantificación

A veces puede resultar útil expresar el muestreo y la cuantificación en termino matemáticos. Si Z y R representan a los conjuntos de números enteros y reales, respectivamente, el proceso de muestreo puede entenderse como una partición en una cuadrícula del plano xy, siendo las coordenadas del centro de cada elemento de la cuadrícula un par de elementos del producto cartesiano Z x Z (también indicado por Z^2), que es el conjunto de todos los pares ordenados de elementos (a, b) siendo a y b enteros pertenecientes a Z. Por tanto *f*(x, y) representa una imagen digital si (x, y) son enteros de Z x Z y *f* es una función que asigna un nivel de gris (es decir, un número real de l conjunto de los números reales R) a cada par de coordenadas (x, y) distinto.

Si los niveles de gris también son números enteros, entonces Z reemplaza a R y una imagen digital se convierte en una función bidimensional (2-D) cuyas coordenadas y valores de amplitud son números enteros.

4.2.1.2 Muestreo no uniforme y cuantificación

Para un valor fijo de resolución espacial, la apariencia de una imagen puede mejorarse en muchos casos empleando un esquema adaptativo en el que el proceso de muestreo dependa de las características de la imagen.

En general, se necesita un muestreo fino en las proximidades de las transiciones bruscas en los niveles de gris, mientras que se puede aplicar un muestreo tosco en las regiones relativamente suaves.

La necesidad de identificar contornos, aunque solamente sea de forma aproximada, es uno de los inconvenientes definidos del muestreo no uniforme. Este método tampoco es práctico para el caso de imágenes que contengan regiones uniformes relativamente pequeñas.

Cuando el número de niveles de gris debe mantenerse reducido, el empleo de niveles desigualmente espaciados es a menudo deseable en el proceso de cuantificación. Un método similar a la técnica de muestreo no uniforme anteriormente visto puede emplearse para la distribución de niveles de gris en una imagen [GON002].

4.2.2 Relaciones básicas entre píxeles

4.2.2.1 Vecinos de un píxel

Un píxel p de coordenadas (x, y) tiene cuatro vecinos horizontales y verticales cuyas coordenadas vienen dadas por:

$$(x + 1, y), (x –1, y), (x, y + 1), (x, y - 1)$$

Este conjunto de píxeles es denominado 4-vecinos de p y se representa como $N_4(p)$. Cada píxel está a una unidad de distancia de (x, y), y algunos de los vecinos de p caen fuera de la imagen digital si (x, y) está en el borde de la imagen.

Los cuatros vecinos en diagonal de p tienen las coordenadas:

$$(x + 1, y + 1), (x +1, y - 1), (x - 1, y + 1), (x –1 , y - 1)$$

Este conjunto de píxeles se representa por $N_D(p)$. Estos, junto a los 4-vecinos, se denominan los 8-vecinos de p, y se representan por $N_8(p)$. Al igual que en el caso anterior, algunos de los 8-vecinos caen fuera de la imagen si (x, y) está en el borde de la misma.

4.2.2.2 Conectividad entre píxeles

La conectividad entre píxeles es un concepto importante empleado para establecer los límites de los objetos y los componentes de áreas de una imagen.

Para determinar si dos píxeles están conectados, debe determinarse si son adyacentes en algún sentido (como ser 4-vecinos) y si sus niveles de gris cumplen un criterio especificado de similitud (como ser iguales). Por ejemplo, en una imagen binaria con valores 0 y 1, dos píxeles pueden ser 4-vecinos pero no estarán conectados a menos que tengan el mismo valor.

Sea V el conjunto de valores de nivel de gris empleados para definir la conectividad, se consideran tres tipos de conectividad:

- 4-conectividad. Dos píxeles p y q con valores dentro de V están 4-conectados si q pertenece a $N_4(p)$.

- 8-conectividad. Dos píxeles p y q con valores dentro de V están 8-conectados si q pertenece a $N_8(p)$.

- m-conectividad (también llamada conectividad mixta). Dos píxeles p y q con valores dentro de V están m-conectados si

i) *q* pertenece a $N_4(p)$, o bien

ii) *q* pertenece a $N_D(p)$ y además el conjunto $N_4(p)$) $N_4(q)$ es vacío. (Este es el conjunto de píxeles que son 4-vecinos de *p* y de *q* cuyos valores están en V).

La conectividad mixta es una modificación de la 8-conectividad que se introdujo para eliminar los múltiples caminos de conexión que aparecen a menudo cuando se emplea la 8-conectividad.

Un píxel *p* es adyacente de un píxel *q* si están conectados. Se puede definir 4-, 8- o m-adyacencia, dependiendo del tipo de conectividad especificada. Dos conjuntos de la imagen, llamémoslos S_1 y S_2, son adyacentes si algún píxel de S_1 es adyacente de algún píxel de S_2.

Un camino desde el píxel *p* de coordenadas (x, y) al píxel *q* de coordenadas (s, t) es una sucesión de diversos píxeles de coordenadas:

$$(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$$

donde $(x_0, y_0) = (x, y)$ y $(x_n, y_n) = (s, t)$, (x_i, y_i) es adyacente a $(x_i - 1, y_i - 1)$, 1 d i d n, y n es la longitud del camino. Así podemos definir 4-, 8- y m-caminos, dependiendo del tipo de adyacencia especificado.

Si *p* y *q* son píxeles de un subconjunto S especificado de la imagen, se dirá que *p* está conectado con *q* dentro de S si existe un camino desde *p* hasta *q* que consista totalmente de píxeles de S. Para cualquier píxel *p* dentro de S, el conjunto de píxeles de S conectados a *p* se denomina componente conexa de S. De esta forma, cualquier par de píxeles de una misma componente conexa están conectados entre sí, y componentes conexas distintas son disjuntas.

La capacidad de asignar etiquetas diferentes a las distintas componentes conexas disjuntas de una imagen es de importancia fundamental en el análisis automatizado de la imagen.

4.3 Mejora de la imagen

4.3.1 Filtros

Hay dos tipos de filtrado, el espacial, que utiliza máscaras directamente en la imagen, y el espectral, que utiliza el espacio de la transformada de Fourier bidimensional de la imagen.

Ambos tipos están relacionados entre sí y se puede demostrar que todo algoritmo espectral tiene un equivalente espacial, y viceversa. En general se denomina filtros espaciales a las máscaras que se utilizan.

Básicamente el proceso de filtrado consiste en realizar una serie de operaciones sobre cada uno de los píxeles que componen la imagen. Si se trabajar con imágenes en color, cada píxel se corresponde con un entero que indica la luminosidad del píxel en cada color, esto quiere decir, que es necesario aplicar las operaciones pertinentes sobre los tres componentes de color R, G y B, como si cada una de ellas fuese una imagen por separado.

4.3.1.1 Filtrado espacial

Los filtros espaciales tienen como objetivo modificar la contribución de determinados rangos de frecuencias a la formación de la imagen. El término espacial se refiere al hecho de que el filtro se aplica directamente a la imagen y no a una transformada de la misma, es decir, el nivel de gris de un píxel se obtiene directamente en función del valor de sus vecinos.

Los filtros espaciales pueden clasificarse basándose en su linealidad: filtros lineales y filtros no lineales. A su vez los filtros lineales pueden clasificarse según las frecuencias que dejen pasar: los filtros paso bajo atenúan o eliminan las componentes de alta frecuencia a la vez que dejan inalteradas las bajas frecuencias; los filtros paso alto atenúan o eliminan las componentes de baja frecuencia con lo que agudizan las componentes de alta frecuencia; los filtros paso banda eliminan regiones elegidas de frecuencias intermedias.

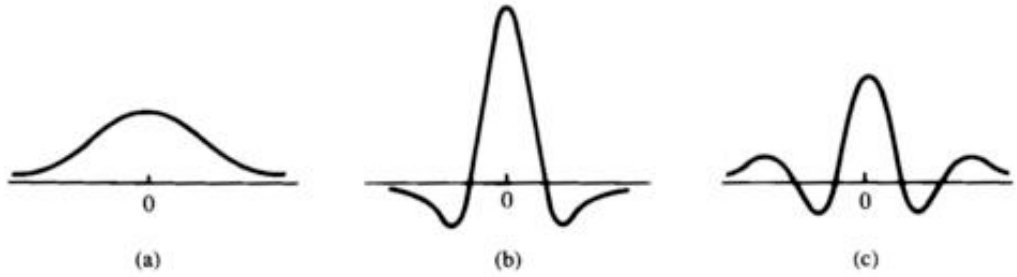


Figura 1. Filtros espaciales. (a) Filtro paso bajo. (b) Filtro paso alto. (c) Filtro paso banda.

La forma de operar de los filtros lineales es por medio de la utilización de máscaras que recorren toda la imagen centrando las operaciones sobre los píxeles que se encuadran en la región de la imagen original que coincide con la máscara y el resultado se obtiene mediante una computación (suma de convolución) entre los píxeles originales y los diferentes coeficientes de las máscaras.

Los filtros espaciales no lineales también operan sobre entornos. Sin embargo, su operación se basa directamente en los valores de los píxeles en el entorno en consideración. Unos ejemplos de filtros no lineales habituales son los filtros mínimo, máximo y de mediana que son conocidos como filtros de rango.

El filtro de mediana tiene un efecto de difuminado de la imagen, y permite realizar una eliminación de ruido de forma eficaz, mientras que el filtro de máximo se emplea para buscar los puntos más brillantes de una imagen produciendo un efecto de erosión, y el filtro de mínimo se emplea con el objetivo contrario, buscar los puntos más oscuros de una imagen produciendo un efecto de dilatación.

Otra clasificación de los filtros espaciales puede hacerse basándose en su finalidad, y así tenemos los filtros de realce (Sharpening) para eliminar zonas borrosas o filtros de suavizado (Smoothing) para difuminar la imagen. También tenemos los filtros

diferenciales que se componen de varios tipos de máscaras (Laplaciano, Prewitt, Sobel, etc.), y se utilizan para la detección de bordes.

4.3.1.2 Filtrado en el dominio de la frecuencia

El filtrado en el dominio frecuencial incluye técnicas que están basadas en la modificación de la transformada de Fourier de la imagen. Los filtros paso bajo atenúan o eliminan los componentes de alta frecuencia en el dominio de Fourier, mientras dejan las bajas frecuencias sin alterar (esto es, éste tipo de filtros dejan "pasar" las frecuencias bajas). Las altas frecuencias son características de bordes, curvas y otros detalles en la imagen, así el efecto de un filtro paso bajo es el de difuminar la imagen.

De igual forma, los filtros paso alto, atenúan o eliminan las bajas frecuencias. Éstas son las responsables de las pequeñas variaciones de las características de una imagen, tal como pueden ser el contraste global y la intensidad media. El resultado final de un filtro paso alto es la reducción de estas características, y la correspondiente aparición de bordes y otros detalles de curvas y objetos.

Desde esta perspectiva, la idea general del filtrado espacial presentada anteriormente es bastante más atractiva e intuitiva que en el dominio de la frecuencia. En la práctica, las pequeñas máscaras espaciales son mucho más empleadas que las transformadas de Fourier debido a su facilidad de implementación y a su velocidad de operación. Sin embargo, es esencial la comprensión de los conceptos en el dominio frecuencial para solucionar los problemas que no se pueden resolver con técnicas espaciales.

4.3.2 Eliminación de ruido en la imagen

Se entiende por ruido en imágenes digitales cualquier valor de un píxel de una imagen que no se corresponde exactamente con la realidad. Cuando se adquiere una imagen digital, ésta está contaminada por ruido.

La mayoría de las veces el ruido proviene del equipo electrónico utilizado en la adquisición de las imágenes (ruido de cuantificación de la imagen, efecto de niebla en la imagen..., etc) y al ruido añadido en los tramos de transmisión (posibles interferencias o errores al transmitir los bits de información).

Se pueden distinguir dos clases diferentes de ruido:

- Ruido gaussiano: Se caracteriza por tener un espectro de energía constante para todas las frecuencias. Cuando se presenta este problema, el valor exacto de cualquier píxel es diferente cada vez que se captura la misma imagen. Este efecto, suma o resta un determinado valor al nivel de gris real y es independiente de los valores que toma la imagen.

- Ruido impulsivo: Se caracteriza por la aparición de píxeles con valores arbitrarios normalmente detectables porque se diferencian mucho de sus vecinos más próximos. La distribución viene dada por:

$$g(x, y)= \begin{cases} 0 & si \quad r(x, y) < p / 2 \\ L - 1 & si \quad p/2 <= r(x, y) < p \\ f(x, y) & si \quad r(x, y) >= p \end{cases}$$

donde r(x,y) es un número aleatorio con distribución uniforme en [0,1) y p es la probabilidad de ocurrencia del ruido aleatorio, es decir, el porcentaje de puntos de la imagen que se verán afectados por el ruido impulsivo del total de puntos de la imagen.

El ruido gaussiano tiene un efecto general en toda la imagen, es decir, la intensidad de cada píxel de la imagen se ve alterada en cierta medida con respecto a la intensidad en la imagen original. Por el contrario, se observa que el ruido impulsivo tiene un efecto más extremo sobre un subconjunto del total de píxeles de la imagen. Un tanto por ciento de los píxeles de la imagen toman arbitrariamente el valor extremo 0 o 255.

Una forma de eliminar el ruido de una imagen es mediante el suavizado de imágenes. Dicho de otro modo, el filtrado paso bajo se emplea no sólo para el suavizado de imágenes, sino también para la eliminación de ruido. De hecho, el filtrado paso bajo es una manera efectiva de reducir el ruido gaussiano en una imagen, mientras que no es tan efectivo con el ruido impulsivo. Como promediar reduce los valores extremos de la vecindad del píxel, hacer un filtro de media tiende a reducir el contraste de las imágenes, pues los valores extremos, altos y bajos, son cambiados por valores medios.

El problema con la utilización de filtros paso bajo para eliminar el ruido de imágenes consiste en que los bordes de los objetos se vuelven borrosos. Los bordes contienen una cantidad enorme de información de una imagen. Filtrando el ruido impulsivo de una imagen, el filtrado de mediana puede ser una mejor opción**.** Los filtros de mediana hacen un mejor trabajo conservando los bordes.

4.3.2.1 Filtrado de mediana

Los filtros de suavizado lineales o filtros paso bajo tienden a "difuminar los ejes" a causa de que las altas frecuencias de una imagen son atenuadas. La visión humana es muy sensible a esta información de alta frecuencia. La preservación y el posible realce de este detalle es muy importante al filtrar. Cuando el objetivo es más la reducción del ruido que el difuminado, el empleo de los filtros de mediana representan una posibilidad alternativa.

A menudo, las imágenes digitales se corrompen con ruido durante la transmisión o en otras partes del sistema. Esto se ve a menudo en las imágenes convertidas a digital de una señal de la televisión. Usando técnicas del filtrado de ruido, el ruido puede ser suprimido y la imagen corrompida se puede restaurar a un nivel aceptable.

En aplicaciones de ingeniería eléctrica, el ruido se elimina comúnmente con un filtro paso bajo. El filtrado paso bajo es satisfactorio para quitar el ruido gaussiano pero no para el

ruido impulsivo. Una imagen corrupta por ruido impulsivo tiene varios píxeles que tienen intensidades visiblemente incorrectas como 0 o 255.
Hacer un filtrado paso bajo alterarán estas señales con los valores extremos sobre la vecindad del píxel. Un método mucho más eficaz para eliminar el ruido impulsivo es el filtrado de mediana.

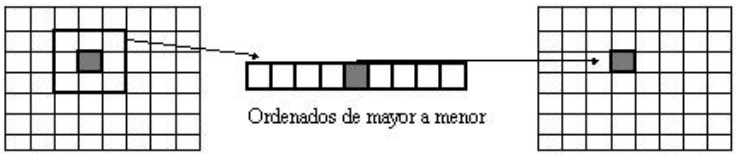


Figura 2. Calculo del valor de mediana de in píxel

En el filtrado de mediana, el nivel de gris de cada píxel se reemplaza por la mediana de los niveles de gris en un entorno de este píxel, en lugar de por la media. Recordar que la mediana M de un conjunto de valores es tal que la mitad de los valores del conjunto son menores que M y la mitad de los valores mayores que M, es decir en un conjunto ordenado de mayor a menor o viceversa, sería el valor de la posición central.

El filtro de la mediana no puede ser calculado con una máscara de convolución, ya que es un filtro no lineal. Podemos ver como este tipo de filtro elimina totalmente el punto que tenia un valor muy diferente al resto de sus vecinos. Como se selecciona el valor de centro, el filtrado de mediana consiste en forzar que puntos con intensidades muy distintas se asemejen más a sus vecinos, por lo que observamos que el filtro de mediana es muy efectivo para eliminar píxeles cuyo valor es muy diferente del resto de sus vecinos, como por ejemplo eliminando ruido de la imagen.

4.4 Segmentación de Imágenes

4.4.1 Introducción

El primer paso en el proceso de análisis de imágenes consiste generalmente en segmentar la misma. Segmentar la imagen es dividir la imagen en las partes u objetos que la constituyen. El nivel al que se realiza esta subdivisión depende de la aplicación en particular. En otras palabras, el proceso de segmentación termina cuando se hayan detectado todos los objetos de interés para la aplicación. En general, la segmentación automática es una de las tareas más complicadas dentro del procesado de imagen. La segmentación va a dar lugar en última instancia al éxito o fallo del proceso de análisis. En la mayor parte de los casos, una buena segmentación dará lugar a una solución correcta, por lo que, se debe poner todo el esfuerzo posible en esta importante etapa del análisis**.**

Los algoritmos de segmentación de imágenes monocromáticas se basan generalmente en dos propiedades básicas de los niveles de gris de la imagen: discontinuidad y similitud.
Dentro de la primera categoría se intenta dividir la imagen basándose en los cambios bruscos en el nivel de gris. Las tareas de interés en esta categoría son la detección de puntos, líneas y bordes en la imagen. Las áreas dentro de la segunda categoría están

basadas en las técnicas de umbrales, crecimiento de regiones, y técnicas de división y fusión.

4.4.2 Detección de discontinuidades

En esta sección se presentaran diversas técnicas para detectar varios tipos de discontinuidades en una imagen digital; como ser puntos, líneas y bordes. El método más común para detectar discontinuidades es la correlación de la imagen con una máscara.

Para una máscara de 3 x 3 como la de la Figura 3 el procedimiento de correlación consiste en realizar el producto de los elementos de la máscara por el valor de gris correspondiente a los píxels de la imagen encerrados por la máscara. La respuesta de la máscara en un punto cualquiera de la imagen es:

$$R = \sum_{i=1}^{9} w_i z_i$$

donde zi es el nivel de gris asociado con el coeficiente de la mascara wi. Habitualmente la respuesta de la máscara está definida con respecto a la posición de su centro. Cuando la máscara está encerrada en un píxel del límite, la respuesta se calcula utilizando el vecindario de píxels parcial apropiado.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figura 3. Una máscara general de 3 X 3

−1	−1	−1
−1	8	−1
−1	−1	−1

Figura 4. Máscara usada para la detección de puntos aislados

4.4.2 Detección de puntos

La detección de puntos aislados es inmediata. Empleando la máscara de la Figura 4, se dirá que se ha detectado un punto en la posición en la cual la misma está centrada la si

$$|R| > T \quad (2)$$

donde T es un umbral no negativo. Básicamente se mide la diferencia entre el píxel central y sus vecinos, puesto que un píxel será un punto aislado siempre que sea suficientemente distinto de sus vecinos. Solamente se considerarán puntos aislados aquellos cuya diferencia con respecto a sus vecinos sea significativa.

4.4.3 Detección de líneas

En este caso se consideran las máscaras de la Figura 5. Si pasamos la primera de las máscaras a lo largo de la imagen, tendrá mayor respuesta para líneas de ancho un píxel orientadas horizontalmente. Siempre que el fondo sea uniforme, la respuesta será máxima cuando la línea pase a lo largo de la segunda fila de la máscara. La segunda máscara de la Figura 3 responderá mejor a líneas orientadas a 45°; la tercera a líneas verticales; y la última a líneas orientadas a □45°.

Estas direcciones se pueden establecer observando que para la dirección de interés las máscaras presentan valores mayores que para otras posibles direcciones. Si denotamos con R1, R2, R3 y R4 las respuestas de las cuatro máscaras de la Figura 3 para un píxel en particular, entonces si se cumple que |Ri| > |Rj | con j <> i, será más probable que dicho píxel esté asociado a la dirección correspondiente a la máscara i.

−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

−1	−1	2
−1	2	−1
2	−1	−1

45°

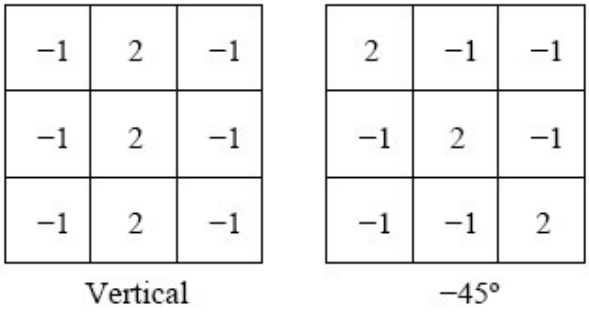


Figura 5. Máscaras usadas para la detección de líneas

4.4.6 Detección de bordes

La detección de bordes es el procedimiento empleado más habitualmente para la detección de discontinuidades. Un borde se define como la frontera entre dos regiones con nivel de gris relativamente diferente. Vamos a suponer a partir de ahora que las regiones de interés son suficientemente homogéneas de modo que la transición entre dichas regiones se puede determinar empleando exclusivamente las discontinuidades en el nivel de gris.

La idea básica detrás de cualquier detector de bordes es el cálculo de un operador local de derivación. En la Figura 6 se puede ver este concepto. En la parte derecha se puede ver una imagen de una banda clara sobre un fondo oscuro, el perfil a lo largo de una línea horizontal y la primera y segunda derivada de dicho perfil. Se observa en el perfil que un borde está (la transición del oscuro al claro) modelado como una discontinuidad suave (no brusco) del nivel de gris. Esto tiene en cuenta el hecho de que en las imágenes reales los bordes están ligeramente desenfocados o emborronados a causa del muestreado.

Como se puede observar en la Figura 6 la primera derivada es positiva para cambio a nivel de gris más claro, negativa para cambio a nivel de gris más oscuro y cero en aquellas zonas con nivel de gris uniforme. La segunda derivada presenta valor positivo en la zona oscura de cada borde, valor negativo en la zona clara de cada borde y valor cero en las zonas de valor de gris constante y justo en la posición de los bordes. El valor de la magnitud de la primera derivada nos sirve para detectar la presencia de bordes, mientras que el signo de la segunda derivada nos indica si el píxel pertenece a la zona clara o a la zona oscura. Además la segunda derivada presenta siempre un cruce por cero en el punto medio de la transición. Esto puede ser muy útil para localizar bordes en una imagen.

Si bien lo expuesto anteriormente se ha limitado al perfil unidimensional, se puede aplicar un argumento similar a un borde de cualquier orientación de una imagen. Simplemente se define un perfil perpendicular a la dirección del borde en cualquier punto que se desee y se interpretan los resultados como se lo hizo anteriormente. La derivada primera en un punto se obtiene utilizando el módulo del gradiente en ese punto. La derivada segunda se obtiene de forma similar usando el laplaciano.

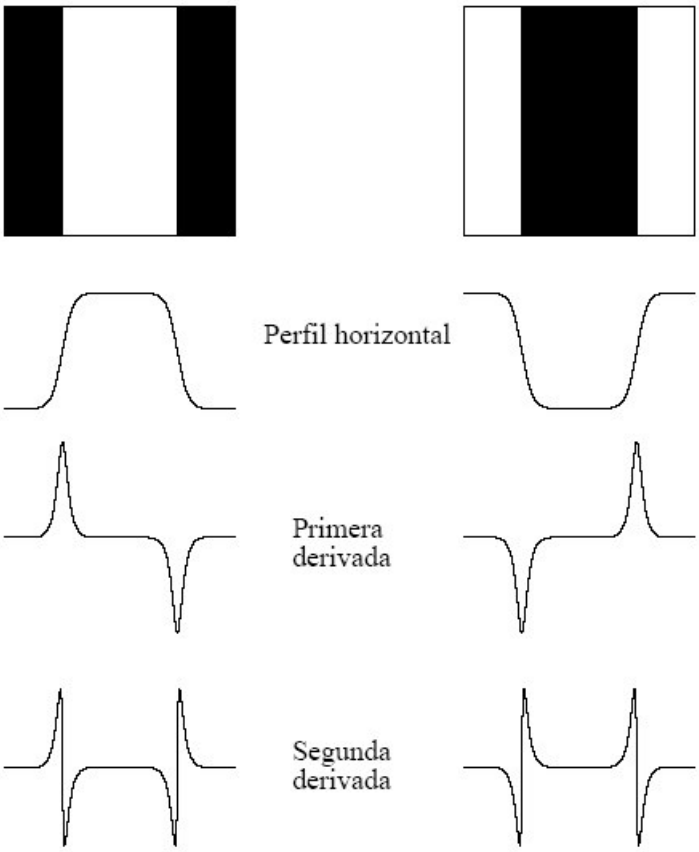


Figura 6. Detección de bordes empleando operadores de derivación. La segunda derivada tiene un cruce por cero en la posición de cada borde.

4.4.6.1 Gradiente

El gradiente de una imagen I(x, y) en la posición (x, y) viene dado por el vector

$$\nabla I = \begin{bmatrix} I_x \\ \\ I_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

El vector gradiente siempre apunta en la dirección de la máxima variación de la imagen I en el punto (x, y). En la detección de bordes es muy importante la magnitud de este vector, denominado simplemente como gradiente de la imagen que viene dado por

$$\nabla I = ||\nabla I|| = \sqrt{I_x^2 + I_y^2}.$$

Esta cantidad es igual a la máxima variación de I(x, y) por unidad de distancia en la dirección del vector. Es una práctica común aproximar el gradiente por sus valores absolutos:

$$\nabla I \approx |I_x| + |I_y|,$$

que son más fáciles de implementar.

La dirección del vector gradiente es también un valor importante. Sea f(x, y) la representación del ángulo de dirección del vector I en (x, y), entonces se tiene que

$$\alpha(x, y) = \tan^{-1} \frac{I_y(x, y)}{I_x(x, y)}$$

donde los ángulos se miden con respecto al eje de las abscisas.

El cálculo del gradiente de una imagen se basa en la obtención de las derivadas parciales en cada posición de píxel. Las derivadas se pueden implementar digitalmente de varias formas.

En la Figura 7 se pueden ver los operadores de Roberts, Prewitt, Sobel y Frei-Chen para determinar las derivadas parciales. Sin embargo, los operadores de Sobel y de Frei-Chen tienen la adicional ventaja de proporcionar un suavizado de la misma. Ya que la derivación acentúa el ruido, el efecto de suavizado es particularmente interesante, puesto que elimina parte del mismo. El requisito básico de un operador de derivación es que la suma de los coeficientes de la máscara sea nula, para que la derivada de una zona uniforme de la imagen sea cero.

