ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA EL RECONOCIMIENTO DE LA DENOMINACIÓN DE DÓLARES AMERICANOS, DIRIGIDO A PERSONAS CON DISCAPACIDAD VISUAL

FELIPE LEONEL GRIJALVA ARÉVALO
JUAN CARLOS RODRÍGUEZ GUERRA

Sangolquí - Ecuador
2010

CERTIFICACIÓN

Certificamos que el presente proyecto de grado fue realizado en su totalidad por los Sres. Felipe Leonel Grijalva Arévalo y Juan Carlos Rodríguez Guerra bajo nuestra dirección.

Ing. Julio Larco	Ing. Luis Orozco
DIRECTOR	**CODIRECTOR**

RESUMEN

El presente proyecto consiste en el desarrollo del prototipo de un sistema para el reconocimiento de la denominación de los dólares norteamericanos de más común circulación en Ecuador, dirigido a personas con discapacidad visual que, por su limitación, encuentran un desafío en esta tarea.

Este trabajo toma como fundamento teórico a las técnicas del Procesamiento digital de imágenes y, fundamentalmente, al método de reconocimiento de imágenes conocido como *Eigenfaces*, basado en la teoría matemática del Análisis de componentes principales.

El sistema ha sido implementado como una aplicación de *software* para un teléfono móvil con sistema operativo *Symbian* S60, 3º edición, *first release*, que se desarrolló a través del entorno de desarrollo integrado (IDE) *Carbide.c++*, en código del lenguaje *Symbian* C++. El sistema es capaz de reproducir mensajes de audio que expresan la denominación del billete en frente de la cámara del dispositivo móvil en una filmación continua sin necesidad de fotografiarlo, mediante el procesamiento de cada *frame* de la misma.

Las pruebas realizadas en dos teléfonos *Nokia*, de modelos N73 y E65, sobre los 4 extremos de 218 billetes de diversa denominación, en distintas condiciones de entorno, demuestran la mejor exactitud del sistema en un 99.8% y una velocidad mínima de procesamiento de 7 *frames* por segundo.

DEDICATORIA

Dedicamos el presente trabajo al alma que encierra el concepto de esa misma palabra; al trabajo:

Al trabajo de aquellas personas que superan día tras día las barreras que el destino les ha impuesto, luchando por mantener su dignidad de seres humanos ante los ojos de un mundo que quizás no ven con los suyos, y que sin embargo se les presenta con una mística gracia, casi siempre imperceptible para las personas que, al no perderlo, aún no hemos advertido el valor que tiene un don en la vida.

Al trabajo de toda la gente preparada que ha sabido comprender que el propósito del abrupto desarrollo actual de la tecnología, y de su talento para comprenderla y utilizarla, no es la ruptura de los vínculos con lo que nos hace seres humanos, en aras de superfluas y ambiguas necesidades recreadas en un mundo artificial, sino la construcción de caminos nunca antes vistos para el reencuentro del ser humano con su pasado, presente y futuro, que lo acerquen y no alejen de sus semejantes, para que esta unión cree una nueva alianza con un mundo real y vivo, que asegure el crecimiento en él y no a costa de él.

Al trabajo de todas y cada una de las personas que han hecho posible nuestra propia preparación; trabajo que, lejos de proveer solamente el bienestar material en nuestro camino, ha forjado nuestras almas con el fuego de su abnegada misión. Tiempo, comprensión, respeto, compañía, confianza…; todos aquellos beneficios recibidos a lo largo de esta etapa de nuestra vida, los hemos de saber utilizar para no defraudar la confianza que han puesto en nuestros conocimientos. Nunca olvidaremos la gran lección de cariño que nos han brindado en su dedicación.

AGRADECIMIENTOS

Me gustaría tener palabras que expresen mi gratitud hacia quienes han participado directa e indirectamente del desarrollo de este proyecto; sin embargo, voy a hacer lo posible por mencionarlos a todos.

Gratitud es lo que más le tengo a *Dios*, por permitirme vivir esta vida, que me admite conocerme más, que me consiente seguir en el camino de crecimiento y evolución, el mismo camino que hacemos todos cada día, a cada instante. Camino en el que espero no caer muchas veces y en el que si caigo, *Dios* me provea de la fortaleza para levantarme.

Mi agradecimiento especial es para mis padres, *Venus* y *Jorge*, quienes han influido en mí de manera trascendente desde los comienzos de mi existencia y son en gran medida, responsables de la persona que soy en la actualidad.

A mis hermanos *Gandy* y *Estefanía* quienes me ayudan de manera significativa, aportando ambos, diariamente con sus cualidades y habilidades, tan distintas y tan eminentes. En muchas ocasiones, de alguna forma, subsidiándome, mostrándome que todos tenemos dones y que no necesariamente son valorados por nuestra sociedad.

Debo decir en honor a la verdad, que mi hermano *Gandy* ha sido quien me motivó a enfocar mi tesis a una población minoritaria, como lo es; la de los no videntes, dado que a diario me muestra que esa "minoría" la componen personas, cada una es hermano, padre, madre, abuela, primo, prima… de alguien y que solo por ese "alguien" y su ser querido, ha valido la pena realizar ésta tesis.

Por su compañía y opinión sincera, está siempre presente en mi mente *Nathali*, quien ha inspirado y motivado mis pequeños y grandes logros. Mi lealtad es especial para ella por amarme tal y como soy.

Felipe Grijalva

AGRADECIMIENTOS

¿Qué hacer cuando un simple *gracias* resulta insuficiente? Para mí, es solamente esperar que al recibir aquella simple dádiva, pueda el alma de cada uno leer entre las líneas que los ojos no hallan, y tal vez así puedan acercarse a comprender la gratitud que siente mi alma cuando volteo al final de este camino y encuentro, algunos borrosos en la lejanía y otros nítidos a mi lado, a todos sus rostros.

Porque mi vida no le pertenece a nadie sino a ustedes, mi única riqueza de cuatro gemas: Porque me entregaste el cuerpo, el alma y el corazón con los pedazos de tu cuerpo, tu alma y tu corazón; y porque nunca alcanzaré a devolverte, ni a ti ni a nadie en el mundo, la infinita cantidad de amor que me has prestado, gracias *Amparo*. Gracias *Carlos*, porque sé que nunca nos dejaremos de querer; porque nada, ni el inclemente tiempo, ni los caminos del destino, podrían borrarte de mi corazón. Gracias *Jean Pierre*, gracias *María José* porque son todo lo bueno que nunca llegaré a ser; porque me hacen saber que soy querido y respetado por las personas que más quiero y respeto.

Por ser mi sangre, aquella que llena mi corazón: Gracias *Mamina* y *Papaca*, nunca dejaré de saber que me han hecho quien soy. Gracias, todos mis tíos, porque en su unión, me enseñan sobre la felicidad. Gracias mis abuelitos, por tenerme en sus oraciones. Gracias *David*, *Diego*, y a todos mis primos, más que eso, mis amigos; porque en ustedes hallo siempre el cariño cuando se me pierde en el vasto mundo.

Porque son la cortina que no deja pasar al frío de la soledad: Porque me han acompañado, querido y ayudado de tantas formas que no puedo numerarlas; porque gracias a ustedes, el mundo tiene más sentido y porque me entregan el fraternal hombro de su comprensión cuando no lo tiene; porque los cuento en todos mis días, gracias *Diego, Israel*, *Santiago*, *Javier*, *Geovanny*, *Juan Carlos*, *Juan*, *Cristina, Ivan, Daniela, hermanos de Naagrum*; y a todos ustedes que no estarán en estas paupérrimas líneas o aún en este triste mundo, pero sí en mi corazón; gracias mis queridos amigos.

A aquella fuerza magnánima, arquitecta del cosmos, que mantiene viva la belleza que el ser humano destruye. A ese Dios, esa Madre Tierra, ese Motor Universal. Gracias.

J.C. Rodríguez

PRÓLOGO

La población con algún tipo de discapacidad visual en Ecuador, según datos de las instituciones CONADIS e INEC, asciende a 363000 personas. Ellas, en su diario vivir, enfrentan una serie de desafíos de toda índole, a los cuales responden mediante una serie de comportamientos predefinidos y que, de acuerdo al medio en donde se apliquen, pueden resultar más o menos eficaces.

En el presente proyecto, se presenta uno de los problemas más delicados de la población con deficiencia visual ya que trata sobre el manejo de dinero, tema que en el entorno globalizado actual es de fundamental interés general. Con respecto a este asunto, el reconocimiento del valor de las monedas por parte de personas no videntes es factible, al constituirse estas de distinto tamaño; este no es el caso para los billetes. De esta forma, los métodos usualmente utilizados por ellos para el reconocimiento de la denominación de un billete en su posesión los ubican en una situación desfavorable, ya sea por la dificultad e incomodidad de los mismos o en los graves inconvenientes económicos que causa el hecho de que ciertas personas de bajos escrúpulos tomen ventaja de su discapacidad visual en una transacción monetaria.

Las limitaciones de las personas con discapacidad visual, presentan soluciones tecnológicas en varias partes del mundo, mas en nuestro país, la mayor parte de la tecnología que se importa o desarrolla, no toma en cuenta a la población no vidente y sus necesidades. De acuerdo al CONADIS, apenas el 9% de la población no vidente ecuatoriana tiene la prestación de uso de alguna ayuda tecnológica para ver, mientras que el resto no tiene el suficiente acceso a tales facultades.

Por estas razones, el trabajo del presente proyecto se enfoca no solo en el desarrollo del prototipo para la solución del problema presentado, sino también en la investigación de las herramientas que permitan hacerlo con las tecnologías de mayor alcance y facilidad de uso para la población no vidente, en general. Tales premisas se ven traducidas en las

prestaciones de portabilidad y penetrabilidad que actualmente ofrecen los teléfonos celulares.

La redacción del trabajo aquí presentado se organiza en 5 capítulos, en los cuales se plantean todos los temas que describen al proyecto desarrollado.

En el Capítulo 1, se hace una introducción al problema que se pretende solucionar con el presente trabajo de tesis, indicando en sus párrafos la información obtenida, de estadísticas y de primera mano sobre la realidad de la población no vidente que finalmente plantea el problema, su importancia y alcance.

En el Capítulo 2, se presenta el fundamento teórico utilizado en el desarrollo del proyecto: el conjunto de técnicas del Procesamiento Digital de Imágenes (PDI) utilizadas para el análisis de la imagen digital de un billete y, especialmente, la descripción del método utilizado para el reconocimiento, *Eigenfaces*, debidamente fundamentado como la aplicación de la matemática del Análisis de Componentes Principales (PCA).

En el Capítulo 3, se detalla el proceso seguido para el desarrollo del prototipo del sistema. Exhibe los criterios para la selección del *hardware* y *software* utilizados y culmina numerando todos los pasos seguidos en el diseño y la programación de la aplicación de *software* que implementa el sistema en las plataformas seleccionadas.

En el Capítulo 4, se describen las pruebas realizadas para la determinación de la eficacia del dispositivo, en su nivel de su porcentaje de éxitos. Se presentan también los resultados obtenidos a partir de estas pruebas y las conclusiones que se infieren a partir de ellos.

En el Capítulo 5 se numeran las conclusiones obtenidas de todo el desarrollo del proyecto, en relación al problema inicialmente planteado, así como las recomendaciones referentes tanto al funcionamiento del dispositivo como al desarrollo de la aplicación, tomando en cuenta su naturaleza.

Finalmente, se incluye una sección de anexos en la cual se hallan algunas de las herramientas utilizadas en el desarrollo de proyecto, los códigos de programación de la aplicación en los distintos lenguajes utilizados, para referencia de futuros desarrollos, y un manual de usuario del sistema finalmente producido, como complemento del mismo.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

GLOSARIO

ANSI, codificación Codificación de caracteres en 8 bits que extiende a la representación ASCII de 7 bits.

CONADIS Consejo Nacional de Discapacidades. Organismo del estado ecuatoriano con la visión de desarrollar acciones de prevención, atención e integración, con el propósito de prevenir las discapacidades y elevar la calidad de vida de las personas con discapacidad.

CPU *Central Processing Unit*. Unidad central de procesamiento de una computadora digital.

DMIPS *Dhrystone Million instructions per second.* Unidad que cuantifica la cantidad de millones de instrucciones por segundo que ejecuta un procesador.

Eigenfaces Rostro propio. Anglicismo que describe un método para el reconocimiento de rostros de personas. Se refiere a los vectores propios o componentes principales de un conjunto de entrenamiento de rostros, para el reconocimiento por computador de los mismos.

Eigenimages Imagen propia. Anglicismo que describe un método para el reconocimiento de imágenes. Se refiere a los vectores propios o componentes principales de un conjunto de entrenamiento de imágenes de cualquier índole, para el reconocimiento por computador de las mismas.

FPS *Frames per second.* Medida de frecuencia a la cual un dispositivo de procesamiento de imágenes genera distintos *frames*.

Frame	Fotograma. En informática, es una imagen particular dentro de una sucesión de imágenes, que está determinada por un número determinado de píxeles.
IDE	*Integrated Development Environment*. Programa informático compuesto por un conjunto de herramientas de programación.
INEC	Instituto nacional de estadística y censos. Institución del estado ecuatoriano con la misión de generar y difundir información estadística útil y de calidad del país con el propósito de facilitar la evaluación del desarrollo de la sociedad y de la economía.
JAWS	*Software* para personas con visión reducida que convierte el contenido de la pantalla de una PC en sonido, de manera que el usuario pueda navegar sin necesidad de verlo.
MATLAB	*MATrix LABoratory. Software* matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M), disponible para las plataformas *Unix*, *Windows* y *Apple Mac* OS X.
MB	*Megabyte*. Unidad de medida de cantidad de datos informáticos.
MHz	*Megahertz*. Unidad que se utiliza muy frecuentemente como para la frecuencia de trabajo de un dispositivo de *hardware*.
NTSC	*National Television System Committee*. Sistema de codificación y transmisión de televisión en color.
Offline	Fuera de línea. Un proceso es *offline* si su ejecución no es en tiempo real y se da a voluntad de un operador, por ejemplo para la producción de una base de datos mediante un programa de entrenamiento.
Online	En línea. Un proceso es *online* si se halla dando uso de algún aparato de una forma automática, en tiempo real, sin intervención

	humana.
PC	*Personal Computer***.** La expresión estándar que se utiliza para denominar a las computadoras personales en general.
PCA	*Principal Component Analysis*. Técnica estadística para la identificación de patrones, que permiten expresar los conjuntos de datos de una forma tal que se resaltan sus similaridades y diferencias
PDI	Procesamiento Digital de Imágenes. Conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información.
Píxel	Acrónimo del inglés *picture element*. Es la menor unidad homogénea que forma parte de una imagen digital.
RAM	*Random Access Memory*. Memoria de acceso aleatoria y volátil de una computadora digital.
RGB	*Red, Green, Blue*. Modelo de color para la composición del mismo en términos de la intensidad de los colores primarios con que se forma: el rojo, el verde y el azul; con el que es posible representar un color mediante la mezcla por adición de los tres colores luz primarios.
Sistema microprocesado	Un tipo de sistema computacional que utiliza un microprocesador como unidad central de procesamiento (CPU). Generalmente, son microcomputadoras que ocupan espacios físicos pequeños, comparados a los que ocupan las computadoras personales (PC).
Viewfinder	*Buffer* de memoria que contiene la imagen que se actualiza periódicamente con la información entregada por los sensores de una cámara digital, desplegada a manera de vídeo en la pantalla de un teléfono celular.

CAPÍTULO 1

INTRODUCCIÓN

1.1 REALIDAD DE LA POBLACIÓN NO VIDENTE EN ECUADOR

Dado que el presente proyecto se enfoca a las personas no videntes del Ecuador, se iniciará la descripción del problema a resolverse con una aproximación estadística de la realidad de la población no vidente en el país.

1.1.1 Estadísticas de la población con deficiencias visuales en Ecuador

De acuerdo a las estadísticas realizadas por el Consejo Nacional de Discapacidades (CONADIS) y el Instituto Nacional de Estadísticas y Censos (INEC), del total de la población de Ecuador, el 13,2% corresponde a personas con algún tipo de discapacidad (1`600.000 personas) de las cuales 363.000 tienen deficiencias visuales [1]. De otra parte, se sabe que solo 17.596 personas con deficiencias visuales se encuentran registradas en el CONADIS [2]. La figura 1.1 muestra la distribución de la población ecuatoriana con discapacidad conforme al tipo de deficiencia [2].

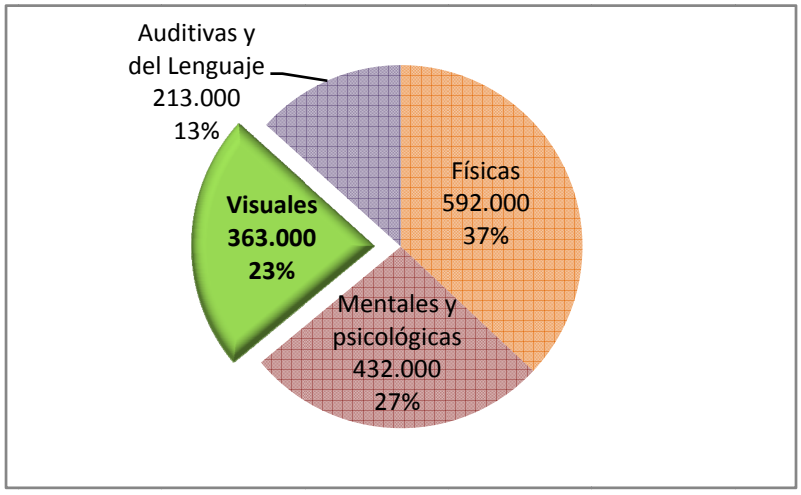


Figura 1.1. Número de personas ecuatorianas con discapacidad de acuerdo al tipo de deficiencia

1.1.2 Ayudas técnicas existentes para ver

De acuerdo a encuestas realizadas en conjunto por el CONADIS y el INEC, el 9% de personas discapacitadas con deficiencias visuales utiliza alguna ayuda técnica para ver mientras que el 17% afirma necesitar alguna ayuda técnica para esta deficiencia [3]. Las ayudas técnicas para ver más utilizadas y necesitadas se resumen en la tabla 1.1.

Tabla 1.1. Porcentaje de personas discapacitadas con deficiencias visuales por uso y necesidad de ayudas técnicas para ver

Tipo de ayuda técnica para ver	% que UTILIZA una ayuda técnica para ver	% que NECESITA una ayuda técnica para ver
Lentes y lupas	50 %	53 %
Instrumentos *Braille*	4.5 %	7.9 %
***Software* JAWS**[1]	5 %	7.8 %
Bastón guía	3.5 %	10.7 %
Equipos electrónicos varios	1.1 %	13.5 %

1.2 EL PROBLEMA DEL RECONOCIMIENTO DE BILLETES

La población no vidente, en su diario vivir, enfrenta un gran número de desafíos, a los cuales responde mediante comportamientos predefinidos y de acuerdo al medio en el que los apliquen, estos pueden resultar más o menos eficaces. Uno de estos es el que se describe en la presente sección.

1.2.1 Planteamiento del problema

Dentro de la sociedad globalizada de los tiempos actuales, el modelo materialista mundialmente aceptado, del capital sobre los medios de producción, hace que el manejo del dinero para las transacciones diarias necesarias en el intercambio comercial, sea primordial para cualquier persona. La población no vidente enfrenta un problema desafiante al encontrarse en una posición en la cual sus integrantes necesitan reconocer la denominación del dinero que manejan. Este problema se presenta específicamente en el caso de los billetes, pues las monedas, al estar construidas de diferentes tamaños, son de

[1] Software para personas con visión reducida que convierte el contenido de la pantalla de una PC en sonido, de manera que el usuario pueda navegar sin necesidad de verlo.

más fácil reconocimiento táctil para cualquier persona; como se puede inferir, no es este el caso para los billetes.