Según la Figura 5 las derivadas según el operador de Sobel vienen dadas por:

$$R_x = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

$$R_y = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

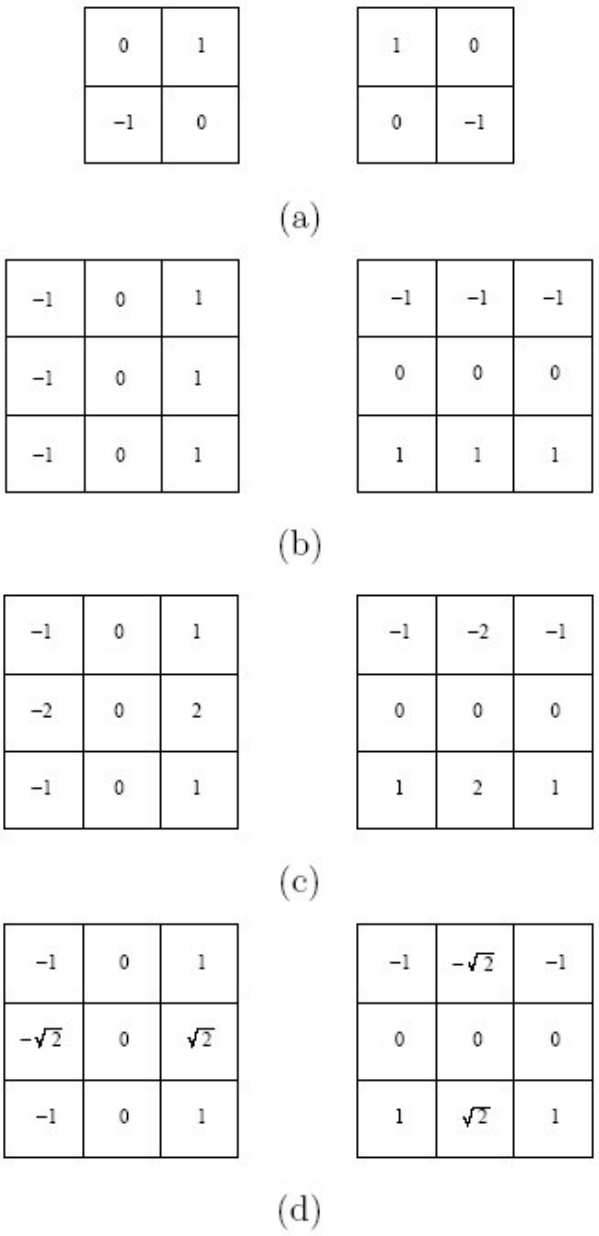


Figura 7. Operadores de derivación: (a) Roberts, (b) Prewitt, (c) de Sobel y (d) de Frei-Chen

4.4.6.2 Laplaciano

El Laplaciano de una imagen I(x, y) es una derivada de segundo orden definida por

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial x^2}.$$

Al igual que en el caso del gradiente se puede implementar en forma digital de varias formas.

0	−1	0
−1	4	−1
0	−1	0

Figura 8. Máscara para el Laplaciano

Puesto que el Laplaciano es un operador de derivación, la suma de los coeficientes debe ser cero. Además, el coeficiente asociado con el píxel central debe ser positivo y los demás coeficientes negativos o ceros. En la Figura 6 podemos ver una máscara para el Laplaciano. En este caso la expresión para determinarlo viene dada por la expresión

$$4z5 – (z2 + z4 + z6 + z8)$$

Aunque el Laplaciano responde a transiciones en la intensidad de la imagen se emplea en pocas ocasiones en la práctica. Debido a que es un operador de segunda derivada es sensible en exceso a la presencia de ruido. Además, este da lugar a bordes dobles y no permite determinar direcciones. En general juega un papel secundario en la detección de bordes para determinar si un píxel está en la zona clara o en la zona oscura del borde a través de su signo.

Un uso más generalizado del Laplaciano es a través de la propiedad de localización de bordes usando la propiedad de los cruces por cero [GON002].

4.4.7 Técnicas de Umbrales

Supongamos que el histograma de los niveles de gris de una imagen I(x, y) es el que se muestra en la figura 9-a:

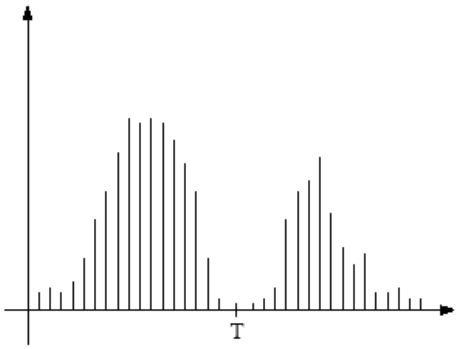


Figura 9-a

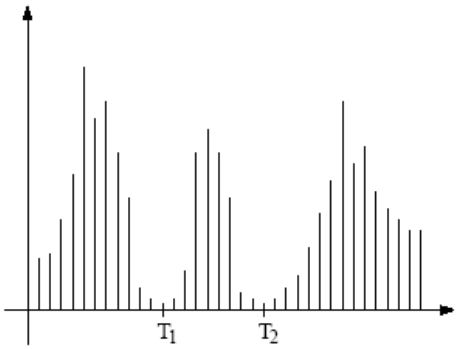


Figura 9-b

La imagen I(x, y) está compuesta de objetos claros sobre un fondo oscuro de tal forma que los niveles de gris están agrupados en dos modos predominantes. Una forma de separar los objetos del fondo consiste en seleccionar un umbral T que separe esos modos.

Entonces, cualquier punto (x, y) para el que se cumpla que

$$I(x, y) > T,$$

se lo etiqueta como objeto; en otro caso, como fondo.

La figura 9-b muestra el histograma de otra imagen en un caso más general. En este caso el histograma de la imagen está caracterizado por tres modos dominantes. Esto ocurrirá cuando tengamos dos tipos de objetos claros sobre fondo oscuro, por ejemplo.

Se puede utilizar el mismo principio para clasificar cada punto (x, y).
Si T1 < I(x, y) <= T2 entonces se lo etiqueta como primer objeto, si I(x, y) > T2 como segundo objeto y si I(x, y) <= T1 como fondo.

En general, este tipo de clasificación con varios umbrales es menos viable, ya que es más difícil determinar esos umbrales que aíslen de forma efectiva las regiones de interés, especialmente cuando el número de modos del histograma aumenta. En este caso es mejor emplear umbrales variables.

En general, un método de umbral se puede ver como una operación en la que se hace un testeo de cada píxel con respecto a una función T de la forma

$$T = T(x, y, p(x, y), I(x, y)],$$

donde I(x, y) es el nivel de gris del punto (x, y) y p(x, y) denota cualquier propiedad local de ese punto (como por ejemplo el nivel de gris medio en un vecindario centrado en (x, y)). El método de umbral dará lugar a otra imagen B(x, y) definida por

$$B(x, y) = \begin{cases} 1 & \text{si } I(x, y) > T \\ \\ 0 & \text{si } I(x, y) \leq T. \end{cases}$$

En este caso un píxel con etiqueta 1 de la imagen B corresponderá a objetos, mientras que un píxel con etiqueta 0 corresponderá al fondo.

Cuando T dependa sólo del nivel de gris I(x, y) se denomina umbral global (en la figura 9-a se puede ver un ejemplo en este caso). Si T depende tanto del nivel de gris I(x, y) como de la propiedad local p(x, y), el umbral se denomina local. Si, además, T depende de las coordenadas espaciales x e y, el umbral se denomina dinámico.

4.4.8 Crecimiento de regiones

El crecimiento de regiones es un procedimiento mediante el cual se agrupan píxeles o subregiones en regiones mayores. El procedimiento más sencillo se denomina agregación de píxeles, que comienza a partir de un conjunto de píxeles semilla, de forma que a partir de cada semilla se crecen regiones añadiendo píxeles a dicha semilla de entre aquellos píxeles vecinos que tienen propiedades similares. El resultado de la segmentación dará lugar como mucho a tantas regiones como semillas haya.

Sin embargo, puede darse el caso de que dos de esas semillas correspondan a píxeles de la misma región. En este caso el crecimiento desde una de las semillas absorberá a la otra, que en este caso deberá ser descartada. Un ejemplo sencillo, pero muy usado en la práctica de similitud es la diferencia absoluta en el nivel de gris. Fijado un umbral T se va calculando la diferencia en valor absoluto del nivel de gris del píxel en cuestión (en el vecindario de la región crecida hasta el momento) con respecto al nivel de gris de la semilla y si no se supera ese umbral T se añade a la región.

Dos problemas fundamentales en el crecimiento de regiones son: por un lado, la selección de las semillas o puntos de partida que representen adecuadamente a las regiones de interés; y por otro, la elección de las propiedades adecuadas que permitan ir añadiendo píxeles durante el proceso de crecimiento [GON002].

4.5 Reconocimiento de Imágenes

La función *diferencia* es la base del reconocimiento de imágenes. Indica la distancia entre dos imágenes. Existen diversas estrategias para reconocimiento de imágenes digitales de acuerdo al tipo de aplicación y de los recursos del sistema.

Utilizando los conceptos derivados del OCR (Optical Carácter Recognition), el primer paso para comparar dos imágenes es vectorizar cada imagen y cada cuadro para luego, comparar las formas de los objetos resultantes. El proceso de vectorización consiste en definir imágenes utilizando la geometría y funciones matemáticas. Los algoritmos existentes para este proceso consumen una gran cantidad de recursos, y la metodología para reconocer la similitud entre estos objetos resulta muy compleja.

La manera más directa de comparar una imagen con otra, es comparar cada píxel de la primera imagen con su correspondiente píxel en la segunda imagen, y acumular las distancia entre cada pareja de píxeles para determinar la distancia general entre ambas.

Aunque esta es una estrategia relativamente buena para comparar imágenes, la cantidad de comparaciones necesarias es muy grande. Por cada comparación debe calcularse la distancia entre los píxeles de las dos imágenes y por cada pareja de píxeles debe compararse cada uno de los tres canales RGB [TRF007]. Si las imágenes son en escala de grises alcanza con comparar uno de los tres componentes RGB ya que tienen el mismo valor para el un píxel determinado.

4.5.1 Método Lineal

La distancia D entre dos píxeles está dada por:

$$D = (\Delta R) + (\Delta G) + (\Delta B)$$

Esta distancia es calculada por cada píxel y por cada canal de color en las imágenes comparadas.

4.5.2 Método Cuadrático

Se puede acentuar el efecto de la diferencia de cada píxel utilizando una diferencia cuadrática o distancia euclidiana.

$$D = \sqrt{(\Delta R^2) + (\Delta G^2) + (\Delta B^2)}$$

Capítulo 5 – Reconocimiento Automático de Texto Braille

5.1 Introducción:

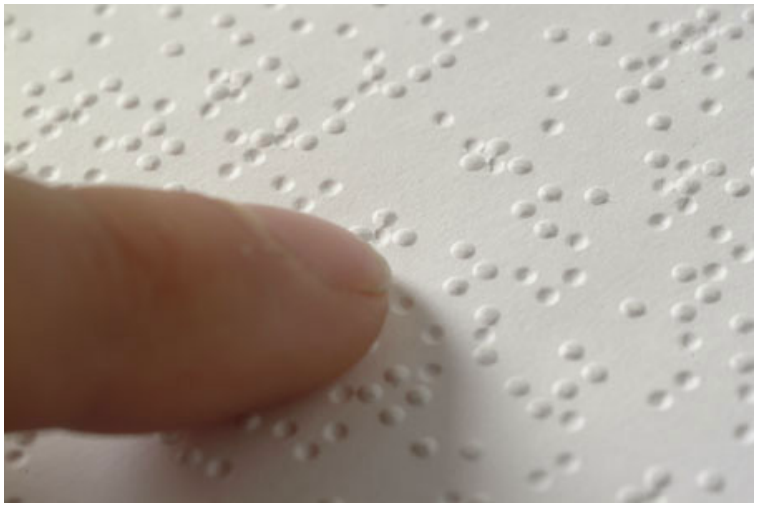
En este capitulo se hablará sobre el reconocimiento automático de texto Braille, eje central de esta tesis. Se expondrán los aspectos a tener en cuenta en el desarrollo del mismo. Se explicarán las etapas que llevan de una hoja Braille en el texto digital correspondiente. Se expondrán los problemas encontrados y las posibles soluciones que se fueron presentando en cada una de ellas hasta llegar a la solución optima.

5.2 Adquisición de un documento Braille:

5.2.1 Características de la imagen obtenida

Como se dijo en un capitulo anterior (ver Sistema Braille) la escritura Braille está compuesta por pequeñas erupciones del papel. A estas erupciones las llamaremos puntos Braille. Observando una hoja Braille a simple vista se notan claramente los puntos Braille en cuestión.

Un ejemplo es el siguiente:



Ejemplo de escritura Braille. En esta figura se observa un documento Braille como se ve a simple vista.

Dadas las características especiales que pueden apreciarse en la imagen anterior, la primera pregunta que uno se hace es: ¿Será posible escanear una hoja Braille?:

La respuesta es sí.

Al escanear una hoja Braille con un escáner normal se obtiene una imagen como la siguiente:

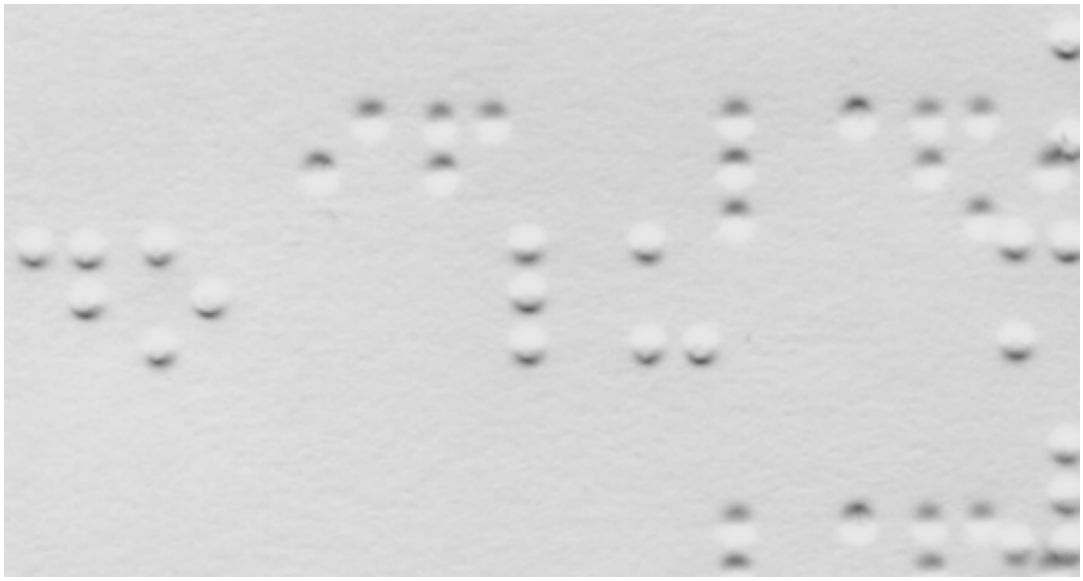


Imagen Braille escaneada con un escáner normal

En esta imagen correspondiente a una hoja Braille escrita en Braille interpunto (a doble cara) se notan claramente dos tipos de puntos: los que están formados por una zona brillante arriba y una zona obscura abajo, y los que están formados por una zona obscura arriba y una brillante abajo.

El fondo de la imagen tiene un valor de luminancia intermedio. Ni tan claro como las zonas brillantes, ni tan oscuro como las zonas obscuras.

Los del primer tipo son los puntos de la cara que la persona vidente detecta cuando pasa su mano por la hoja.

Los del segundo tipo son los puntos correspondientes al texto de la otra cara. Estos puntos no se detectan al tacto y sólo aparecen en hojas escritas en Braille interpunto (hojas a doble cara). De ésta forma queda claro que al realizar el escaneo de la misma se obtiene la información de los puntos de las dos caras de la hoja.

En este tipo de hojas se presenta una dificultad extra: hay dos tipos de patrones para buscar. Además se debe poder discriminar esos dos patrones por que representan cosas muy distintas (los puntos de cada cara). Por otro lado, superando esta dificultad se podrá reconocer las dos caras de la hoja en una sola pasada de la hoja por el escáner.

5.2.2 Parámetros de adquisición

Es muy importante obtener una imagen de buena calidad para que las etapas posteriores del reconocimiento funcionen correctamente.

Dos de los parámetros para la adquisición de la imagen más importantes son:

- La resolución de escaneo:

  La resolución de escaneo es la cantidad de puntos por pulgada ("dot per inch" o comúnmente llamada DPI) que se utiliza para escanear el documento. A mayores valores de DPI se tendrá una imagen de mayor calidad, es decir las zonas brillantes y obscuras de las que hablamos anteriormente serán más grandes y más fáciles de aislar y reconocer.

  Por otro lado el tamaño en memoria de la imagen adquirida será mayor también. Esto se traduce en un aumento del tiempo empleado en el reconocimiento. A veces este aumento de tiempo suele ser innecesario.

  Realizando el procedimiento de escaneo a resoluciones entre 150 y 300 DPI se obtuvieron buenos resultados. Se recomienda escanear a 150 DPI a menos que el documento esté muy deteriorado.

  Las pruebas realizadas a 300 dpi mostraron puntos Braille más fáciles de reconocer pero iban en detrimento de la performance de los algoritmos desarrollados para el reconocimiento.

- El contraste:

  Se realizaron pruebas escaneando hojas braille con diferentes niveles de contraste. Algunas imágenes obtenidas no fueron aptas para el reconocimiento.

  Especialmente las imágenes con muy poco contraste como la siguiente:

En esta imagen se notan claramente las zonas obscuras correspondientes a la mitad de los puntos Braille. No así las zonas brillantes, que en este caso, se confunden con el fondo de la misma.

5.3 Problemas:

5.3.1 Ruido en la adquisición

El proceso de escaneo de la hoja puede generar ruido. Este ruido se manifiesta como pequeñas zonas con nivel de luminancia parecido al de las zonas brillantes de los puntos braille o al de las zonas obscuras, pero más pequeñas.

Otro tipo de ruido es el generado por pequeñas torceduras en el papel produciendo un efecto similar al caso anterior.

Estas pequeñas manchitas introducidas, pueden llegar a ser confundidas con las zonas brillantes u obscuras de los puntos. Es decir, se pueden llegar a detectar erróneamente como medios puntos Braille reales.

Más adelante se verá que esto no representa un problema mayor, por la forma en que se detectan los puntos braille como unidad. No obstante, se puede aplicar un filtro de mediana para eliminar el ruido introducido luego de la etapa de adquisición de la imagen (etapa de preprocesamiento).

5.3.2 Inclinación de la hoja

Es común que el usuario introduzca la hoja en el escáner con una cierta inclinación. Esto supone un problema a resolver. Se necesitará detectar la inclinación de la misma y luego aplicar un proceso de corrección sobre la imagen escaneada. Además, en las hojas Braille puede pasar que los renglones tengan una leve inclinación entre si.

5.3.3 Documentos en mal estado

Las hojas braille pueden deteriorase con el uso. Los puntos se achatan, deforman, etc. En algunos casos el nivel de deterioro no será un problema para el reconocimiento. En otros, el reconocimiento podrá tener algunos errores (léase confundir un símbolo Braille con otro) o será prácticamente inviable.

5.3.4 Las distancias entre puntos no siempre coinciden

Si bien las distancias entre puntos en la escritura Braille son standard generalmente no coinciden. En muchos casos hay un margen de error que si bien es pequeño debe ser considerado en los algoritmos desarrollados para el reconocimiento.

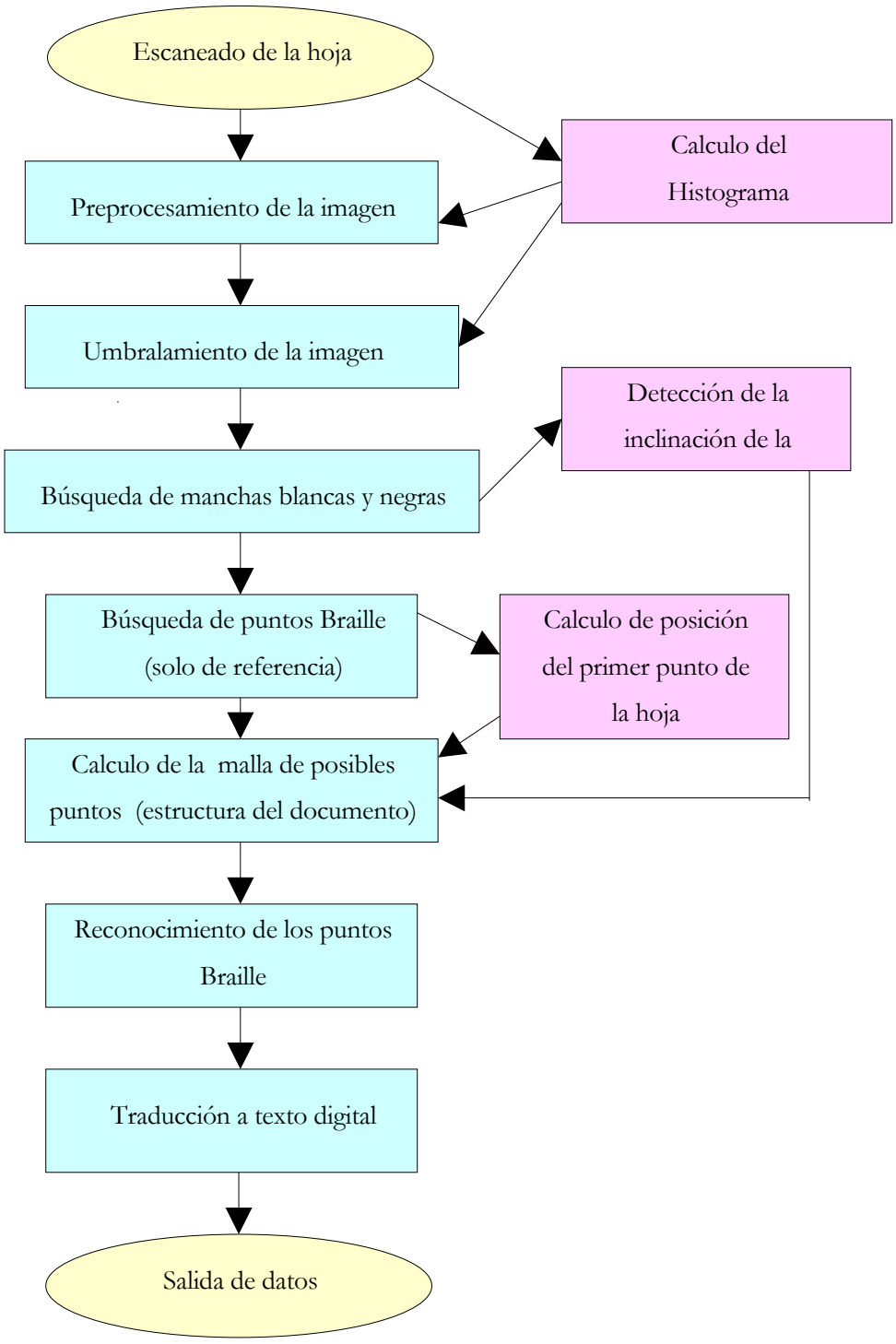
5.3.5 Hojas Braille de tamaño mayor que A4

Las hojas mayormente usadas en la escritura Braille son más grandes que el formato A4. En la actualidad los escáneres A3 son mucho más caros que los A4 y por otro lado estos últimos son los más utilizados.

5.3.6 Braille Interpunto

En la escritura Braille existen hojas escritas a doble cara. Esto lleva el consiguiente de que al escáner se obtienen los puntos de las dos caras al mismo tiempo y esto hace mucho mas complejo el proceso de segmentación puesto que deben poder aislarse los puntos de cada cara.

5.4 Etapas para el reconocimiento automático de texto Braille:



Para realizar el reconocimiento, siguió un esquema similar al propuesto por [UVO097]. En la figura anterior se pueden observar las distintas etapas involucradas.

A continuación se explicarán detalladamente las etapas del mismo:

5.4.1 Escaneo de la hoja

Se obtiene una representación digital del documento para ser procesada por las próximas etapas. Para realizar el escaneado se utilizó el estandar Twain. Este standard provee una API, con alrededor de 50 funciones, que puede ser utilizada desde una aplicación determinada. De esta forma la aplicación desarrollada se independiza del dispositivo de adquisición y se asegura que funcionará con cualquiera que soporte el standard Twain (actualmente un gran número de ellos).

Se trabajó con imágenes en escala de grises. La resolución de escaneado recomendada es de 150 dpi, pudiéndose también trabajar a 300 dpi.

Una restricción adicional es que el escáner sea color. Con esto se asegura que al menos uno de los 3 haces de luz (RGB) del mismo barre el documento a ser escaneado con un cierto grado de inclinación, permitiendo de esta forma la generación de las zonas brillantes y obscuras de las que se habló anteriormente. Si el escáner es monocromo y el haz (en este caso es uno solo) es perpendicular al documento a escanear, no se generaran las zonas brillantes y obscuras.

5.4.2 Pre-procesamiento

Esta etapa puede incluir un conjunto de varias operaciones sobre la imagen adquirida con el escáner a fin de dejar disponible para las etapas posteriores una imagen que pueda ser mejor procesada e interpretada:

- Si la imagen no está en escala de grises habrá que convertirla. Si la imagen está siendo escaneada (no es de archivo) habrá que setear los objetos Twain para realizar el escaneo en escala de grises.

- Si es necesario habrá que ajustar el contraste de la misma. En las pruebas realizadas las imágenes que mejor se adaptaron son imágenes de fondo más bien grisáceo que blanquecinas.

- Si es necesario habrá que aplicar algún tipo de filtro para minimizar el ruido introducido por el escáner al momento de la adquisición. Para solucionar este problema se puede aplicar un filtro de mediana a la imagen adquirida.

5.4.3 Cálculo del Histograma

Se calcula el histograma de luminancias de la imagen obtenida en la etapa anterior. De esta forma podemos saber el porcentaje de píxeles de cada nivel de gris de la imagen.

Para realizar su cálculo, se inspecciona la imagen píxel por píxel. Para cada píxel se observa su nivel de gris y se toma registro de la cantidad de píxeles para cada nivel de gris.

Esta información se guarda en una estructura tipo arreglo con una entrada para cada nivel de gris (256 entradas). El nivel de gris 0 representa el color negro mientras que el nivel de gris 255 representa el blanco.

Al finalizar el proceso se tendrá asociada, para cada entrada del arreglo, la cantidad de píxeles del nivel de gris que representa esa entrada.

Ejemplo:

c. píxeles	700	500	700	800	600	200	120	140	.....	10	7	4	2	0	1
nivel de gris	0	1	2	3	4	5	6	7		250	251	252	253	254	255

En este ejemplo se ve un histograma de una imagen bastante obscura, pues tiene muchos píxeles de colores cercanos al negro (de la escala de grises) y pocos píxeles de color blanco.

5.4.4 Umbralamiento de la imagen

Con este proceso lo que se intenta es convertir las zonas brillantes y obscuras (de las que ya se ha hablado anteriormente) en manchas blancas y negras respectivamente.

Se usaran la imagen resultado de la etapa de pre-procesamiento y el histograma de luminancias de la misma (el calculado anteriormente).

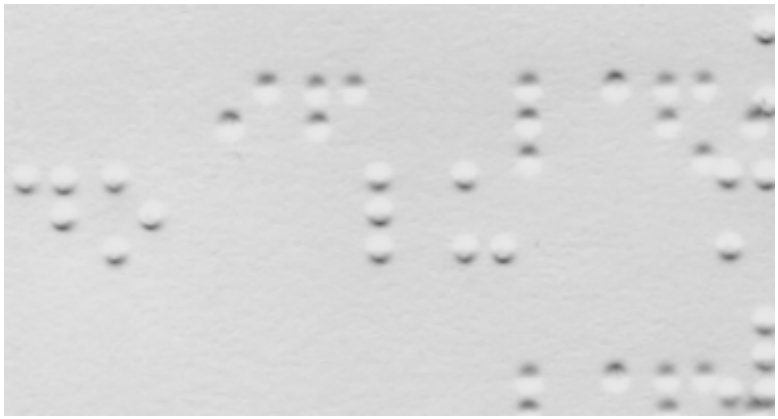
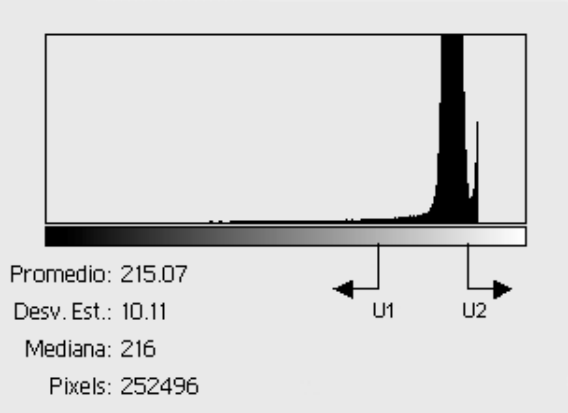


Imagen Original

Se aplica un proceso de doble umbral a la imagen en cuestión. Como resultado se obtendrá una imagen a tres colores.

Lo primero que se hace es calcular dos umbrales (niveles de gris) que llamaremos U1 y U2. Estos se calculan a partir del histograma de luminancias de la imagen.



Histograma de luminancias de la imagen original

Se toman los puntos que dejan por debajo / encima un porcentaje determinado del área total del histograma (colas del histograma). Se han obtenido buenos resultados con un porcentaje del 3 % (para documentos a una cara), de los cuales 1% es la cola izquierda y 2% es la cola derecha del mismo.

U1 es el nivel de gris para el cual, el área debajo de la curva que viene dada por la función histograma desde 0 hasta dicho punto, representa el 1% de la cantidad total de píxeles de la imagen.

U2 es el nivel de gris para el cual, el área debajo de la curva que viene dada por la función histograma desde dicho punto hasta 255, representa el 2% de la cantidad total de píxeles de la imagen.

Una vez obtenidos estos umbrales, se aplica a cada píxel de la imagen una función g. Suponiendo que representamos la imagen como una función f, donde f(x, y) es el nivel de gris para el píxel de la fila "x" columna "y", la función g a aplicar sería la siguiente:

- Si f(x, y) <= U1 entonces f(x, y) = 0 (se reemplaza por un píxel negro)
- Si f(x, y) >= U2 entonces f(x, y) = 255 (se reemplaza por un píxel blanco)
- Si f(x, y) > U1 y f(x, y) < U2 entonces f(x, y) = 128 (se reemplaza por un nivel de gris intermedio)

Imagen después de aplicar los umbrales

En la imagen anterior se puede ver la imagen en tres colores, resultado de haber aplicado los umbrales calculados. Se notan claramente las manchas blancas y negras. Ahora es más fácil buscar manchas negras y blancas, que zonas obscuras y brillantes.

Aplicando umbrales diferentes a los expuestos anteriormente las manchas blancas y negras eran muy pequeñas (algunas hasta desaparecían) o eran tan grandes que llegaban a juntarse. Es decir, se pegaban unas con otras, formando una unidad.

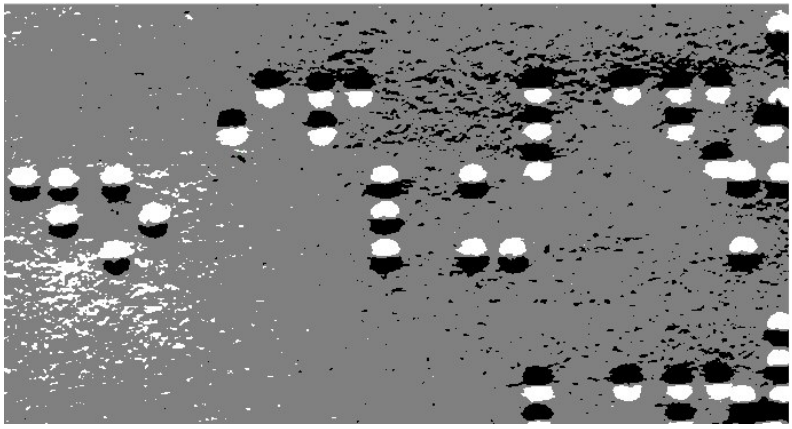


Imagen después de aplicar umbrales distintos a los óptimos.

5.4.5 Búsqueda de manchas blancas y negras

En esta etapa lo que se pretende es encontrar los centros de las manchas blancas y negras. La salida de esta será dos listas: una con las coordenadas de los centros de las manchas blancas y otra con las coordenadas de los centros de las manchas negras.

En un principio se desarrolló un método que recorría la imagen tratando de encerrar las manchas. Pero este resultaba ser bastante más lento para imágenes inclinadas (léase más de 3 grados) y podía no detectar algunas manchas. Por esto es que se desarrolló otro

método, el cual, trabaja sobre la relación de píxeles vecinos a fin de encontrar las manchas. Lo interesante de este método es que resultó ser más rápido que el anterior para diferentes ángulos de inclinación y además asegura que encontrará todas las manchas.

Explicación del método 1 (el descartado):

Este método consiste en encerrar cada mancha con un rectángulo (imaginario). En particular, el rectángulo mínimo que la contiene. De esta forma es fácil, en un paso posterior, calcular el centro de la mancha (será el centro del rectángulo mismo).

Es importante destacar que las dimensiones se miden siempre en milímetros y solo son pasadas a píxeles cuando es necesario. De esta manera se logra independencia de la resolución de escaneo de la imagen.

Para el caso de búsqueda de manchas blancas, se inspecciona cada píxel de la imagen umbralizada. Si éste es blanco se centra en el (se toma como centro sus coordenadas) un rectángulo de base "b" y altura "h" (éstas dimensiones fueron deducidas experimentalmente).

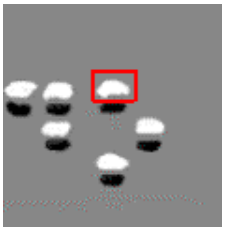
Luego se verifica que ninguno de los píxeles de la imagen en el borde de este rectángulo sean blancos. Esto es para ver si se ha encerrado la mancha con el mismo, pues para algunos píxeles blancos tomados como centro del rectángulo, puede pasar que uno o varios de sus bordes atraviesen la mancha en cuestión.

Si alguno llega a serlo se descarta el píxel en el que se había centrado el rectángulo y se sigue con el próximo.

En el caso de la imagen de la izquierda, el borde derecho del rectángulo contiene píxeles blancos. En el caso de la otra imagen los bordes superior e izquierdo contienen píxeles blancos.

Si el píxel tomado como centro del rectángulo es blanco pero ninguno de los píxeles del borde lo es, esto significa que estamos encerrando la mancha.

Tenemos las coordenadas de un rectángulo que encierra a la mancha, pero lo que queremos es hallar el rectángulo mínimo que encierra a la misma.



En el caso de esta imagen el rectángulo encierra a la mancha pero este no es el rectángulo de dimensiones mínimas que la encierra.

Para encontrarlo lo que hacemos es ir cerrando los bordes del rectángulo hasta que cada uno tope con uno o más píxeles blancos.

Ahora si, este es el rectángulo mínimo que encierra a la mancha.

Como se comentó al principio de este apartado, una vez encontrando el rectángulo mínimo, su centro es fácil de calcular y de esta forma obtenemos el centro de la mancha en cuestión.

El procedimiento para buscar las manchas negras es similar, solo que en vez de investigar los píxeles blancos se investigan los negros.

Explicación del método 2 (el usado en la herramienta de reconocimiento):

Se barre la imagen píxel a píxel, de izquierda a derecha y de arriba abajo. Sean p el píxel en cada paso del proceso, r el vecino superior de p, t el vecino izquierdo de p, q el vecino diagonal superior izquierdo de p y s el vecino diagonal superior derecho de p. La naturaleza de la secuencia barrido asegura que cuando se llega a p los píxeles r, t, q, y s ya han sido procesados.

Teniendo en cuenta los conceptos establecidos arriba se sigue el siguiente procedimiento: Si el píxel p no es del color de manchas buscado se continua hasta la próxima posición de barrido.

Si el valor de p es del color de manchas buscado se examinan r, t, q y s. Si ninguno es del color de la manchas a buscar se asigna una nueva etiqueta a p (este píxel es la primera ocurrencia para la mancha a la que pertenece dicho píxel).

Si solo uno de r, t, q, s es del color buscado se asigna la etiqueta de ese píxel a p. Si más de uno de estos píxeles es del color buscado y tienen la misma etiqueta se asigna esta a p; pero si tienen etiquetas diferentes se asigna el valor de una de estas a p y se hace anotación de la equivalencia de etiquetas.

Al final del barrido todos los píxeles del color de mancha buscado han sido etiquetados pero algunas de estas etiquetas pueden ser equivalentes. Es decir, una mancha puede estar compuesta por píxeles con etiquetas distintas, pero que son equivalentes.

Para solucionar este inconveniente se clasifican todos los pares de etiquetas equivalentes en clases de equivalencia, se asigna una etiqueta diferente a cada clase y luego se da una segunda pasada sobre la imagen reemplazando cada etiqueta por la etiqueta asignada a su clase de equivalencia. Se aprovecha esta segunda pasada para calcular las dimensiones de las manchas y sus respectivos centros.

El proceso de búsqueda de manchas se realiza dos veces, una vez para hallar las blancas y otra para hallar las negras.

Si la imagen presenta un ruido de adquisición importante, este se presentará (una vez umbralada la imagen) como pequeñas manchitas blancas o negras (bastante más pequeñas que las que se están buscando).

El método anterior asegura que encontrará todas las manchas, inclusive las manchitas. Para solucionar este inconveniente lo que se hace es filtrar la colección de manchas halladas para quedarse con las que son mayor a un área determinada.

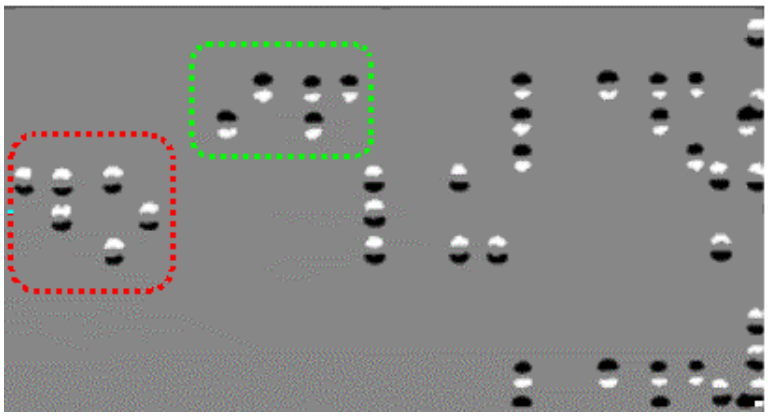
5.4.6 Búsqueda de puntos (solo de referencia)

En esta etapa del reconocimiento debemos hallar las coordenadas de los centros de los puntos Braille de la hoja. Recordemos que los puntos Braille vienen dados por parejas cercanas de manchas blancas y negras cercanas.

Se trabajó con la imagen a tres colores, resultado de la etapa de umbralamiento; y las coordenadas de los centros de las manchas, halladas en la etapa anterior. Es importante aclarar que para el correcto funcionamiento del algoritmo que se expondrá más adelante las listas de manchas estén ordenadas por coordenadas Y creciente y para iguales Y, coordenadas X creciente.

Debemos tener presente que si estamos trabajando con hojas escritas a doble cara (braille interpunto) la imagen a tres colores contiene las manchas blancas y negras de las dos caras.

Los puntos de una cara estarán formados por una mancha blanca arriba y una negra debajo (aproximadamente en la misma vertical), mientras que los puntos de la otra estarán formados por una mancha negra arriba y una blanca abajo.

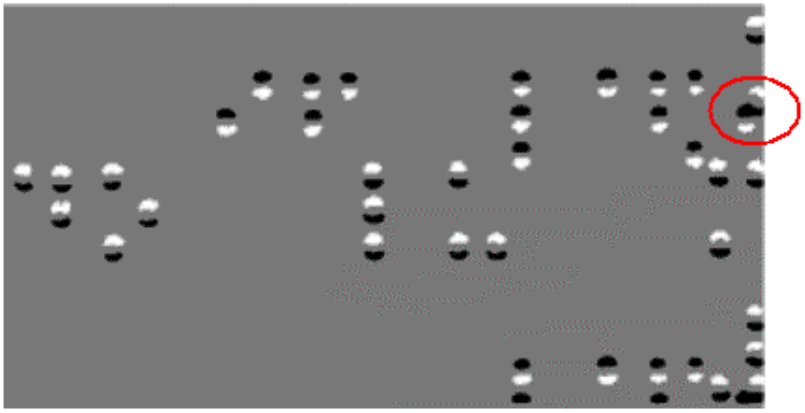


Las manchas encerradas por el cuadro punteado rojo corresponden a la cara "A" de la hoja. Las encerradas por el cuadro punteado verde corresponden a la cara "B" de la misma.

Debe quedar claro que lo que se quiere en esta etapa es obtener solo algunos puntos Braille, no todos los de la hoja. Estos son solo de referencia y serán necesarios en etapas posteriores del reconocimiento.

Lo importante en esta etapa es no detectar puntos falsos, es decir puntos que no existen en la hoja. Esto puede suceder si al armar una pareja se toma una mancha de un punto real de la hoja con una mancha colada en la imagen por alguna causa. También puede suceder debido a las manchas cercanas que llegan a pegarse al aplicar los umbrales (etapa de umbralamiento). Lo último ocurre cuando hay puntos hacia arriba y abajo demasiado cerca.

En la imagen que se muestra a continuación puede verse claramente (dentro del circulo rojo) dos manchas blancas pero solo una negra. En realidad deberían ser dos manchas negras: una correspondiente a la parte inferior de un punto de la cara "A" y otra correspondiente a un punto de la cara "B" de la hoja.



Dentro del círculo rojo se observan dos manchas blancas (una de cada cara), pero solo una negra. Esto se debe a que las manchas cercanas se pegan al aplicar los umbrales.

Se desarrolló un método para obtener las coordenadas de los centros de los puntos, utilizando las coordenadas de los centros de las manchas y la imagen a tres colores. Las listas de coordenadas de manchas están ordenadas por coordenada "y" creciente. Para iguales valores de "y" se ordena por "x" creciente.

Explicación del método:

Se recorre la lista de manchas blancas. Se selecciona la primera y se verifica que sobre ésta no haya una mancha negra. Si es así, se busca si hay una mancha negra debajo de ella. En caso afirmativo se habrá encontrado un punto Braille de la cara "A" de la hoja (se encontró una ocurrencia del patrón mancha blanca arriba, mancha negra abajo).

Se calcula su centro con la información de coordenadas de las manchas en cuestión (la blanca y la negra) y se lo guarda en una lista de puntos para cara "A". Luego la mancha negra (la que está por debajo de la mancha blanca) es removida de la lista de manchas negras.

La distancia máxima para verificar si una mancha está por arriba o por debajo de otra se ajustó experimentalmente y es de alrededor de 1.0 mm. La misma es pasada a píxeles al momento de la comparación, ya que las coordenadas de los centros de las manchas están expresadas en esa unidad.

Para calcular la cantidad de píxeles que representan una cierta cantidad de milímetros se utiliza la siguiente formula f:

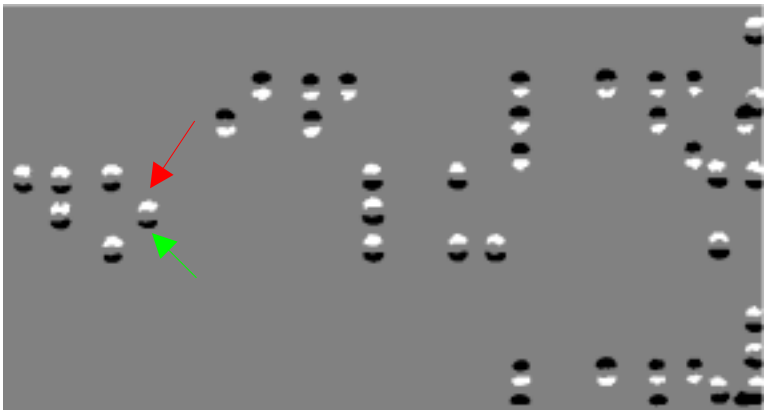
$$f(x) = (DPI\_RES * x) / 25,4$$

Donde DPI_RES es la resolución (cantidad de píxeles por pulgada) de la imagen y 25,4 es la cantidad de milímetros que representa una pulgada.

A continuación se sigue con las demás manchas blancas de la lista hasta tratarlas todas.

Para hallar los puntos de la otra cara (cara "B") se trabaja con la lista de manchas negras que sobraron. Para cada una se verifica que abajo haya una mancha blanca (ésta será seguro una mancha blanca para la que no se encontró pareja negra arriba y por eso fue descartada en la instancia anterior).

En caso afirmativo se calcula el centro del punto hallado con la información de coordenadas de los manchas en cuestión (la negra y la blanca) y se lo guarda en una lista de puntos para cara "B".



En esta imagen se ve una mancha blanca debajo de la flecha roja y una mancha negra arriba de una mancha negra. Estas manchas corresponden a un punto de la cara "A" del documento Braille.

Al final de esta etapa se tienen dos listas: la de puntos de la cara "A" y la de puntos de la cara "B".

5.4.7 Búsqueda de los 10 puntos más cercanos al origen

En esta etapa como en las siguientes se trabajará con la información de las listas de puntos de cada cara, de forma independiente.

Se recorre la lista de puntos de la cara que se está procesando, buscando los diez puntos de coordenadas más cercanas al (0, 0). La coordenada (0, 0) corresponde al borde superior izquierdo de la imagen.

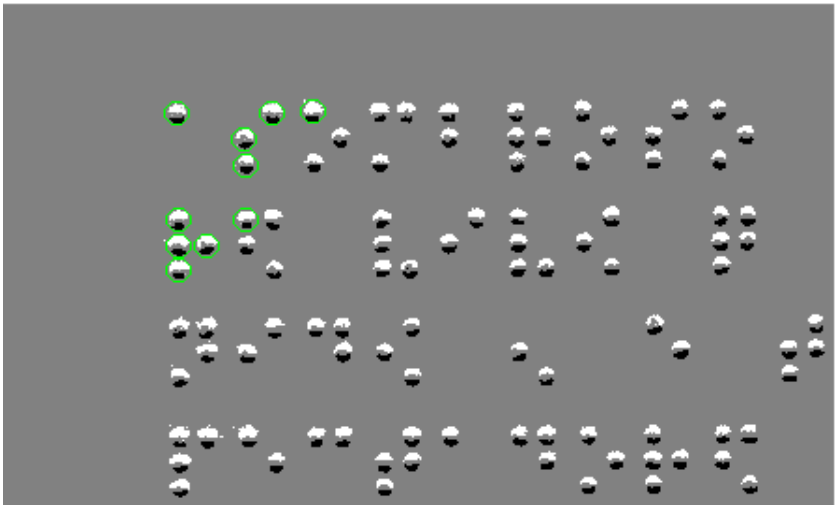
Para calcular la distancia al origen se utiliza la conocida formula:

$$d^2 = (x_1 - x_0)^2 + (y_1 - y_0)^2$$

que en este caso se reduce a

$$d^2 = x_1^2 + y_1^2$$

Los puntos encontrados se guardan en una lista. Esta será usada en etapas posteriores del reconocimiento.



Los puntos Braille marcados con verde son los diez más cercanos al borde superior izquierdo.

5.4.8 Detección de la inclinación de la hoja

En esta etapa, se detecta el ángulo de inclinación de la hoja sobre el escáner. Este valor será de gran utilidad en las etapas posteriores del reconocimiento.

Según la bibliografía estudiada [UVO097], uno de los métodos más utilizados para la detección de la inclinación de documentos sobre el escáner es la transformada de Hough.

No obstante, se exploraron otras ideas, aprovechando las características de los documentos Braille:

La primera idea fue fabricar una línea con la primera fila de puntos encontrados y establecer su ángulo de inclinación. Se trabajo sobre el dominio de coordenadas de puntos hallados, no sobre el de la imagen.

Primero se hallaba el primer punto de la hoja y luego se intentaba formar una línea imaginaria por donde pasaran este punto y otros. Se probaban diferentes ángulos. El que maximizaba la cantidad de puntos en la línea (con un pequeño margen de error), era el ángulo de inclinación buscado.

El método resultó ser bastante exacto, pero en algunos casos no devolvía los resultados esperados. Uno de los problemas principales es que era muy dependiente del correcto cálculo del primer punto de la hoja.

Teniendo en cuenta esto, se desarrolló un método más preciso, que no depende de la correcta detección del primer punto de la hoja. Este método trabaja en el dominio de la imagen en vez de en el de los puntos (coordenadas).

Explicación del método desarrollado:

Se barre la imagen con diferentes ángulos. Para cada ángulo se suman los píxeles negros o blancos de cada fila. Luego se calcula la varianza de las sumas por fila. El ángulo que maximice la varianza es el ángulo de inclinación de la hoja.

Este algoritmo es bastante lento. Para aumentar la performance del mismo se tuvieron en cuenta algunas consideraciones:

- Se trabaja sobre una mitad de la imagen. Se elige la mitad que tenga mas información de puntos (mas probabilidad de texto Braille). Para realizar esta elección se pueden contar la cantidad de píxeles blancos y negros de cada mitad. Se selecciona la mitad que tenga maás píxeles blancos y negros.

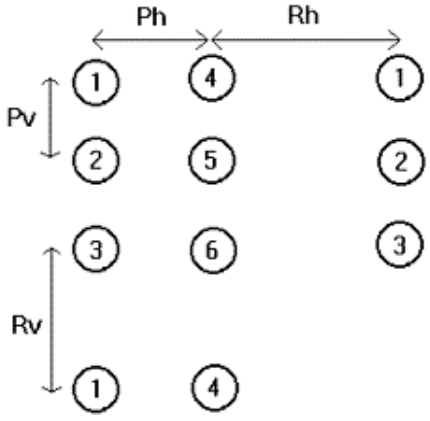
- Se hace una reducción de la resolución horizontal de la imagen de 1 en 5. Es decir al realizar las sumas por fila se avanza de a 5 píxeles. Esto acelera considerablemente el algoritmo y no supone una perdida de precisión en el cálculo.

- Se utilizó un método de aproximación. En cada paso se busca el ángulo de inclinación con más precisión pero dentro del intervalo de error del paso anterior.

De esta forma se aumenta notablemente el la performance del algoritmo a la vez que se mantiene la robustez y precisión del mismo.

5.4.9 Cálculo de posición del primer punto de la hoja

Como se dijo anteriormente (ver capitulo Sistema Braille), las distancias entre puntos en la escritura Braille siguen un patrón definido.



Malla Braille tradicional

Donde Ph, Rh, Pv y Rv son distancias conocidas y expresadas en milímetros. Conociendo la resolución de escaneado, esas distancias se pueden pasar a píxeles. Además, las filas y columnas estarán inclinadas con inclinación conocida (que podría ser

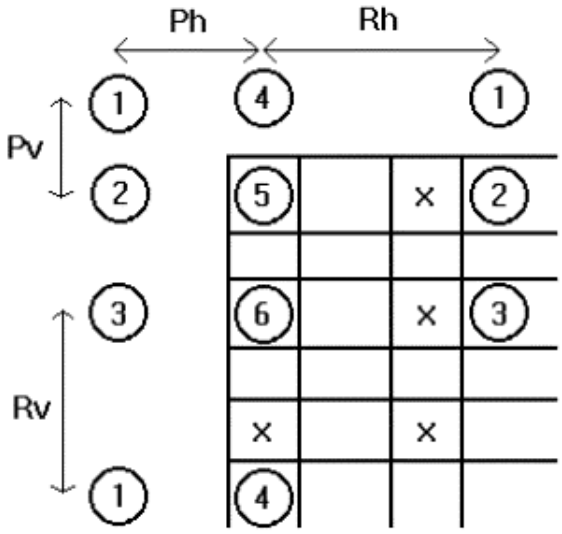
nula). Se usará la suposición de que las filas y columnas son siempre perpendiculares (aunque a veces no sea así).

Una vez conocido el ángulo de inclinación uno puede centrarse en un punto y moverse las distancias adecuadas hasta encontrar todos los demás (por lo menos todos los verdaderos).

Para ello se necesitan dos cosas:

- Que el punto inicial sea verdadero.
- Saber a que posición (de 1 a 6) corresponde dentro de su carácter.

Se utiliza la lista de puntos más cercanos al origen (calculada en la etapa anterior) y se considera cada uno como posible punto inicial. Para cada punto se suponen las 6 posiciones posibles. Después se recorre la malla (llamamos malla al conjunto de posibles puntos de la hoja) contando los puntos de entrada que son "encajados". La máxima de esas 60 cuentas determina el punto de referencia inicial.



Determinación del punto inicial por ensayo y error

En la figura de arriba se ve el momento en el que algoritmo supone que el punto 5 está en la posición de carácter 1. Para ello se centra en el una malla de búsqueda a fin de encontrar todos los puntos de la hoja.

Sin embargo, como se parte de una suposición falsa, se buscan puntos en muchas posiciones incorrectas (en la figura están marcadas con una x). De esta forma, cuando el número de puntos encontrados sea máximo, se habrá encontrado el punto inicial.

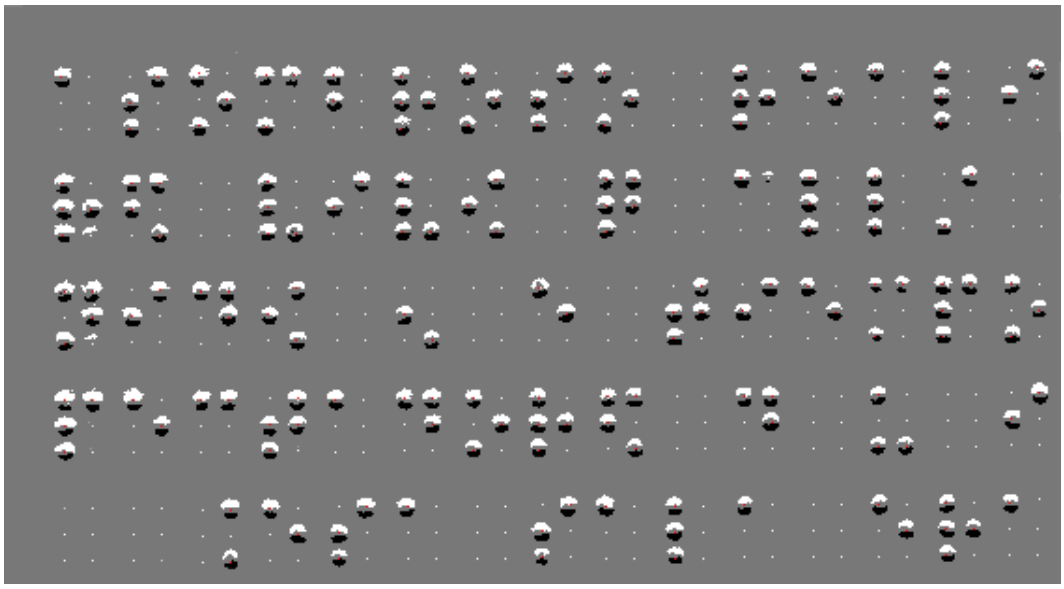
Para hacer más robusto este algoritmo frente al ruido se tienen en cuenta dos aspectos:

- Se permite cierto margen de error en la búsqueda de puntos (encajado).

- No se realiza la búsqueda en toda la hoja sino en un número limitado de filas y columnas a partir del punto de estudio. Esto permite que el algoritmo sea mas rápido y además compensa pequeños errores en el ángulo de inclinación calculado.

5.4.10 Cálculo de todos los posibles puntos de la hoja

Como se dijo antes, llamamos malla Braille al conjunto de posibles puntos de la hoja. La construcción de la misma se realiza partiendo del punto de referencia inicial (calculado en la etapa anterior), colocando sobre el una plantilla de tamaño un carácter. Esta plantilla se moverá en todas direcciones hasta los límites de la hoja, determinando de esta forma, todos los posibles puntos.