La información de primera mano, obtenida del diálogo directo con varias personas pertenecientes a la población no vidente realizado a manera de entrevista sobre el tema mencionado, haciendo uso de la guía del anexo A1, produce un conjunto de conclusiones sobre los problemas y las necesidades de la población no vidente con respecto al reconocimiento de la denominación de los billetes, resumidas en los siguientes párrafos:

El problema del reconocimiento de la denominación de billetes por parte de la población no vidente es un fenómeno real, y se manifiesta, ya sea en la dificultad e incomodidad de los métodos usualmente utilizados para la discriminación de los billetes o en los graves inconvenientes económicos que causa el hecho de que ciertas personas de bajos escrúpulos tomen ventaja de la discapacidad visual de la persona no vidente en una transacción monetaria.

El método generalmente utilizado por las personas no videntes para la discriminación de los billetes en su posesión es el de obtener la información sobre la denominación del billete de una segunda persona, para después realizar alguna clase de identificación física del mismo, actuando sobre el papel (*e.g.* doblez realizado en una esquina del billete), que es diferente para cada billete de distinta denominación, antes de la realización de cualquier transacción monetaria. Sin embargo, este método presenta la gran inconveniencia de no ser factible todo el tiempo pues, como se puede inferir, las transacciones monetarias más comunes son las de carácter informal, que deben ser realizadas *in situ*. En cambio, para la obtención de nuevos especímenes de billetes, dependen del testimonio de cada persona con la cual realicen la transacción, situación que se plantea aquí como no deseable.

1.2.2 Formulación del problema

Establecido y reconocido el problema por las personas entrevistadas, se corrobora su necesidad de contar con algún tipo de dispositivo autónomo para el reconocimiento de la denominación de los billetes.

De acuerdo a las preferencias de las personas entrevistadas pertenecientes a la población no vidente, el requerimiento más importante que debe cumplir un dispositivo de tal connotación es la portabilidad. Una idea extraída de la experiencia de entrevista es la

característica de que las funciones que debería cumplir tal dispositivo fueran implementadas dentro de un dispositivo ya existente, con grandes y conocidas capacidades de portabilidad, así como de penetrabilidad en el mercado actual, como lo son los teléfonos móviles, como se apreció en la empatía en las personas no videntes entrevistadas por tal idea.

De este análisis, se llega a la formulación del problema presentado en el presente proyecto, definido a través de la siguiente pregunta: ¿CÓMO SE PUEDE IMPLEMENTAR UN SISTEMA PARA EL RECONOCIMIENTO DE LA DENOMINACIÓN DE DÓLARES AMERICANOS (LA MONEDA EN CIRCULACIÓN EN ECUADOR) ENFOCADO A PERSONAS NO VIDENTES, MEDIANTE TECNOLOGÍAS DE FÁCIL ACCESO PARA ESTA POBLACIÓN?.

El objetivo del presente proyecto es el diseño de un prototipo de tal sistema en un teléfono móvil de características detalladas en los próximos capítulos, para responder esta formulación así como la implementación del sistema para la comprobación física de la hipótesis que enmarca la misma.

1.2.3 Importancia del proyecto

En general, la población no vidente enfrenta una serie de desafíos de esta y otra índole en su diario vivir, algunos de los cuales tienen actualmente el planteamiento e incluso la implementación de alguna clase de solución en varias partes del mundo, especialmente, de carácter electrónico, mas en nuestro país, la mayor parte de la tecnología que se importa o desarrolla, no toma en cuenta a la población no vidente y sus necesidades, por lo que las soluciones son obtenidas sólo por una pequeña parte de esta población; usualmente, la de mayores recursos económicos, mientras el resto no consigue solventar sus necesidades. Esto lleva a la conclusión de que la implementación de un sistema que dé una solución al problema del reconocimiento de la denominación de los billetes, se enfoca en un beneficio social a una parte de la población generalmente no tomada en cuenta en los proyectos de aplicación electrónica. Además, el presente trabajo, plantea la iniciativa para el desarrollo de nuevos proyectos, en Ecuador, enfocados a la solución del resto de problemas que enfrenta esta y otras minorías. Tal desarrollo contribuye al mejoramiento de la calidad de vida de los seres humanos, cualquiera que fuere su condición. Esto, en la

opinión de los autores de este proyecto, debería considerarse como el más alto logro del desarrollo tecnológico.

1.2.4 Alcance y delimitación de la solución propuesta

En este inciso, cabe destacar las características generales de la solución propuesta, en relación al alcance que tendrá el presente proyecto, que son las siguientes:

- El proyecto pretende la producción de un prototipo del sistema formulado, compuesto por una aplicación de *software*, instalada en un teléfono móvil de características detalladas, que implemente el sistema de una forma completamente funcional.
- El sistema implementado en el teléfono móvil antes mencionado, en plena funcionalidad, será capaz de ofrecer a una persona no vidente, mediante un procedimiento adecuado de manejo del mismo, la opción del reconocimiento de la denominación de los dólares de los Estados Unidos de Norteamérica de común circulación en la República del Ecuador: uno, cinco, diez y veinte dólares; así como también de dos especímenes de menor circulación: cincuenta y cien dólares. Se excluyen del reconocimiento a las denominaciones de dos dólares y mil dólares. Cabe destacar que los especímenes que aquí se definen como “de común circulación” son los cuales presentan una apariencia semejante a las imágenes que se adjuntan en el anexo A2.
- El prototipo del sistema desarrollado no discriminará billetes falsos de verdaderos.
- El alcance del sistema implementado en el prototipo, en lo concerniente a los asuntos técnicos, como el dispositivo móvil en el cual se implementa y las condiciones de funcionamiento del mismo, se detallará en los siguientes capítulos.

CAPÍTULO 2

TEORÍA DEL PROCESAMIENTO DIGITAL DE IMÁGENES

2.1 INTRODUCCIÓN

Dado que los dólares de distinta denominación, como moneda de cambio en nuestro entorno, tienen como característica principal la particularidad que exhiben las imágenes de los billetes que los representan, la cotidiana diferenciación de los mismos es un continuo proceso de visión para las personas facultadas con el sentido de la vista. En el caso particular de las personas que carecen de este don, la visión se transforma en un problema que puede ser resuelto para menesteres específicos, como el reconocimiento de la denominación de billetes, mediante la utilización de la tecnología a través de un sistema electrónico que se encargue de la digitalización de la imagen y el envío de la misma a un sistema microprocesado (computadora personal, microcontrolador, DSP, dispositivos y teléfonos móviles), para su adecuado procesamiento y el posterior análisis que permita su reconocimiento [4].

Esta es la clase de problema que el Procesamiento Digital de Imágenes (PDI) trata de resolver, mediante sus procesos. El PDI es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información [4]. Un sistema como el que se requiere en el presente proyecto, adoptaría esta teoría mediante la implementación de varios de sus procesos, diagramados de forma general en la figura 2.1.

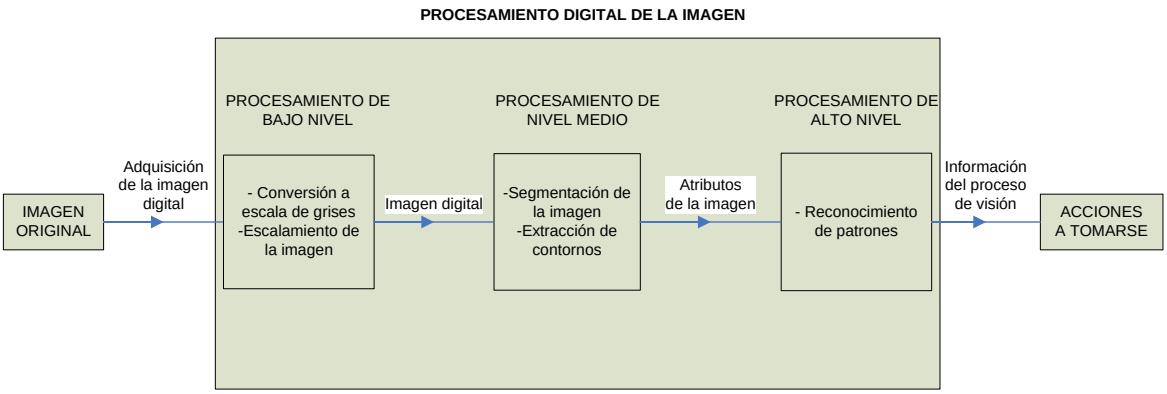


Figura 2.1. Etapas de procesamiento que requiere la imagen del sistema

En el presente capítulo, se generalizarán los conceptos, de la vasta teoría del Procesamiento Digital de Imágenes, utilizados en el desarrollo del proyecto en descripción.

2.2 PROCESOS DE BAJO NIVEL

Dentro del paradigma adoptado en la anterior introducción para la categorización del Procesamiento Digital de Imágenes, los primeros pasos para cualquier tarea de visión por computador se dan mediante los procesos de bajo nivel, mismos que comprenden operaciones primitivas, llamadas también de pre procesamiento de una imagen de entrada para convertirla en una imagen con características más adecuadas para las futuras operaciones que se harán a partir de ella; esto es, un mejoramiento de la imagen original. Una operación de bajo nivel se caracteriza por el hecho de que tanto sus entradas como sus salidas son imágenes [4].

2.2.1 Representación de una imagen en código RGB

Una imagen digital se obtiene mediante un adecuado proceso de adquisición de datos, que captura la información de la imagen original situada en nuestro espacio tridimensional y la entrega en una matriz de valores de dos dimensiones, que representará a la imagen digital [4], [5]. Este proceso es llevado a cabo mediante sistemas de adquisición de imágenes, muchas veces ya incorporados en los sistemas microprocesados; usualmente, son las cámaras digitales. Posteriormente a la adquisición de la imagen digital, esta se representa dentro del sistema espacialmente como una matriz con una resolución de $m \times n$ elementos (m filas y n columnas). En la figura 2.2 se muestran los elementos básicos existentes en un sistema de adquisición de imágenes.

Cada elemento de la matriz que constituye la imagen digitalizada, se conoce como *píxel* y tiene un valor numérico asignado que se corresponde con el nivel de luminosidad del punto correspondiente en la escena captada de la imagen original, usualmente en el rango de 0 a 255 (para sistemas de adquisición que utilizan 8 bits), en el cual, 0 representa al negro absoluto y 255 al blanco absoluto. Esta luminosidad puede ser presentada como un nivel de gris, que también es conocida como intensidad en ese punto, como se muestra en la figura 2.2.

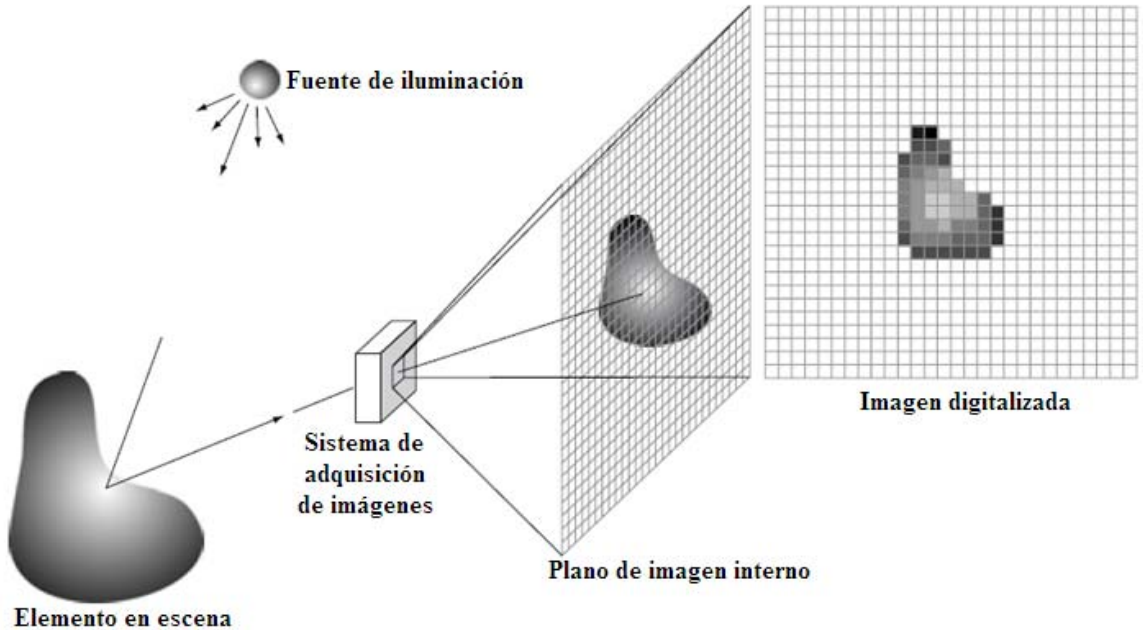


Figura 2.2. Partes básicas de un sistema de adquisición de imágenes

Sin embargo, en muchos sistemas microprocesados actuales con subsistemas de adquisición de imágenes integrado, es común la obtención de la imagen digital como una imagen a color. En este caso, los elementos de la imagen digital vienen dados por tres valores de intensidad, que representan cada uno de los componentes básicos del color en cuestión. Estos componentes son el Rojo (*R*), Verde (*G*) y Azul (*B*), representados con esas letras capitales por sus nombres en inglés: *Red*, *Green*, *Blue*. Este es el conocido código RGB [5].

Según este código, un conjunto *(R,G,B)* de valores *(0,0,0)* es el negro absoluto; el *(255,255,255)*, es el blanco absoluto; el *(255,0,0),* es el rojo puro; el *(0,255,0)*, es el verde puro y el *(0,0,255)*, es el azul puro. La combinación de distintos valores proporciona otros colores. Debido a lo anterior, se dirá que una imagen en color posee tres bandas espectrales: rojo, verde y azul, siendo cada una de ellas una matriz independiente de números, en el rango de 0 a 255, para imágenes que se representan con 8 bits por el método de adquisición [5]. Cabe mencionar que RGB es el código más frecuente para representación de imágenes a color, pero no es el único modelo de color, ya que existen otros, que representan a las imágenes a color de formas distintas a bandas espectrales. Esta representación en cuestión, involucra tres matrices distintas que representan a la imagen digital, cada una con distintos niveles de intensidad representados por distintos valores numéricos.

2.2.2 Conversión de una imagen RGB a escala de grises

La conversión de una imagen RGB a escala de grises trata básicamente de transformar la información de los tres canales de color *(R, G, B)* a un solo canal de intensidad de gris, para el establecimiento de una sola imagen digital en una matriz de valores de intensidad, con información equivalente de la imagen real. No existe una fórmula única para realizar dicha transformación ya que dependerá de la naturaleza del sensor utilizado al obtener la imagen en color. Se presentan a continuación las formas más comunes de conversión de una imagen RGB a escala de grises:

- La fórmula de conversión del estándar NTSC es la fórmula más utilizada para calcular la luminancia efectiva de cada pixel [6]:

$$gris = 0{,}2989\,R + 0{,}5870\,G + 0{,}1140\,B \tag{2.1}$$

- El promedio de los tres canales de color es la manera más sencilla y práctica de obtener los niveles de gris:

$$gris = \frac{R + G + B}{3} \tag{2.2}$$

- Un efecto a menudo presente en imágenes tomadas con cámaras digitales de teléfonos móviles es que los 3 canales de color difieren en su calidad, nitidez y contraste. Debido a dicho efecto, se suele utilizar la fórmula 2.3 para la conversión de RGB a gris [7]:

$$gris = \frac{R + G}{2} \tag{2.3}$$

El valor numérico de estas variables estará en el rango prefijado por el sistema de adquisición de imágenes (*e.g.* si es de 8 bits, el rango es de 0 a 255).

2.2.3 Escalamiento de una imagen

El escalamiento o redimensionado, junto a la traslación y a la rotación, se ubica dentro de las operaciones geométricas elementales de imágenes [5]. Este tipo de transformaciones implican un cambio en la disposición de los píxeles respecto a un sistema de coordenadas, de suerte que el resultado es una transformación de la geometría de la

imagen original. Para su análisis es necesario el establecimiento de una adecuada nomenclatura de la distribución espacial de los píxeles en dos entornos, a los que se llamarán: *malla original* y *malla final;* la primera es la matriz de resolución original $m \times n$ y la segunda, aquella de resolución final $m_f \times n_f$. Estas matrices representan, respectivamente, a la imagen original de entrada de la operación (en este caso, una imagen en escala de grises), y a la imagen resultante de la operación o imagen escalada.

En ambas imágenes, los píxeles, asignados a valores de intensidad de gris, asumen una estructura matricial, siendo identificados espacialmente por cada una de sus coordenadas, referenciadas con índices, que en la malla original se conocerán como *(i, j)* y en la final como *(q, r)*, de la forma en que se muestra en la figura 2.3, que exhibe el ejemplo de la operación de escalamiento de una imagen, de reducción.

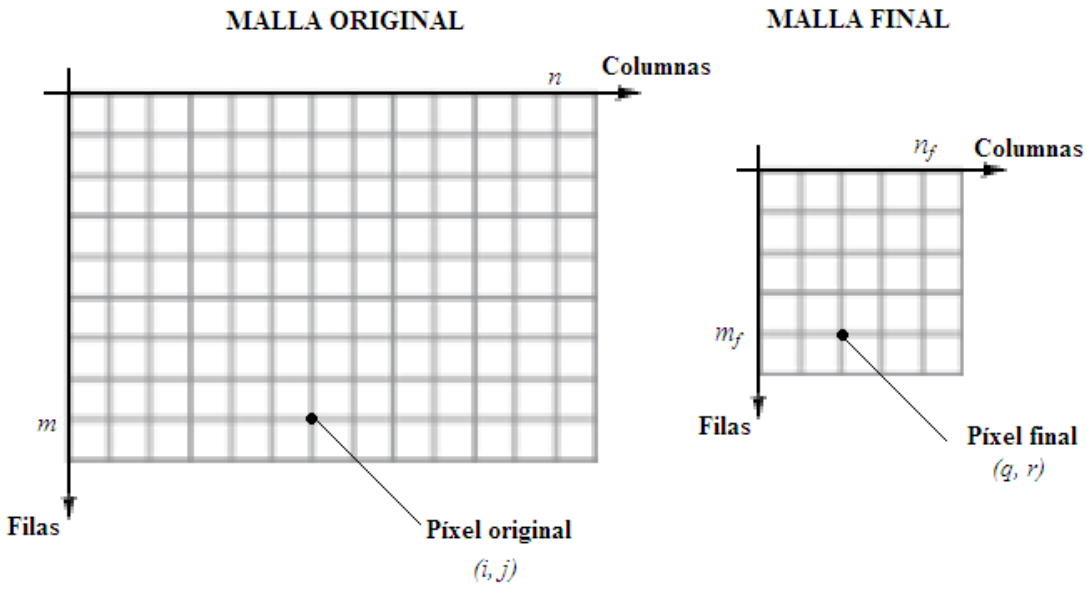


Figura 2.3. Mallas original y final de una operación de escalamiento de reducción

Una operación geométrica cualquiera consiste en la distribución de los valores de intensidad de los píxeles de la malla original en la malla final, mediante una adecuada asignación de nivel de gris a cada píxel de coordenadas *(q, r)* de la malla final, provenientes de ciertas coordenadas *(i', j')* de la malla original, cuyos índices se han seleccionado mediante una transformación geométrica adecuada. En general, este proceso se realiza mediante los siguientes pasos:

- A partir de las coordenadas de cada uno de los píxeles *(q, r)* de la malla final, se determina un conjunto de coordenadas *(i', j')*, que corresponden a los

píxeles seleccionados de la malla original para asignarse a los de la malla final de destino. Esto se realiza mediante una *función de transformación (i', j') = f((q, r))* que es particular de cada distinta operación geométrica. En este caso, la función será una de escalamiento.