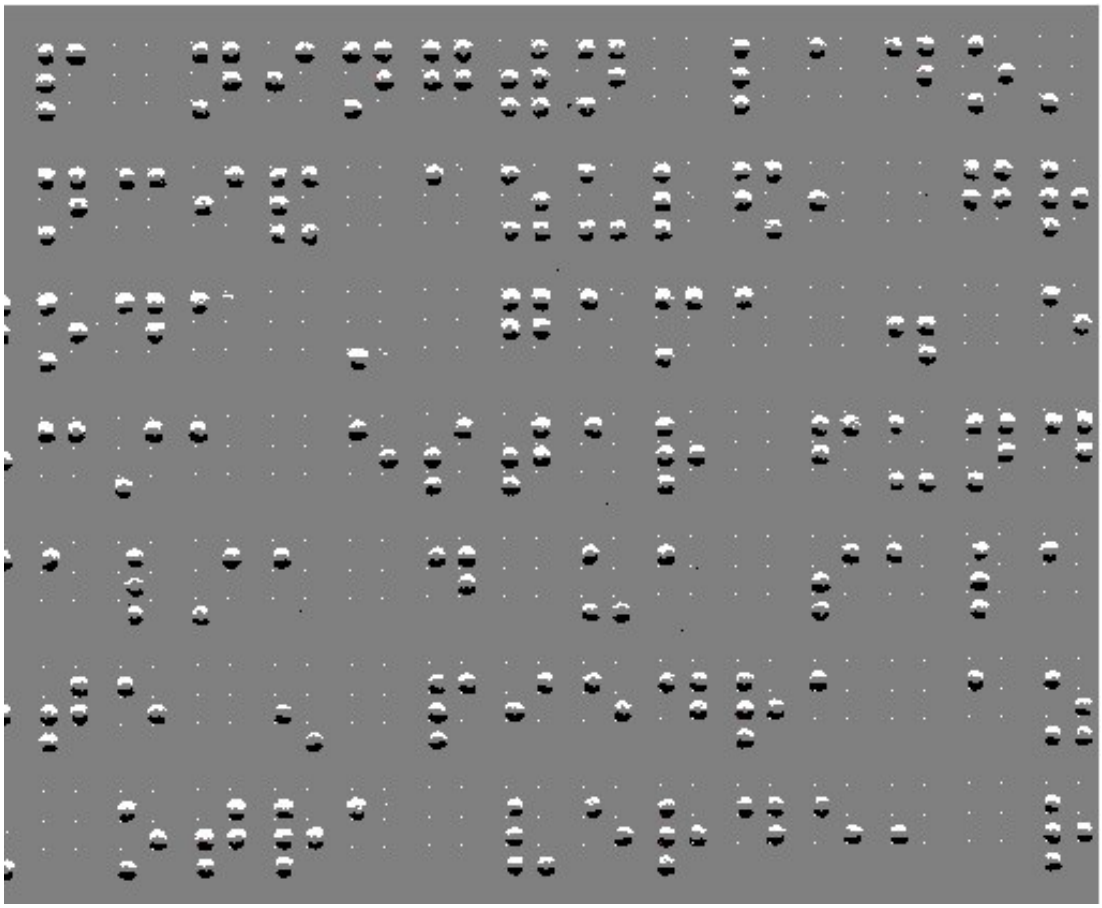
Los puntitos blancos son los posibles puntos Braille de la hoja. Los que están en rojo son puntos de la malla que coinciden con los puntos Braille reales de la hoja. Cada conjunto de 6 puntitos es una plantilla de carácter Braille.

El movimiento de la plantilla se hace teniendo en cuenta las distancias estándar entre caracteres y el ángulo de inclinación calculado en una de las etapas anteriores. Es decir, de ser necesario, la malla se construye inclinada con el ángulo de inclinación calculado.

Se detecto un problema al desplazar la plantilla sobre la imagen. Al avanzar hacia la derecha se iba perdiendo precisión. Las plantillas quedaban desfasadas con respecto a la posición donde deberían haber quedado.

Esto es por que generalmente las distancias no coinciden exactamente y el error se va acumulando a medida que se va avanzando sobre la fila de caracteres. Sucede algo similar al desplazar la plantilla verticalmente para escanear la siguiente fila de caracteres.

En la siguiente figura se puede ver un ejemplo de esta situación:



Los puntitos blancos son los posibles puntos Braille de la hoja. Los que están en rojo son puntos de la malla que coinciden con los puntos Braille reales de la hoja. Cada conjunto de 6 puntitos es una plantilla de carácter Braille.

Para solucionar este inconveniente se realizó la creación de la malla de forma adaptativa. Es decir, al posicionar la plantilla se busca en la lista de puntos Braille (calculada en 5.4.6) para ver cuantos puntos son encajados en esa posición. Si no se encaja ninguno se sigue con el próximo carácter, pero si se encajan puntos la plantilla (para esa posición) se centra basándose en ellos.

El próximo movimiento de la plantilla será relativo a la posición actual de la plantilla adaptada. Esta forma de crear la malla es necesaria para que el resultado no dependa en exceso del punto de referencia inicial. Además de esta forma se corrigen pequeños errores en el calculo del ángulo de inclinación de la hoja. Las pruebas realizadas han demostrado que el algoritmo funciona perfectamente con inclinaciones de hasta 2 grados, sin tener presente el ángulo de inclinación para corrección.

A veces también puede suceder que algún carácter esté escrito bastante corrido de su posición esperada. Esto puede generar un desincronismo difícil de corregir. En estos casos puede ser que los caracteres sucesivos no sean reconocidos correctamente.

Es importante aclarar que no hay que preocuparse por los puntos Braille que se hayan detectado erróneamente en la etapa de búsqueda de puntos (ver 5.4.6), puesto que simplemente estarán descolocados y serán descartados al momento de construir la malla.

5.4.12 Reconocimiento de los puntos Braille

Esta es la etapa del reconocimiento propiamente dicho. En esta instancia se dispone de la información de coordenadas de todos los posibles puntos Braille de la hoja (malla Braille).

De esta forma se sabe en que ubicaciones de la imagen se debe investigar para determinar cuales puntos están activos y cuales no. A la malla se la puede ver como una lista de caracteres Braille (plantillas). Cada carácter mantiene la información de sus seis puntos.

En un principio se utilizo la imagen umbralizada (obtenida en una etapa anterior). Resulto ser que algunos puntos no llegaban a reconocerse, puesto que las manchas eran muy pequeñas.

Para solucionar esto umbralizó la imagen escaneada, con umbrales más grandes. De esta forma las manchas se agrandan y esto ayuda a disminuir el porcentaje de errores al momento de reconocer los puntos.

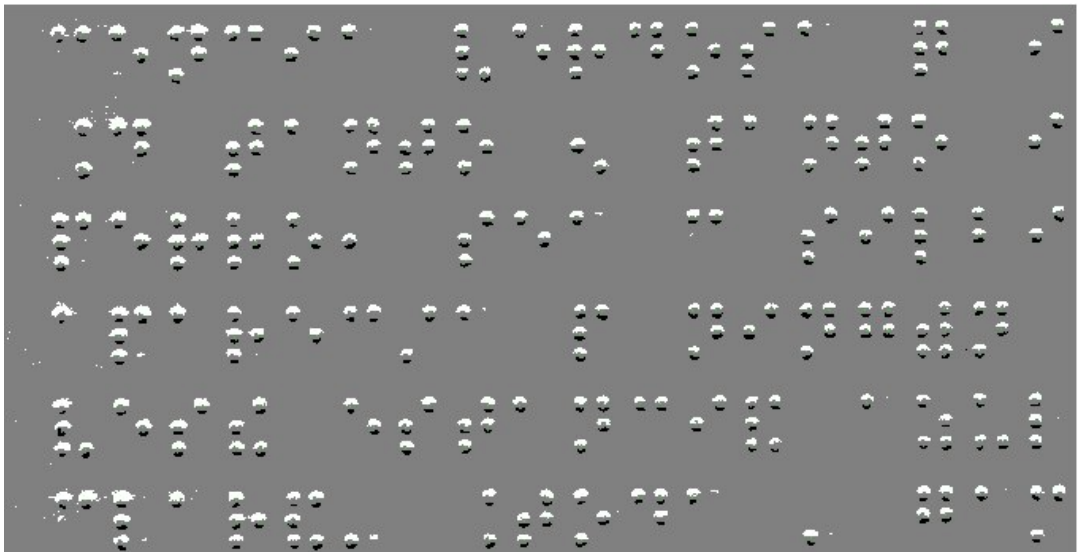
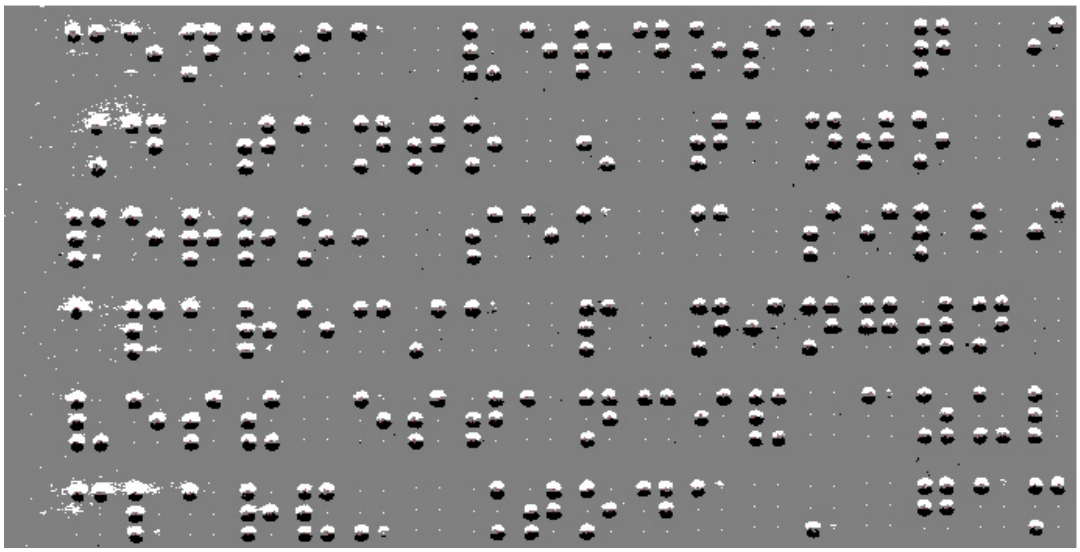


Imagen umbralizada con umbrales para detección de manchas. Notar que algunas manchas sobre todo negras son bastante pequeñas. Esto puede complicar el proceso de reconocimiento final para determinar cuales puntos están activos y cuales no.



La misma imagen umbralizada, pero con umbrales más grandes. De esta forma ,al agrandarse las manchas, se intenta minimizar los errores al momento de reconocer los puntos.

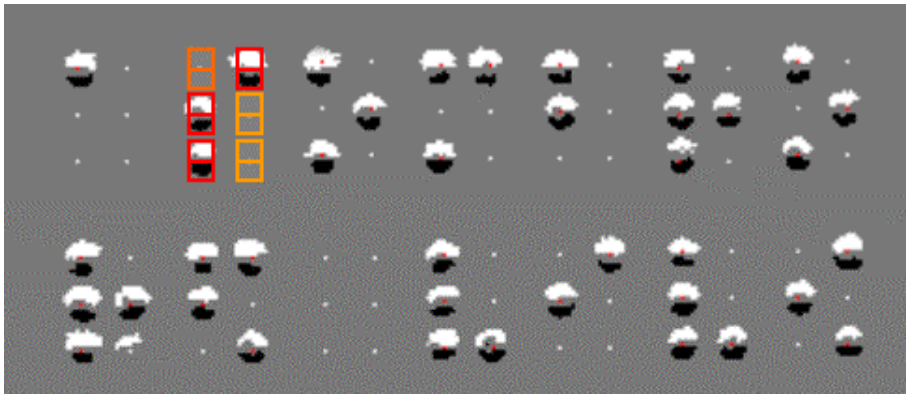
Explicación del método:

Se recorre la malla de caracteres Braille. Se toma el primer carácter de la malla y se investiga en la imagen en las coordenadas de cada punto del mismo para ver si se encuentra el siguiente patrón:

Mancha blanca arriba y mancha blanca abajo, en el caso de cara "A" de la hoja, o mancha negra arriba y mancha blanca abajo para el caso de cara "B" de la hoja.

Para determinar si existen las manchas se cuenta la cantidad de píxeles del color de la mancha correspondiente en dos rectángulos imaginarios de 2 mm * 0.8 mm. Uno por sobre el posible punto y otro por debajo de este.

Si la cantidad de píxeles es mayor a 5, tanto para la mancha blanca como para la negra, se asume que ese punto Braille está activo. Estos valores se ajustaron experimentalmente y corresponden al caso de imágenes de 150 dpi. No obstante debería trabajarse con valores independientes de la resolución de escaneado de la imagen.



Para determinar si existen las manchas se cuenta la cantidad de píxeles del color de la mancha correspondiente en un rectángulo imaginario del tamaño promedio de una mancha. Un rectángulo por sobre el posible punto y otro por debajo este. Si la cantidad de píxeles es mayor a un valor para los dos tipos de mancha, se asume que ese punto Braille está activo. En esta figura, en el segundo carácter se ha determinado que están activos los puntos de los rectángulos rojos.

Luego se realiza el mismo procedimiento sobre los demás caracteres de la malla. Al finalizar esta etapa, se sabe para cada carácter de la malla cuales de sus puntos están activos y cuales no.

5.4.13 Transformación a texto digital

El objetivo de esta etapa es traducir los caracteres Braille hallados, en caracteres digitales. En esta instancia se sabe, para cada carácter Braille de la malla, cuales son los puntos que están activos y cuales no (se realizó en la etapa anterior). Tomando ventaja de ello se realiza el siguiente cálculo para cada uno de los caracteres:

$$c(p) = 32 * v(p_6) + 16 * v(p_5) + 8 * v(p_4) + 4 * v(p_3) + 2 * v(p_2) + v(p_1)$$

donde $v(p_x)$ es 1 si el punto p_x del carácter que se está transformando se determinó como activo y 0 en caso contrario.

El valor c(p) calculado va a estar en el rango de 0 a 63 y se utiliza como índice de acceso para la tabla que contiene el alfabeto. En cada posición de esta tabla se ubica el carácter digital correspondiente.

En la escritura Braille existen los llamados símbolos dobles (ver capítulo 3). Estos son tomados como valores especiales de c(p). El valor 40 representa al símbolo doble de mayúscula. De esta forma cuando se encuentra que c(p) = 40 significa que el próximo carácter será en mayúscula.

Otro valor especial de c(p) es c(p) = 60 (símbolo doble de número). Cuando se encuentre c(p) = 60 significará que los próximos caracteres son números.

Entonces, dependiendo del valor c(p), se realiza un tratamiento distinto al momento de la traducción:

- Si el valor c(p) es distinto de 40 y 60 se busca con este índice en la tabla y se obtiene el carácter de texto correspondiente.

- Si el valor c(p) es igual a 40 significa que el o los próximos caracteres están en mayúscula. Entonces se inspecciona el próximo carácter de la malla. Si el valor de c(p) para este carácter es distinto de 40 se busca en la tabla con este índice obteniendo el carácter de texto correspondiente y se lo pasa a mayúscula; mientras que si este segundo carácter es nuevamente igual a 40 (esto indica que toda la palabra es mayúscula – ver Capitulo Braille) se inspeccionan los próximos caracteres hasta que el valor de c(p) sea igual a cero 0 (es un carácter de espaciado), obteniéndose los caracteres para los índices c(p) calculados y se los pasa todos a mayúsculas.

- Si el valor c(p) es igual a 60 significa que los próximos caracteres son números. En este caso también se utiliza el valor de c(p) como índice, pero de otra tabla del alfabeto. Esto se resolvió así por que la escritura Braille utiliza los mismos símbolos con los que representa las primeras letras del abecedario para representar los dígitos numéricos. De otra forma se tendrían dos posibles valores para el mismo índice sobre la misma tabla.

5.4.13 Almacenamiento del texto digital

Una vez realizado el reconocimiento, el texto digital obtenido es guardado en un archivo de texto plano (ascii) para luego ser impreso o utilizado como entrada de otro programa. Por ejemplo podría ser usado como entrada de un programa corrector ortográfico. El archivo se guarda con extensión .txt.

5.5 Reconocimiento de hojas de tamaño mayor a A4

Las hojas más comunes en la escritura Braille son de tamaño mayor que el formato A4, mientras que los escáneres convencionales más comunes son los A4. En el mercado se pueden conseguir escáneres A3 (de tamaño mayor que A4), pero a precios bastante mas altos que el de los anteriores.

No obstante, se puede escanear una hoja A3 en un escáner A4, por trozos. Esto implica colocar la hoja de costado sobre el escáner y realizar tantos escaneos consecutivos como sea necesario.

Primer escaneo Segundo escaneo Resultado del merge

Además se deben tener algunas consideraciones al momento de realizar el reconocimiento automático:

5.5.1 Rotación de la imagen:

Una vez obtenida la imagen escaneada la imagen debe rotarse 90° en sentido antihorario. Las siguientes etapas trabajaran con la imagen rotada.

Para obtener la imagen rotada se aplica a la imagen original una transformación. Suponiendo que pixAux es el array de píxeles de la imagen, que width y height son el ancho y alto de la imagen antes de ser rotada y que pix es el array de píxeles de la imagen ya rotada; el algoritmo de rotación sería el siguiente:

```
//roto la imagen 90° en sentido antihorario
int pos = 0;
for(int j=width-1;j>=0;j--) {
        for(int i=0;i<height;i++) {
                pix[pos] = pixAux[i * width + j];
                pos++;
        }
}
```

Tener en cuenta que una vez rotada la imagen el nuevo valor de ancho (width) de la nueva imagen será el alto (height) de la imagen antes de ser rotada y el valor de alto de la nueva imagen será el ancho de la imagen antes de ser rotada.

5.5.2 Manchas arriba y abajo

En la sección 5.4.6 (búsqueda de puntos) se dijo que un punto Braille estaba compuesto por una mancha blanca arriba y una mancha blanca abajo, y que ese era el patrón a buscar. En este caso al haber rotado la imagen un punto Braille se compone por una mancha blanca a la izquierda y una mancha negra a la derecha, siendo este el nuevo patrón a buscar.

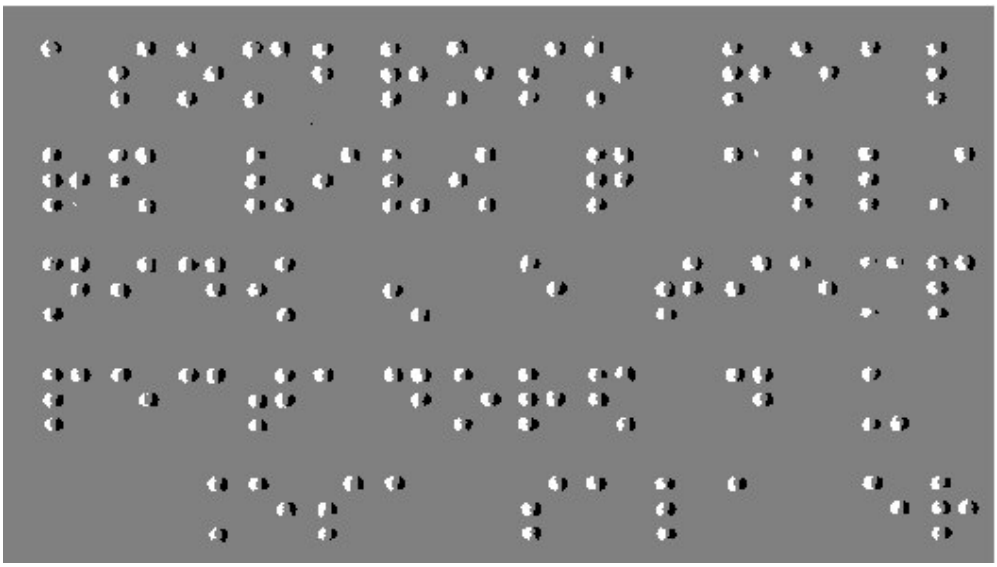


Imagen rotada y umbralizada. Notar como la disposición de las manchas blancas y negras forman los puntos Braille.

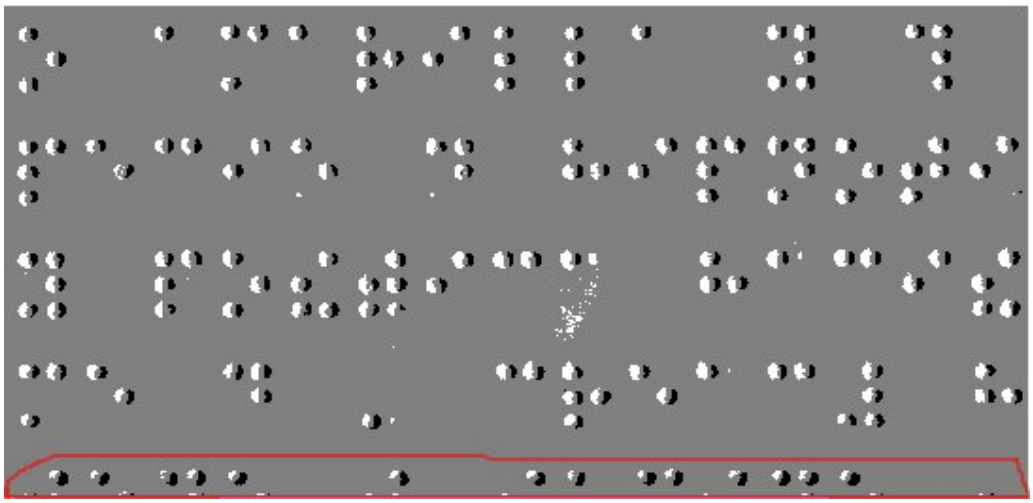
Esto supone una variación en los algoritmos expuestos en la sección 5.4.6 (búsqueda de puntos). La clave está en tener presente que ahora el ancho de la imagen será el alto de la imagen antes de rotar (ídem alto de la nueva imagen por ancho de la imagen sin rotar) y que al momento de buscar las parejas de manchas blancas y negras no se busca abajo, sino al costado.

5.5.3 Merge en la traducción

Al escanear y traducir por trozos un el texto de un documento quedará segmentado. Para solucionar este problema lo que se hace es un 'merge' entre el resultado de la traducción del trozo actual y el resultado de la traducción del trozo anterior.

Para esto, es necesario que al menos dos líneas de texto completas se solapen en los sucesivos escaneos. Luego, se trabaja en el dominio del texto juntando los dos trozos y eliminando la parte de los trozos de texto que quedaría repetida.

Para realizar el merge, primero se obtiene la anteúltima línea de texto del primer trozo de texto No se utiliza la última, por que es probable que una parte haya quedado fuera de la plancha del escáner y presentándose incompleta verticalmente y de esta forma se haya traducido con un alto porcentaje de error.



En la figura se puede ver la última línea de este trozo de una hoja Braille encerrada por la línea roja. En este caso esta línea quedó incompleta y debe descartarse.

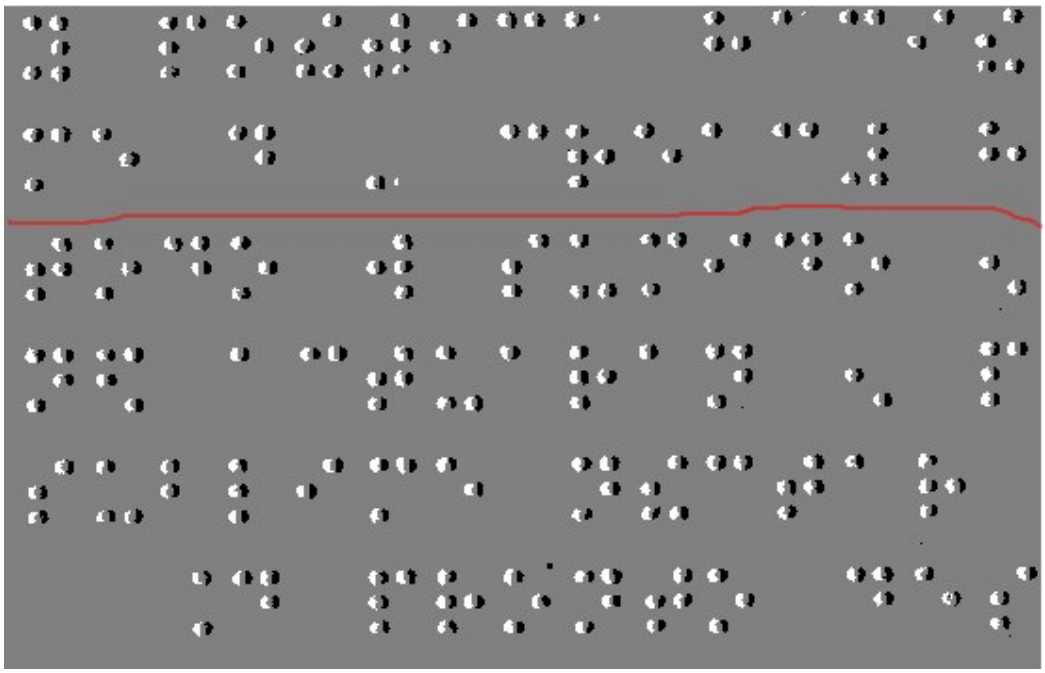


Imagen del segundo trozo escaneado. El texto traducido correspondiente a los puntos por encima de la línea roja es descartado al momento de realizar el proceso de merge.

Capítulo 6 – Java OBR (Aplicación desarrollada)

6.1 Lenguaje de desarrollo

Para la implementación de esta herramienta se decidió utilizar Java como lenguaje de desarrollo, dadas sus interesantes características para el tratamiento de imágenes, su portabilidad y su condición de lenguaje libre.

6.1.1 Características generales del lenguaje

Las principales características del lenguaje se resumen en la siguiente lista:

- Simple y familiar
- Orientado a objetos
- Distribuido
- Robusto
- Seguro
- Arquitectura neutral
- Multithread
- Alta performance
- Interpretado
- Dinámico

Los programas Java corren sobre una JVM (Java Virtual Machine). Como existe una JVM para cada hardware - sistema operativo el programa puede haber sido desarrollado por ejemplo en una máquina windows y luego correr en un linux cualquier otro sistema que tenga instalado una versión de JVM compatible.

6.1.2 Manejo de imágenes en JAVA

La API de Java provee un conjunto de clases con métodos muy interesantes para el tratamiento de imágenes digitales. Casi la totalidad de estas clases se ubican en el paquete java.awt. A Continuación se presentará las clases y métodos más representativas.

6.1.2.1 La clase Image

La clase *Image*, como cualquier otra, está formada por una serie de constantes, variables, constructores y métodos. Aunque la clase *Image* dispone de un constructor, es una clase abstracta, por lo que no se puede instanciar ningún objeto llamando directamente a este constructor. Se puede conseguir un objeto *Image* indirectamente por la invocación del método *getImage()* de las clases *Applet* o *Toolkit*. El método *getImage()* utiliza un hebra de ejecución o tarea separado para cargar la imagen. El resultado práctico de la invocación a getImage() es la asociación entre una referencia de tipo *Image* y un fichero localizado en algún lugar que contiene la imagen que interesa.

Otra forma de obtener un objeto *Image* es invocar al método *createImage()* de las clases *Component* y *Toolkit*. La lista siguiente muestra una pequeña descripción de los métodos de la clase *Image*:

flush(), libera todos los recursos que utiliza el objeto *Image*.
getGraphics(), crea un contexto gráfico para pintar una imagen en segundo plano.

getHeight(ImageObserver), determina la altura de la imagen.
getWidth(ImageObserver), determina la anchura de la imagen.
getProperty(String,ImageObserver), obtiene una propiedad de la imagen por su nombre.
getScaledInstance(int,int,int), crea una versión escalada de la imagen.
getSource(), devuelve el objeto que produce los píxeles de la imagen.

6.1.2.2 La clase Color

Esta clase es bastante simple, encapsula los colores utilizando el formato *RGB*. En este formato los colores se definen por sus componentes Rojo, Verde y Azul, representado cada uno de ellos por un número entero en el rango 0-255. En el caso del API Java2D, el espacio de color que se usa es el *RGB*, un estándar del consorcio W3.
Hay otro modelo de color llamado *HSB* (matiz, saturación y brillo). La clase *Color* proporciona métodos de conveniencia para poder realizar la conversión entre un modelo y otro sin dificultad.
La clase *Color* proporciona variables estáticas finales que permiten el uso de cualquiera de los treinta colores, simplemente especificando su nombre. Para usar estas variables sólo es necesario referenciar el color por su nombre de variable, por ejemplo *objeto.setBackground(Color.red)*.

Los constructores de esta clase son tres, dos de los cuales permiten instanciar un nuevo objeto de tipo *Color* indicando las cantidades de rojo, verde y azul que entran en su composición mediante valores enteros en el rango 0 a 255: *Color(int, int ,int)* El otro de los constructores permite especificar la contribución de rojo, verde y azul al color final y admite valores flotantes en el rango 0.0 a 1.0: *Color(float, float, float)*
El tercer constructor admite solamente un número entero en el cual se especifica el valor RGB, de forma que la cantidad de color rojo que interviene en el color final está especificada en los bits 16-23 del argumento, la cantidad de verde en los bits 8-15 y la cantidad de azul en los bits 0-7. *Color(int)*.