- Determinar los valores de las intensidades a ser asignadas a las coordenadas de cada uno de los píxeles en la malla final, *(q, r)*, a partir de los valores de intensidad conocidos, de la malla original, de los píxeles en las coordenadas *(i', j')* antes determinados. Esto es necesario pues, el conjunto de coordenadas *(i', j')* no siempre se corresponde con valores existentes de coordenadas *(i, j)*, de tal forma que se necesita un criterio para la selección de la intensidad de gris a ser asignada en cada coordenada *(q, r)* de la malla final, el más conveniente para mantener la geometría de la operación. Esta operación se conoce como *interpolación*.

La operación de escalamiento en cuestión es una variación del tamaño de la imagen original, que puede realizarse a lo largo de cualquiera de los ejes coordenados de filas y columnas, como los mostrados en la figura 2.3, que pueden ser identificados como eje *x* y eje *y*. De esta forma, el escalado se presenta mediante un factor de escala S_x en la dirección *x* y S_y en la dirección *y*. Cuando el factor toma valores entre *0* y *1*, se produce una reducción de la imagen y, cuando los valores son mayores que *1*, se produce un aumento. Para esta operación, se determina la correspondiente función de transformación antes mencionada, mediante las siguientes ecuaciones:

$$q = S_x i \tag{2.4}$$

$$r = S_y j \tag{2.5}$$

La aplicación de esta ecuación implica la determinación previa de los factores de escala, mediante la relación por cociente existente entre las dimensiones de la malla final, contenedora de la imagen escalada, y la malla original, de la imagen de entrada:

$$S_x = \frac{m_f}{m} \tag{2.6}$$

$$S_y = \frac{n_f}{n} \tag{2.7}$$

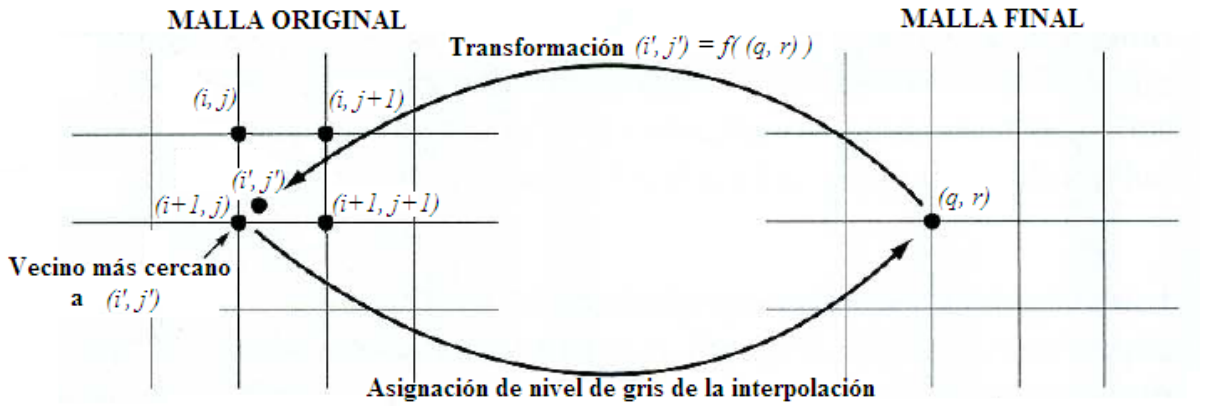


Figura 2.4. Proceso para la asignación de nivel de gris a partir de interpolación

Una vez fijados los factores, se obtiene una imagen reducida (ampliada) por dichos factores. Las dimensiones de la imagen se verán disminuidas (ampliadas) en esa misma proporción. El objetivo, entonces, consiste en obtener los valores de la nueva imagen; para ello, dadas las coordenadas *(i, j)*, se calculan las coordenadas *(q, r)* de cada uno de los píxeles de la malla final, obteniéndose las coordenadas pseudo discretas *(i', j')*, mediante la aplicación de las ecuaciones 2.4 y 2.5, de la siguiente forma: $(i', j') = \left(\frac{q}{S_x}, \frac{r}{S_y} \right)$. De esta manera, se tiene un conjunto de $m_f \times n_f$ coordenadas *(i', j')*, una para cada coordenada *(q, r)* de la malla final, que corresponden a valores de coordenadas en la malla original.

Las nuevas coordenadas *(i' j')*, en cualquier transformación geométrica, pueden ser vistas como coordenadas de píxeles superpuestos en la malla original, que pueden o no coincidir con un valor existente de coordenada *(i, j)*. En vista de que cada coordenada *(i', j')* informa la posición de cada píxel seleccionado de la malla original, cuya intensidad será trasladada a la malla final en la posición correspondiente dada por *(q, r)*, antes establecida, se puede notar que es necesaria la determinación de una valores enteros de coordenadas *(i, j)* correspondientes a cada coordenada *(i', j')* para la selección del píxel adecuado. Esto se logra mediante la operación de interpolación. Esta operación se puede considerar como el cálculo del valor de la intensidad de un píxel, en una posición cualquiera, como una función de los píxeles que le rodean. En este caso, esta operación asignará, en la malla final, al píxel de posición *(q, r)*, un valor de gris en función de los píxeles que rodean al punto de posición *(i', j')* en la malla original, como se muestra en la figura 2.4.

Existen varios tipos de interpolaciones; el utilizado en el ejemplo de la figura 2.4 es la *Interpolación por vecino más próximo*. En este tipo de interpolación, se supone que el píxel, al ser interpolado, toma el mismo valor de intensidad que el más cercano de entre los cuatro que le rodean; asignando, de esta manera, al píxel *(q, r)* de la malla final, el nivel de gris del vecino más cercano al punto *(i', j'),* superpuesto en la malla original; en el caso del ejemplo, se asigna el valor de intensidad de *(i+1, j)* a *(q, r)*. La determinación de este vecino más cercano se la puede realizar en base a ciertos criterios, de los cuales, uno de los más usados es la realización de un redondeo de los valores de las coordenadas *(i', j')*, mediante la relación de redondeo más común:

$$f(x) = \begin{cases} \lfloor x \rfloor, & x < \lfloor x \rfloor + \frac{1}{2} \\ \lfloor x \rfloor + 1, & x \geq \lfloor x \rfloor + \frac{1}{2} \end{cases} ; donde: \lfloor x \rfloor = máx\{k \in \mathbb{Z} \mid k \leq x\} \qquad (2.8)$$

Finalmente, cabe destacar el hecho de que la aplicación de una operación de interpolación en imágenes, no tiene una implementación única, y puede realizarse con métodos más formales (como la aplicación de núcleos matemáticos de interpolación), como con métodos más heurísticos, siempre y cuando se mantenga la filosofía de la asignación de un nivel de gris de la malla original a cada píxel de la malla final, en función de la resolución espacial de los píxeles. La aplicación de estas operaciones sobre una imagen en escala de grises completa las posiciones de la malla final para producir el resultado esperado, la imagen escalada.

2.2.4 Normalización de una imagen

La normalización de una imagen es el proceso que elimina los efectos de eventuales variaciones de iluminación y ruido debidas a las condiciones inherentes a la adquisición de la imagen [8]. Matemáticamente normalizar es transformar las estadísticas de una imagen de modo que su media sea cero y su varianza uno de acuerdo a la ecuación 2.7:

$$g_N = \frac{g - \eta}{\sigma}; \quad donde:$$

$$\begin{aligned} &\eta: \ \textit{Media de la imagen original} \\ &\sigma: \ \textit{Desviación estándar de la imagen original} \\ &g: \ \textit{Matriz de la imagen original} \\ &g_N: \textit{Matriz de la imagen normalizada} \end{aligned} \qquad (2.9)$$

Al calcular la matriz de la imagen normalizada g_N con la ecuación 2.9, los valores de dicha matriz suelen ser números decimales cercanos a uno e incluso negativos; se puede notar que estos valores, en el dominio de las imágenes digitales, no tienen mucho sentido de ser, teniendo en cuenta que una imagen por lo general solo toma valores enteros entre *0* y *255*. Por este motivo, la matriz de la imagen normalizada debe ser considerada como cualquier señal en dos dimensiones ya que el concepto de imagen como representación visual pierde su sentido.

Adicionalmente, la naturaleza decimal de la matriz g_N es un problema al momento de implementar el proceso de normalización en plataformas de *hardware* sin unidades de procesamiento de punto flotante o con poca eficiencia de este al momento de realizar operaciones con decimales. A fin de superar este inconveniente, se suele modificar g_N de manera que la media y varianza tomen valores típicos de una imagen (*e.g.* media *100*, varianza *80*); con esto se logra que el rango de g_N sea lo suficientemente amplio para poder redondear o truncar sus valores sin incurrir en un error como el que se cometería al truncar una imagen g_N con media cero y varianza uno. La ecuación que permite realizar este ajuste es la siguiente:

$$g_N = \frac{g-\eta}{\sigma}\sigma_N + \eta_N \;;\quad donde:$$

η: *Media de la imagen original*
σ: *Desviación estándar de la imagen original*
g: *Matriz de la imagen original* (2.10)
η_N: *Media deseada de la imagen*
σ_N: *Desviación estándar deseada de la imagen*
g_N: *Matriz de la imagen normalizada*

En el desarrollo de este estudio, la fórmula 2.10 se adopta como la ecuación para la implementación de la normalización. Para finalizar, la figura 2.5 muestra el efecto de normalizar una imagen bajo dos condiciones de iluminación diferentes.

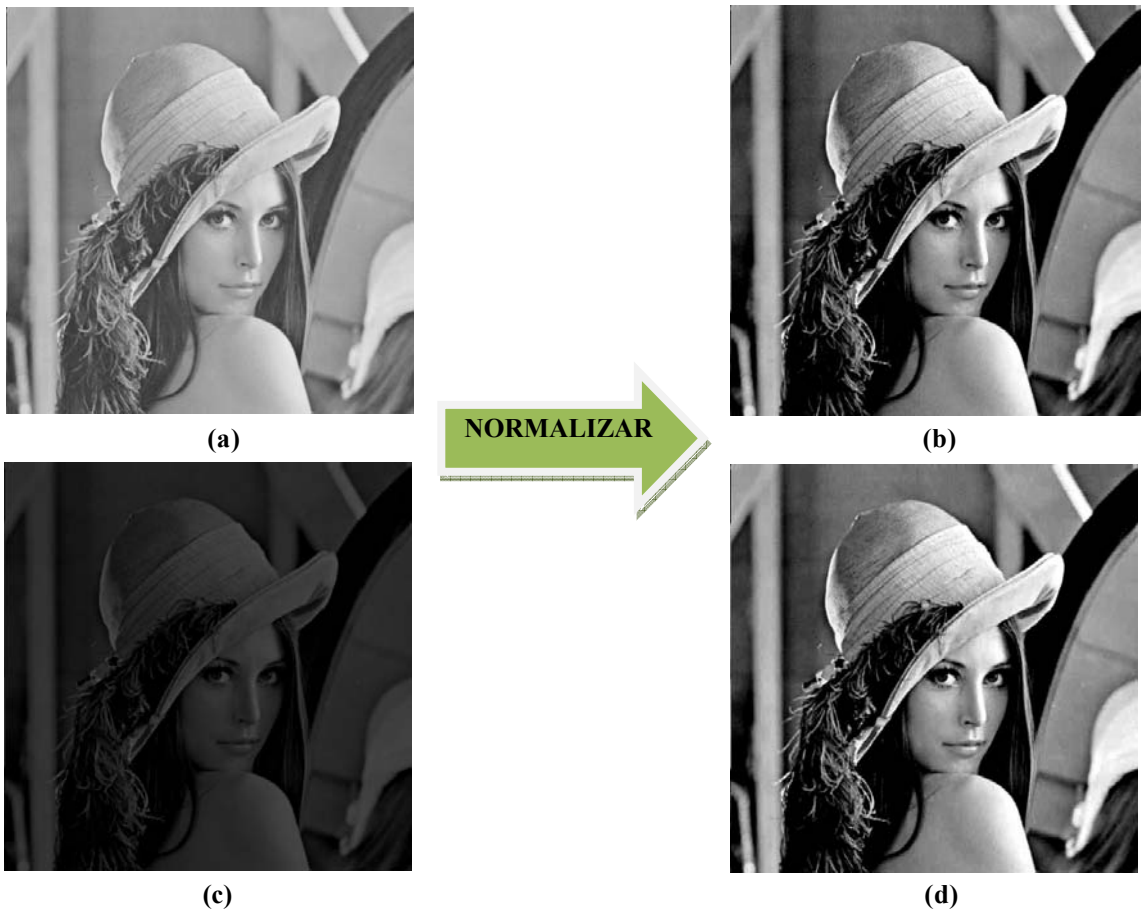


Figura 2.5. Efecto del proceso de normalización

(a) Imagen con iluminación alta (b) Normalización de la imagen con iluminación alta

(c) Imagen con iluminación baja (d) Normalización de la imagen con iluminación baja

2.3 PROCESOS DE NIVEL MEDIO

En la teoría del Procesamiento Digital de Imágenes, una vez que los procesos más básicos de pre procesamiento de una imagen en estudio se han encargado de producir una versión lo suficientemente apropiada de la imagen digital inicialmente obtenida para un análisis más profundo, se somete a la misma a diversos procesos más complejos, los de nivel medio. Los procesos de nivel medio sobre imágenes implican tareas tales como la segmentación de una imagen para la división de la misma en los objetos o regiones que la componen así como la descripción de aquellos objetos para una reducción apropiada de la información para un posterior procesamiento matemático. Un proceso de nivel medio se caracteriza por el hecho de que sus entradas son generalmente imágenes, pero sus salidas

son ahora atributos extraídos de esas imágenes [4]. Algunos de estos procesos se detallarán en esta sección.

2.3.1 Umbralización de una imagen

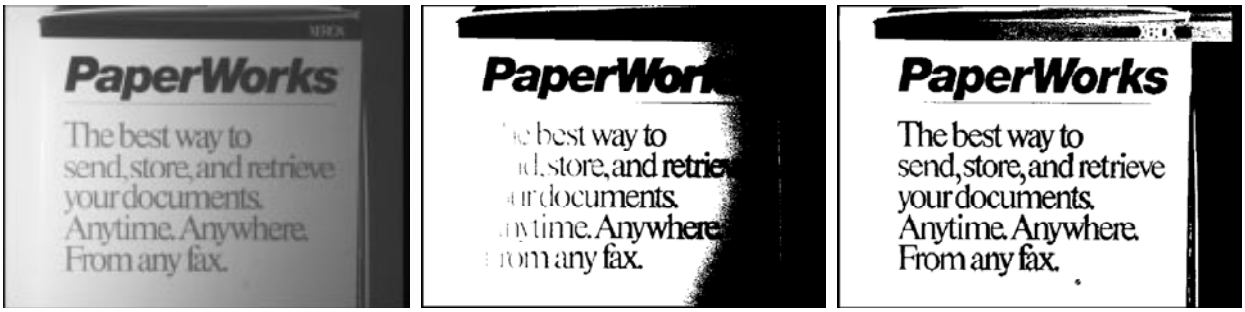
2.3.1.1. Comparación entre los métodos de umbralización

La umbralización o binarización suele ser uno de los primeros pasos en muchos de los algoritmos de análisis de imágenes. El objetivo de binarizar una imagen es convertir dicha imagen a una representación de dos niveles, por ejemplo, una imagen en blanco y negro.

En el caso que compete al presente estudio, en imágenes en tonos de gris, cada pixel es comparado con un umbral; de acuerdo a la manera en que dicho umbral es calculado, las técnicas de binarización pueden ser divididas en dos grandes categorías: binarización global y binarización local o adaptativa. La tabla 2.1 resume las principales características de ambos tipos de binarización, mientras que la figura 2.6, a manera de ilustración, muestra el resultado de aplicar diferentes tipos de umbralización a una imagen.

Tabla 2.1. Características de la binarización global y local

Binarización	Umbral	Robustez frente a iluminación no uniforme	Velocidad de procesamiento	Métodos
Global	Umbral único	Baja	Rápido	*Otsu* [9]
Local	Umbral calculado de acuerdo a la vecindad del pixel	Alta	Lento	*Wellner* [10] *Sauvola* [11] Media local [12]



(a) **(b)** **(c)**

Figura 2.6. Umbralización de una imagen usando diferentes métodos
(a) Imagen Original (b) Binarización global (c) Binarización Local

2.3.1.2. Método de umbralización adaptativa usando la media local de una imagen

Si se considera una imagen en gris en la cual $g(x,y) \in [0,255]$ es la intensidad de un pixel en la posición (x,y), de manera general, en métodos de umbralización adaptativa, el objetivo es calcular un umbral $t(x,y)$ para cada pixel de forma que la imagen binarizada $b(x,y)$ sea:

$$b(x,y) = \begin{cases} 0\,, & g(x,y) \le t(x,y) \\ 1\,, & en\ otro\ caso \end{cases} \tag{2.11}$$

En el presente método, el umbral $t(x,y)$ es calculado usando la media $\eta(x,y)$ de las intensidades de los pixeles en una ventana de un tamaño $W \times W$ cualquiera, centrada en el pixel de posición (x,y):

$$t(x,y) = \tau\,\eta(x,y) \tag{2.12}$$

Donde τ es un parámetro de ponderación que depende de la aplicación. Por lo general toma valores entre 0 y 1.

Usar la media local ponderada de la ecuación 2.12 da buenos resultados inclusive en imágenes degradadas y con iluminación no uniforme bastante marcada. Sin embargo, para calcular el umbral local $t(x,y)$, es necesario calcular la media local $\eta(x,y)$ para cada píxel. Obtener $\eta(x,y)$ tiene una complejidad computacional de $O(W^2N^2)$ para una imagen de $N \times N$ pixeles con una ventana de tamaño $W \times W$ [12]. Esta complejidad impone verdaderos problemas en aplicaciones que requieren ser procesadas en tiempo real. Para lograr acelerar el tiempo de procesamiento de manera que no dependa del tamaño de la ventana, *Bradley et al* proponen una forma eficiente de calcular la media local usando la integral de una imagen como se describirá en el siguiente inciso [12].

2.3.1.3. Integral de una imagen para calcular medias locales

El concepto de integral de una imagen fue popularizado en el campo de la visión computarizada por *Viola y Jones* [13]. La integral $\zeta(x,y)$ de una imagen de entrada g está definida como una imagen en la cual la intensidad en la posición (x,y) es igual a la suma de las intensidades de todos los pixeles ubicados arriba y a la izquierda de dicha posición

en la imagen original [12]. Por lo tanto, la intensidad de la integral de una imagen en la posición (x, y) puede ser matemáticamente escrita como:

$$\zeta(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{y} g(i,j) \tag{2.13}$$

Para comprender de mejor manera este concepto, la figura 2.7 muestra un ejemplo práctico de una imagen y su integral. En este ejemplo, en la posición *(x = 1, y = 2)*, la integral de la imagen es igual a *14 + 2 + 6 + 0 + 11 + 5 = 38*. En la posición *(x = 1, y = 1)*, la integral es igual a 6 + 0 + 11 + 5 = 22.