Para el uso de colores el JDK dispone de una serie de métodos, de los cuales, la página oficial de Java proporciona una completa descripción en la documentación del AWT.
Los métodos que se presentan a continuación devuelven un entero representando el valor RGB para un determinado objeto *Color*:

getRed(), devuelve el componente rojo del color como un entero en el rango 0 a 255.
getGreen(), devuelve la cantidad de verde que entra en la composición del objeto Color, en el rango 0 a 255.
getBlue(), devuelve el componente azul del color con un entero en el rango 0 a 255.
getRGB(), devuelve un entero representando el color RGB, utilizando los bits para indicar la cantidad de cada uno de los componentes rojo, verde y azul que entran en su composición.
Los bits 24 a 31 del entero que devuelve el método son 0xff, los bits 16 a 23 son el valor rojo, los bits 8 a 15 son el valor verde y los bits 0 a 7 indican el valor del color azul. Siempre en el rango 0 a 255.

Los siguiente métodos devuelven un objeto de tipo *Color*, de forma que se pueden utilizar para crear objetos de este tipo:

brighter(), crea una versión más brillante del color.

darker(), crea una versión más oscura del color.
decode(String), convierte una cadena a un entero y devuelve el color correspondiente.

Los métodos que se indican a continuación, también devuelven un objeto de tipo *Color*, pero están especializados para trabajar con el sistema a través de la clase *Properties*.

getColor(String), busca un color entre las propiedades del sistema. El objeto *String* se utiliza como el valor clave en el esquema clave/valor utilizado para describir las propiedades en Java. El valor es entonces utilizado para devolver un objeto *Color*.
getColor(String,Color), busca un color entre las propiedades del sistema. El segundo parámetro es el que se devuelve en el caso de que falle la búsqueda de la clave indicada en el objeto *String*.
getColor(String,int), busca un color entre las propiedades del sistema. El segundo parámetro es utilizado para instanciar y devolver un objeto *Color* en el caso de que falle la búsqueda de la clave indicada en el objeto *String*.

Los siguientes métodos se utilizan para realizar la conversión entre el modelo de color *RGB* y el modelo *HSB*:

getHSBColor(float,float,float), crea un objeto de tipo *Color* basado en los valores proporcionados por el modelo HSB.
HSBtoRGB(float,float,float), convierte los componentes de un color, tal como se especifican en el modelo HSB, a su conjunto equivalente de valores en el modelo RGB.
RGBtoHSB(int,int,int,float[]), convierte los componentes de un color, tal como se especifican en el modelo RGB, a los tres componentes del modelo HSB.

6.1.2.3 El método drawImage()

El método *drawImage()* de la clase *Graphics* se utiliza para presentar una imagen en pantalla.

En la clase *Graphics* están disponibles varias versiones sobrecargadas de este método, algunas de las más importantes se muestran a continuación.

drawImage(Image,int,int,Color,ImageObserver), pinta la imagen que se indica situando su esquina superior izquierda en la coordenada que se pasa (segundo y tercer parámetro), en el contexto gráfico actual. Los píxeles transparentes se pintan en el color de fondo que se indica. Esta operación es equivalente a rellenar un rectángulo del ancho y alto de la imagen dada con un color y luego pintar la imagen sobre él, aunque probablemente más eficiente.

drawImage(Image,int,int,ImageObserver), hace lo mismo que el anterior pero los píxeles transparentes de la imagen no se ven afectados. Este método retorna inmediatamente en todos los casos, incluso aunque la imagen completa no se haya terminado de cargar y no se haya presentado completamente en el dispositivo de salida.

drawImage(Image,int,int,int,int,Color,ImageObserver), pinta la imagen que se pasa dentro del rectángulo que se indica en los parámetros, escalando esa imagen si es necesario. El método retorna inmediatamente aunque la imagen no se haya cargado completamente. Si la representación de la imagen en el dispositivo de salida no se ha completado, entonces *drawImage()* devuelve el valor *false*.

drawImage(Image,int,int,int,int,ImageObserver), pinta la imagen que se pasa dentro del rectángulo que se indica en los parámetros, escalando esa imagen si es necesario. El método retorna inmediatamente, aunque la imagen no se haya cargado completamente. Si la representación de la imagen en el dispositivo de salida no se ha completado, entonces *drawImage()* devuelve el valor false. El proceso que pinta las imágenes notifica al observador a través de su método *imageUpdate()*.

El método *imageUpdate()* pertenece a la clase *Component*. Al ser llamado, encola una llamada a repaint() permitiendo que se pinte un trozo más de imagen sin que se interfiera la tarea o hebra (thread) de control principal. La propiedad *awt.image.incrementaldraw* determina si la imagen puede ser pintada en piezas, tal como se ha indicado antes. El valor por defecto para esta propiedad es *true*, en cuyo caso el sistema va pintado trozos de imagen según van llegando. Si el valor es *false*, el sistema esperará a que la imagen esté completamente cargada antes de pintarla.

Existe una segunda propiedad, *awt.image.redrawrate* que determina el período mínimo en milisegundos, que debe haber entre llamadas a *repaint()* para imágenes. El valor por defecto es 100, y solamente se aplica cuando la propiedad anterior es *true*. Tanto esta propiedad como la anterior se pueden modificar, pero la modificación sirve solamente para la ejecución actual del programa.

El último parámetro (observer) que se pasa al método *drawImage()* se refiere al objeto que debe ser notificado cuando la imagen esté disponible. Como ya se comentó, cuando se realiza una llamada a *drawImage(),* se lanza una tarea que carga la imagen solicitada. Hay un observador que monitorea el proceso de carga. La tarea que está cargando la imagen notifica a ese observador cada vez que llegan nuevos datos. Para este parámetro se puede utilizar perfectamente *this* como observador en la llamada a *drawImage()*, es más, cualquier componente puede servir como observador para imágenes que se pintan sobre él.

6.1.2.4 La interfaz ImageProducer

Se trata de una interface para objeto que puede producir imágenes para la clase *Image*. Cada imagen contiene un *ImageProducer* que se utiliza en la reconstrucción de esa imagen cuando es necesario. Por ejemplo, cuando se escala la imagen, o cuando se cambia el tamaño de la misma. Este interfaz declara varios métodos que deben ser implementados por los objetos que las implementan. Algunos de ello se expondrán un poco más adelante.

6.1.2.5 La clase MediaTracker

Esta es una clase de utilidad general diseñada para controlar el estado de los objetos de tipo media, que en teoría pueden ser tanto clips de sonido, imagenes o cualquier otro objeto media.

Un objeto *MediaTracker* se utiliza a través de una instancia de la clase *MediaTracker* sobre el objeto *Component* que se quiere monitorear, e invocando al método *addImage()* para cada una de las imágenes que se quiere controlar. A cada una de estas imágenes se puede asignar un identificador único, o también se puede asignar el mismo identificador a un grupo de imágenes. Este identificador controla el orden de prioridad en que se cargarán

las imágenes, de forma que imágenes con un identificador bajo tienen preferencia sobre otras con identificador más alto. El identificador también se puede utilizar para identificar un conjunto único de imágenes. En otras palabras, asignando el mismo identificador a varias imágenes, se las puede manejar a la vez.

Se puede determinar el estado de una imagen (o grupo de imágenes) invocando alguno de los distintos métodos sobre el objeto *MediaTracker*, pasándole como parámetro el identificador de la imagen (o grupo de imágenes).
El objeto *MediaTracker* también se puede utilizar para bloquear la ejecución a la espera de la carga completa de una imagen.

La clase *MediaTracker* proporciona cuatro constantes, que se utilizan como valores de retorno de algunos de sus métodos, tal como su nombre sugiere, indican el estado en que se encuentra una imagen dada:

- ABORTED, la carga de la imagen (o cualquier objeto media) ha sido abortada
- COMPLETE, realizara satisfactoriamente la carga completa
- ERRORED, se ha encontrado algún tipo de error en la carga
- LOADING, se está cargando la imagen (o cualquier objeto media) .

Un objeto *MediaTracker* tiene la posibilidad de controlar el estado de algunas, o todas las imágenes que están siendo cargadas para el objeto *Component* que se ha pasado como parámetro cuando se instanció. Cuando se añade una imagen a la lista, hay que proporcionar un identificador numérico para esa imagen, que será luego el que sea utilizado por otros métodos para poder indicar el estado en que se encuentra una imagen determinada.

Los métodos que se presentan a continuación se utilizan para crear y mantener la lista de imágenes:

addImage(Image,int), incorpora una nueva imagen a la lista de imágenes que está siendo controlada por el objeto *MediaTracker*.
addImage(Image,int,int,int), incorpora una imagen escalada a la lista.
removeImage(Image), elimina la imagen indicada de la lista.
removeImage(Image,int), elimina la imagen que corresponde al identificador que se pasa como parámetro de la lista.
removeImage(Image,int,int,int), elimina la imagen que corresponde al ancho, alto e identificador, de la lista de imágenes.

El objeto *MediaTracker* se puede utilizar para hacer que la tarea, o hebra de ejecución, se bloquee hasta que una o más imágenes de su lista haya completado su carga. Esto se consigue con los siguientes métodos:

waitForAll(), inicia la carga de todas la imágenes controladas por el objeto *MediaTracker*.
waitForAll(long), inicia la carga de todas la imágenes, devolviendo *true* si todas las imágenes se han cargado y *false* en cualquier otro caso.
waitForID(int), inicia la carga de las imágenes que corresponden al identificador que se pasa.
waitForID(int,long), inicia la carga de las imágenes que corresponden al identificador que se pasa; devuelve *true* si todas las imágenes se han cargado y *false* en cualquier otro caso.

Por supuesto, es posible utilizar métodos que no bloquean la tarea que carga las imágenes para comprobar el estado de una o más imágenes de la lista. Esto permite continuar haciendo el trabajo mientras las imágenes siguen cargándose. Estos métodos devuelven *true* o *false* para indicar si la carga se ha completado. Se puede observar que hay dos versiones sobrecargadas de cada uno de los métodos. La versión con el parámetro *bolean* comenzará la carga de cualquier imagen que no se esté cargando en caso de que ese parámetro *bolean* sea *true*. La otra versión no iniciará la carga de ninguna imagen.

checkAll() y checkAll(boolean), comprueban si todas las imágenes que están siendo controladas por el objeto *MediaTracker* han finalizado la carga.
checkID(int) y *checkID(int,boolean),* comprueban si todas las imágenes que corresponden al identificador especificado han concluido su carga.

El hecho de que los métodos anteriores indiquen que la carga de las imágenes ha sido completa, no garantiza que esa carga esté libre de errores. Los siguiente métodos se utilizan para determinar si se ha producido algún problema durante la carga de las imágenes:

getErrorsAny(), devuelve una lista de todas las imágenes (o cualquier media) en los que se ha producido un error en la carga.
getErrorsAny(int), devuelve una lista de todas las imágenes correspondientes a un determinado identificador, en los que se ha producido un error en la carga.
isErrorAny(), comprueba el estado de error de todas las imágenes.
isErrorID(int), comprueba el estado de error de todas las imágenes correspondientes a un determinado identificador.

Los siguientes dos métodos devuelven un valor entero formado por la operación OR entre los valores de estado de todas las imágenes que se requieren. En el primer caso, de todas las que controla el objeto *MediaTracker*; y en el segundo, de las que corresponden al identificador que se especifica:

statusAll(boolean)
statusID(int,boolean)

6.1.2.6 Interfaces

Las tres interfaces más interesantes utilizadas en el tratamiento de imágenes digitales son *ImageProducer*, *ImageConsumer* e *ImageObserver*. A Continuación, una pequeña descripción de cada una de ellas:

Las clases que implementan la interface *ImageProducer* sirven como fuentes de píxeles. Los métodos de estas clases pueden generar los píxeles a partir de la pantalla, o puede interpretar cualquier otra fuente de datos, como un fichero GIF. No importa cómo genere los datos, el principal propósito de un productor de imágenes es proporcionar píxeles a una clase *ImageConsumer*.

Los productores de imágenes operan como fuentes de imágenes, y el modelo *productor/consumidor* que se sigue en el tratamiento de imágenes es el mismo que se utiliza en el modelo *fuente/receptor* de eventos.

Los métodos declarados en la interface *ImageProducer* hacen posible que uno o más objetos *ImageConsumer* se registren para mostrar su interés en una imagen. El productor de la imagen invocará a los métodos declarados en el interfaz *ImageConsumer* para enviar píxeles a los consumidores de imágenes.

Un productor de imágenes puede registrar muchos consumidores de sus píxeles, de la misma forma que una fuente de eventos puede registrar múltiples receptores de sus eventos.
En la interface *ImageProducer* hay varios métodos para manipular la lista de consumidores, por ejemplo para añadir nuevos consumidores interesados en los píxeles del productor, o para eliminar consumidores de la lista.

La interface *ImageConsumer* declara métodos que deben ser implementados por las clases que reciben datos desde un productor de imágenes. El principal de estos métodos es *setPíxeles()*.

void setPíxeles(int, int, int, int, ColorModel, byte[], int, int)

La interface *ImageObserver* ya ha sido comentada anteriormente.

6.1.2.7 La clase ColorModel

Dependiendo de cómo se quiera la representación de la imagen en pantalla, un solo píxel puede contener mucha información o casi ninguna. Por ejemplo, si se quieren dibujar una serie de líneas del mismo color sobre un fondo de un color diferente, la única información que debe contener cada píxel es si debe estar encendido o apagado, es decir, que solamente sería necesario un bit de información para representar al píxel.

En el otro extremo de la información que puede recoger un píxel se encuentran los gráficos complejos de los juegos de ordenador, que requieren 32-bits para representar un píxel individual. Estos 32 bits se dividen en grupos de cuatro octetos de bits, o bytes. Tres de estos grupos representan la cantidad de colores fundamentales: *rojo*, *verde* y *azul*, que forman parte del color del píxel; y el cuarto byte (normalmente conocido con byte *alpha*) se utiliza para representar la transparencia (en un rango de 0 a 255) siendo 0 la transparencia total y 255 la opacidad completa. En teoría, este formato permite la utilización de 16 millones de colores en cada píxel individual, aunque puede ser que el monitor o la tarjeta gráfica no sean capaces de invidualizar electrónicamente tal cantidad de colores.

Cada uno de los extremos se puede considerar como un *modelo de color*. En Java, un modelo de color determina cuántos colores se van a representar dentro del AWT. La clase *ColorModel* es una clase abstracta que se puede extender para poder especificar representaciones de color propias.

Un modelo de color es necesario porque hay muchos métodos que reciben un *array* de bytes y convierten esos bytes en píxeles para su presentación en pantalla, es decir, convierten los bytes de datos en píxeles visuales sobre la pantalla. Por ejemplo, con un modelo de color directo simple, cada grupo de cuatro bytes se podría interpretar como la representación de un valor del color de un píxel individual. En un modelo de color indexado simple, cada byte en el *array* se podría interpretar como un índice a una tabla de

enteros de 32 bits donde cada una de esas entradas representa el valor del color del asociado al píxel.

El AWT proporciona tres subclases de *ColorModel*: *IndexColorModel*, *ComponentColorModel* y *PackedColorModel*. Sin embargo, se puede definir cualquier subclase necesaria para una aplicación en concreto.

6.1.2.8 La clase IndexColorModel

Cuando se utilizan imágenes de alta resolución, se consume gran cantidad de memoria. Por ejemplo, una imagen de dimensiones 1024x768, contiene 786.432 píxeles individuales. Si se quieren representar estos píxeles con sus cuatro bytes en memoria, se necesitarán 3.145.728 bytes de la misma para esta imagen.

Para evitar este rápido descenso de memoria, se han incorporado varios esquemas de forma que las imágenes se puedan representar de forma razonable sin comprometer demasiado el uso de memoria. Este esquema conocido como "paleta de colores" y es el que implementa esta clase.

6.1.2.9 La clase DirectColorModel

Esta clase extiende a la clase *PackedColorModel* e implementa el formato completo de color de 32-bit, en el que cada píxel está formado por los cuatro bytes, representando el canal alpha y las cantidades de rojo, verde y azul que componen cada píxel.

6.1.2.10 La clase FilteredImageSource

Esta clase es una implementación del interfaz *ImageProducer* que toma una imagen y un objeto de tipo *ImageFilter* para generar una nueva imagen que es una versión filtrada de la imagen original. Se pueden realizar una gran variedad de operaciones de filtrado, como son el desplazamiento y sustitución de colores, la rotación de imágenes, etc.

6.1.2.11 La clase ImageFilter

Esta clase es una implementación de la interface *ImageConsumer*. Además de los métodos declarados en la interface, hay métodos para implementar un filtro *nulo* (que no realiza modificación alguna sobre los datos que se le pasan).

Hay varias subclases que extienden esta clase base para permitir la manipulación de la imagen, como son *RGBImageFilter*, *CropImageFilter*, *ReplicateScaleFilter* y *AreaAveragingScaleFilter*. Se puede subclasificar *ImageFilter* para crear un filtro propio.

6.1.2.12 La clase MemoryImageSource

Esta clase es una implementación del interfaz *ImageProducer* que utiliza un *array* de datos para generar los valores de los píxeles de una imagen. Dispone de varios constructores, cuyos parámetros son (todos o en parte) lo siguientes:

- Ancho y alto en píxeles de la imagen que va a ser creada.
- Modelo de color que se empleará en la conversión, si el constructor no requiere este parámetro, utilizará el modelo de color RGB por defecto.
- Un array de bytes o enteros conteniendo los valores a ser convertidos en píxel según el modelo de color especificado.
- Un offset para indicar el primer píxel dentro del array.
- El número de píxeles por línea en el array.
- Un objeto de tipo Hashtable conteniendo las propiedades asociadas de la imagen, en caso de que la haya.

En todos los casos, el constructor genera un objeto *ImageProducer* que se utiliza como un *array* de bytes o enteros para generar los datos de un objeto *Image*. Esta clase puede pasar múltiples imágenes a consumidores interesados en ellas, recordando a la funcionalidad *multiframe* que ofrece el formato gráfico *GIF89a* para producir animaciones.

6.1.2.13 La clase PixelGrabber

Esta es una clase de utilidad para convertir una imagen en un *array* de valores que corresponden a sus píxeles. Lo hace a través del método grabPixels().

PixelGrabber implementa el interface *ImageConsumer*, que puede ser acoplado a un objeto *Image* o *ImageProducer* para realizar cualquier manipulación de esa imagen.

6.1.2.14 La clase BufferedImage

BufferedImage deriva de la clase *Image* y permite acceder a un objeto *Image* como si fuera un buffer de datos.
Cuando un gráfico es complejo o se usa repetidamente, se puede reducir el tiempo que tarda en mostrarse renderizándolo primero en un buffer fuera de pantalla y luego copiando el buffer en ella. Esta técnica, llamada de doble buffer, se usa frecuentemente para animaciones.

Un *BufferedImage* puede usarse fácilmente como un buffer fuera de pantalla. Para crear un *BufferedImage* cuyo espacio, color, profundidad y distribución de píxeles corresponden exactamente a la ventana en la que son dibujados, se llama al método *Component.createImage*. Si necesitamos control sobre el tipo de la imagen fuera de la pantalla, podemos construir directamente un objeto *BufferedImage* y usarlo como un buffer.

Para dibujar dentro de una imagen almacenada, se llama al método *BufferedImage.createGraphics* para obtener el objeto *Graphics2D*; luego se llama a los métodos de dibujo apropiados del *Graphics2D*. Todo el API de dibujo de Java 2D puede usarse cuando se dibuja sobre un *BufferedImage* que está siendo utilizado como un buffer fuera de pantalla.
Cuando se esté listo para copiar el *BufferedImage* en la pantalla, simplemente se llamará al método *drawImage* sobre el *Graphics2D* de nuestro componente se le pasará el *BufferedImage*.

Los tipos de imágenes definidos más usuales que posee *BufferedImage* son:

- TYPE_BYTE_GRAY representa a un byte sin signo en escala de gris. No está indexado.
- TYPE_INT_ARGB representa una imagen con 8-bits de color RGBA, 8 para cada uno, empaquetado en un valor entero.
- TYPE_INT_RGB representa una imagen con 8-bits de color RGB, 8 para cada uno, empaquetado en un valor entero.
- TYPE_BYTE_INDEXED representa una imagen de bytes indexados.

Los constructores para esta clase son:

BufferedImage(ColorModel cm, WritableRaster raster, boolean isRasterPremultiplied, Hashtable properties) crea un nuevo *BufferedImage* con un objeto *ColorModel* y un objeto Raster especificado.
BufferedImage(int width, int height, int imageType) crea un nuevo BufferedImage con *TYPE* predeterminado.
BufferedImage(int width, int height, int imageType, IndexColorModel cm) crea un nuevo *BufferedImage* con *TYPE* predeterminado: *TYPE_BYTE_BINARY* ó *TYPE_BYTE_INDEXED*.

Algunos de los métodos más importantes de esta clase son:

createGraphics(), crea un objeto Graphics2D, el cual puede ser usado para dibujar en este *BufferedImage*.
flush(), libera los recursos que esta usando este objeto.
getColorModel(), retorna el obejto ColorModel que representa al mismo.
getHeight(), devuelve el alto del objeto BufferedImage
getRGB(int x, int y), devuelve el valor entero RGB del píxel especificado en las coordenadas x e y.
getRGB(int startX, int startY, int w, int h, int[] rgbArray, int offset, int scansize), devuelve un vector con los valores enteros RGB de la porción de imagen especificada.
getSource(), devuelve un objeto ImageProducer de los píxeles que forman la imagen.
getSubimage(int x, int y, int w, int h), retorna un objeto *BufferedImage* que es una subimagen especificada por los valores de las coordenadas pasadas como parámetros.
getWidth(), devuelve el ancho del objeto.
setRGB(int x, int y, int rgb), establece en la coordenada (x,y) el valor entero RGB especificado.

La información para el desarrollo de esta sección se obtuvo de [JSU006].

6.2.1 JDK 1.4.2

El Java Development Kit (JDK) es el Kit para desarrollo de aplicaciones Java. Es un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar programas Java. Los programas más importantes del JDK son:

- Interprete en tiempo de ejecución (JRE):

  Permite la ejecución de los programas Java (*.class). Su sintaxis es:

Java [Opciones] ClaseAEjecutar [Argumentos]

- Opciones: Especifica opciones relacionadas con la forma en que el intérprete Java ejecuta el programa

- ClaseAEjecutar: Especifica el nombre de la clase cuyo método main() se desea ejecutar. Si la clase reside en un paquete se deberá especificar su ruta mediante en la forma paquete.subpaquete.clase_a_ejecutar.

- Argumentos: Especifica los argumentos que se recibirán en los parámetros del método main(String[] args), si es que el programa necesita de parámetros para la ejecución. Un ejemplo podría ser un programa que realiza alguna operación sobre una imagen, probablemente se reciba por parámetro la ruta completa de la imagen a ser tratada, y una ruta para la imagen de salida.

- **Compilador:**

  Se utiliza para compilar archivos de código fuente Java (habitualmente .java), en archivos de clases Java ejecutables (*.class). Se crea un archivo ".class" para cada archivo fuente .java.

  La sintaxis desde la línea de comandos es:

  javac [Opciones] ArchivoACompilar

  - Opciones: Especifica opciones de cómo el compilador ha de crear las clases ejecutables.

  - ArchivoACompilar: Especifica la ruta del archivo fuente a compilar, normalmente un archivo con extensión ".java".

- **Visualizador de Applets (appletViewer)**

  Es una herramienta que sirve para probar applets. Los applets son aplicaciones Java que corren sobre un navegador web. De esta forma con el visualizador de applets se puede visualizar el programa como se vería en un navegador.

  Al ser activado desde una línea de comandos abre una ventana en la que muestra el contenido del applet.

  La sintaxis de activación es:

  appletViewer [Opciones] Applet

  - Opciones: Especifica cómo ejecutar el applet Java.

- Applet: Indica una URL o una ruta de disco que contiene una página HTML con un applet Java embebida.

- **Depurador**

  Es una utilidad de línea de comandos que permite depurar aplicaciones Java. No es un entorno de características visuales, pero permite encontrar y eliminar los errores de los programas Java de una manera muy exacta.

  Se activa con la sintaxis:

  jdb [Opciones]

  - Opciones: Se utiliza para especificar ajustes diferentes dentro de una sesión de depuración.

- **Desensamblador de archivo de clase:**

  Se utiliza para desensamblar un archivo de clase. Su salida por defecto, muestra los atributos y métodos públicos de la clasedesensamblada, pero con la opcion –c también desensambla los códigos de byte, mostrándolos por pantalla. Es útil cuando no se tiene el código fuente de una clase de la que se quisiera saber cómo fue codificada.

  La sintaxis es la siguiente:

  javap [Opciones] [NombreClases]

  - Opciones: Especifica la forma en la que se han de desensamblar las clases.

  - NombresClase: Especifica la ruta de las clases a desensamblar, separadas por espacios.

- **Generador de cabecera y archivo de apéndice:**

  Se utiliza para generar archivos fuentes y cabeceras C para implementar métodos Java en C (código nativo). Esto se consigue mediante la generación de una estructura C cuya distribución coincide con la de la correspondiente clase Java.

  La sintaxis es la siguiente:

  javah [Opciones] NombreClase

  - NombreClase: Nombre de la clase desde la cuál se van a generar archivos fuente C.

  - Opciones: Forma en la que se generarán los archivos fuente.

- **Generador de documentación:**

  Es una herramienta útil para la generación de documentación API directamente desde el código fuente Java. Genera páginas HTML basadas en las declaraciones y comentarios javadoc, con el formato /** comentarios */

  La documentación que genera es del mismo estilo que la documentación que se obtiene con el JDK.

  Su sintaxis es:

  javadoc Opciones NombreArchivo

  - Opciones: Opciones sobre qué documentación ha de ser generada.

  - NombreArchivo: Paquete o archivo de código fuente Java, del que generar documentación

- **Applets de demostración**

  El JDK incluye una serie de applets de demostración, con su código fuente completo.

- **Código fuente de la API**

  El código fuente de la API se instala de forma automática, cuando se descomprime la JDK, aunque permanece en formato comprimido en una archivo llamado "scr.zip" localizado en el directorio Java que se creó durante la instalación.

Para el desarrollo de esta sección se utilizó [JSU006] y [GLJ006].

6.2.2 Morena Framework

6.2.2.1 Introducción

Morena 6.3 es un sucesor directo de otro producto anterior llamado JavaTwain. El producto original se hizo público en la versión 2.0 a finales de la década del 90. Consistía de una simple clase wrapper Java para la API TWAIN sin manejo de errores y con solo un conjunto pequeño de geters y seters para las capacidades más comunes de los dispositivos de adquisición. De esta forma, introdujo la filosofía de ésta línea de productos – interface Java nativa simple, que oculta la complejidad e inconsistencia de una API de adquisición de imágenes.

El último release a la fecha es Morena 6.3. Esta versión provee una gran independencia de la API, instalación simple y documentación asociada.

6.2.2.2 TWAIN

TWAIN (Technology Without Interesting Name) es una API de captura de imagenes para sistemas operativos Microsoft Windows y Apple Macintosh, que se introdujo en 1992. Este standard es mantenido por el TWAIN Working Group, una organización que representa la industria de la imagen [TWN099] y fue creado por varias firmas comerciales dedicadas al hardware / software para procesado de imagen (HP, Aldus, Caere, Kodak, Logitech y Adobe).

La función principal de TWAIN es estandarizar la comunicación entre la aplicación y el dispositivo de captura. De este modo se hace posible construir aplicaciones independientes del modelo y fabricante del dispositivo. El hecho de que este estándar se haya generalizado para soportar también otras posibles fuentes de imagen (como cámaras o bases de datos de imágenes) abre la puerta a poder tener diferentes fuentes para la imagen de entrada.

Twain se basa en la definición de una API (Application Programs Interface) que permiten a nuestra aplicación interactuar con la fuente de imagen. Cualquier fabricante que afirme soportar el estándar TWAIN debe proporcionar los programas de bajo nivel adecuados (drivers TWAIN).