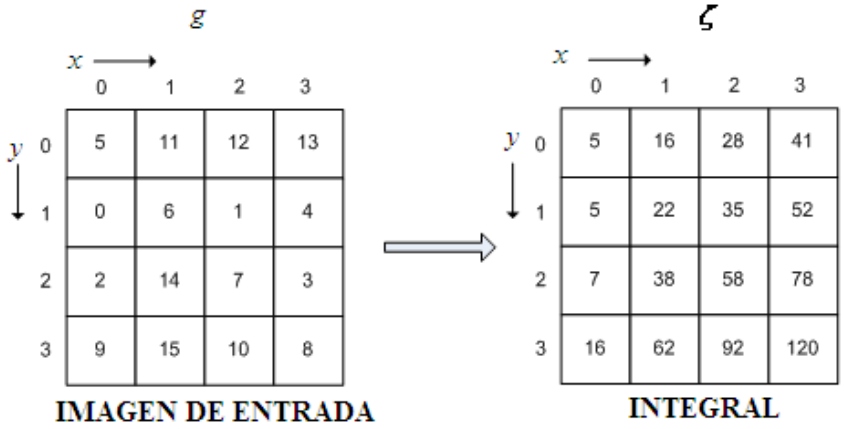


Figura 2.7 Integral de una imagen de 4×4 píxeles

Una vez que se obtiene la integral de la imagen, la media local $\eta(x,y)$ puede ser calculada simplemente usando una suma y dos restas para cualquier tamaño de ventana W [12]:

$$\eta(x,y) = \frac{1}{W^2}\left(\zeta\left(x+\frac{W}{2}, y+\frac{W}{2}\right) + \zeta\left(x-\frac{W}{2}, y-\frac{W}{2}\right) - \zeta\left(x+\frac{W}{2}, y-\frac{W}{2}\right) - \zeta\left(x-\frac{W}{2}, y+\frac{W}{2}\right)\right) \tag{2.14}$$

Utilizando la fórmula de la ecuación 2.12 se reduce la complejidad computacional de $O(W^2N^2)$ a $O(N^2)$ [12].

El método de umbralización expuesto anteriormente se puede encontrar en mayor detalle en la referencia [14]. De otra parte, un desarrollo matemático más extenso sobre la integral de una imagen puede ser encontrado en [13].

2.3.2 Erosión de una imagen

La erosión es un proceso que pertenece a la categoría de las operaciones morfológicas sobre imágenes, procedimientos conocidos porque simplifican las imágenes y aún así, preservan las formas principales de los objetos [5]. Una aplicación específica de este tipo de operaciones es el suavizado de los bordes de una región. Esto es útil pues, con las técnicas de umbralización, en general, los bordes se presentan algo ruidosos, apareciendo sobre ellos pequeños valles o protuberancias. Estos pueden suprimirse mediante transformaciones morfológicas [5]. Como se puede apreciar, estas técnicas van de la mano con los procesos de nivel medio de segmentación de una imagen.

2.3.2.1 Principio de una transformación morfológica

La morfología matemática, supone que las imágenes reales pueden ser modeladas utilizando conjuntos de puntos tales como los elementos del conjunto E^2, que es el *espacio euclídeo* 2D. El tratamiento en un sistema microprocesado usa el equivalente digital a este espacio, conjuntos cuyos elementos son pares de números enteros, en el contexto de las imágenes *binarias*, que corresponden a las coordenadas de cada píxel de la imagen digital [5].

Así, una imagen binaria es un conjunto de puntos o pares *(i, j)*. En la representación de la imagen binaria, en la figura 2.8, el origen, marcado con un asterisco (*) tiene coordenadas *(0,0)* y las coordenadas de cualquier otro punto se interpretan como una posición en fila y columna respecto del origen. En esta figura, y para el estudio de esta sección, se considera que los puntos que pertenecen a un objeto dentro de la imagen binaria se marcan con *"1"*, siendo el fondo de la imagen lleno de valores de *"0"*.

$$D=\begin{bmatrix} *0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.8. Ejemplo de conjunto de puntos *D*

En la figura 2.8, el conjunto de puntos *D* considera exclusivamente los valores lógicos 1, y viene dado por:

$$D = \{(0,1),(0,2),(0,3),(1,1),(1,2),(1,3),(2,2),(2,3),(3,2),(3,3)\}.$$

Una transformación morfológica $\phi(D)$, viene dada por la relación del conjunto de puntos *D* (la imagen) con otro pequeño, *B*, llamado *elemento estructural*, que se expresa con respecto a un origen local *O*, llamado punto representativo.

Un elemento estructural comúnmente utilizado en operaciones morfológicas para filtrado es el conjunto [4]:

$$B = \{(-1,-1),(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)\}$$

Representado en la figura 2.9 en la cual el punto representativo se marca por un asterisco (*).

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & *1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 2.9. Elemento estructural B comúnmente usado en operaciones morfológicas

La transformación morfológica $\phi(D)$, aplicada a la imagen *D*, significa que el elemento estructural *B* se desplaza por toda la imagen. Suponiendo que *B* se posiciona sobre algún punto de la imagen, el píxel de la imagen correspondiente al punto representativo *O*, de *B*, se denomina píxel actual. El resultado de la relación entre *D* y *B* (propia de cada operación morfológica distinta) puede ser cero o uno, valor que se almacena en la posición del píxel actual, en la imagen nueva que se produce.

2.3.2.2 Definición y efectos de la erosión

La transformación morfológica de la erosión, con su operador simbolizado por $\otimes$, combina a los conjuntos *D* y *B* utilizando la relación:

$$D \otimes B = \left\{ \vec{d} \in E^2 \mid \vec{d} + \vec{b} \in D, \forall \vec{b} \in B \right\}; \quad donde: \tag{2.15}$$

$\vec{d}$: *Vector de posición de los "unos" de la imagen D*

E^2: *Espacio euclídeo 2D*

$\vec{b}$: *Vector de posición de los "unos" del elemento estructural B*

La fórmula 2.15 expresa que cada par *(i, j)* del conjunto de la imagen, *D*, que contiene a algún objeto, es decir, que tiene un valor de intensidad de *"1"*, y que es representado por el vector de posición $\vec{d}$, es comprobado. El resultado son los puntos de la imagen para los cuales todos los posibles $\vec{d}+\vec{b}$ están en *D*, siendo los pares representados por el vector $\vec{b}$, los puntos con valores de intensidad *"1"* del elemento estructural, *B*. A continuación, se muestra un ejemplo del conjunto de puntos *D* erosionados por un elemento estructural *B* cualquiera:

$$D = \{(0,2),(1,2),(2,0),(2,1),(2,2),(2,3),(3,2),(4,2)\}$$
$$B = \{(0,0),(0,1)\}$$
$$D \otimes B = \{(2,0),(2,1),(2,2)\}$$

Aquí, se puede apreciar que, por ejemplo, para el punto $\vec{d}$ = *(0,2)* de *D*, todos los posibles $\vec{d}+\vec{b}$ son: *(0,2) + (0,0) = (0,2)* y *(0,2) + (0,1) = (0,3)*; de los cuales *(0,3)* no es un elemento de *D*, por lo que el par *(0,2)* no se incluye en el conjunto de respuestas de la erosión. En cambio, para el punto *(2,1)*, de *D*, todos los posibles $\vec{d}+\vec{b}$ son: *(2,1) + (0,0) = (2,1)* y *(2,1) + (0,1) = (2,3)*; los cuales, ambos son elementos de *D*, por lo que el par *(2,1)* se incluye como un elemento del conjunto de respuestas de la erosión.

Gráficamente, la erosión se realiza de la siguiente forma: Se recorre la imagen, *D*, de una dirección a otra (por ejemplo, de izquierda a derecha y de arriba abajo) y allí donde se encuentre un uno, se sitúa el origen del elemento estructural sobre dicho uno; si todos los unos del elemento estructural, al ser superpuestos con la periferia de la imagen original, coinciden con los unos de la imagen, se marca el píxel de la imagen donde está el origen del elemento estructural con el valor de uno. A continuación se puede apreciar el mismo ejemplo anterior, de una manera gráfica:

$$D \otimes B = \begin{bmatrix} *0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} *1 & 1 \end{bmatrix} = \begin{bmatrix} *0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Los efectos de la erosión sobre una imagen pueden apreciarse en la figura 2.10 tomada de la referencia [4], en la cual se aplica esta operación con el elemento estructural

B de la figura 2.9, sobre la imagen binaria de una huella digital como parte de un proceso de filtrado de la imagen.

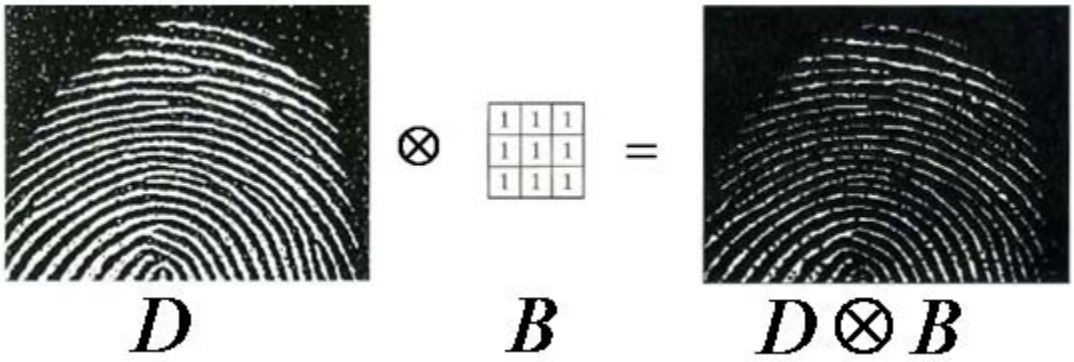


Figura 2.10. Efectos de la erosión

Se observa que, en la imagen original, *D*, se halla ruido manifestado como elementos claros sobre el fondo oscuro, así como elementos oscuros sobre las porciones claras de la imagen binaria. En la imagen erosionada se aprecia la eliminación del ruido del fondo (las porciones claras de ruido). Esto sucede pues todos los componentes claros (puntos de valor "uno") de este tipo de ruido son físicamente más pequeños que el elemento estructural utilizado (matriz 3×3 de puntos de valor "uno"), según se puede entender de la definición de la erosión. Sin embargo, también se puede apreciar que el tamaño de los elementos oscuros de ruido sobre las porciones claras de la imagen, incrementan su tamaño [4].

Se puede notar aquí, la importancia del elemento estructural utilizado y la razón por la cual es altamente común en operaciones de filtraje de ruido. Adicionalmente a su eficacia, este elemento estructural incrementa la prestación de la erosión, pues el hecho de que los ocho píxeles de la periferia de *B* deban comprobar a los píxeles de la misma periferia en cada punto de la imagen *D*, hace que se pueda arreglar como una operación lógica sencilla, como se verá en el siguiente inciso.

2.3.2.3 Erosión mediante el uso de los 8 vecinos cercanos

Un pixel *p*, de coordenadas *(i, j)*, tiene un conjunto de 8 vecinos, ubicados en sus extremos horizontales y verticales y en sus diagonales, cuyas coordenadas vienen dadas como se muestra en la figura 2.11.

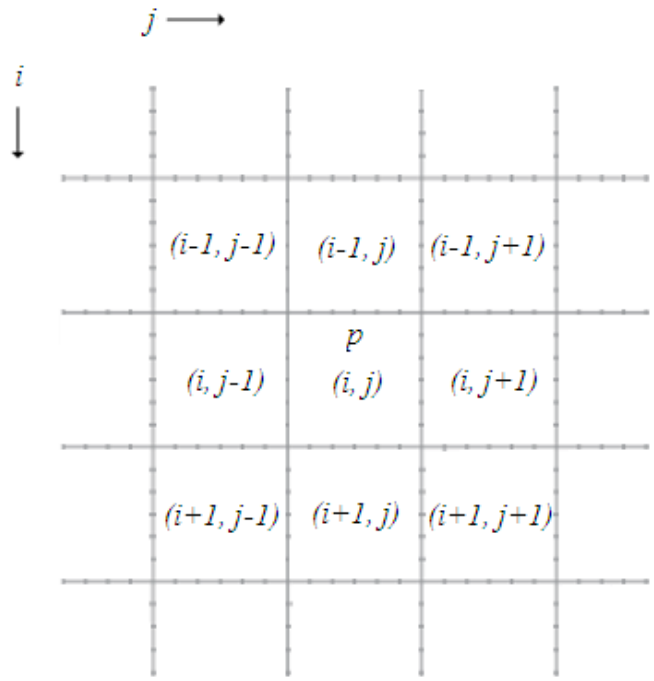


Figura 2.11. 8 vecinos cercanos de un píxel *p*

Este conjunto de píxeles, llamado los 8 vecinos cercanos de *p*, se nota por $N_8(p)$. Cada píxel dista la unidad de *(i, j)*, y algunos de los vecinos de *p* caen afuera de la imagen digital, si *(i, j)* está en el borde de la imagen [4].

Según el método gráfico de implementación de la operación de erosión enunciado en la sección 2.3.2.2, si a una imagen binaria *D*, se la recorre y allí donde haya un uno se sitúa al origen del elemento estructural *B* de la figura 2.9, para que se pueda asignar un uno al píxel de la imagen donde está el origen del elemento estructural en cuestión, todos los píxeles de la periferia del punto de la imagen *D* deben ser unos. Esta periferia es justamente $N_8(p)$, del píxel *p* en el que se ha situado el origen de *B*.

Como se puede apreciar, al utilizar este común elemento estructural, la erosión se convierte en una simple operación lógica aplicada a todos y cada uno de los píxeles de la imagen binaria original *D*, para producir los píxeles correspondientes en la imagen erosionada, *F*, dada mediante la fórmula 2.16:

$$p_F = p_D \,\&\, N_8(p_D) \tag{2.16}$$

Donde p_D es el valor lógico de cada píxel de la imagen binaria original, en la posición *(i, j)*; p_F es el valor lógico de cada píxel de la imagen erosionada, en esa misma

posición *(i, j)*; el símbolo *&* simboliza la operación lógica AND y la variable $N_8(p_D)$ se entiende aquí como una operación AND combinada entre los 8 vecinos cercanos del píxel p_D. Se debe advertir que esta forma de implementación para la erosión sólo será posible con el uso del elemento estructural *B* dado en la figura 2.9.

2.3.3 Perfil de proyección de una imagen

El perfil de proyección de una imagen es una característica muy utilizada en reconocimiento de patrones de escritura como los algoritmos expuestos en [15] y [16]. Para entenderlo se debe considerar una imagen binaria $g(Q,R)$ de Q filas y R columnas donde se define lo siguiente [15]:

- **Perfil vertical:** suma de todos los pixeles negros perpendiculares al eje *y*; Este perfil es representado por un vector P_V de tamaño Q definido como:

$$P_V[i] = \sum_{j=1}^{Q} g(i,j) \tag{2.17}$$

- **Perfil horizontal:** suma de todos los pixeles negros perpendiculares al eje *x*; Este perfil es representado por un vector P_H de tamaño R definido como:

$$P_H[j] = \sum_{i=1}^{R} g(i,j) \tag{2.18}$$

La figura 2.12 presenta un ejemplo de perfil vertical y horizontal. Es importante aclarar que las definiciones 2.17 y 2.18 asumen que los pixeles de interés son de color negro como en el caso del ejemplo. Si no fuese así, las ecuaciones 2.17 y 2.18 deben considerar la suma de todos los pixeles blancos en lugar de pixeles negros.

Figura 2.12. Perfil de proyección vertical P_V y horizontal P_H de una imagen

2.4 PROCESOS DE ALTO NIVEL

El espectro del procesamiento digital de una imagen, dentro del paradigma adoptado en la definición de este capítulo, finaliza con los procesos de alto nivel que, de forma básica, son técnicas de reconocimiento de objetos en imágenes. Estos procesos se encargan de dar sentido a un conjunto de objetos individuales previamente identificados como tales. Su aplicación ulterior es la realización de las funciones cognoscitivas normalmente asociadas con la visión [4] por parte del sistema electrónico que recibe la información resultante de uno de estos procesos. Al llegar al nivel más alto de procesamiento, las técnicas utilizadas son de una complejidad notable, siendo muchas de ellas basadas principalmente en la clasificación de patrones en las imágenes. En la presente sección se detallará el *Análisis de Componentes Principales*, una de las técnicas más comunes para hallar patrones en datos de dimensión elevada [17], como lo son las imágenes.

2.4.1 Introducción al Análisis de Componentes Principales

El Análisis de Componentes Principales, PCA (por su nombre en inglés, *Principal Component Analysis*), es una técnica estadística para la identificación de patrones en conjuntos de datos, que permiten expresar los conjuntos de datos de una forma tal que se resaltan sus similaridades y diferencias [18]. Dado que los patrones, en conjuntos de datos aleatorios de gran dimensión, en donde ya no se tiene la prestación de la representación gráfica, son difíciles de relacionar, PCA se convierte en una herramienta poderosa de análisis, ya que provee la gran ventaja de que es capaz de comprimir los datos, una vez hallados sus patrones, reduciendo su número de dimensiones, sin mucha pérdida de información. PCA en la actualidad es en realidad una herramienta estándar en el análisis de todo tipo de datos, aplicada en diversos campos, desde neurociencia hasta visión por computador, al ser un método simple para la extracción de información relevante a partir de complejos y confusos conjuntos de datos, que con mínimo esfuerzo de cálculo ofrece un método para revelar las muchas veces escondidas estructuras simplificadas que yacen debajo [19].

De la presente introducción se puede inducir la relación que hay entre el PCA y el reconocimiento de imágenes. Una aproximación inicial con respecto a esta analogía se establece al notar que las imágenes, en un sentido, no son más que conjuntos de datos de

las intensidades de sus píxeles; números que encierran las características de los objetos que contienen las imágenes que representan. Si, en este punto, se considera el reconocimiento de una imagen como el peso de un resultado de una comparación entre la imagen de entrada de un sistema y una imagen de muestra, esta operación dependerá de los patrones que exhiban ambas imágenes y será más o menos pesada según las resoluciones espaciales de las imágenes (el número de píxeles). La aplicación de los principios del PCA en este ámbito se ve entonces más claramente.

2.4.2 Preliminares matemáticos

2.4.2.1. Media, desviación estándar y varianza

La media y la desviación estándar son herramientas estadísticas que sirven para analizar las relaciones existentes entre los elementos de un conjunto de datos. En este contexto se puede definir a la media estadística como una medida de tendencia central resultado de una serie determinada de operaciones con un conjunto de datos y que, en determinadas condiciones, puede representar por sí sola a todo el conjunto [20]. Existen distintos tipos de medias, tales como la media geométrica, la media ponderada y la media aritmética; sin embargo la que interesa al presente estudio es la media aritmética, promedio o simplemente media. De manera más formal, la media aritmética, de un conjunto finito X_i, es igual a la suma de todos sus valores dividida entre el número de elementos n del conjunto:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{2.19}$$

Expresada de forma más intuitiva, se puede decir que la media (aritmética) es la cantidad total de la variable distribuida a partes iguales entre cada observación. También la media aritmética puede ser denominada como centro de gravedad de una distribución, el cual no es necesariamente la mitad. Una de las limitaciones de la media es que se ve afectada por valores extremos; valores muy altos tienden a aumentarla mientras que valores muy bajos tienden a bajarla, lo que implica que puede dejar de ser representativa de la población.

Por ejemplo, los siguientes conjuntos de datos tienen exactamente la misma media, *10*, pero son totalmente diferentes: [*0 8 12 20*] y [*8 9 11 12*]. Es aquí donde la desviación estándar marca la diferencia entre ambos conjuntos ya que mide qué tan dispersos están los datos. Formalmente la desviación estándar σ se define como la media aritmética de las desviaciones de cada elemento X_i de un conjunto X con respecto a la media $\overline{X}$ de dicho conjunto. Matemáticamente se expresa con la fórmula 2.20:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2}{n-1}} \tag{2.20}$$

La ecuación 2.20 puede ser modificada dividiendo para n en lugar de $n-1$ cuando el tamaño del conjunto de datos sea igual a la población. Un análisis profundo de las diferencias entre ambas formas de cálculo de la desviación estándar se presenta en [21]. De otra parte, existe una ecuación de la desviación estándar, la usada en este proyecto, que resulta mucho más sencilla al momento de implementar:

$$\sigma = \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n} X_i^{\,2}\right) - \overline{X}^2} \tag{2.21}$$

Finalmente, a partir del concepto de desviación estándar, se define otra manera de determinar qué tan dispersos están los elementos de un conjunto de datos; este concepto es la varianza que no es más que la desviación estándar al cuadrado:

$$\mathrm{var}(X) = \sigma^2 = \frac{1}{n}\left(\sum_{i=1}^{n} X_i^{\,2}\right) - \overline{X}^2 \tag{2.22}$$

2.4.2.2 Covarianza y matriz de covarianza.

Las herramientas estadísticas de la sección 2.4.2.1 son calculadas netamente en conjuntos de datos de una sola dimensión (*e.g.* estaturas de un conjunto de personas en una clase o notas de un grupo de personas en su examen final). Sin embargo, muchos conjuntos de datos tienen más de una dimensión, en cuyo caso, el análisis estadístico trata usualmente de encontrar las relaciones entre dichas dimensiones. Por ejemplo, se puede tener un conjunto de datos en 2 dimensiones, como las estaturas de un grupo de personas y las notas de su examen final. A partir de dicho conjunto, se puede establecer estadísticamente si la

estatura de un estudiante tiene efecto en sus notas. Recordando que la desviación estándar y la varianza operan solamente en una dimensión, solo se podría calcular dichas estadísticas para cada dimensión independientemente una de la otra. Sin embargo, es más útil tener una medida que relacione cómo varía una dimensión respecto de la otra; es precisamente la covarianza dicha medida.

La covarianza se mide siempre entre 2 dimensiones. Es decir, si se tiene un conjunto de datos en 3 dimensiones (X,Y,Z), entonces se podría calcular la covarianza entre las dimensiones (X,Y), (Y,Z) y (X,Z). Si se mide la covarianza en una misma dimensión, o sea, la covarianza entre las dimensiones (X,X), (Y,Y) y (Z,Z) se obtiene las varianzas de X, Y, Z respectivamente. Matemáticamente, la covarianza entre dos dimensiones X e Y, cada una con n elementos, se calcula con la ecuación 2.23 [18]:

$$\operatorname{cov}(X,Y)=\sigma^2_{XY}=\frac{\sum_{i=1}^{n}\left(X_i-\overline{X}\right)\left(Y_i-\overline{Y}\right)}{n-1} \tag{2.23}$$

El valor absoluto entregado por la ecuación 2.23 mide el grado de redundancia de un conjunto de datos mientras que su signo evalúa la proporcionalidad de la razón entre ambas variables. Es así que se puede afirmar lo siguiente:

- Si $\sigma^2{}_{XY}>0$: hay dependencia directa (positiva), es decir, al crecer X, Y también crece.
- Si $\sigma^2{}_{XY}=0$: Una covarianza 0 se interpreta como la no existencia de una relación lineal entre las dos variables estudiadas.
- Si $\sigma^2{}_{XY}<0$: hay dependencia inversa o negativa, es decir, al crecer X, Y decrece.

Como la covarianza se mide entre dos dimensiones, si se tiene un conjunto de datos de dimensión 3 o superior, se puede medir más de una covarianza. Por ejemplo, para un conjunto de 3 dimensiones (X,Y,Z), es posible calcular $\sigma^2{}_{XY}$, $\sigma^2{}_{XZ}$ y $\sigma^2{}_{YZ}$. De hecho, para un conjunto de datos de dimensión *n*, se puede obtener $\frac{n!}{2(n-2)!}$ diferentes valores de covarianza [18].

Una manera útil de representar todos los posibles valores de covarianza es mediante una matriz. De esta manera, la ecuación 2.24 define la matriz de covarianza como una matriz C perteneciente al espacio vectorial de todas las matrices cuadradas $n \times n$ de números reales, tal que un elemento de ella, $c_{i,j}$ es igual a la covarianza entre *i* e *j* [18]:

$$[\mathrm{C}]_{\mathrm{n}\times\mathrm{n}} = \left\{\mathrm{C} \in \mathcal{M}^{\mathbb{R}}_{\mathrm{n}\times\mathrm{n}} | c_{i,j} = \sigma^2_{ij}, c \in C\right\} \tag{2.24}$$

Para comprender de una mejor manera qué representa la matriz de covarianza, se puede considerar el siguiente ejemplo de un conjunto en 3 dimensiones (X,Y,Z):

$$C = \begin{pmatrix} \sigma^2{}_{XX} & \sigma^2{}_{XY} & \sigma^2{}_{XZ} \\ \sigma^2{}_{YX} & \sigma^2{}_{YY} & \sigma^2{}_{YZ} \\ \sigma^2{}_{ZX} & \sigma^2{}_{ZY} & \sigma^2{}_{ZZ} \end{pmatrix} \tag{2.25}$$

Cabe recalcar que la diagonal principal de la matriz de covarianza contiene los valores de las varianzas de cada dimensión. Además, como $\sigma^2{}_{XY} = \sigma^2{}_{YX}$, la matriz de covarianza es simétrica a lo largo de su diagonal principal.

2.4.2.3 Valores y vectores propios

En la teoría del Álgebra Lineal, se presenta el problema de una transformación lineal de un espacio vectorial en sí mismo $f : V \to V$, en el cual, se desean conocer aquellos vectores que tienen la misma dirección que su transformación por f; es decir, se buscan vectores no nulos tales que u y $f(u)$, para un cierto escalar λ, se verifique la relación: $f(u) = \lambda u$. A estos vectores u se les conoce como *vectores propios* o vectores característicos de la transformación lineal f y a los escalares λ se les da el nombre de *valores propios* o valores característicos de la transformación lineal f [22]. En jerga matemática, también se les conoce por sus anglicismos, *eigenvector* y *eigenvalue*.

En la figura 2.13, se muestra el resultado de la aplicación de la transformación lineal, en su forma matricial, $f(u) = Tu$ tanto a un vector no propio, $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$, como a un vector propio, $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ de esa transformación, mediante la multiplicación de la matriz de la

transformación lineal T, $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ por cada uno de estos vectores. En el primer ejemplo, el vector resultante no es un múltiplo entero del vector original, mientras que en el segundo ejemplo, el vector resultante es 1 vez el vector de entrada, ya que es el mismo.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 1\begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

(b)

Figura 2.13. Ejemplo de la transformación lineal aplicada a: (a) Un vector no propio (b) Un vector propio

El vector del segundo ejemplo es un elemento del espacio vectorial $\mathbb{R}^2$, el plano en dos dimensiones, en el cual, es una flecha que apunta desde el origen al punto *(2, 2).* La transformación lineal que se ha seleccionado en este ejemplo, representada por la matriz T, es aquella que refleja a los vectores de entrada en el eje $y = x$, como lo indica la figura 2.14, en la que se muestran los vectores de entrada u y w, así como sus correspondientes vectores de salida v, y z, de la transformación lineal del ejemplo anterior.

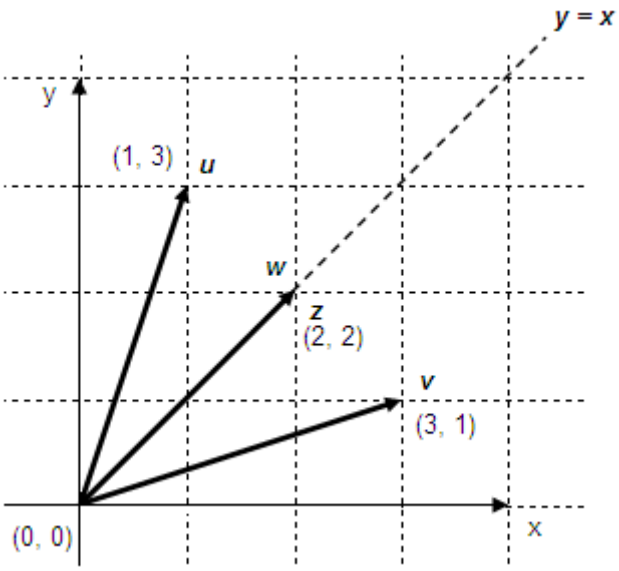


Figura 2.14. Representación gráfica de las transformaciones lineales del ejemplo de la figura 2.13

Mediante este ejemplo gráfico en el sencillo espacio 2D, se puede entender el concepto tras los vectores propios. Se aprecia que para cualquier vector que se encuentre en el eje $y = x$, su transformación de reflexión es un múltiplo de él mismo (en este caso, una vez él mismo), tal como sucede con el segundo vector de ejemplo; entonces, se pueden generalizar a todos los vectores que se encuentren en el eje $y = x$ como todos los vectores αw, donde α es cualquier escalar y w puede ser cualquier vector que yace en el eje $y = x$, como el segundo vector del ejemplo anterior. De esta forma, el vector w se convierte uno en el que se tipifican todos sus múltiplos, porque no importa la longitud del vector, siempre yacerá en el eje $y = x$. El vector w es, entonces, un vector propio de la transformación lineal de la reflexión. Los vectores propios tienen las siguientes propiedades importantes [18]:

- Sólo pueden ser hallados en transformaciones lineales con matrices asociadas cuadradas y no toda matriz cuadrada tiene vectores propios. Dada una matriz *n* x *n* con vectores propios, existen *n* de ellos.
- Incluso si se escala el tamaño de un vector propio antes de la multiplicación por su transformación lineal, se obtendrá el mismo múltiplo del vector de entrada como resultado a la salida.
- Los vectores propios asociados a una transformación lineal son ortogonales entre sí. Por esta razón, es posible la representación de los datos en el espacio vectorial V , en términos de los vectores propios.
- Es una práctica usual el normalizar los vectores propios, una vez hallados; esto es, hacer que su longitud sea exactamente la unidad. Ya que es la dirección lo que determina al vector propio y no su tamaño. La aplicación de la anterior propiedad y esta consideración hace que de hecho, un conjunto de vectores propios de una transformación lineal, puedan ser una base orto normal de su espacio vectorial [19].

Los valores propios de una transformación lineal están íntimamente relacionados. En la figura 2.15 se aprecia una aplicación de una propiedad de los vectores propios, antes mencionada. Aún escalando al vector de entrada $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ antes de su transformación, se mantiene el mismo múltiplo del vector de entrada en la misma; en este caso: 1.

$$3\begin{bmatrix}2\\2\end{bmatrix}=\begin{bmatrix}6\\6\end{bmatrix}$$
$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}6\\6\end{bmatrix}=1\begin{bmatrix}6\\6\end{bmatrix}$$

Figura 2.15. Escalamiento del vector propio antes de su transformación (se mantiene el mismo valor propio)

Justamente, el número 1 es el valor propio asociado al vector propio. En este ejemplo, sin importar el múltiplo del vector propio que se tome antes de multiplicarlo por la matriz de la transformación lineal, siempre se obtendrá una vez el vector escalado como resultado.

El cálculo de los vectores y valores propios de una transformación lineal, se realiza mediante métodos matemáticos que son más o menos complejos dependiendo de la dimensión de la matriz de transformación. Sin embargo, en la actualidad existen una gran cantidad de herramientas de *software* con librerías para calcular estas cantidades, mismas que siempre deberán relacionar a los valores propios con sus respectivos vectores propios.

2.4.2.4 Distancia de *Mahalanobis*

En Estadística, la Distancia de *Mahalanobis* es una medida de distancia introducida por *Chandra Mahalanobis* en 1936. Su utilidad radica en que es una forma de determinar la similitud entre dos variables aleatorias multidimensionales. Se diferencia de la distancia euclídea en que tiene en cuenta la correlación entre las variables aleatorias [23].

Formalmente, la distancia de *Mahalanobis*, $d_m(\vec{x},\vec{y})$, entre dos variables aleatorias $\vec{x}$ e $\vec{y}$, con la misma distribución de probabilidad y matriz de covarianza C se define como:

$$d_m(\vec{x},\vec{y})=\sqrt{(\vec{x}-\vec{y})^T C^{-1}(\vec{x}-\vec{y})} \tag{2.26}$$

Para entender la utilidad de la distancia de *Mahalanobis*, se puede considerar el siguiente ejemplo práctico [23]: Un pescador quiere poder medir la *similitud* entre dos salmones porque quiere clasificarlos en dos tipos para su venta y poder así vender los

grandes a mayor precio. Para cada salmón mide su anchura y su longitud. Con estos datos construye un vector de características $\vec{x}_i = (x_{1i}, x_{2i})$ para cada salmón *i*.

La longitud de los salmones pescados es una variable aleatoria que toma valores entre 50 y 100 cm, mientras que su anchura está entre 10 y 20 cm. Si el pescador usase la distancia euclídea:

$$d_e(\vec{x}_1, \vec{x}_2) = \sqrt{(x_{11} - x_{12})^2 + (x_{21} - x_{22})^2} \tag{2.27}$$

O en notación vectorial:

$$d_e(\vec{x}_1, \vec{x}_2) = \sqrt{(\vec{x}_1 - \vec{x}_2)^T (\vec{x}_1 - \vec{x}_2)} \tag{2.28}$$

Al ser las diferencias de anchura menos cuantiosas que las de longitud, les estará dando menos importancia. Por esta razón, el pescador decide incorporar la estadística de los datos a la medida de distancia, ponderando según su varianza: las variables con menos varianza tendrán más importancia que las de mayor varianza. De esta forma pretende igualar la importancia de la anchura y la longitud en el resultado final. La ecuación quedaría:

$$d_2(\vec{x}_1, \vec{x}_2) = \sqrt{\left(\frac{x_{11} - x_{12}}{\sigma_1}\right)^2 + \left(\frac{x_{21} - x_{22}}{\sigma_2}\right)^2} \tag{2.29}$$

Donde σ_i es la desviación estándar de la componente *i* de los vectores de medidas.

O en notación vectorial:

$$d_2(\vec{x}_1, \vec{x}_2) = \sqrt{(\vec{x}_1 - \vec{x}_2)^T S^{-1} (\vec{x}_1 - \vec{x}_2)} \tag{2.30}$$

Donde *S* es una matriz diagonal cuyos elementos en la diagonal son: $s_{ii} = \sigma_i^2$

Pero la ecuación 2.30 tiene un problema y es que la longitud y anchura de los salmones *no son independientes*; es un problema, pues de hecho la anchura depende *en cierta forma* de la longitud, pues es más probable que un salmón largo sea también más ancho. Para incorporar la dependencia entre las dos variables, el pescador puede sustituir la matriz diagonal *S* por la matriz de covarianza C:

$$d_m(\vec{x}_1, \vec{x}_2) = \sqrt{(\vec{x}_1 - \vec{x}_2)^T C^{-1} (\vec{x}_1 - \vec{x}_2)} \tag{2.31}$$

Ecuación que da precisamente la distancia de *Mahalanobis.*

Finalmente, si la matriz de covarianza C es diagonal, se puede generalizar la ecuación 2.30 reduciendo la distancia de *Mahalanobis*, en un espacio vectorial de dimensión *N*, a la ecuación 2.32:

$$d_m(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}} \tag{2.32}$$

2.4.3 El método del PCA

Para cumplir los objetivos planteados en la sección 2.4.1, la primera cuestión que el Análisis de Componentes Principales se plantea es si existe una base que exprese de manera diferente el conjunto de datos del que se dispone, distinta a la base canónica de su espacio vectorial [19]; es decir, de tal forma que resalte las similitudes y diferencias del conjunto. Esta reformulación de datos puede ser vista como una transformación lineal del conjunto de datos, mediante una adecuada matriz de transformación. El principio básico del método que describe al PCA se representa en la ecuación 2.33:

$$P\Gamma = \Omega \tag{2.33}$$

Donde Γ es una matriz que representa al conjunto original de datos, Ω es una nueva representación de ese conjunto de datos y P es la matriz asociada a una transformación lineal llamada *cambio de base*, que representa geométricamente una rotación y re escalado, en la cual, sus filas son un conjunto de nuevos vectores de base para expresar a las columnas de Γ.

A continuación se plantea paso a paso el procedimiento para la obtención de este nuevo conjunto de datos [18], junto a varias nociones de las implicaciones matemáticas que envuelve el mismo. Para una justificación matemática más fuerte sobre el siguiente procedimiento, se recomienda la lectura de la referencia [19].

Paso 1.- Obtener el conjunto de datos

Se obtiene una colección de los *M* datos de muestra del experimento en curso, de *N* dimensiones cada uno, como columnas en la matriz de datos originales:

$$\Gamma = \begin{bmatrix} \Gamma_1 & \Gamma_2 & \dots & \Gamma_M \end{bmatrix} = \begin{bmatrix} \gamma_{11} & \gamma_{21} & \cdots & \gamma_{M1} \\ \gamma_{12} & \gamma_{22} & \cdots & \gamma_{M2} \\ \vdots & \vdots & & \vdots \\ \gamma_{1N} & \gamma_{2N} & \cdots & \gamma_{MN} \end{bmatrix} \tag{2.34}$$

Donde cada dato *i* es un vector Γ_i de *N* componentes γ_{ij}; es decir: $\Gamma_i = \left(\gamma_{i1}, \gamma_{i2}, ..., \gamma_{ij}, ..., \gamma_{iN-1}, \gamma_{iN}\right)$.

Para ejemplificar, se toma el ejemplo dado en la sección 2.4.2.4, los datos tomados por un pescador para medir la similitud entre salmones. Si en este experimento, se utilizan tres muestras de salmones, se tiene *M*=3; con dos dimensiones de representación de características (ancho y longitud) se tiene *N*=2. El conjunto de datos Γ, entonces, serían de los salmones Γ_1, Γ_2, Γ_3, expresados en términos de cada ancho *a* y longitud *l*, en la matriz:

$$\Gamma = \begin{bmatrix} \Gamma_1 & \Gamma_2 & \Gamma_3 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ l_1 & l_2 & l_3 \end{bmatrix} \tag{2.35}$$

En este paso como en los próximos, se establecerá una forma de expresión de cada variable del método, estándar para el desarrollo del presente proyecto, con el objetivo de simplificar, en el proceso, la expresión de los cálculos. Esto no significa que haya una sola forma de expresión matemática de cada variable. Las presentes convenciones están basadas en las referencias [19], [17].

Paso 2.- Sustraer la media

Sustrayendo de cada una de las dimensiones de cada elemento del conjunto de datos, la media a través de esa dimensión. Se hace para tener un conjunto de datos cuya media sea cero, es decir, un conjunto normalizado [18]. Sigue la ecuación:

$$\Gamma_i - \bar{\Gamma} = \Gamma_i - \begin{bmatrix} \frac{1}{M}\sum_{j=1}^{M}\gamma_{j1} \\ \frac{1}{M}\sum_{j=1}^{M}\gamma_{j2} \\ \vdots \\ \frac{1}{M}\sum_{j=1}^{M}\gamma_{jN} \end{bmatrix}, \forall i = 1, 2, ..., M \tag{2.36}$$

Donde $\bar{\Gamma}$ es la imagen media del conjunto de datos.