Siempre que se usa TWAIN existen tres elementos:

- La fuente de imagen (el hardware)
- El administrador de fuente (los drivers TWAIN)
- El programa de aplicación

TWAIN define más de 50 posibles operaciones del programa de aplicación sobre el administrador de fuente. Al desarrollar la aplicación se debe tener en cuenta que no todas las operaciones son posibles en cualquier momento. Es decir, el conjunto de fuente – administrador de fuente tiene un "diagrama de estados" asociado y el programa de aplicación debe saber cual es el estado actual. Se tendrá un conjunto de operaciones válidas para cada estado y habrá operaciones que provocaran el cambio del estado actual.

Los estados son:

- pre-sesión (o de reposo)
- administrador cargado en memoria (pero no abierto todavía)
- administrador abierto
- fuente cargada y abierta
- fuente activa
- fuente preparada para entregar imagen
- imagen transfiriéndose

Normalmente una aplicación seguirá el diagrama de estados en el orden anterior.

6.2.2.3 SANE

SANE (Scanner Access Now Easy) es una API de captura de imagenes para sistemas operativos Unix y Unix-like, pero también disponible para OS/2 y Microsoft Windows. Fue introducida en 1996, estando a la fecha en la versión 1.0. SANE es un proyecto open source y esta disponible bajo la GNU (General Public License).

6.2.2.4 Requerimientos técnicos

Morena es solo un Wrapper Java de la API TWAIN o SANE para captura de imágenes. Esto implica que se necesitan al menos estos tres componentes, para su correcto funcionamiento:

- Java2 VM (versión 1.3 o mayor).
- TWAIN DSM (Data Source Manager) para maquinas windows o MacOS X, o el Sane network escuchando en una máquina unix local o remota.
- Algún hardware de adquisición de imágenes con su driver correctamente instalado y configurado.

Morena funciona en cualquier máquina y / o sistema operativo con estos tres componentes disponibles. En raras ocasiones Morena puede hacer un crash de la Java VM. Esto probablemente significa, que se tiene un driver Twain instalado, el cual no es capaz de correr sin heap o stack constrains de VM. No se puede hacer mucho para solucionar esto, excepto instalar una nueva versión del driver.

6.2.2.3 Instalación

Desde la versión 6.3, no se necesita instalación. Solo hay que poner morena.jar, morena_windows.jar (or morena_osx.jar) y morena_license.jar en el CLASSPATH. Los componentes nativos están en morena_windows.jar (o morena_osx.jar) y son instaladados automáticamente en la carpeta .morena, del directorio raiz, de ser necesario.

6.2.2.4 Adquisición

Para la adquisición de una imagen con Morena lo primero que se debe hacer es seleccionar la fuente, que puede ser un escáner, cámara instalado en el sistema. La siguiente línea de código muestra como seleccionar la fuente:

MorenaSource source = Morena.selectSource(null);

El método *selectSource()* de la clase Morena despliega en pantalla una secuencia de cajas de diálogo para que el usuario seleccione si se va a usar la API TWAIN o SANE, para ingresar el nombre del host donde está corriendo el demonio SANE y para seleccionar uno de los dispositivos instalados. El valor de retorno es una instancia de la clase *MorenaSource* que representa un dispositivo en particular. La característica más importante es que que *MorenaSource* implementa la inteface *ImageProducer*, que es la API nativa Java para imágenes.

MorenaSource provee varios métodos para utilizar las capacidades del dispositivo del adquisición. Los más importantes son:

setVisible(boolean), si el parámetro es true despliega un cliente de escaneo.
setColorMode(), selecciona modo de escaneo color.
setGrayScaleMode(), selecciona modo de escaneo en escala de grises.
setResolution(double), permite setear la resolución de escaneado.
setBrightness(double), permite setear el brillo para el escaneado.
setContrast(double), permite setear el contraste para el escaneado.

Una vez seleccionada la fuente se puede realizar la adquisición de la imagen. Para ello se utiliza la clase *MorenaImage* de la siguiente manera:

```
MorenaImage image = new MorenaImage(source);
```

La clase *MorenaImage* funciona como un buffer en memoria para la imagen adquirida. Esta clase implementa las interfaces *ImageConsumer* e *ImageProducer*.

Las operaciones de Morena pueden disparar la excepción MorenaException.

A continuación, un ejemplo de uso:

```
import SK.gnome.morena.*;

public class EjemploMorena
{ public static void main(String[] args) throws MorenaException
  { MorenaSource source=Morena.selectSource(null);
    System.err.println("Selected source is "+source);
    if (source!=null)
    { MorenaImage image=new MorenaImage(source);
      System.err.println("Size of acquired image is "
            +image.getWidth()+" x "
            +image.getHeight()+" x "
            +image.getPixelSize());
    }
  }
}
```

Para la realización de ésta sección se utilizó la referencia [MOR005]

6.2.3 Entorno de desarrollo Eclipse

Eclipse es un IDE (Entorno de desarrollo integrado) para desarrollo de aplicaciones Java. Este es un producto de la compañía IBM que puede ser descargado de internet libremente de www.eclipse.org . Es parte de un proyecto de software libre que lleva el mismo nombre. [ECL006].

Una característica interesante es que está basado en plugins. Un plugin es un conjunto de archivos (entre ellos clases java) que le agregan funcionalidad. Un ejemplo de estos plugins es Jigloo que permite desarrollar interfaces de usuario de manera visual.

Eclipse esta desarrollado en Java y por lo tanto corre sobre una JVM (Java Virtual Machine).

Se pueden crear clases, compilarlas y luego hacer un debug de las mismas, todo desde el mismo entorno.

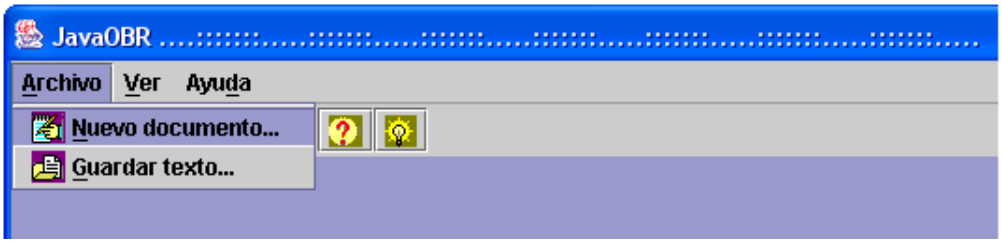
6.3 Arquitectura de la aplicación desarrollada

En esta sección se hablará sobre los aspectos de implementación de la herramienta desarrollada para realizar el reconocimiento automático de texto Braille, llamada JavaOBR. Como su nombre lo indica, está desarrollada en Java.

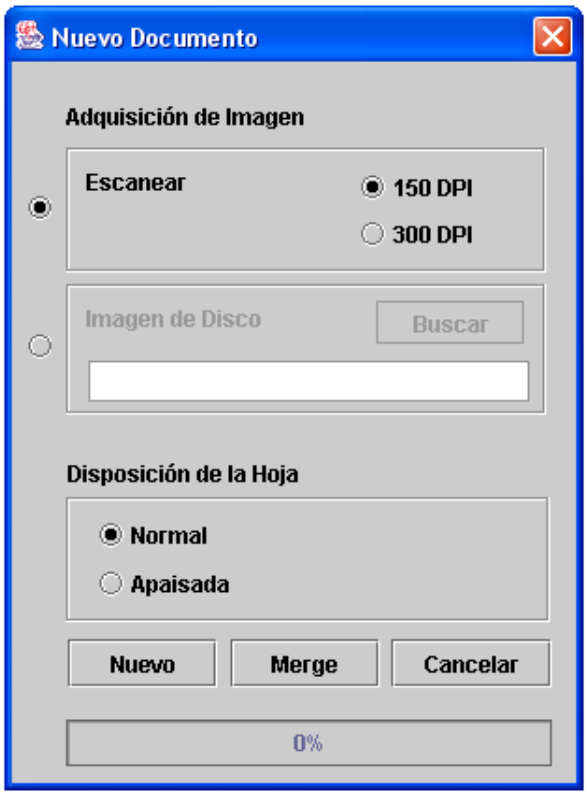
Se utilizaron varias clases del paquete java.awt. Para el escaneo de las imágenes se utilizo el Framework Morena. Para el desarrollo de las interfaces de usuario se utilizo tecnología Java Swing.

6.3.1 Funcionalidad

Se empezará describiendo el menú de operaciones de la herramienta:



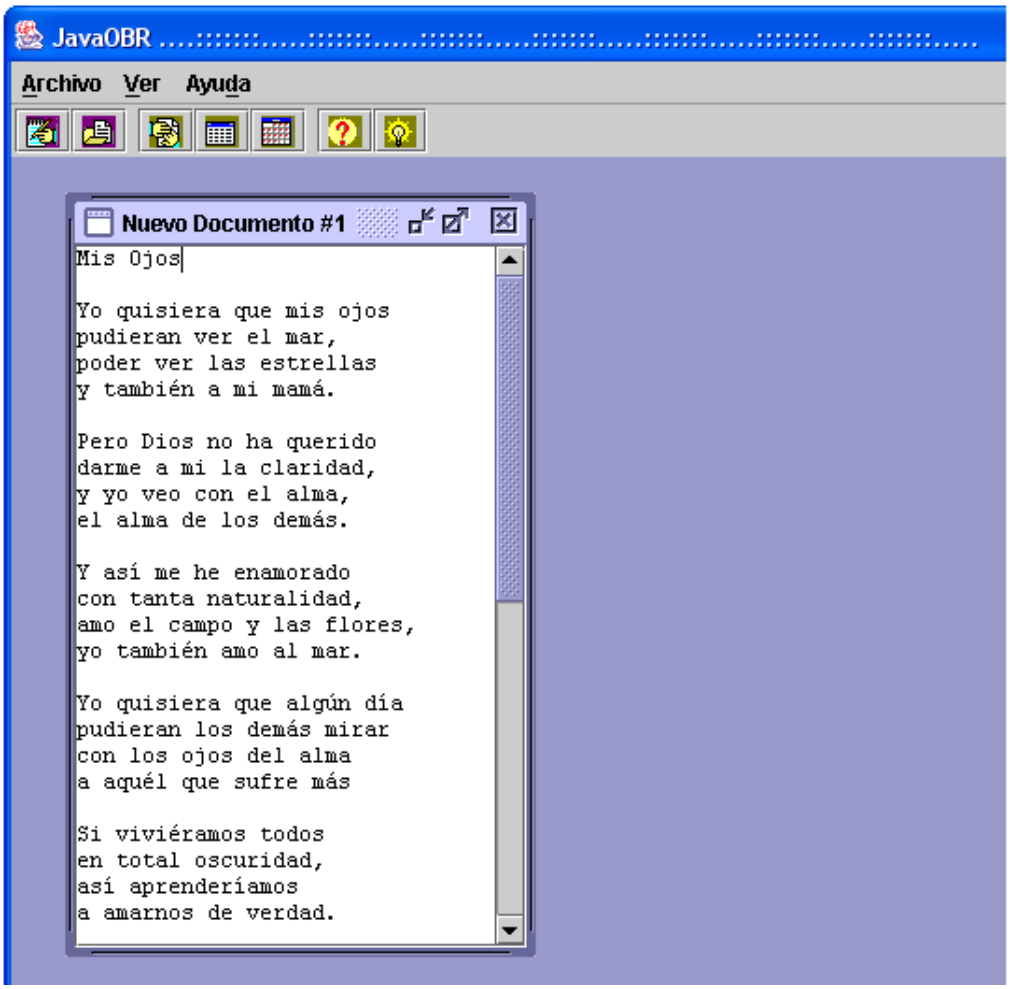
Para traducir un nuevo documento se debe elegir la opción “Nuevo documento” en el menú Archivo. Inmediatamente se despliega la siguiente pantalla:



Se pueden seleccionar dos modos de adquisición. El modo 'Escaneo', que permite escanear a resoluciones de 150 y 300 dpi; y el modo 'Imagen de Disco', que permite seleccionar una imagen de disco previamente escaneada a 150 dpi.

Disposición de la hoja sobre el escáner: puede ser 'Normal' o 'Apaisada'.Una vez elegidas las opciones de trabajo anteriores se procede a realizar el reconocimiento y traducción.

Si se quiere procesar un nuevo documento, se utiliza el comando 'Nuevo'. Una vez realizado el reconocimiento se despliega en una nueva ventana el texto traducido**.**



Para el caso de hojas de tamaño mayor que el escáner, por ejemplo supongamos un escáner A4 y hojas de tamaño mayor a A4 se escanea un trozo de la hoja con el comando 'Nuevo' eligiendo en disposición 'Apaisada' y luego se escanea el otro trozo (con la misma disposición de la hoja) con el comando 'Merge'. De esta forma se realiza el merge de los dos trozos luego del segundo escaneo.

6.3.2 Modelo de clases

Se realizó un modelo orientado a objetos. Se utilizaron varios patrones de diseño, entre ellos:

- Command: para modelar las etapas del reconocimiento automático.
- Model View Controller: para separar las vistas, del modelo de la aplicación.
- Singleton: para tener disponible una sola instancia de algunas clases particulares.

A continuación una breve descripción de las clases más importantes del modelo:

- BrailleOCR

Esta clase encapsula el comportamiento del OCR Braille. Los métodos más importantes son escanearYProcesar() y procesar(). El primero escanea la imagen y le pasa el resultado a procesar(). El segundo es el encargado de realizar el reconocimiento y traducción a texto digital.

El método procesar() utiliza como colaboradores a objetos que resuelven las diferentes etapas del reconocimiento automático. A saber: UmbralizarFilter, BuscadorDeManchas, BuscadorDePuntos, BuscadorDePuntoInicial, FactoryMalla, BuscadorDeInclinacion, BrailleTransformer6P, TextMerger.

- BrailleOCRThread

Esta clase hereda el comportamiento de un thread (hilo de ejecución). Este thread al ejecutarse instancia un objeto BrailleOCR y llama al método escanearYProcesar(). Al ejecutarse este proceso en un hilo de ejecución diferente, se permite interacción con la interfaz de usuario para por ejemplo cancelar el mismo.

- UmbralizarFilter

La responsabilidad de esta clase es umbralizar la imagen que se le pasa como parámetro.
El método aplicarDobleUmbral() de esta clase trabaja sobre los píxeles de la imagen aplicando los umbrales que le son pasados como parámetro. Para obtener los umbrales óptimos utiliza la clase Histograma**.**

- BuscadorDeManchas

Su responsabilidad es la de buscar las manchas blancas y negras resultado de la etapa de umbralización. Se llama dos veces al método buscarManchas(), una vez para buscar las blancas y otra para las negras. Cada llamada devuelve una colección de objetos ManchaBlanca o ManchaNegra según el caso.

- BuscadorDePuntos

Se encarga agrupar manchas blancas y negras a fin de formar los puntos Braille de la imagen. El metodo buscarPuntos() de esta clase recibe como parámetro dos colecciones, una de manchas blancas y otra de manchas negras. Retorna una colección con los puntos Braille encontrados.

- BuscadorDePuntoInicial

Esta clase tiene la responsabilidad de buscar la posición del primer punto de la hoja. Este es el punto a partir del cual se generará la malla Braille luego (ver 5.4.9). El método buscar() de esta clase devuelve este punto.

- BuscadorDeInclinacion

Esta clase se utiliza para detectar el ángulo de inclinación de la hoja sobre el escáner. Para ello se utiliza el metodo buscarInclinación() de la misma. Este método utiliza otros

métodos privados a la clase, a saber, calcularVarianza(), getZonaMasPoblada(), sumarPixelesNYB(), para detectar la inclinación.

- CaracterBraille

Esta clase representa un carácter Braille. Tiene variables de instancia que representan las coordenadas de cada uno de los puntos Braille que lo forman. Un método muy importante es el adaptate(). Este se utiliza al momento de generar la malla Braille para adaptar el caracter generado de modo que haga matching con los puntos Braille encontrados.

Otro método importante es el convertir(), que se utiliza para corregir los puntos del caracter según la inclinación detectada.

- Histograma

Esta clase encapsula la lógica para poder trabajar con un histograma de imagen. El método getValorCotaInferiorA() recibe como parámetro un valor que indica el porcentaje que representa la cola izquierda del histograma y devuelve el nivel de gris para el que se obtiene dicho porcentaje. El método getValorCotaSuperiorA() recibe como parámetro un valor que indica el porcentaje que representa la cola derecha del histograma y devuelve el nivel de gris (punto de inicio de la cola) para el que se obtiene dicho porcentaje.

- FactoryMalla

Con esta clase se crea la malla de posibles caracteres Braille a partir de un punto de inicio. El punto de inicio es la coordenada del punto 1 del primer carácter de la hoja.

La malla se construye con el método construirMalla(). Este método devuelve un objeto de la clase Malla. La clase Malla encapsula una matriz de objetos CaracterBraille.

- Mancha

Un objeto de ésta clase representa a una mancha de las obtenidas al realizar el proceso de umbralado a la imagen. Tiene dos subclases: ManchaBlanca y ManchaNegra.

- BrailleTransformer6P

Esta clase es la encargada de realizar el reconocimiento para saber si un punto braille está activo o no (ver 5.4.11). Esto lo hace a través del método activo(). El método calcularNum() recibe como parámetro la información del estado (activo - no activo) de cada uno de los 6 puntos que conforman un carácter braille y devuelve un número entre 0 y 255 representando un carácter ASCII.

- TextMerger

Esta clase tiene la responsabilidad de realizar el merge de texto cuando se trabaja con tamaños de hoja más grandes que el tamaño del escáner. El método merge() recibe como

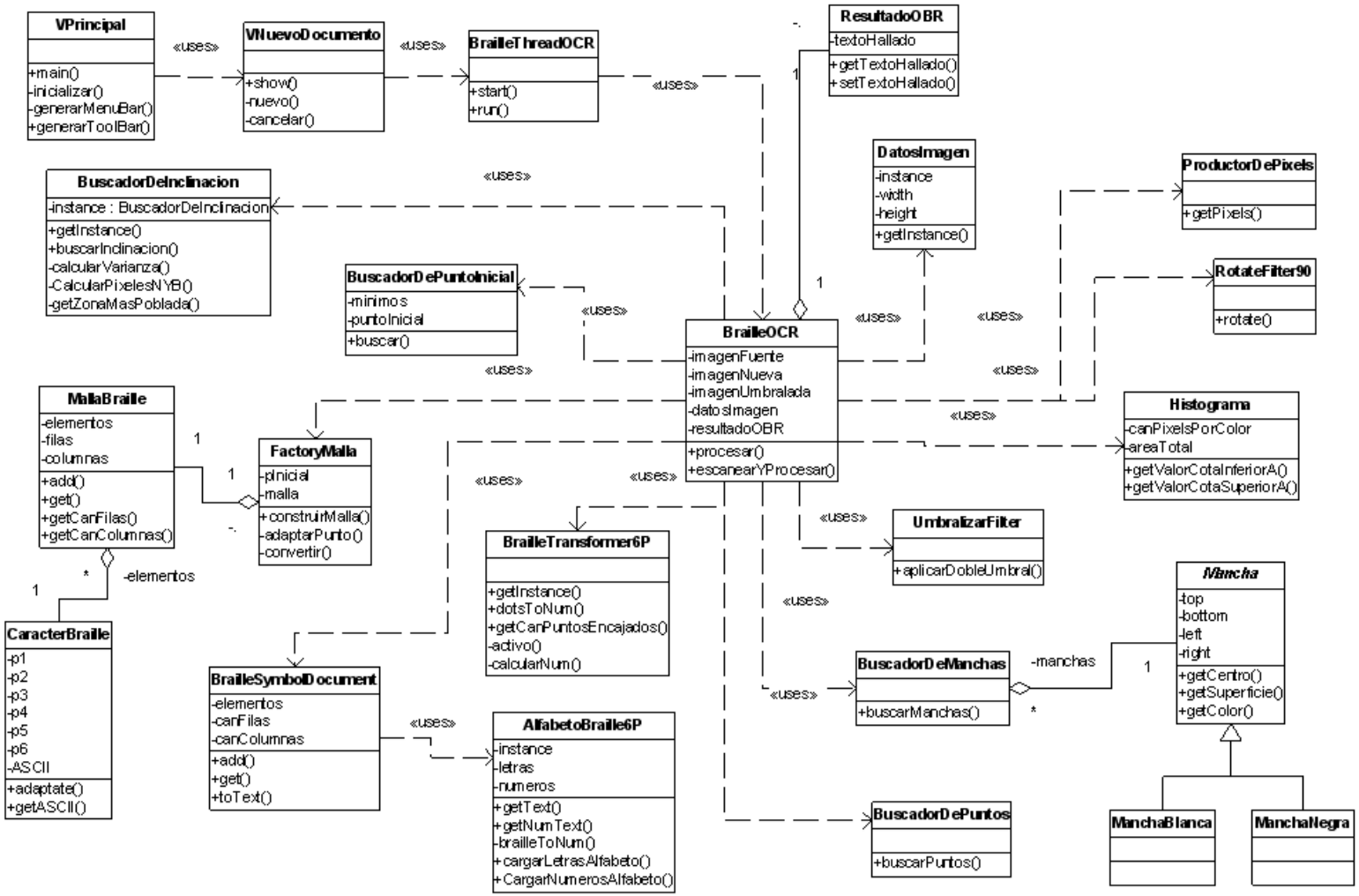
parámetro dos Strings y devuelve el String resultado del merge de los dos anteriores, eliminando la duplicación de las zonas donde hay overlaping de texto (ver 5.5.3).

- Vprincipal

Esta clase es la ventana principal de la aplicación. Está desarrollada utilizando componentes Swing. Contiene el metodo estático main() para poder correr la aplicación. Esta clase es una subclase de JFrame. Contiene una instancia de la clase JDesktopPane para poder trabajar con frames internos a la ventana principal. Estos frames se usaran para mostrar el texto traducido.

- VNuevoDocumento

Esta clase es una subclase de JDialog. Contiene varios JButtons uno de los cuales permite realizar el reconocimiento instanciando la clase BrailleOCRThread.


VPrincipal
+main()
-inicializar()
-generarMenuBar()
+generarToolBar()
«uses»
VNuevoDocumento
+show()
-nuevo()
-cancelar()
«uses»
BrailleThreadOCR
+start()
+run()
«uses»
ResultadoOBR
-textoHallado
+getTextoHallado()
+setTextoHallado()
BuscadorDeInclinacion
-instance : BuscadorDeInclinacion
+getInstance()
+buscarInclinacion()
-calcularVarianza()
-CalcularPixelesNYB()
-getZonaMasPoblada()
«uses»
DatosImagen
-instance
-width
-height
+getInstance()
ProductorDePixels
+getPixels()
BuscadorDePuntoInicial
-minimos
-puntoInicial
+buscar()
«uses»
BrailleOCR
-imagenFuente
-imagenNueva
-imagenUmbralada
-datosImagen
-resultadoOBR
+procesar()
+escanearYProcesar()
«uses»
RotateFilter90
+rotate()
«uses»
MallaBraille
-elementos
-filas
-columnas
+add()
+get()
+getCanFilas()
+getCanColumnas()
FactoryMalla
-plnicial
-malla
+construirMalla()
-adaptarPunto()
-convertir()
Histograma
-canPixelsPorColor
areaTotal
+getValorCotaInferiorA()
+getValorCotaSuperiorA()
«uses»
BrailleTransformer6P
+getInstance()
+dotsToNum()
+getCanPuntosEncajados()
-activo()
-calcularNum()
«uses»
UmbralizarFilter
+aplicarDobleUmbral()
«uses»
CaracterBraille
-p1
-p2
-p3
-p4
-p5
-p6
-ASCII
+adaptate()
+getASCII()
BrailleSymbolDocument
-elementos
-canFilas
-canColumnas
+add()
+get()
+toText()
«uses»
AlfabetoBraille6P
-instance
-letras
-numeros
+getText()
+getNumText()
-brailleToNum()
+cargarLetrasAlfabeto()
+CargarNumerosAlfabeto()
«uses»
BuscadorDeManchas
+buscarManchas()
-manchas
Mancha
-top
-bottom
-left
-right
+getCentro()
+getSuperficie()
+getColor()
BuscadorDePuntos
+buscarPuntos()
ManchaBlanca
ManchaNegra

6.4 Instalación de la aplicación

6.4.1 Requerimientos

6.4.1.1 Hardware

- Pentium III 450 o superior.
- 128 MB de memoria RAM.
- 50 MB de disco rígido libre.
- Escáner Twain / Sane compatible.

6.4.1.2 Software

- Sistemas Operativos que soporten Java (win 98, 2000, XP, Linux).
- Versión de la JVM (maquina virtual de Java) 1.4.2 o superior, para el sistema operativo que se este usando, instalada correctamente. La JVM se puede descargar gratuitamente de www.java.com.
- Driver Twain correctamente instalado o demonio Sane corriendo en la maquina o red local.. Estos drivers vienen en los CD's de intalación del dispositivo de escaneo a usar o se pueden descargar de las webs de los mismos.

6.4.2 Procedimiento

La instalación consiste simplemente en copiar el directorio JOBR en algún directorio de su disco. Si no se tiene instalada una JVM compatible (ver 6.4.1.2) deberá instalarla para el correcto funcionamiento de la herramienta.

Capítulo 7 – Conclusiones

7.1 Pruebas y Resultados

A continuación se presentan los casos de prueba de la herramienta desarrollada. Para tal fin se utilizaron 10 hojas tamaño A4 las cuales en promedio tenían 4900 puntos (aproximadamente 800 caracteres). Estas hojas fueron otorgadas gentilmente por la "Editora Nacional Braille Julián Vaquero".

La pruebas consistieron en escanear y realizar el reconocimiento con la herramienta para ver cuantos caracteres reconocía correctamente y cuales eran los tiempos de procesamiento.

Se utilizó un scanner Hewlett-Packard HP 3800 y una computadora con un procesador AMD Sempron 2800+ y memoria de 512 MB.

Los resultados fueron los siguientes:

Estas pruebas arrojaron que el tiempo promedio de proceso fue de 7 segundos para hojas escaneadas a 150 DPI y 12 segundos para hojas escaneadas a 300 DPI.

El porcentaje de caracteres reconocidos correctamente fue del 99% para ambos casos.