Paso 3.- Calcular la matriz de covarianza

La matriz de covarianza del conjunto de datos original, Γ, describe las características de ruido y redundancia existentes en el conjunto [19]. La matriz de covarianza en el espacio de datos discretos tiene varias aproximaciones, de las cuales, la seleccionada en este proyecto es la siguiente:

$$C_\Gamma = \Gamma\Gamma^T \tag{2.37}$$

Donde Γ^T es la transpuesta de la matriz del conjunto original de datos.

Dado que el conjunto de datos se haya expresado de la forma indicada en los dos anteriores pasos, la ecuación 2.37 *aproxima* adecuadamente la relación vista en la sección 2.4.2, como se puede comprobar mediante su desarrollo matemático.

El objetivo es, entonces, *minimizar* la redundancia del sistema, medido por las magnitudes de las covarianzas, así como *maximizar* la señal de los datos respecto al ruido, lo que es a su vez determinado por las varianzas en cada dimensión; estas medidas se observan claramente en la matriz de covarianza C_Γ antes aproximada [19]. El método consiste en la producción de un sistema con una nueva matriz de covarianza, C_Ω, que tenga las siguientes características [19]:

- Todos los términos no pertenecientes a su diagonal, deben ser cero. De aquí, que C_Ω debe ser una matriz diagonal, descorrelatada. Por esto la matriz de covarianza C_Γ debe ser diagonalizada, como se verá en el siguiente paso del PCA.

- Cada dimensión sucesiva en la diagonal de C_Ω debe estar ordenada descendentemente de acuerdo a la varianza; orden que indicaría las dimensiones con mayor importancia en su señal.

Paso 4.- Calcular los valores y vectores propios de la matriz de covarianza del conjunto de datos original

El método del PCA tiene como objetivo hallar una transformación lineal adecuada, *P*, para expresar de una forma diferente al conjunto original de datos Γ , obteniendo un nuevo conjunto Ω,que tenga una matriz de covarianza C_Ω diagonal y con sus elementos (las varianzas) ordenadas descendentemente para indicar la jerarquía de las dimensiones en su importancia. Esto se indica mediante el siguiente desarrollo:

$$C_\Omega = \Omega\Omega^T = (P\Gamma)(P\Gamma)^T = P(\Gamma\Gamma^T)P^T = PC_\Gamma P^T \tag{2.38}$$

Donde P^T es la transpuesta de la matriz de la transformación lineal de cambio de base.

Justamente, la expresión $PC_\Gamma P^T$ es la que se utiliza para la diagonalización de una matriz, dado que esta sea simétrica, como es el caso en la matriz de covarianza y que la matriz P^T sea una matriz ortogonal asociada a un cambio de base [22]. La cuestión es entonces, hallar la matriz P^T. De otra parte, se sabe que una matriz simétrica es diagonalizable por una matriz de sus vectores propios ortonormales [19]. De aquí, se deduce que P^T es la matriz que contiene a todos los *N* vectores propios ortonormales de la matriz de covarianza $N \times N$ antes obtenida.

Se obtienen entonces, mediante cualquier método seleccionado, los valores y vectores propios de la matriz de covarianza C_Γ , respectivamente, en las matrices:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_N \end{bmatrix} \tag{2.39}$$

$$U = \begin{bmatrix} \upsilon_{11} & \upsilon_{21} & \dots & \upsilon_{N1} \\ \upsilon_{12} & \upsilon_{22} & \dots & \upsilon_{N2} \\ \vdots & \vdots & & \vdots \\ \upsilon_{1N} & \upsilon_{2N} & \dots & \upsilon_{NN} \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_N \end{bmatrix} \tag{2.40}$$

En la ecuación 2.40, cada vector propio *i* es un vector u_i de *N* componentes υ_{ij}; es decir: $u_i = \left(\upsilon_{i1},\ \upsilon_{i2},\ ...,\ \upsilon_{ij},\ ...,\ \upsilon_{i\,N-1},\ \upsilon_{iN}\right)$.

Cada vector propio u_i ubicado en cada columna de la matriz U, está asociado al valor propio en la correspondiente columna de la matriz Λ. Finalmente, la matriz P^T, es la versión ortonormal de la matriz de vectores propios, que, al ser ya ortogonal, requiere, como paso final la normalización de sus vectores propios:

$$P^T = \begin{bmatrix} \frac{\upsilon_{11}}{|u_1|} & \frac{\upsilon_{21}}{|u_2|} & \dots & \frac{\upsilon_{N1}}{|u_N|} \\ \frac{\upsilon_{12}}{|u_1|} & \frac{\upsilon_{22}}{|u_2|} & \dots & \frac{\upsilon_{N2}}{|u_N|} \\ \vdots & \vdots & & \vdots \\ \frac{\upsilon_{1N}}{|u_1|} & \frac{\upsilon_{2N}}{|u_2|} & \dots & \frac{\upsilon_{NN}}{|u_N|} \end{bmatrix} \tag{2.41}$$

Esta matriz, transpuesta, es la que requeriría el método del PCA para la operación de cambio de base del conjunto original $\Omega = P\Gamma$, como se puede deducir de la ecuación 2.38. Lo que se obtiene de este proceso, es un conjunto de vectores que, como se esperaba, expresan de una mejor manera al conjunto de datos originales. En la figura 2.16, se presenta la aplicación de los anteriores pasos a un conjunto de datos en dos dimensiones de un ejemplo de la referencia [18]. En la figura, se muestra la representación de los datos del conjunto (puntos) en las dos bases canónicas del espacio vectorial 2D (ejes *x* e *y*), así como el gráfico de los 2 vectores propios obtenidos del anterior proceso (líneas diagonales punteadas).

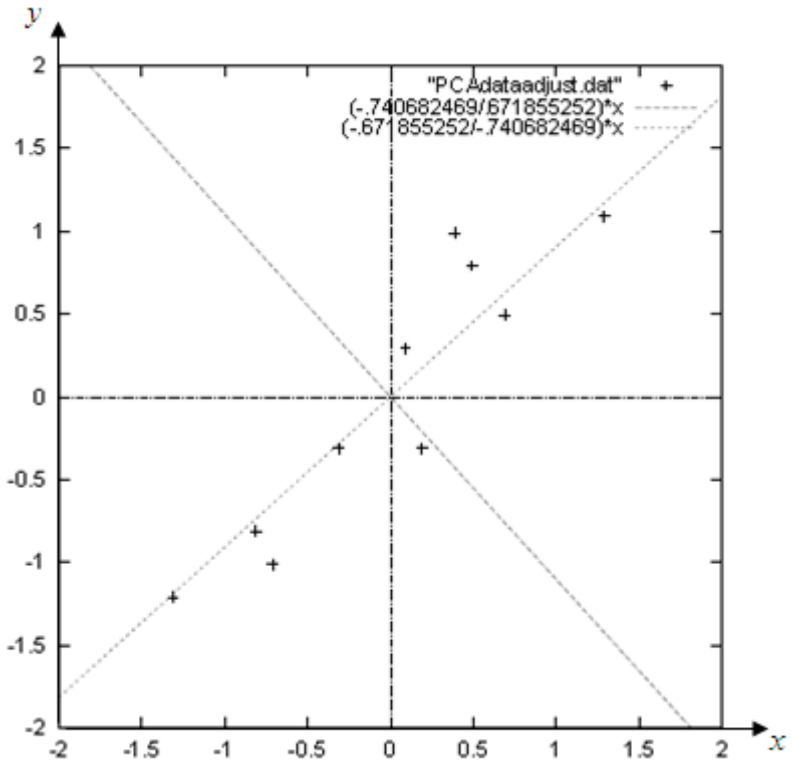


Figura 2.16. Gráfico de ejemplo de un conjunto de datos normalizados y los vectores propios de su matriz de covarianza [18]

Se aprecia claramente que los vectores propios son ortogonales y, de forma más importante, que proveen información acerca de los patrones en el conjunto de datos. Uno de los vectores propios atraviesa a los puntos de los datos, de la forma que lo haría una línea de mejor ajuste, indicando, en las medidas a través de ella (dadas por las proyecciones de cada punto en la línea), la relación de cercanía entre los distintos puntos de datos. El otro vector propio muestra patrones menos importantes en los datos, la distancia que hay de ellos al primer vector propio. Se deduce entonces, que mediante este proceso, se han extraído vectores que caracterizan a los datos, que ahora podrían ser proyectados a este nuevo espacio mediante la transformación de cambio de base, como se verá después. La meta del PCA, sin embargo, es simplificar aún más el nuevo conjunto de datos a obtenerse, mediante la reducción de sus dimensiones, como se expresó al inicio de la presente sección, 2.4. Los siguientes pasos se ocupan de esta tarea.

Paso 5.- Selección de componentes para la formación de un vector de características

Es aquí donde la noción de la compresión de datos y reducción de dimensión se aplica. Una vez obtenidos los valores y vectores propios asociados a los mismos, se establece que los componentes principales del conjunto de datos original son los vectores propios correspondientes a los valores propios que tengan los más altos valores absolutos [18], [19], por cuanto representan, en una eventual transformación del conjunto, los valores más altos de las varianzas en la matriz de covarianza que se obtiene. Es aquí, entonces, donde se ordena en forma descendente, tanto a la matriz Λ como a la matriz P^T, antes obtenidas, de acuerdo a los valores propios más altos (recordar que cada valor propio tiene su correspondiente vector propio asociado). Esto produce componentes en orden de significancia. Ahora, se pueden ignorar a los componentes con menor importancia. En general, se truncan las matrices anteriores de la dimensión $N \times N$ a una dimensión $N \times K$ siendo K el número de valores (vectores) propios seleccionados. El resultado es que el conjunto final de datos tendrá menor dimensión que el original ($K < N$). El vector de características es, entonces, simplemente una matriz que toma a los K vectores propios seleccionados, como columnas; de aquí, que es la versión final de la matriz antes hallada, P^T, pero ahora, de dimensión $N \times K$. Esto provee finalmente la matriz P de cambio de base requerida por el método del PCA, dada por la ecuación 2.42:

$$P^T = \begin{bmatrix} u_1 & u_2 & \dots & u_K \end{bmatrix}$$

$$\therefore$$

$$P = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_K \end{bmatrix} = \begin{bmatrix} \upsilon_{11} & \upsilon_{12} & \dots & \upsilon_{1N} \\ \upsilon_{21} & \upsilon_{22} & \dots & \upsilon_{2N} \\ \vdots & \vdots & & \vdots \\ \upsilon_{K1} & \upsilon_{K1} & \dots & \upsilon_{KN} \end{bmatrix} \tag{2.42}$$

Dado que cada elemento u_{ij} sea un vector propio normalizado de la matriz de covarianza del conjunto original de datos y haya pasado por el proceso de ordenación descendente y selección enunciado en el presente inciso.

Paso 6.- Derivación del nuevo conjunto de datos

Es el paso final del PCA, así como el más sencillo. Una vez obtenida la matriz de cambio de base requerida por el método, solo queda aplicar la ecuación 2.33, $\Omega = P\Gamma$, para la obtención del nuevo conjunto de datos Ω, de dimensiones $K \times M$. Como se puede seguir del desarrollo matemático de esta ecuación, geométricamente, lo que hace es proyectar a todos los *M* vectores del conjunto de datos original, que son los datos del experimento en curso y se hallan originalmente en la dimensión *N*, en un nuevo conjunto de bases o vectores unitarios, obteniéndose, en realidad, el mismo conjunto de *M* vectores de datos, ahora en la dimensión *K*; esto es, el conjunto de datos original, solamente en términos de los vectores propios seleccionados. La ventaja de esta transformación, como se determinó en el desarrollo del método, es que estos nuevos vectores son realmente aquellos que mejor describen a los patrones que existen entre ellos, eliminando los datos de las dimensiones más correlacionadas entre sí, para evitar la redundancia y dejando en su lugar sólo a las dimensiones más representativas de la información que conllevan los datos.

En esta sección se ha desarrollado toda la matemática de la técnica de reconocimiento de patrones sugerida al inicio de la presente sección; sin embargo, queda aún sin respuesta la pregunta: ¿Cómo se aplica el PCA al reconocimiento de imágenes? La siguiente sección, con la cual se finaliza el marco teórico del presente estudio, detallará un algoritmo que aplica todos los principios del PCA específicamente en el reconocimiento de imágenes. Como se verá, este nuevo método presenta algunas variaciones de la matemática del PCA aquí vista, propias de la naturaleza de las imágenes, que, sin embargo, mantienen los principios del Análisis de Componentes Principales.

2.4.4 Reconocimiento de imágenes mediante "*Eigenimages*"

Una de las formas de reconocer patrones en imágenes, normalmente es extraer información relevante de una imagen, codificarla lo más eficientemente posible y comparar dicha imagen codificada con una base de datos de modelos codificados de manera similar. Lo anterior se puede lograr matemáticamente buscando los *componentes principales* de una distribución de imágenes, o lo que resulta lo mismo, hallar los vectores propios de la matriz de covarianza de un conjunto de imágenes según el método del PCA, detallado en la sección 2.4.3. Estos vectores propios pueden ser considerados como un conjunto de características que exhiben las variaciones entre imágenes, donde cada pixel de una imagen

contribuye en mayor o menor grado en la construcción de cada vector propio; es por esto que los vectores propios pueden ser representados como una especie de *imágenes fantasmas*, a las cuales, en el entorno del reconocimiento de imágenes, se les suele denominar mediante el anglicismo *eigenimages*, que se entiende como *imágenes propias*, derivadas, como ellas, del nombre *vectores propios* (*eigenvector*).

Precisamente, el núcleo de reconocimiento de patrones implementado en el presente proyecto está basado en el artículo *"Face Recognition using Eigenfaces"* desarrollado por *Pentland* & *Turk* [17]. La diferencia es que en el presente trabajo se debe reconocer imágenes de billetes y no de rostros; por esta razón, semánticamente hablando, es más apropiado usar el anglicismo *eigenimages* en lugar de *eigenfaces*. A manera de ejemplo, la figura 2.20 muestra algunas *eigenimages* obtenidas de un conjunto de imágenes de billetes.

Lo interesante de las *imágenes propias* es que cada imagen en el conjunto de entrenamiento puede ser representada como una combinación lineal de las *eigenimages*. El número de posibles *eigenimages* es igual al número de imágenes usadas en el conjunto de entrenamiento. Sin embargo, las imágenes pueden ser aproximadas usando solo las "mejores" *eigenimages*, o sea, aquellas que tienen valores propios más altos y que por consiguiente contribuyen con la mayor parte de la varianza (información) dentro del conjunto. El objetivo primordial de querer usar solo unas pocas *eigenimages* es por cuestiones de eficiencia computacional. Las K mejores *eigenimages* construyen un sub-espacio de dimensión K que, siguiendo los objetivos del presente proyecto, se conocerá de ahora en adelante como *espacio de billetes*. Al igual que las funciones sinusoidales son funciones base para la descomposición de *Fourier* (de hecho son funciones propias de los sistemas lineales), las *eigenimages* son vectores base para la descomposición de un conjunto de imágenes en sus componentes principales [17].

Más atrás aún, la idea de usar *eigenimages* está motivada por una técnica desarrollada por *Sirovich* & *Kirby* [24] para representar eficientemente imágenes de rostros usando el análisis de componentes principales. Ellos sostenían que una colección de imágenes de rostros puede ser aproximadamente reconstruida almacenado una pequeña colección de pesos (coeficientes) para cada rostro y un pequeño conjunto de imágenes estándar (*eigenfaces*).

Generalizando, si un conjunto de imágenes puede ser reconstruido como la combinación lineal de un conjunto de imágenes características (*eigenimages*) ponderadas de acuerdo a una colección de pesos (coeficientes), entonces se obtiene una potente herramienta para reconocer imágenes que calcula las características *esenciales* (pesos) a partir de imágenes conocidas (conjunto de entrenamiento) y que reconoce si una imagen desconocida corresponde a determinada categoría comparando sus pesos, obtenidos también a partir de las *eigenimages*, con los pesos asociados a las imágenes conocidas.

El proceso de reconocimiento mediante *eigenimages* a seguirse es el siguiente:

a. Calcular las *eigenimages* de un conjunto de entrenamiento para definir el *"espacio de billetes"*. Escoger las K *"mejores eigenimages"*
b. Obtener los pesos de cada imagen del conjunto de entrenamiento proyectando dichas imágenes en cada una de las K *eigenimages* escogidas en el punto anterior.
c. Para una imagen de entrada, calcular un conjunto de pesos proyectando dicha imagen en cada una de las K *eigenimages* calculadas en el entrenamiento.
d. Determinar si la imagen de entrada es un billete chequeando si está suficientemente cerca del *espacio de billetes* usando alguna métrica de distancia.

A continuación, se detalla cada uno de estos pasos:

2.4.4.1 Cálculo de las *eigenimages*

En esta sección se detallará el procedimiento de cálculo de las *eigenimages*. Para un análisis matemático más profundo, se recomienda leer y comprender la sección 2.4.3. El procedimiento, que se realiza *offline*[2], es el siguiente:

1. Obtener un conjunto de entrenamiento de imágenes $I_1, I_2, ..., I_M$. Las M imágenes de muestra deben estar centradas, escaladas a una misma resolución y normalizadas de acuerdo a la ecuación 2.10.

[2] *Offline* o *fuera de línea* en reconocimiento de imágenes es un proceso de entrenamiento que, utilizando imágenes conocidas, genera información (*e.g. eigenimages)* que posteriormente servirá para clasificar imágenes desconocidas. En el presente estudio, el entrenamiento *offline* es llevado a cabo por MATLAB como se verá en el capítulo 3.

Figura 2.17. Ejemplo de conjunto de entrenamiento normalizado, M=24

2. Representar cada imagen I_i de resolución $N \times N$ como un vector columna Γ_i de tamaño N^2. De ahora en adelante toda imagen será considerada un vector.

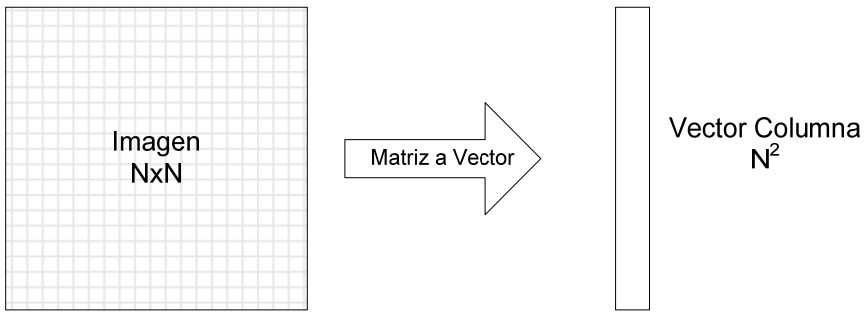


Figura 2.18. Representación de una imagen como matriz y como vector

3. Calcular la imagen media Ψ.

$$\Psi = \frac{1}{M}\sum_{i=1}^{M} \Gamma_i \quad (vector\ N^2 \times 1) \tag{2.43}$$

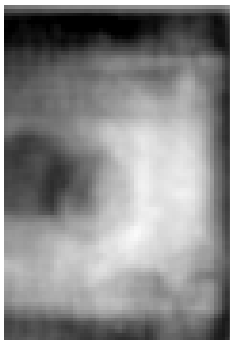


Figura 2.19. Imagen media del conjunto de entrenamiento de la figura 2.17

4. Restar la imagen media de cada imagen del conjunto.

$$\Phi_i = \Gamma_i - \Psi \quad \forall i = 1...M \quad (vector\ N^2 \times 1) \tag{2.44}$$

Donde cada Φ_i es una imagen normalizada con respecto a la imagen media Ψ.