Se presenta a continuación los gráficos correspondientes al escaneo y procesamiento de una de estas hojas con la herramienta desarrollada:

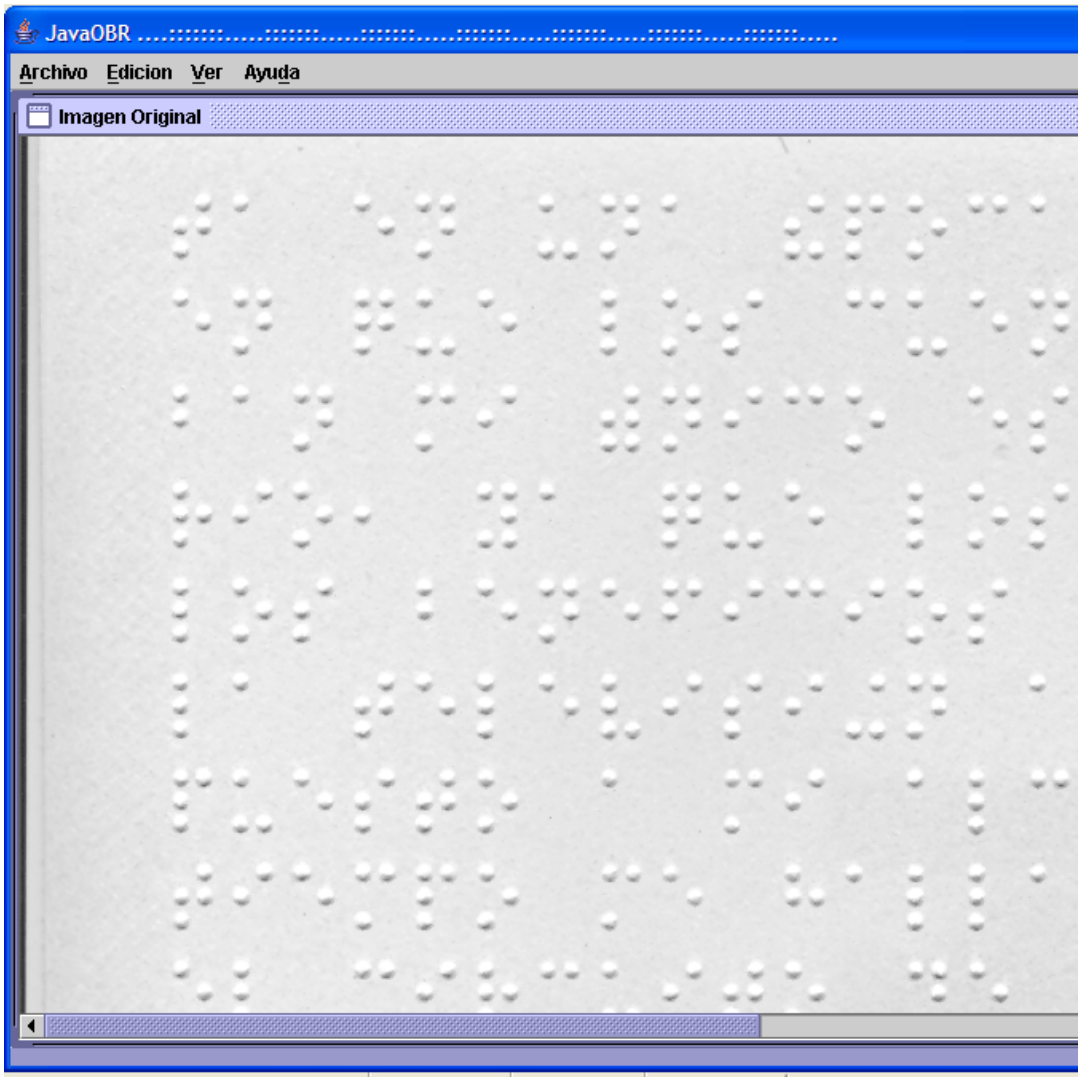


Imagen de la hoja Braille escaneada con la Herramienta

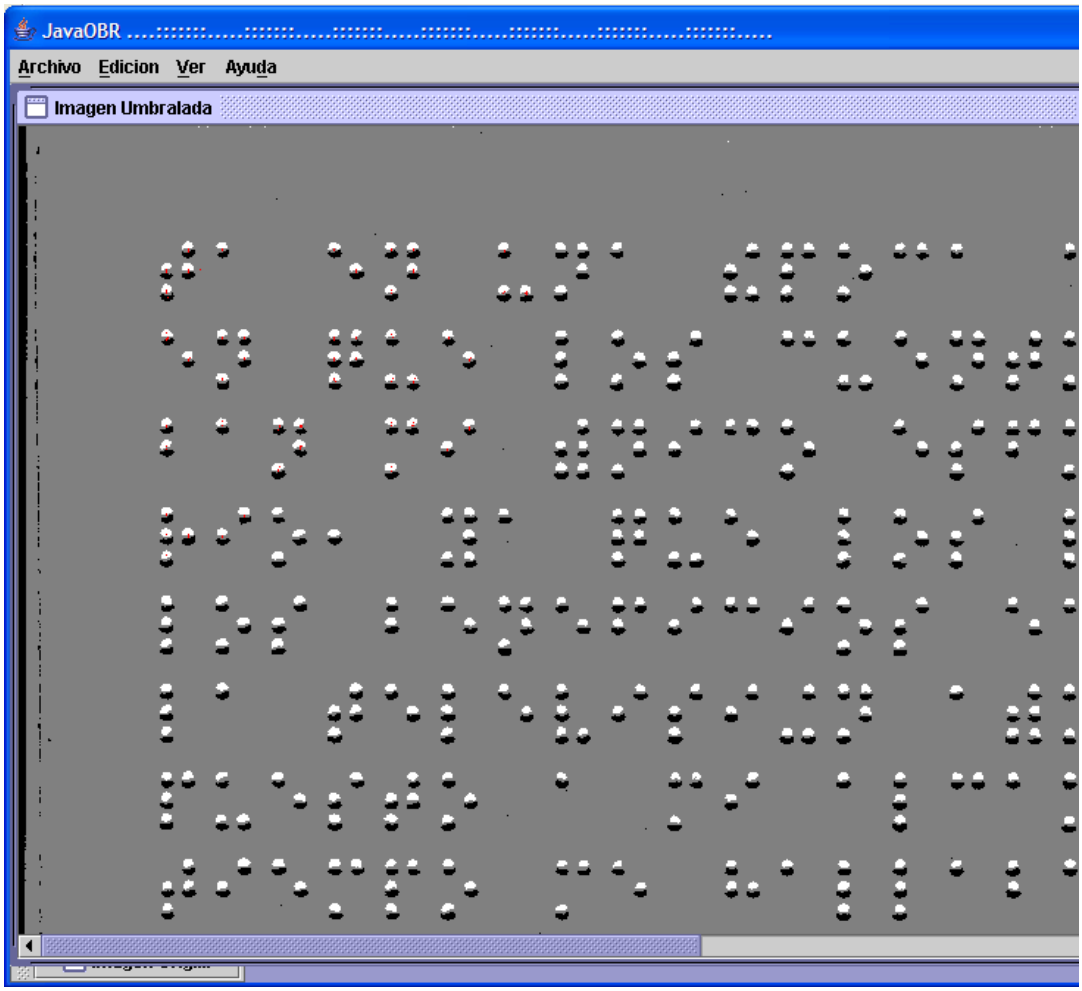


Imagen de la Hoja Braille después de aplicar los filtros para obtener las manchas blancas y negras

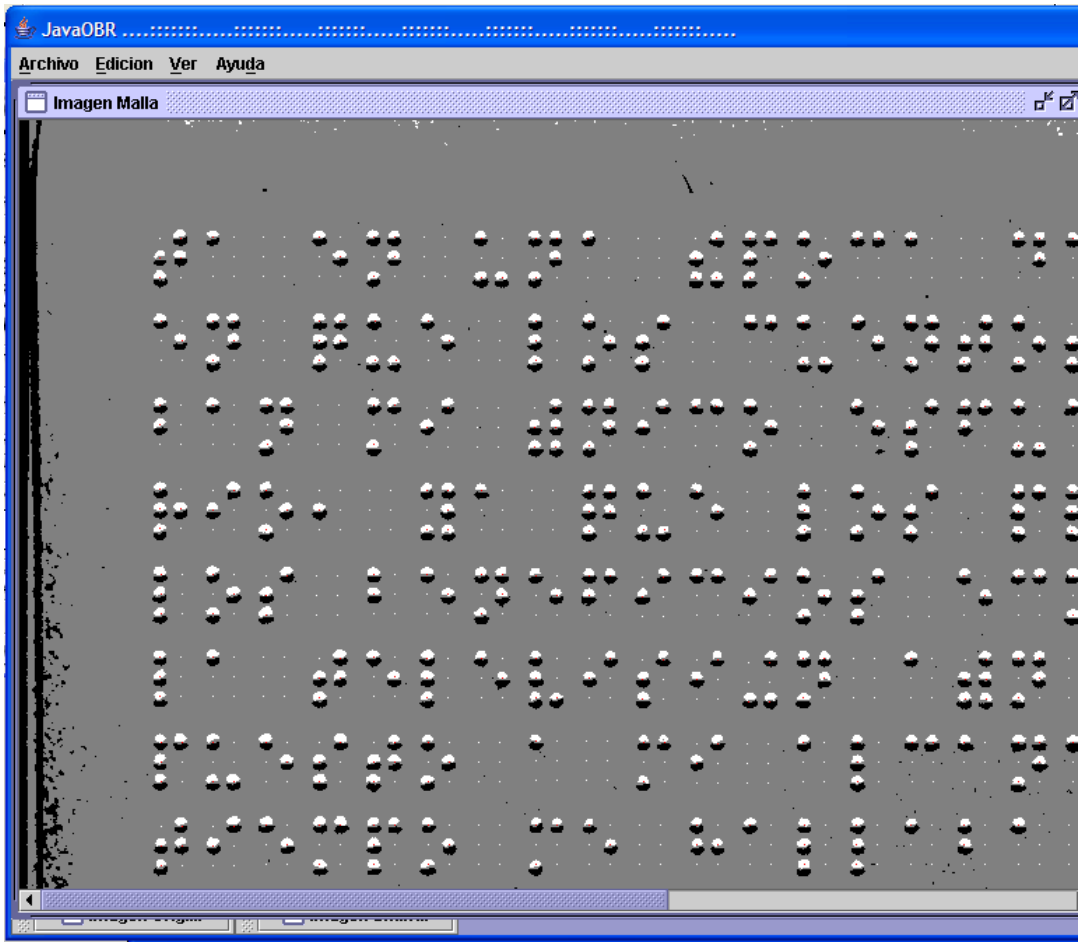
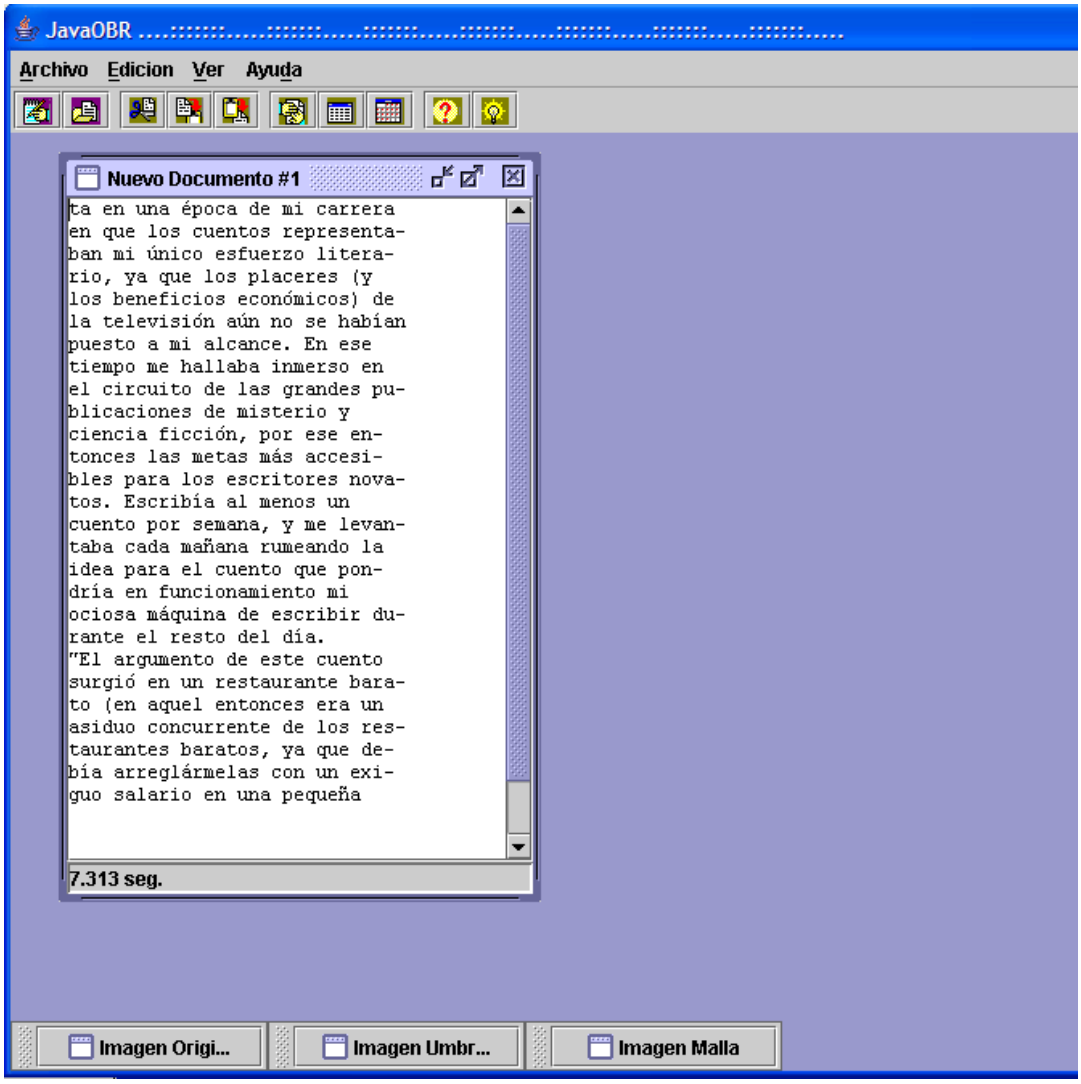


Imagen de Hoja Braille luego de construir la malla. La malla esta compuesta por los puntitos blancos



Traducción en texto digital finalizado el proceso de reconocimiento

7.2 Conclusiones generales

Se logró desarrollar una herramienta que permite reconocer los caracteres formados por los puntos de una hoja Braille y realizar su traducción obteniendo el texto digital correspondiente, a partir de la imagen obtenida por un escáner convencional.

La misma está totalmente desarrollada en Java y su distribución y uso es libre siendo, según lo investigado, un desarrollo único en Argentina y Latinoamérica.

Trabaja con hojas simple faz, corrigiendo la inclinación de la misma si fuera necesario. Este giro está limitado a +- 5 grados. Si se detecta que la inclinación es mayor se pedirá al usuario que corrija la posición y vuelva a realizar la operación.

La segmentación de las hojas escritas a doble faz no se desarrolló por ser extremadamente compleja y se la propone como línea futura de investigación.

El tiempo de proceso de traducción por hoja es de alrededor de 7 segundos, dependiendo de la cantidad de puntos en la misma. El porcentaje de aciertos, según los casos de prueba realizados, es del 99%.

En el desarrollo de esta tesis se ha tenido la posibilidad de hablar con personas videntes y no videntes de institutos como la "Biblioteca Braille" de la ciudad de La Plata, la fundación T.I.F.L.O.S o la "Editora Nacional Braille y Libro Parlante", expresando ellos entusiasmo y contento por el desarrollo de una herramienta de esta clase.

7.2 Líneas futuras

- Procesamiento de hojas doble faz: La escritura Braille puede realizarse a doble faz. Este tipo de escritura se llama Braille interpunto. La segmentación de imágenes de documentos Braille de este tipo es compleja y costosa.

- Caracteres Braille de 8 Puntos: Existe el llamado Braille Informático en donde los caracteres Braille están formados por dos columnas de cuatro puntos cada una en vez de dos columnas de tres puntos como el Braille tradicional.

- Detección Automática de tamaño y distancia entre puntos: Agregar algoritmos y procesos de detección automática de este tipo de parámetros podría ser de gran utilidad para los usuarios de esta herramienta.

Apéndice I – Escáneres

Introducción

Los escáneres son periféricos diseñados para registrar caracteres escritos, o gráficos en forma de fotografías o dibujos, impresos en una hoja de papel facilitando su introducción en la computadora convirtiéndolos en información binaria comprensible para ésta. Existen escáner monocromos (blanco y negro) y color.

Estos periféricos generalmente se conectan al puerto paralelo o a un puerto USB de la computadora. Otra opción es mediante una conexión SCSI.

Funcionamiento

El principio de funcionamiento de un escáner es la digitalización, es decir, la conversión de una información analógica a datos que pueden ser procesados y almacenados en una computadora; para ello, se vale de una serie de componentes internos que posibilitan este objetivo.

Una fuente de luz va iluminando línea por línea la imagen o documento en cuestión, y la luz reflejada es recogida por los elementos que componen el CCD (Charged-Couple Device). Este dispositivo convierte la luz recibida en información analógica. Por último, un DAC (Digital-Analog Converter) convierte los datos analógicos en valores digitales.

En los escáneres blanco y negro la inclinación del haz de luz que ilumina la imagen la elige el fabricante, pudiendo ser perpendicular a la misma. En los escáneres color se tienen tres haces (uno para cada color del RGB) y como los tres no pueden ser perpendiculares por que deben encontrarse en un punto, al menos uno de ellos está inclinado (no es perpendicular).

Resolución

Cuando se habla de una resolución óptica de 600 ppp (puntos por pulgada), estamos indicando que el dispositivo CCD posee 600 elementos. Cuanta mayor sea la resolución, más calidad tendrá el resultado. En la actualidad, lo mínimo son 300 ppp, aunque 600 ppp es una resolución más conveniente si vamos a digitalizar fotografías. No obstante, la mayoría de escáneres pueden alcanzar mayor resolución, mediante la interpolación; se trata de un algoritmo por el cual el escáner calcula el valor situado entre dos píxeles digitalizados, a partir del valor de estos. Por ello, hay que saber diferenciar entre la resolución óptica (real) y la interpolada.

Profundidad de color

Este parámetro se expresa en bits e indica el número de tonalidades de color que un píxel puede adoptar. Lo normal en la actualidad es un valor de 24 bits. Aunque hasta hace poco los escáneres de blanco y negro, tonos de grises o 256 colores eran muy populares, es verdad que los 24 bits de color se han convertido en un estándar. Esto es lógico si se tiene en cuenta que en la actualidad cualquier placa de video es capaz de mostrar esta cantidad de colores.

Sin embargo, hay escáneres capaces de utilizar 30 o incluso 36 bits de color, pero la mayoría lo hacen a nivel interno, para disminuir el intervalo entre una tonalidad y la siguiente; posteriormente, lo que envían al PC son únicamente 24 bits. Por otro lado, muy pocos programas pueden gestionar esos bits adicionales de color.

Tipos de escáneres

Existen cinco tipos de escáneres, pero no todos son ideales para la digitalización de imágenes:

De sobremesa o planos

Un escáner plano es el tipo más versátil. Es ideal para escanear páginas de un libro sin tener que desprenderlas. Generalmente lucen como fotocopiadoras pequeñas ideales para un escritorio, y se utilizan para los objetos planos. Sus precios pueden variar de acuerdo con la resolución de escaneado, pero salvo que se utilicen para realizar presentaciones muy importantes, no se necesita adquirir uno de un costo tan alto.

De mano

Son los llamados escáneres "portátiles", son los menos costosos, con todo lo bueno y lo malo que implica esto. Hasta hace unos pocos años eran los únicos modelos con precios asequibles para el usuario medio, ya que los de sobremesa eran extremadamente caros. Esta situación a cambiado tanto que en la actualidad los escáneres de mano están casi inutilizados por las limitaciones que presentan en cuanto a tamaño del original a escanear (generalmente puede ser tan largo como se quiera, pero de poco más de 10 cm de ancho máximo) y a su baja velocidad. Otro problema es la carencia de color en los modelos más económicos.

Casi todos ellos carecen de motor para arrastrar la hoja. El usuario el que debe pasar el escáner sobre la superficie a escanear. Todo esto es muy engorroso, pero resulta ideal para copiar imágenes pequeñas como firmas, logotipos y fotografías, además es eficaz para escanear rápidamente fotos de libros encuadernados, artículos periodísticos, facturas y toda clase de pequeñas imágenes.

De rodillo

Unos modelos de aparición relativamente moderna, se basan en un sistema muy similar al de los aparatos de fax: un rodillo de goma motorizado arrastra a la hoja, haciéndola pasar por una rendija donde está situado el elemento capturador de imagen.

Este sistema implica que los originales sean hojas sueltas, lo que limita mucho su uso al no poder escanear libros encuadernados sin realizar antes una fotocopia (o arrancar las páginas), salvo en modelos muy particulares que permite separar el cabezal de lectura y usarlo como si fuera un escáner de mano.

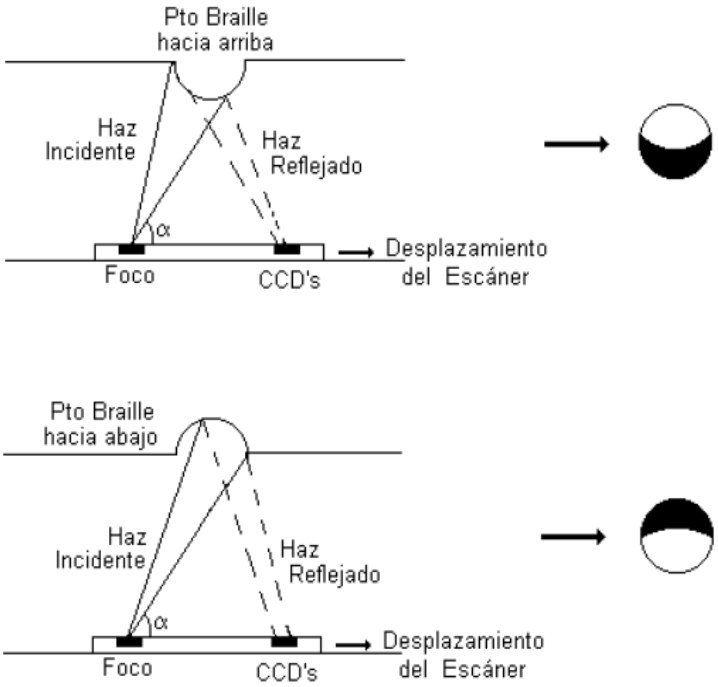
A favor tienen el hecho de ocupar muy poco espacio, incluso existen modelos que se integran en la parte superior del teclado. En contra se tiene que su resolución rara vez supera los 400x800 dpi (píxeles por pulgada), aunque esto es más que suficiente para el tipo de trabajo con hojas sueltas al que van dirigidos.

Escáneres para transparencias

Poseen una resolución mejor que los anteriores y por eso también son un poco más caros. Pueden digitalizar transparencias desarrollando un trabajo de muy buena calidad. Estos tampoco son tan utilizados como los planos, pero en aquellas lugares en donde utilizan el formato de diapositiva y transparencia para sus impresiones, son una herramienta realmente indispensable.

Escaneo de Texto Braille

En las siguientes figuras se puede observar como los haces de luz de un escáner inciden sobre una hoja braille al deslizarse el aparato adquisidor.



Puede notarse que el haz presenta una inclinación, es decir no es perpendicular a la hoja. Esto es lo que permite que las hojas Braille puedan ser escaneadas. Al pasar la luz del escáner sobre los puntos genera un patrón como el que se observa en las figuras. La inclinación del haz la elige el fabricante del escáner. Los escáneres color tienen tres haces de luz, uno para cada color. Por lo tanto, al menos uno de los haces no puede ser

perpendicular por que los 3 haces deben hacer foco en el mismo punto, pero no pueden partir del mismo punto. Aunque se escanee en escala de grises en un escáner color se garantiza que se podrá escanear documentos Braille.

El resultado de escanear una hoja Braille se describe mejor en el Capítulo VI de esta tesis.

Para la realización de este apéndice se utilizó [FOT006], [HAR006], [UVO097]

Apéndice II – Formatos de Imágenes

Introducción

Una imagen puede almacenarse en un archivo siguiendo diferentes formatos. Unos son más sencillos que otros, muchos utilizan compresión de datos, y cada uno tiene sus ventajas y sus desventajas, pero todos ellos tienen algunas características en común:

Siempre se utiliza una cabecera en el archivo que identifica el tipo de archivo del que se trata, y contiene información necesaria para interpretar el mismo, como el tamaño de la imagen o el número de colores.

A continuación se encuentran, generalmente comprimidos con un algoritmo específico de ese formato, los datos de la imagen. Una vez descomprimidos, estos datos indican el color de cada píxel de la imagen.

La imagen puede tener más o menos colores, y en función del número de colores serán necesarios más o menos bits por píxel para indicar el color del que se trata. Cuantos más colores, mejor calidad tendrá la imagen, pero más ocupará el archivo.

Normalmente el número de colores es 16, 256 ó 16 millones, lo que requiere 4, 8 ó 24 bits por píxel. En el caso de utilizar 16 ó 256 colores, debe especificarse a que color real corresponde cada uno de esos colores, es decir, que cantidades de Rojo, Verde y Azul serán utilizadas para representar el color en la pantalla.

La tabla que asocia a cada color las correspondientes cantidades de Rojo, Verde y Azul se llama paleta de colores. Puede ser modificada en función de la imagen, por lo que es necesario guardarla en el archivo. En el caso particular de utilizar 16 millones de colores, no se utiliza paleta de colores, pues la relación entre número de color y cantidad de Rojo, Verde y Azul es implícita: De los 24 bits por píxel, 8 especifican la cantidad de Rojo, otros 8 la cantidad de Verde, y otros 8 la de Azul.

Entre los formatos de imágenes más usados se encuentran BMP, JPG, GIF, PNG y TIF. A continuación se expone una breve descripción de cada uno de ellos.

BMP

Este formato de imágenes es propietario de la compañía Microsoft. No tiene mucho de bueno porque no es reconocido en plataformas diferentes a la PC que trabajan con Windows. Los modos de trabajo son en 16, 256 y 16 millones de colores. Consiste de una cabecera y a continuación el valor de cada píxel.

Ventajas:

- Son imágenes de alta calidad y se leen muy rápidamente.
- Es un formato bastante sencillo.
- Es reconocido en cualquier computadora que trabaje sobre Windows.

Desventajas:

- No se puede comprimir de ninguna manera posible o forma conocida así que hay que conseguir pasarlo a otra extensión.

- Solo es reconocido en plataformas Windows y además los archivos son bastante más grandes que en otros formatos.

JPG

Este formato reduce una imagen casi una décima parte (o mas), pero pagando el precio de perdida de calidad de la imagen. Y esto no queda aquí sino que si hacemos modificaciones a la imagen y la seguimos guardando en el mismo formato llegara a un punto en que la misma ya no sirva para nada. Al momento de guardar la imagen se puede seleccionar la calidad (nivel de compresión).

Ventajas:

- Convierte la imagen mas compleja en un archivo muy pequeño, permitiendo aprovechar al máximo el espacio en disco.

Desventajas:

- Utiliza un método de compresión destructiva que daña la calidad de la imagen.
- No permite definir colores como transparentes.

GIF

Es formato GIF (Graphic Interchange Format) es propietario de la compañía Compuserve. Las dos maneras de utilizarlo con una sola imagen o con mas de una han hecho que sea le formato favorito de los usuarios de Internet muchas de las animaciones que vemos en la red son pequeños archivos GIF.

Ventajas:

- Ahorra mucho espacio en disco para la calidad que puede ofrecer.
- El algoritmo que tiene no daña las imágenes.
- Puede contener animaciones.
- Se puede trabajar en modo escala grises aprovechando sus de 256 colores posibles.
- Permite definir colores como transparentes

Desventajas:

- Tiene capacidad de 256 colores como máximo.
- Hay que tener programas especiales si queremos ver las animaciones fuera de Internet o crear las propias.

PNG

El formato de archivo PNG (Portable Network Graphics) fue desarrollado como alternativa a los formatos GIF y JPEG. No es propietario de ninguna compañía. A diferencia de GIF, admite imágenes en color real y basadas en paletas de colores. Permite guardar imágenes transparentes (a diferencia de JPEG).

Ventajas:

- Los archivos PNG emplean un avanzado sistema de compresión sin pérdidas.
- No se necesita de ninguna licencia para trabajar con este formato.
- Se puede trabajar en modo escala de grises de 8 y 16 bits.
- Permite definir colores como transparentes

Desventajas:

- No es práctico (aún) para el uso en páginas web debido a que no todos los navegadores lo soportan

TIF

Este es el formato hecho para profesionales del diseño gráfico, porque puede guardar todos los detalles que contiene una imagen. Fue creado por la compañía Adobe Systems que es una mas de sus aportaciones al diseño de imágenes para la plataforma Apple.

Ventajas:

- Contiene mucha información útil.
- Gran calidad de las imágenes, con compresión no destructiva.

Desventajas:

- Ocupa mucho espacio.
- Solo algunos programas a parte de lo desarrollados por Adobe Systems pueden reconocer la codificaron Mac/PC sin problemas.

Para realizar este apéndice se utilizaron las siguientes referencias, [IMF006], [IFF006], [GON002]

Apéndice III – Presentación CACIC 2006

Reconocimiento Automático de Texto Braille

Héctor Ferraro, Claudia Russo
hpferraro@hotmail.com
crusso@lidi.info.unlp.edu.ar

Instituto de Investigación en Informática LIDI
Facultad de Informática. Universidad Nacional de La Plata.
La Plata, Buenos Aires, Argentina.

IV Workshop de Computación Gráfica, Imágenes y Visualización
XII Congreso Argentino de Ciencias de la Computación CACIC 2006

Abstract

This job presents the development of ad-hoc methods to generate a tool to allow reconognize the characters of a Braille text and traduce those in digital text. This kind of tool me be useful to reprint Braille documents, as help to learn Braille, as soon as medium of integration for blind students in all levels or integration in the social boundary of blind people. For this, we develop, in others, two ad-hoc methods: one to segments the scaned image and another to detects the inclination angle of the Braille document at the scaning moment.

Key words: **Digital Images, Automatic recognize, Digital image procesing.**

Este trabajo presenta el desarrollo de métodos ad-hoc para generar una herramienta que permita reconocer los caracteres de un texto Braille y traducirlos a texto digital. Este tipo de herramienta puede ser útil para reimprimir documentos Braille, como ayuda en el aprendizaje de Braille o como medio de integración para estudiantes no videntes en todos los niveles o integración en el ámbito social, de personas no videntes. Para esto, se desarrollo entre otros, un método que segmenta la imagen escaneada a ser tratada y otro para detectar la inclinación de la hoja en el momento del escaneo.

Palabras clave: **Imágenes digitales, Reconocimiento automático, Procesamiento digital de imágenes.**

1. Introducción

La habilidad de reconocer letras y dígitos (caracteres) es fundamental para interpretar lenguajes impresos. Para una computadora un carácter en una página no es más que otra imagen u objeto a ser reconocido. El reconocimiento óptico de caracteres consiste en reconocer de forma automática, letras, dígitos, o algún otro símbolo, de imágenes que han sido opticamente exploradas en líneas horizontales (imágenes raster).

Braille es el sistema de lecto-escritura que utilizan las personas no videntes. Fue inventado por el francés Luis Braille en 1870. Un documento Braille está formado por una serie de caracteres separados a una distancia standard. Cada carácter está compuesto de seis puntos que pueden estar activos o no, es decir presentar salientes en el papel (erupciones) o no. Cada combinación de puntos activos y no activos en un carácter representa un símbolo distinto. De esta forma, con seis puntos se pueden representar 64 símbolos distintos. Los documentos braille pueden ser escritos a una cara o a doble cara, esta última forma se conoce como Braille inter-punto [3] [4].

El caso del reconocimiento de texto Braille tiene diferencias y similitudes con el reconocimiento óptico de caracteres: Hay que resolver el problema de la inclinación de la hoja. Se utilizan técnicas de umbralización de la imagen para segmentar la imagen. Con respecto a las diferencias, en el caso de reconocimiento de texto Braille se intenta detectar la malla (todos los posibles puntos Braille de la hoja) para luego ver cuales puntos están activos y cuales no y para el reconocimiento propiamente dicho no hace falta utilizar técnicas de clasificación de los objetos a reconocer.

A continuación se plantea un esquema de solución al problema planteado. Se describirán las etapas y algoritmos desarrollados para reconocer los caracteres Braille de un documento Braille y transformarlos en texto digital.

2. Solución planteada:

En la Figura 1 pueden verse las etapas para el reconocimiento de texto braille:

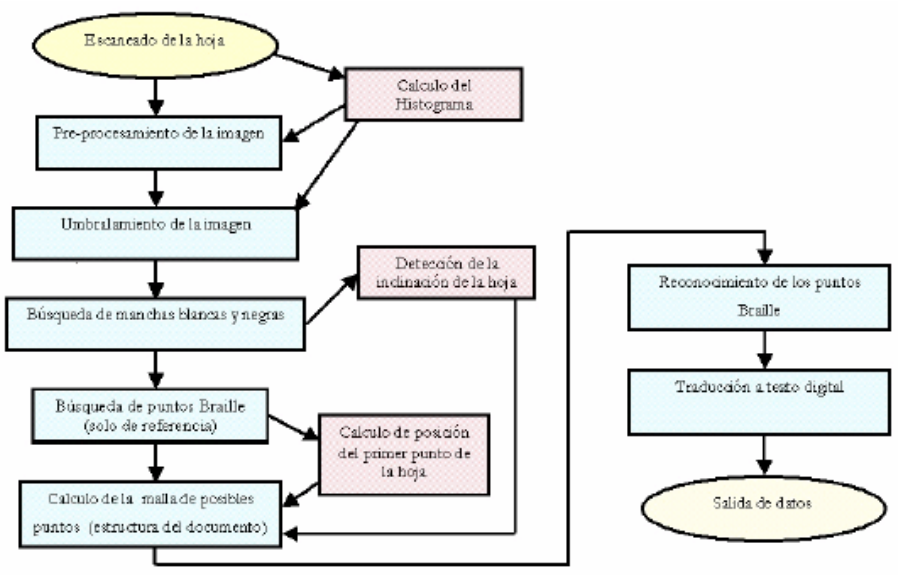


Figura 1. Imagen después de aplicar los umbrales

2.1 Escaneado del documento

Para realizar el escaneado se utilizó el estandar Twain. Este standard provee una API que puede ser utilizada desde una aplicación determinada. De esta forma la aplicación desarrollada se independiza del escáner y se asegura que funcionará con cualquier escáner que trabaje con Twain (actualmente un gran número de ellos) [8]. Se trabajó con imágenes en escala de grises. La resolución de escaneado recomendada es de 150 dpi.

Características de la imagen escaneada:

Al escanear un documento braille se observan unas zonas brillantes, unas mas obscuras y el fondo es de un brillo intermedio. Un punto Braille está formado por una zona brillante y una obscura. Esto queda ilustrado en la Figura 2.

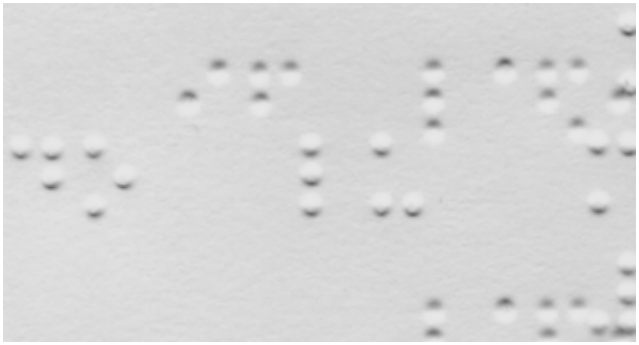


Figura 2. Un trozo de un documento braille escaneado

2.2 Preprocesamiento de la imágen

Es importante disponer de una buena imagen de entrada. Esta etapa puede incluir un conjunto de varias operaciones sobre la imagen adquirida con el escáner a fin de dejar disponible para las etapas posteriores una imagen que pueda ser mejor procesada e interpretada:

- Si la imagen no está en escala de grises habrá que convertirla.
- Si es necesario habrá que ajustar el contraste de la misma.
- Si es necesario habrá que aplicar algún filtro para minimizar el ruido introducido por el escáner al momento de la adquisición.

2.3 Calculo del histograma

En esta etapa disponemos de la imagen en escala de grises. Se calcula el histograma de luminancias de la misma. Calcular el histograma consiste en contar la cantidad de píxeles de la imagen para cada nivel de gris. Los niveles de gris están en el rango 0..255 (escala de grises). Esta información se guarda en una estructura de datos tipo arreglo a fin de dejarla disponible para ser usada por las etapas posteriores [1].

2.4 Umbralamiento de la imagen

En esta etapa se utilizan la imagen resultado de la etapa de pre-procesamiento y el histograma de luminancias antes calculado. Se aplica sobre la imagen un algoritmo de doble umbral obteniendo como resultado una imagen a tres colores.

Para hallar los umbrales se utiliza el histograma de luminancias calculado anteriormente. Se toman los puntos que dejan por debajo/encima un porcentaje determinado del área total del histograma (colas del histograma). Se han obtenido buenos resultados con un porcentaje del 3 % (para documentos a una cara), de los cuales 1% es la cola izquierda y 2% es la cola derecha del mismo [1] [5].

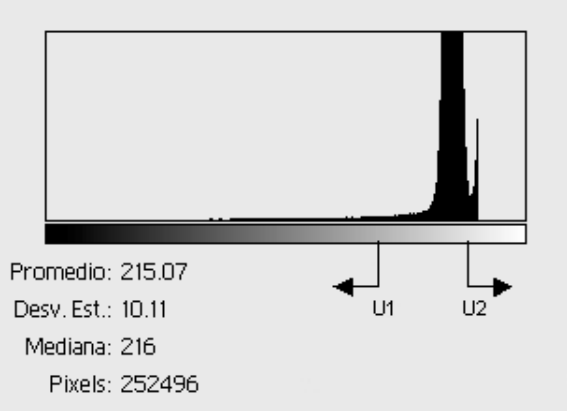


Figura 3. Histograma de luminancias de la imagen original

U1 es el nivel de gris para el cual, el área debajo de la curva que viene dada por la función histograma desde 0 hasta dicho punto, representa el 1% de la cantidad total de píxeles de la imagen.

U2 es el nivel de gris para el cual, el área debajo de la curva que viene dada por la función histograma desde dicho punto hasta 255, representa el 2% de la cantidad total de píxeles de la imagen.

Una vez obtenidos los umbrales se aplica a cada píxel de la imagen una función g. Suponiendo que representamos la imagen como un función f, donde f(x, y) es el nivel de gris para el píxel de la fila "x" columna "y", la función g a aplicar sería la siguiente:

- Si f(x, y) <= U1 entonces f(x, y) = 0 (se reemplaza por un píxel negro)
- Si f(x, y) >= U2 entonces f(x, y) = 255 (se reemplaza por un píxel blanco)
- Si f(x, y) > U1 y f(x, y) < U2 entonces f(x, y) = 128 (se reemplaza por el nivel intermedio de la escala de grises)

De esta forma las zonas brillantes se convertirán en manchas blancas, las zonas oscuras en manchas negras y las zonas de brillo intermedio ahora serán grises. Esto puede observarse en la Figura 4.

Figura 4. Imagen después de aplicar los umbrales

2.5 Búsqueda de manchas blancas y negras

En ésta etapa se buscan los centros de las manchas blancas y negras. Se trabaja sobre la imagen umbralada obtenida en el paso anterior.

Explicación del método desarrollado:

Se barre la imagen píxel a píxel, de izquierda a derecha y de arriba abajo. Sean p el píxel en cada paso del proceso, r el vecino superior de p, t el vecino izquierdo de p, q el vecino diagonal superior izquierdo de p y s el vecino diagonal superior derecho de p. La naturaleza de la secuencia barrido asegura que cuando se llega a p los píxeles r, t, q, y s ya han sido procesados.

Teniendo en cuenta los conceptos establecidos arriba se sigue el siguiente procedimiento: Si el píxel p no es del color de manchas buscado se continua hasta la próxima posición de barrido. Si el valor de p es del color de manchas buscado se examinan r, t, q y s. Si ninguno es del color de la manchas a buscar se asigna una nueva etiqueta a p (este píxel es la primera ocurrencia para la mancha a la que pertenece dicho píxel). Si solo uno de r, t, q, s es del color buscado se asigna la etiqueta de ese píxel a p. Si más de uno de estos píxeles es del color buscado y tienen la misma etiqueta se asigna esta a p; pero si tienen etiquetas diferentes se asigna el valor de una de estas a p y se hace anotación de la equivalencia de etiquetas. Al final del barrido todos los píxeles del color de mancha buscado han sido etiquetados pero algunas de estas etiquetas pueden ser equivalentes. Es decir, una mancha puede estar compuesta por píxeles con etiquetas distintas, pero que son equivalentes [1].

Para solucionar este inconveniente se clasifican todos los pares de etiquetas equivalentes en clases de equivalencia, se asigna una etiqueta diferente a cada clase y luego se da una segunda pasada sobre la imagen reemplazando cada etiqueta por la etiqueta asignada a su clase de equivalencia. Se aprovecha esta segunda pasada para calcular las dimensiones de las manchas y sus respectivos centros.

El proceso de búsqueda de manchas se realiza dos veces, una vez para hallar las blancas y otra para hallar las negras.

2.6 Búsqueda de puntos (solo de referencia)

En esta etapa se buscan algunos puntos Braille, no todos los de la hoja. Estos son solo de referencia y serán necesarios en etapas posteriores del reconocimiento. Se trabaja con la imagen a tres colores, resultado de la etapa de umbralamiento; y las coordenadas de los centros de las manchas halladas en la etapa anterior.

Los puntos de una cara estarán formados por una mancha blanca arriba y una negra debajo (aproximadamente en la misma vertical), mientras que los puntos de la otra estarán formados por una mancha negra arriba y una blanca abajo (aproximadamente en la misma vertical).

Lo importante en esta etapa es no detectar puntos falsos. Los errores se deben a las manchas cercanas que llegan a pegarse al aplicar los umbrales (etapa de umbralamiento). Esto ocurre cuando hay puntos hacia arriba y abajo demasiado cerca.

Explicación del método desarrollado para hallar los puntos:

Se selecciona la primera mancha de la lista de manchas blancas y se verifica que sobre ésta no haya una mancha negra. Si es así se busca si hay una mancha negra debajo de ella. En caso afirmativo se habrá encontrado un punto Braille de la cara "A" de la hoja (se encontró una ocurrencia del patrón mancha blanca arriba, mancha negra abajo). Se calcula su centro con la información de coordenadas de las manchas en cuestión (la blanca y la negra) y se lo guarda en una lista de puntos para cara "A". Luego la mancha negra (la que está por debajo de la mancha blanca) es removida de la lista de manchas negras.

La distancia máxima para verificar si una mancha está por arriba o por debajo de otra se ajustó experimentalmente y es de alrededor de 1.0 mm. La distancia es pasada a píxeles al momento de la comparación, ya que las coordenadas de los centros de las manchas están expresadas en esa unidad.

Para calcular la cantidad de píxeles que representan una cierta cantidad de milímetros se utiliza la siguiente formula f:

$$f(x) = (DPI\_RES * x) / 25,4$$

Donde DPI_RES es la resolución (cantidad de píxeles por pulgada) de la imagen y 25,4 es la cantidad de milímetros que representa una pulgada.

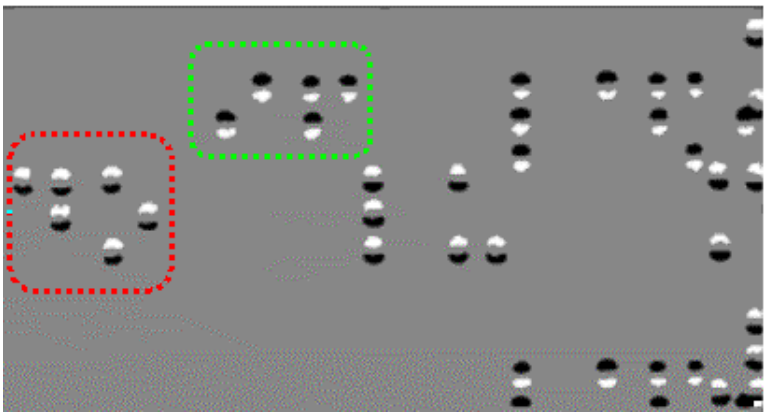


Figura 5. Las manchas encerradas por el cuadro punteado rojo corresponden a la cara "A" de la hoja. Las encerradas por el cuadro punteado verde corresponden a la cara "B" de la misma.

Este procesamiento se repite para cada mancha de la lista de manchas blancas.

Para hallar los puntos de la otra cara (cara "B") se trabaja con la lista de manchas negras que sobraron. Para cada una se verifica que abajo haya una mancha blanca (ésta será seguro una mancha blanca para la que no se encontró pareja negra arriba y por eso fue descartada en la instancia anterior). En caso afirmativo se calcula el centro

del punto hallado con la información de coordenadas de las manchas en cuestión (la negra y la blanca) y se lo guarda en una lista de puntos para cara “B”.

2.7 Búsqueda de los 10 puntos más cercanos al origen

En esta etapa se trabajará con la información de las listas de puntos de cada cara de forma independiente.
Se recorre la lista de puntos de la cara que se está procesando, buscando los diez puntos de coordenadas más cercanas al (0, 0). La coordenada (0, 0) corresponde al borde superior izquierdo de la imagen. Los puntos encontrados se guardan en una lista que será usada en etapas posteriores del reconocimiento.

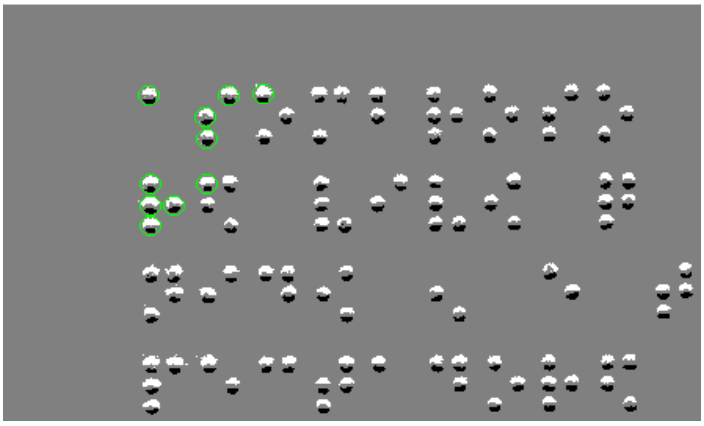


Figura 6. Los puntos Braille marcados con verde son los diez más cercanos al borde superior izquierdo.

2.8 Detección de la inclinación de la hoja

En esta etapa se detecta el ángulo de inclinación de la hoja sobre el escáner. Este valor será de gran utilidad en las etapas posteriores del reconocimiento.

Explicación del método desarrollado:

Se barre la imagen con diferentes ángulos. Para cada ángulo se suman los píxeles negros o blancos de cada fila. Luego se calcula la varianza de las sumas. El ángulo que maximice la varianza es el ángulo de inclinación de la hoja.

Este algoritmo es muy lento. Para aumentar la performance se tuvieron en cuenta algunas consideraciones:

- Se trabaja sobre una mitad de la imagen. Se elige la mitad que tenga mas información de puntos (mas probabilidad de texto Braille). Para realizar esta elección se pueden contar la cantidad de píxeles blancos y negros de cada mitad. Se selecciona la mitad que tenga mas píxeles blancos y negros.

- Se hace una reducción de la resolución horizontal de la imagen de 1 en 5. Es decir al realizar las sumas por fila se avanza de a 5 píxeles. Esto acelera considerablemente el algoritmo y no supone una perdida de precisión.

- Se utilizó un método de aproximación. En cada paso se busca el ángulo de inclinación con más precisión pero dentro del intervalo de error del paso anterior. La misma se hace en tres pasos.

Con estas consideraciones se aumenta notablemente el speed-up del algoritmo desarrollado a la vez que se mantiene la robustez y precisión del mismo [2].

2.9 Calculo de posición del primer punto de la hoja

Las distancias entre puntos en la escritura Braille siguen un patrón definido. Conociendo la resolución de escaneado esas distancias se pueden pasar a píxeles. Además, las filas y columnas estarán inclinadas con inclinación conocida (que podría ser nula). Se usará la suposición de que las filas y columnas son siempre perpendiculares aunque a veces esto no sea así.

Una vez conocido el ángulo de inclinación es posible centrarse en un punto y moverse las distancias adecuadas hasta encontrar todos los demás (por lo menos todos los verdaderos).

Para ello son necesarias dos cosas:

- Que el punto inicial sea verdadero.

- Saber a que posición (de 1 a 6) corresponde dentro de su carácter.

Se utiliza la lista de puntos más cercanos al origen (calculada en una etapa anterior) y se considera cada uno como posible punto inicial. Para cada punto se suponen las 6 posiciones posibles. Después se recorre la malla (llamamos malla al conjunto de posibles puntos de la hoja) contando los puntos de entrada que son “encajados”. La máxima de esas 60 cuentas determina el punto de referencia inicial [5].

Para hacer más robusto este algoritmo frente al ruido se tienen en cuenta dos cuestiones:

- Se permite cierto margen de error en la búsqueda de puntos (encajado).

- No se realiza la búsqueda en toda la hoja sino en un número limitado de filas y columnas a partir del punto de estudio. Esto permite que el algoritmo sea más rápido y además compensa pequeños errores en el ángulo de inclinación calculado.

2.10 Calculo de todos los posibles puntos de la hoja

Como se dijo anteriormente, llamamos malla Braille al conjunto de posibles puntos de la hoja. La construcción de ésta se realiza partiendo del punto de referencia inicial (calculado en la etapa anterior), colocando sobre el una plantilla de tamaño un carácter. Esta plantilla se moverá en todas direcciones hasta los límites de la hoja determinando, de esta forma, todos los posibles puntos. El movimiento de la plantilla se hace teniendo en cuenta las distancias estándar entre caracteres y el ángulo de inclinación detectado.

La creación de la malla es adaptativa: Al posicionar la plantilla se busca en la lista de puntos Braille (calculada en una etapa anterior) para ver cuantos puntos son encajados en esa posición. Si no se encaja ninguno se sigue con el próximo carácter, pero si se encajan puntos (para esa posición) la plantilla se centra basándose en ellos.

El próximo movimiento de la plantilla será relativo a la posición actual de la plantilla adaptada. De ésta forma el resultado no depende en exceso del punto de referencia inicial. Además se corrigen pequeños errores en el cálculo del ángulo de inclinación de la hoja.
Los puntos Braille que se hayan detectado erróneamente en la etapa de búsqueda de puntos simplemente estarán descolocados y serán descartados al momento de construir la malla.

2.11 Reconocimiento de los puntos Braille

En la etapa anterior se calculó la malla Braille. Es decir la posición de los posibles puntos Braille. En esta etapa se determinará cuales de estos puntos están activos y cuales no. Se trabaja con la malla y la imagen umbralada a un porcentaje mayor para aumentar el tamaño de las manchas blancas y negras para disminuir el error en el encajado de puntos [5]. La malla está organizada en caracteres Braille (conjuntos de seis puntos Braille).

Se recorre la malla: Para cada carácter en la malla se determina que puntos del mismo están activos. Para ello se inspecciona la imagen por arriba para ver si hay una cierta cantidad de píxeles blanco y por debajo para ver si hay una cierta cantidad de píxeles negros. Si es así se anota el punto como activo, caso contrario será inactivo.

2.12 Transformación a texto digital

A fin de realizar la transformación a texto digital (ascii) es necesaria una estructura para almacenar el alfabeto Braille. Esta estructura es una tabla con 64 entradas donde el índice de la primera posición es 0 y el índice de la última es 63.

En esta etapa sabemos para cada carácter Braille de la malla cuales son los puntos que están activos y cuales no (se realizó en la etapa anterior). Tomando ventaja de ello se realiza el siguiente cálculo para cada uno de los caracteres:

$$c(p) = 32 * v(p_6) + 16 * v(p_5) + 8 * v(p_4) + 4 * v(p_3) + 2 * v(p_2) + v(p_1)$$

donde $v(p_x)$ es 1 si el punto p_x del carácter se determinó como activo y 0 en caso contrario.

El valor c(p) calculado va a estar en el rango de 0 a 63 y se utiliza como índice de acceso para la tabla que contiene el alfabeto. Dependiendo del valor c(p) se realizan se realiza un tratamiento distinto al momento de la traducción:

- Si es distinto de 40 y 60 se busca con este índice en la tabla y se obtiene el carácter de texto correspondiente.

- Si es igual a 40 significa que el o los próximos caracteres están en mayúscula [3] [4]. Entonces se inspecciona el próximo carácter de la malla. Si el valor de c(p) para este carácter es distinto de 40 se busca en la tabla con este índice obteniendo el carácter de texto correspondiente y se lo pasa a mayúscula; mientras que si es igual a 40 (esto indica que toda la palabra es mayúscula) se inspeccionan los próximos caracteres hasta que el valor de c(p) sea igual a cero 0 (es un carácter de espaciado) obteniéndose los caracteres para los índices c(p) calculados y se los pasa todos a mayúsculas.
- Si es igual a 60 significa que los próximos caracteres son números [3] [4]. En este caso también se utiliza el valor de c(p) como índice, pero de otra tabla del alfabeto. Esto se resolvió así por que la escritura Braille utiliza los mismos símbolos con los que representa las primeras letras del abecedario para representar los dígitos numéricos. De otra forma se tendrían dos posibles valores para el mismo índice sobre la misma tabla.

3. Conclusiones y Resultados Obtenidos:

Utilizando el esquema de solución propuesto en [5] se desarrollaron métodos ad-hoc para las diferentes etapas del reconocimiento automático de texto Braille obteniendo resultados óptimos. En particular se desarrolló un método para detectar la inclinación de la hoja que resultó más eficiente que el propuesto en [5]. Actualmente se está implementando la solución en Java.

4. Líneas Futuras:

Desarrollo de un algoritmo de segmentación para aislar los puntos de las diferentes caras en el caso de trabajar con braille interpunto (hojas doble faz).
Detección automática de distancias. Aunque las distancias entre puntos Braille están estandarizadas, existen varios tipos de hojas Braille con juegos de distancias distintos. Hasta ahora se le ingresa a los algoritmos las distancias entre caracteres y entre puntos de caracteres.

5. Bibliografía y Referencias:

[1] Digital Image Processing, 2ND ED. Gonzalez, R.C.; Woods, R.E. McGraw-Hill 2002.

[2] Optical Recognition of Braille writing using standard equipment – IEEE

[3] http://www.funcaragol.org/

[4] http://members.tripod.com/DE_VISU/brlest.html

[5] http://www.gpi.tsc.uvigo.es/pub/papers/urs95_2.pdf

[6] http://www.sighted.com/english/obr.html

[7] Introducción al reconocimiento de patrones clásico. Oscar Bría.

[8] Twain www.twain.org

[9] IEEE Transactions on Pattern Analysis and Machine Intelligence

[10] IEEE - Vision, Image and Signal Processing

[11] W. Pratt, Digital Image Processing, 2nd Edition. John Wiley and Sons In., U.S.A., 1991.

Bibliografía y Referencias

UVO097	Paper sobre Proyecto de la Universidad de Vigo y la O.N.C.E O.C.R para texto Braille. Año 1997 http://www.gpi.tsc.uvigo.es
GON002	Gonzalez, R.C., Woods, R.E., Digital Image Processing, Addison-Wesley, 1993.
APO096	A. Antonacopoulos. "Automatic Reading of Braille Documents". Handbook of Optical Character Recognition and Document Image Analysis. World Publishing Co. 1996.
IEE005	Web de la IEEE. www.i3e.org
OBR000	Web de la empresa de la Aplicación comercial OBR. www.neovision.cz/prods/obr/
MTF094	Paper IEEE. Optical Recognition of Braille Writing Using Standard Equipment Jan Mennens, Luc van Tichelen, Guido Francois and Jan. J.Engelen
VNG097	Paper IEEE. Regular Feature Extraction for Recognition of Braille C M Ng Dept. of Computing and Mathematics, The HK Technical College (Chai wan), Hong Kong, China. Vincent Ng, Y Lau Dept. of Computing and Mathematics, The HK Technical College (Chai wan), Hong Kong, China.
SEI005.	Estado actual de la segmentación de Imágenes http://www.aisa.uvigo.es/thesis/fvazquez/cap2.pdf
TRF007	Técnicas de Reconocimiento de Imágenes para la Creación de Fotomosaicos Tesis de Grado
BRA087	J. J. Della Barca y Otros. "Normativa Braille Unificada para los Países Hispano-Parlantes". Reunión para la Unificación de la Normativa Braille en Países de Habla Hispana (patrocinada por la O.N.C.E. y la Fundación Braille de Uruguay). Montevideo (Uruguay). Junio, 1987.
DEV099.	Web de De Visu Información sobre y para personas ciegas o con deficiencia visual. http://members.tripod.com/DE_VISU/index.html
BBL005.	Web de la Biblioteca Braille de la ciudad de La Plata Buenos Aires, Argentina
AFB005.	Web de American Foundation of de Blind. http://www.afb.org
FMC005.	Fundació de Cecs Manuel Caragol http://www.funcaragol.org
FBU005.	Web de la Fundación Braille de Uruguay. http://www.fbraille.com.uy/index.htm
TJA006	Tutorial de Java http://www.cica.es/formacion/JavaTut/Intro/tabla.html

JSU006	Página de Java Sun. http://java.sun.com/
GLJ099	Guía de iniciación al Lenguaje Java http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/Index.htm
TWN099	Web oficial de Twain http://www.twain.org/
MOR005	Proyecto Morena http://www.gnome.sk/index.html
ECL006	Web del proyecto Eclipse www.eclipse.org
FOT006	Web de Fotonostra http://www.fotonostra.com/grafico/escaners.htm
HAR006	Web de Hardware http://www.duiops.net
IMF006	Image Formats (web) http://www.image-formats.com/
IFF006	Image File Formats (web) http://www.yourhtmlsource.com/images/fileformats.html