5. Calcular la matriz de covarianza C. En la sección 2.4.3 se detallan las motivaciones y efectos de esta operación.

$$C = \frac{1}{M}\sum_{i=1}^{M} \Phi_n \Phi_n{}^T = AA^T \quad (matriz\ N^2 \times N^2) \tag{2.45}$$

Donde $A = \begin{bmatrix} \Phi_1 & \Phi_2 & ... & \Phi_M \end{bmatrix}$ $(matriz\ N^2 \times M)$

6. Calcular los vectores propios u_i de $C = AA^T$. Como la matriz de covarianza C es de dimensión $N^2 \times N^2$, determinar los N^2 vectores propios es algo casi imposible para un tamaño típico de una imagen. Afortunadamente, se pueden determinar los vectores propios resolviendo primero una matriz equivalente mucho más pequeña de dimensión $M \times M$. Esta matriz es $L = A^T A$ de dimensión $M \times M$ cuyos vectores propios v_i responden a la ecuación 2.46 [17]:

$$A^T A v_i = u_i v_i \tag{2.46}$$

Al multiplicar ambos lados por A, se obtiene:

$$AA^T Av_i = u_i Av_i \tag{2.47}$$

Se sabe que Av_i son los vectores propios de la matriz $C = AA^T$. En conclusión, la matriz L (ecuación 2.48) y la matriz de covarianza $C = AA^T$ tienen *los mismos valores propios*, y sus vectores propios, v_i y u_i están relacionados con la ecuación $u_i = Av_i$ [17]. La matriz L, se calcula así:

$$L = A^T A \tag{2.48}$$

Es importante recordar que la matriz $L = A^T A$ puede tener hasta M valores y vectores propios mientras que la matriz $C = AA^T$ puede tener hasta N^2 valores y vectores propios. Además los M valores propios de $L = A^T A$ (con sus correspondientes vectores propios) corresponden a los M más altos valores propios de $C = AA^T$ (con sus correspondientes vectores propios).

Del análisis anterior, los M mejores vectores propios u_i de la matriz de covarianza $C = AA^T$ de dimensión $N^2 \times N^2$ se calculan a partir de los vectores propios v_i de la matriz $L = A^T A$ de dimensión $M \times M$ con la fórmula 2.48:

$$u_i = Av_i \tag{2.49}$$

Finalmente, hay que recordar que los vectores propios u_i obtenidos con la ecuación 2.49, deben ser normalizados de modo que la norma de $\|u_i\| = 1$, como se expresa en la sección 2.4.3.

7. De los M vectores propios calculados en el paso anterior, escoger solo K vectores propios correspondientes a los K más altos valores propios. No existe fórmula para saber el valor de K; por ejemplo, en la referencia [17] para $M = 16$, se escogió $K = 7$. Los valores propios escogidos son las *eigenimages* que servirán para el reconocimiento.

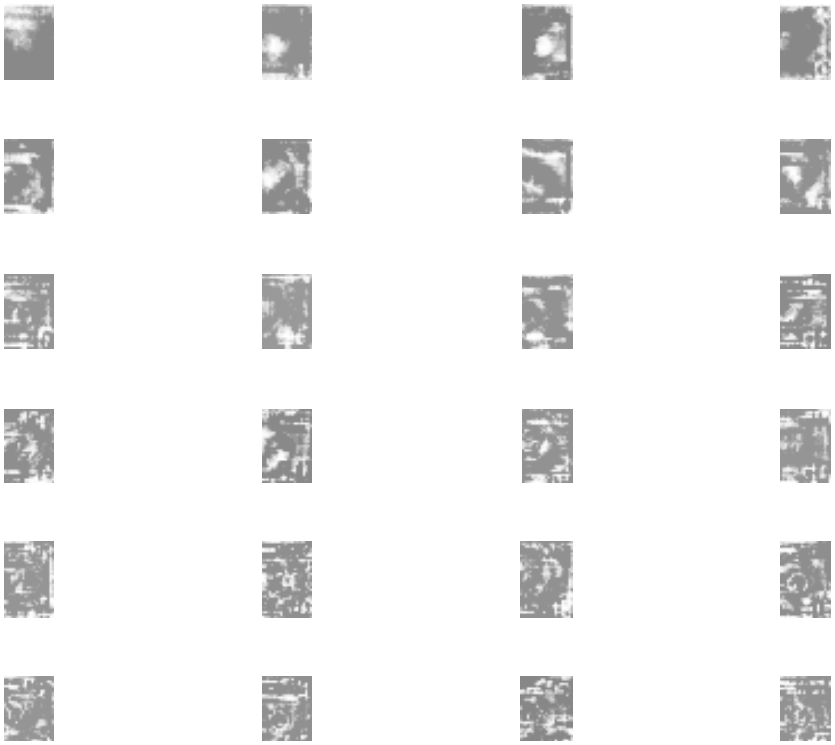


Figura 2.20. Representación de las *eigenimages* del conjunto de entrenamiento de la figura 2.17, K=24

8. Representar cada imagen del conjunto de entrenamiento en el "espacio de billetes" proyectando dicha imagen en cada una de las *K* bases (*eigenimages*). Para lograrlo, es conveniente representar las *eigenimages* en una matriz u de dimensión $N^2 \times K$ de modo que sus columnas sean las *eigenimages* u_i. Entonces, para obtener los pesos del conjunto de entrenamiento se puede usar la ecuación 2.50:

$$\Omega = u^T A \tag{2.50}$$

Donde Ω es una matriz de dimensión $K \times M$ cuyas columnas contienen los K coeficientes de la proyección de cada imagen del conjunto de entrenamiento; es decir, que es el conjunto de imágenes de muestra (de dimensión N^2), ahora representados en la nueva dimensión *K*.

2.4.4.2 Uso de *eigenimages* para identificar una imagen

Una vez que las *eigenimages* han sido creadas, la identificación *online*[3] se convierte en una tarea de reconocimiento de patrones. Es así que una nueva imagen de entrada normalizada (Γ_{in}) es transformada en sus componentes dentro del *"espacio de billetes"* mediante una simple operación:

$$\omega_i = u_i^T \left(\Gamma_{in} - \Psi \right) \forall \; i = 1,2,...,K \tag{2.51}$$

Los valores calculados con la ecuación 2.51 forman un vector de pesos de la imagen de entrada $\Omega^T{}_{in} = \left[\omega_1 \quad \omega_2 \quad ... \quad \omega_K \right]$ que valora la contribución de cada *eigenimage* o base para representar a la imagen de entrada. Luego, para determinar la clase a la que pertenece este vector, se debe calcular la distancia de dicho a vector a cada uno de los vectores proyectados del conjunto de entrenamiento. Por sus características, la distancia usada en este proyecto es la *distancia de Mahalanobis* (ver sección 2.4.2.4). Una imagen es clasificada como billete de determinado valor cuando la mínima distancia del vector Ω_{in} a cada proyección del conjunto de entrenamiento es menor que un determinado umbral. Caso contrario, la imagen es descartada.

2.4.4.3 Ventajas y desventajas de *eigenimages*

Hasta ahora se ha visto que *eigenimages* es un poderoso método que permite pasar de un complejo espacio de dimensión N^2 a otro mucho más manejable de dimensión K. Entre sus principales ventajas, se tiene que reduce drásticamente la complejidad computacional por el hecho de reducir la dimensión del espacio. Además, las operaciones matemáticas para procesar una imagen de entrada *online* son simples sumas y productos, lo cual hace que este método sea factible de ser procesado en plataformas de *hardware* con velocidad limitada. Otra gran ventaja de PCA es que para lograr una tasa de acierto alta no requiere de un conjunto de entrenamiento extremadamente grande como requieren otros algoritmos de aprendizaje (*e.g.* Adaboost [13]).

Sin embargo PCA también tiene una serie de desventajas. De una parte, se ha comprobado que tiene poca robustez frente a iluminación no uniforme [17]. Por otro lado,

[3] *Online* o *en línea* en reconocimiento de imágenes es el conjunto de procesos que utilizan la información (*e.g. eigenimages*) calculada durante el entrenamiento (*offline*) con el objetivo de clasificar una imagen desconocida. En el presente estudio, los procesos *online* son realizados por el teléfono celular como se verá en el capítulo 3.

no es un método invariante a la escala, esto implica que debe existir un pre procesamiento para escalar la imagen de entrada. A pesar de todos estos inconvenientes, PCA todavía es atractivo por su sencillez y desempeño.

Finalmente, se recomienda la lectura de la referencia [17] así como la sección 2.4.3 para una comprensión más profunda sobre la forma de trabajo del PCA, técnica que fundamenta al método de reconocimiento de patrones mediante *eigenimages*. En el presente trabajo, se incluye una implementación de *eigenimages* en la herramienta de *software*, MATLAB que aplica todos los conceptos presentados en esta sección.

CAPÍTULO 3

DESARROLLO DEL PROTOTIPO

3.1 VISTA PRELIMINAR DEL SISTEMA

El presente capítulo detalla el procedimiento seguido para la implementación del prototipo del sistema que se ha definido en el alcance y delimitación de la solución propuesta del presente proyecto (sección 1.2.4), para dar respuesta a la hipótesis del planteamiento del problema, utilizando aquellos criterios obtenidos acerca de los requerimientos de la población no vidente.

La idea básica del sistema es la aplicación de la teoría matemática del PCA (sección 2.4.3) mediante el seguimiento de los pasos de la metodología del reconocimiento de imágenes mediante *eigenimages*, detallados a lo largo de la sección 2.4.4, con el uso de un sistema *online* explicado mediante el esquema general mostrado en la figura 3.1.

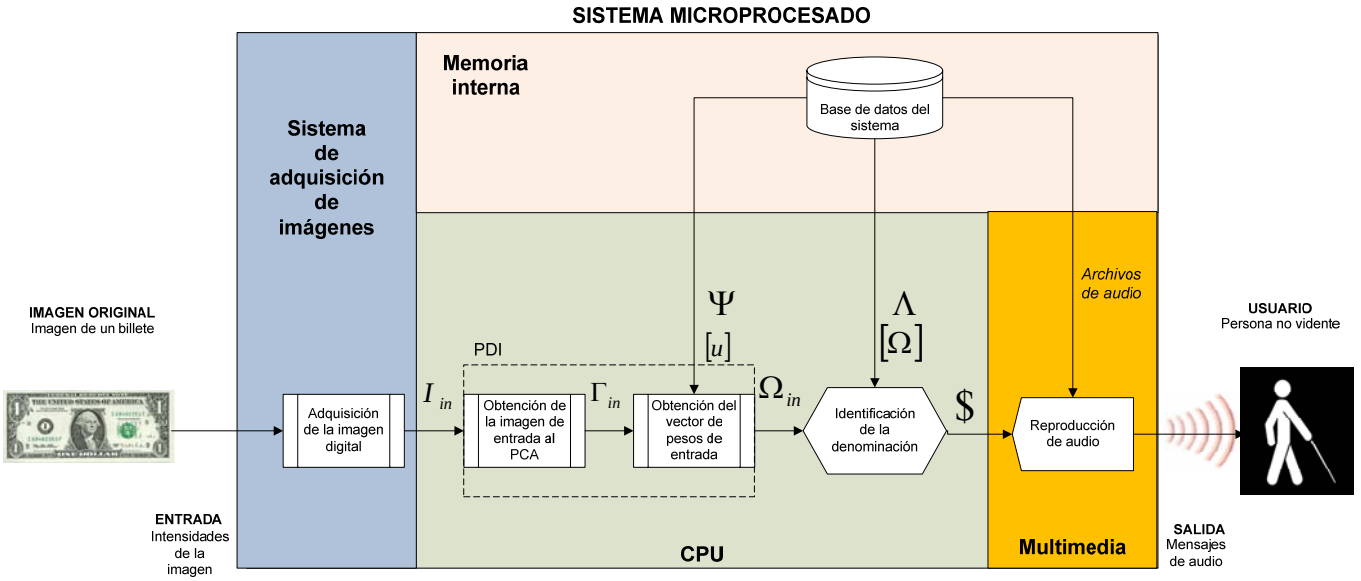


Figura 3.1. Esquema general del sistema a implementarse

En una breve descripción del sistema, los bloques indicados en la figura 3.1 cumplen, respectivamente, las siguientes funciones:

- **Adquisición de la imagen digital:** El sistema toma, como entrada, a la imagen de un billete. Mediante un adecuado subsistema de adquisición de imágenes, debe

interpretar la información de la imagen original para la formación de una imagen digital de entrada llamada I_{in}. Esta imagen se conocerá como *imagen de entorno* y se supone que contiene en su interior la imagen de un billete sobre algún fondo.

- **Obtención de la imagen de entrada al PCA:** La imagen de entorno I_{in} pasa, entonces, a un bloque de PDI implementado por el CPU del sistema. Dentro de este bloque, se somete a procesos de PDI de bajo y medio nivel (sección 2.2 y 2.3) para la adecuada segmentación de la imagen de entorno que permita la obtención de lo que se conocerá como la *imagen de entrada al PCA*, Γ_{in}, que consiste en el corte normalizado de una determinada porción del billete de la imagen de entorno, adecuado para su entrada a la matemática del PCA. En la sección 3.2.1.1 se detallarán las características de la imagen de entrada al PCA.

- **Obtención del vector de pesos de entrada:** Siguiendo el PDI de la imagen digital obtenida a la entrada del sistema, se somete a la imagen de entrada al PCA, Γ_{in}, a la metodología de *eigenimages* para la identificación de una imagen, detallada en la sección 2.4.4.2, mediante su normalización (en el contexto de PCA) con el uso de Ψ, que es la imagen media de un conjunto de imágenes de muestra de las distintas denominaciones a reconocerse, y la proyección de Γ_{in} al llamado *espacio de billetes*, con el uso de $[u]$, que es la matriz de los vectores propios del conjunto de muestras (o *eigenimages*). Estos conceptos se refieren a la teoría de la sección 2.4.4. Como se puede notar, aquí se hace uso de la memoria interna del sistema, en la cual deben hallarse almacenadas las variables: Ψ, $[u]$. A la salida de este bloque se obtiene lo que se conoce como el *vector de pesos de entrada*, Ω_{in} que, según la teoría de la sección 2.4.4, consiste en una representación de la imagen de entrada al PCA en una nueva dimensión reducida de componentes principales. El conjunto de imágenes de muestra referido en este bloque se determina en la sección 3.2.1.

- **Identificación de la denominación:** Continuando el trabajo del CPU, este debe encargarse de tomar al vector de pesos de entrada Ω_{in} y realizar la respectiva comparación con cada uno los *vectores de pesos del conjunto de muestras* del método de *eigenimages*, que según la teoría del reconocimiento de la sección

2.4.4.2, están contenidos en la matriz $[\Omega]$. La comparación se realizará, como se anotó en dicha sección, mediante la métrica de la distancia de *Mahalanobis*, que hace uso de los valores propios del conjunto de muestras, Λ. El resultado de esta comparación, es la identificación de la denominación del billete de entrada, por lo que la salida de este bloque, es ya el dato de la denominación del billete, referido en el esquema 3.1 como la variable $\$$. Este bloque hace uso también de la memoria interna del sistema, que tiene almacenadas las variables: $[\Omega]$, Λ.

- **Reproducción de audio:** El último paso del sistema consiste en el uso de la información de la denominación obtenida del billete para la emisión del mensaje a la persona no vidente, a través de las capacidades multimedia del sistema microprocesado para la reproducción de un mensaje audible indicando la denominación. En este boque, se hace uso, una vez más, de la memoria del sistema, en la cual, estarán almacenados archivos de audio con los mensajes pertinentes. Los detalles de estos archivos se señalarán en la sección 3.4.

Todos los detalles concernientes al funcionamiento respecto a las posibles eventualidades en el sistema (*e.g.* que la imagen de entrada no sea un billete o que el método no pueda hallar una respuesta en un intervalo de tiempo adecuado) serán determinados en los requerimientos del programa (de *software*) en la sección 3.3.2.2.

El sistema microprocesado que encierra a todos los bloques de la figura 3.1, será un teléfono celular móvil, como se determinó en el alcance del proyecto, en la sección 1.2.4. Todas las justificaciones para la selección de las plataformas de *hardware* y *software* del sistema serán justificadas respectivamente en las secciones 3.3.1 y 3.3.2.

Adicionalmente, un requerimiento fundamental en el esquema de la figura 3.1, es una adecuada *base de datos* que contenga las variables necesarias para el reconocimiento de la imagen: Ψ, $[u]$, $[\Omega]$, Λ; así como los archivos de audio para la reproducción de los mensajes al usuario. Estos datos deben establecerse de antemano, en un adecuado programa *offline* de entrenamiento. En la sección 3.2 se detallará el procedimiento seguido para el desarrollo de esta base de datos. Se deja, entonces, la descripción del desarrollo del sistema microprocesado esquematizado en la figura 3.1 para la sección 3.3 y en adelante.

3.2 DESARROLLO DE LA BASE DE DATOS DEL SISTEMA

Como se ha establecido en la vista preliminar del sistema que se desea implementar como solución, es necesario para este desarrollo una adecuada base de datos que contenga las principales variables que produce el método de *eigenimages* para el reconocimiento de imágenes. La presente sección describe la forma en la que se han implementado los pasos de la sección 2.4.4 para la obtención de las variables que conformarán la base de datos del sistema a implementarse. Se inicia este procedimiento por el establecimiento del conjunto de imágenes de muestra.

3.2.1 Establecimiento del conjunto de imágenes de muestra para *eigenimages*

Según lo establecido en los primeros pasos de los algoritmos de la sección 2.4.4.1, el método de las *eigenimages* inicia con la obtención de un conjunto de imágenes de muestra, que son el conjunto de *patrones que reconocerá el sistema* mediante la comparación de una imagen de entrada con cada una de sus imágenes de muestra. En este proyecto, todas estas son imágenes de los billetes de las distintas denominaciones a reconocerse. Es necesario en este punto, entonces, determinar las características de estas imágenes.

3.2.1.1 Características de las imágenes

Para este proyecto se ha decidido que, tanto las imágenes del conjunto de muestras como las de entrada al sistema deben ser imágenes en escala de gris (más manejables según la sección 2.2.2, al componerse de una sola matriz) de una *porción* del billete, porque la obtención de la imagen de un billete completo resulta una tarea más difícil para una persona no vidente, al requerirse para ella un enfoque más alejado del objetivo. Además, la imagen que se obtendría en tales condiciones de lejanía entre el billete y el sistema de adquisición de imágenes, tendría una apariencia tal que no ofrecería rasgos distintivos entre las denominaciones distintas de cada billete. La apariencia de estas porciones se muestra en la figura 3.2.

Figura 3.2. Porción del billete a usarse en el proyecto para su reconocimiento

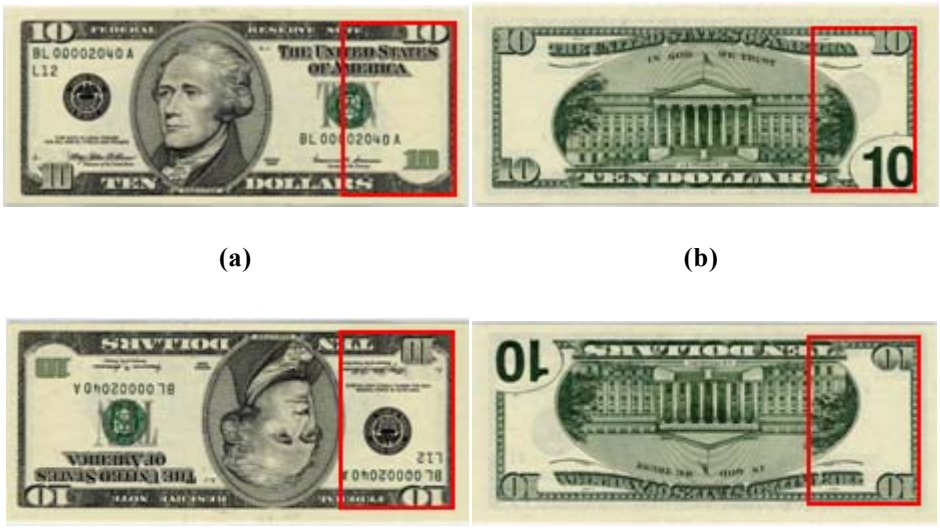
El ejemplo de la figura 3.2 muestra la imagen de la porción de un extremo de un billete de un dólar. Se ha seleccionado tal porción porque pruebas realizadas durante la experimentación con el método del PCA con porciones de menor área y en otros sitios ofrecieron resultados poco satisfactorios. Se concluyó de aquello que la información sobre las características del billete de cada denominación en la estética de la figura 3.2, en los extremos laterales de los billetes, es la necesaria y suficiente para obtener buenos resultados de reconocimiento con la matemática del PCA.

Las imágenes de la estética de la figura 3.2 se han determinado con una resolución, de ancho por alto, de *60×90 píxeles* (90 filas×60columnas)**.** La razón de esta selección es que, en las pruebas del método del PCA, al trabajar con resoluciones menores o iguales a 4800 píxeles, se produjeron resultados desfavorables. La resolución seleccionada es de 60 $\times$90 = 5400 píxeles. Las cantidades de 90 filas y 60 columnas se realizaron pensando en la proporción de $\frac{2}{3}$ de ancho a alto, como un número racional manejable para los futuros cálculos en el dispositivo microprocesado que implementará el sistema *online*.

3.2.1.2 Obtención de las imágenes de muestra

Para esta tarea, se ubicó el conjunto de los 8 billetes mostrados en el anexo A2 como los billetes de muestra, que son los que se han establecido en el alcance del proyecto (sección 1.2.4). A continuación, se procedió a la toma de fotografías de los billetes haciendo uso de las mismas herramientas para el sistema *online* que serán detalladas en la sección 3.3, asegurando así que el sistema de adquisición para las imágenes de entrada sea el mismo para las de muestra y así procurar que la comparación a realizarse no sea ambigua. Las tomas realizadas son de las características detallada en la sección 3.2.1.1.

Ahora, dado que una persona no vidente, al tomar el billete no puede saber en qué orientación está, se ha determinado aquí el concepto de *extremos* de un billete. Los 4 extremos de los que consta un billete son los detallados en la figura 3.3, en la cual se muestra el orden que se ha asignado para cada extremo en este proyecto.



(a) **(b)**

(c) **(d)**

Figura 3.3. Extremos considerados para la toma de imágenes:
(a)Extremo uno (b) Extremo dos (c) Extremo tres (d) Extremo cuatro

En el ejemplo mostrado en la figura 3.3, se muestran los extremos tomados para un billete de 10 dólares. Los mismos extremos se toman para los billetes del resto de denominaciones del proyecto, considerando que la toma de la imagen por parte de la persona no vidente será aproximando el billete al foco del sistema de adquisición de imágenes por el lado izquierdo, de tal forma que la toma se realiza de un extremo derecho como cualquiera de los de la figura3.3. Si se consideran los cuatro casos de la figura 3.3, se cubren todas las posibilidades de imagen que se tendría para su reconocimiento, de cualquier forma que la persona no vidente haya tomado el billete para su enfoque.

Bajo este criterio, se tomaron imágenes de cada uno de los 4 extremos de las 6 denominaciones: $1, $5, $10, $20, $50 y $100; orientando a los billetes de 7 formas distintas, mediante el criterio de que distintas posiciones relativas de la toma del billete proveerán un mayor porcentaje de éxito del método de *eigenimages* [18]. La organización para la toma de las imágenes es en los siguientes subconjuntos de tipos de muestras:

- Imágenes de los 4 extremos de cada una de las 6 denominaciones, *en forma recta*, como ejemplifica la parte (a) de la figura 3.4.
- Imágenes de los 4 extremos de cada una de las 6 denominaciones, en *ligera inclinación horaria*, como ejemplifica la parte (b) de la figura 3.4.

- Imágenes de los 4 extremos de cada una de las 6 denominaciones, en *ligera inclinación anti horaria*, como ejemplifica la parte (c) de la figura 3.4.
- Imágenes de los 4 extremos de cada una de las 6 denominaciones, en *ligera inclinación hacia atrás*, como ejemplifica la parte (d) de la figura 3.4.
- Imágenes de los 4 extremos de cada una de las 6 denominaciones, *con un ligero hundimiento en la mitad*, como ejemplifica la parte (e) de la figura 3.4.
- Imágenes de los 4 extremos de cada una de las 6 denominaciones, de billetes de *poca circulación*, como ejemplifica la parte (f) de la figura 3.4.
- Imágenes de los 4 extremos de cada una de las 6 denominaciones, de billetes *desgastados*, como ejemplifica la parte (g) de la figura 3.4.

Por esta organización, se obtiene en total un conjunto de: $4\ extremos \times 6\ denominaciones \times 7\ tipos\ de\ muestras = 168\ imágenes\ de\ muestra.$

Las imágenes así obtenidas fueron organizadas en un directorio llamado “Muestras” y almacenadas como archivos de extensión *pgm* (*Portable GrayMap format*). La razón de la selección de esta extensión es que es un formato usado para escala de grises, sin compresión, de tal forma que la información de la imagen de los archivos sea lo más fiel posible a la imagen originalmente obtenida mediante el sistema de adquisición de imágenes.

Los *nombres* de los archivos para las distintas imágenes de muestra siguen un código que se puede comprender con un ejemplo específico: *b20.4.7.pgm*. Este ejemplo es el nombre del archivo de la imagen de un billete (*b*) de 20 dólares (20), del cuarto extremo (4) y del subconjunto de billetes desgastados (7).

(a)

(b)

(c)

(d)

(e)

(f)

(g)

Figura 3.4. Ejemplos del conjunto de imágenes originales. Se exhiben ejemplos de las muestras de billetes: (a) Rectos (b) Con ligera inclinación horaria (c) Con ligera inclinación anti horaria (d) Con ligera inclinación hacia atrás (e) Con ligero hundimiento en la mitad (f) De poca circulación (g) Desgastados

3.2.2 Descripción del programa de entrenamiento

El objetivo del programa de entrenamiento es la obtención de las variables que componen la base de datos del sistema que se desea implementar como solución en el presente proyecto, diagramado en la figura 3.1. Este programa sigue el flujo de la figura 3.5.

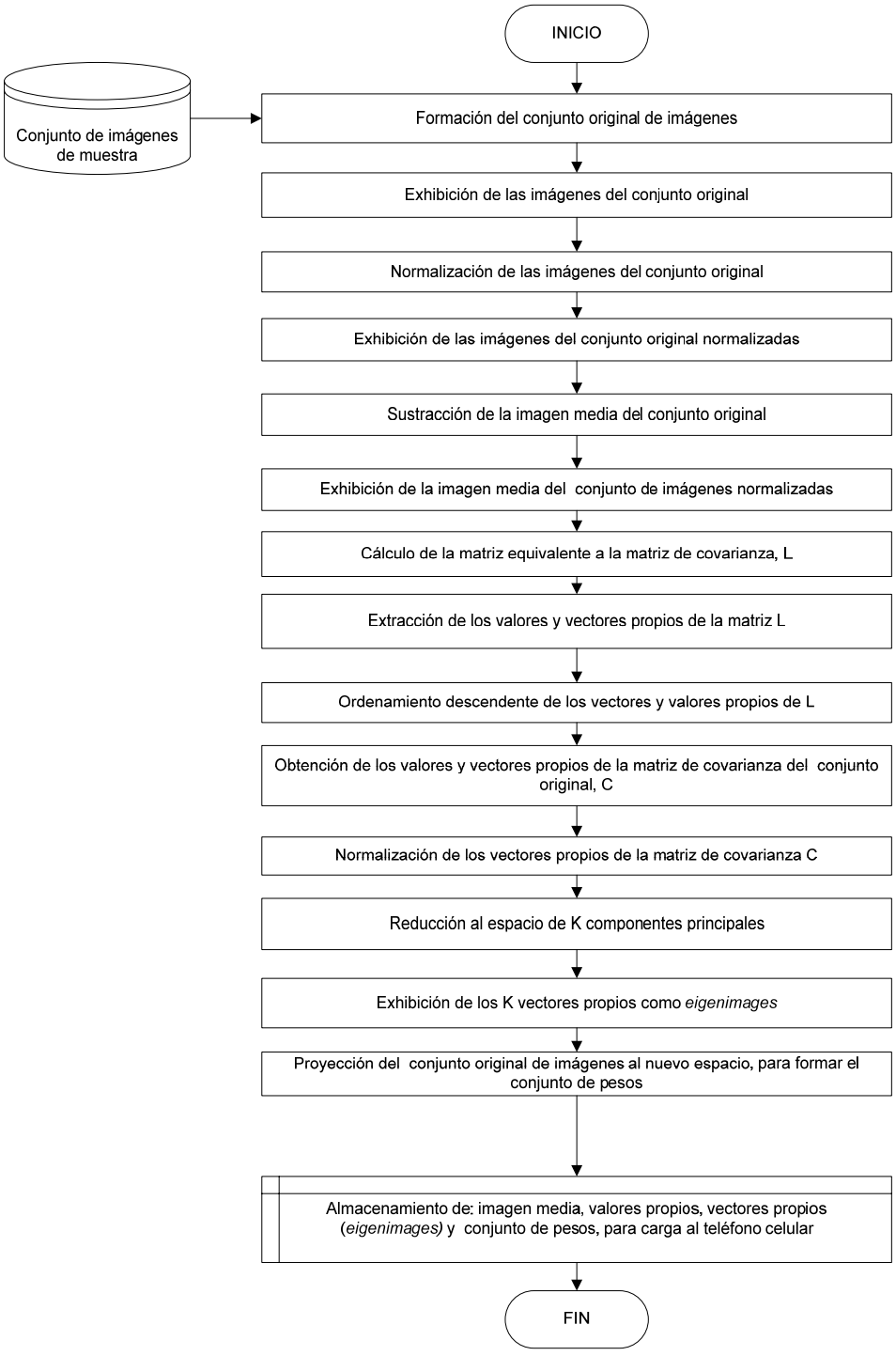


Figura 3.5. Diagrama de flujo del programa de entrenamiento

El programa recibe como entrada al conjunto de imágenes de muestra determinado en la sección 3.2.1 y entrega como salida las variables necesarias para el reconocimiento de la imagen de entrada en el sistema microprocesado *online*, definidas dentro de los pasos para la obtención de *eigenimages* en la sección 2.4.4.1. Estas salidas son:

- La *imagen media* del conjunto de imágenes de muestra: Ψ.
- El conjunto de *K* bases, vectores propios o *eigenimages* seleccionados como componentes principales del conjunto de imágenes de muestra, en la matriz $[u]$, que se conocerá como la matriz de vectores propios de la matriz de covarianza *C*.
- Las imágenes de muestra proyectadas al nuevo espacio de billetes de dimensión *K*, en la matriz $[\Omega]$. Se conocerá a este conjunto como *conjunto de pesos*.
- El conjunto de *K* valores propios del conjunto de imágenes de muestra, en el vector Λ, de valores propios de la matriz de covarianza *C*.

Se pretende almacenar estos datos en un conjunto archivos ordenados en un solo directorio que componga la base de datos del sistema de la figura 3.1.

Este programa ha sido implementado en la herramienta de *software* MATLAB, puesto que satisface la capacidad de procesamiento que requiere el algoritmo y provee la prestación de la facilidad de implementación en operaciones matriciales. El código de este programa principal, en MATLAB, se halla en el anexo A4.1. A continuación, se describe el procedimiento en cada bloque del diagrama en la figura 3.5.

3.2.2.1 Formación del conjunto original de imágenes

Tiene como objetivo la formación de una estructura matemática matricial con la información del conjunto de imágenes de muestra para que su análisis sea posible. Cumple con el paso 2 del método de *eigenimages,* planteado en la sección 2.4.4.1, así como la organización recomendada en el paso 1 de la sección 2.4.3, del método del PCA.

Recibe como entrada al conjunto de 168 imágenes de muestra determinado en la sección 3.2.1.2. Produce, como salida, la matriz del conjunto original de datos (imágenes) Γ de la expresión 2.34. El diagrama de este procedimiento se presenta en la figura 3.6.

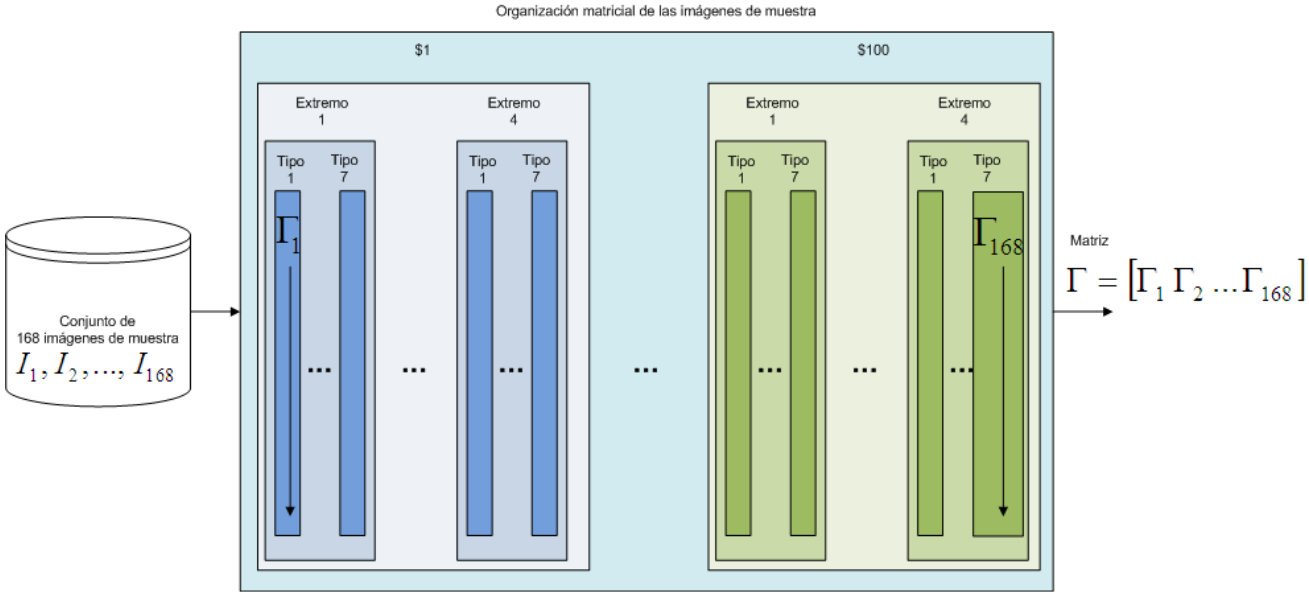


Figura 3.6. Organización del conjunto original de imágenes

Este procedimiento inicia con la lectura de cada una de las M imágenes I_i del conjunto de muestras, mediante la apertura de cada uno de sus archivos en el directorio "Muestras" mediante la función de MATLAB, `imread`, que guarda en RAM las imágenes como matrices de resolución $N = 90 \times 60$. Después realiza un ordenamiento de cada imagen I_i como vectores columna Γ_i de dimensión $N \times 1$, dentro de la matriz Γ que tiene dimensiones $N \times M$, donde $M=168$ es el número de imágenes en el conjunto de muestra y $N=5400$ es la resolución de píxeles de cada imagen de muestra del conjunto. Así, se organiza a todas las imágenes de entrada como vectores columna de la variable matriz en MATLAB, `gamma`. La organización particular en esta matriz, como se ve en la figura 3.6, es de la siguiente manera:

- Primero, en 6 grupos de columnas en orden ascendente de denominación ($1, $5, $10, $20, $50 y $100); en cada uno de estos grupos:
- 4 grupos de columnas según el extremo del billete tomado; en cada uno de estos grupos:
- 7 grupos de columnas según el tipo de muestra al que pertenece el billete (recto, inclinado horariamente, etc.).

El procedimiento se realiza mediante la función definida por el usuario, `LeerImagenes` que, además, produce un dato de tipo *estructura*, propio de MATLAB, con información general de cada muestra (denominación, extremo tomado, tipo de

muestra, etc.). El código de esta función se puede revisar en el anexo A4.2, la cual también utiliza el etiquetado de las imágenes según su nombre de archivo, implementada por la función detallada en el anexo A4.3.

3.2.2.2 Exhibición de las imágenes del conjunto original

Tiene como objetivo la exhibición del conjunto original de imágenes obtenido en el anterior procedimiento para la comprobación visual de que la organización hecha allí se ha realizado correctamente.

Recibe como entrada a la matriz Γ de todas las imágenes del conjunto de muestras, en el orden en que aparecen allí y selecciona, para su gráfico, a todas las imágenes cuyo tipo de subconjunto de muestra (de los 7 establecidos en la sección 3.2.1.2) es uno predefinido por el usuario de este procedimiento. Usa para esto la referencia del nombre del archivo que sigue el código detallado en la sección 3.2.1.2. Si la matriz está ordenada, el resultado de este proceso es la exhibición de las imágenes de los 4 extremos de cada una de las 6 denominaciones del subconjunto de muestra predefinido, en sus respectivos órdenes ascendentes. Un ejemplo de una de estas exhibiciones se muestra en la figura 3.7, en la cual se halla graficado el primer subconjunto de imágenes de muestra, las rectas.

Figura 3.7. Exhibición del conjunto de imágenes originales

Se implementa mediante la función definida por el usuario `MostrarImg`, cuyo código se puede revisar en el anexo A4.4. El tipo de subconjunto de muestras a graficarse, predefinido por el usuario se determina con una variable, modificable en el rango del 1 al 7, en el programa principal del anexo A4.1, mediante la selección del valor de la variable `plotset`.

3.2.2.3 Normalización de las imágenes del conjunto original

Tiene como objetivo eliminar las variaciones de iluminación y ruido de las imágenes en el conjunto original de muestras, para igualar las estadísticas (media y varianza) de todas estas imágenes, según lo anotado en la teoría de la normalización de imágenes en la sección 2.2.4. También, sigue las recomendaciones del primer paso de la teoría de obtención de *eigenimages* de la sección 2.4.4.1. Sigue el diagrama de flujo de la figura 3.8.

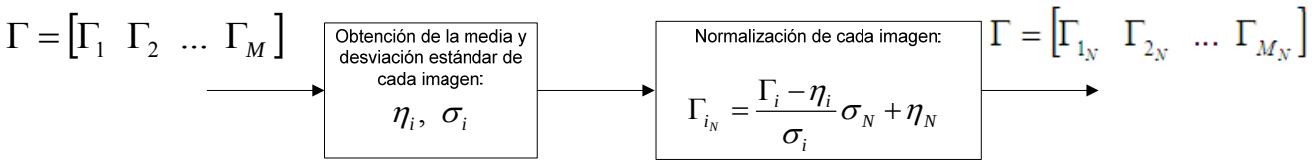


Figura 3.8. Proceso de normalización de las imágenes del conjunto original de muestras

Recibe como entrada a la matriz Γ de las imágenes del conjunto de muestra como vectores columna. Devuelve, como salida a la misma matriz Γ pero con todas sus imágenes normalizadas, igualmente, como vectores columna. Lo hace aplicando la ecuación 2.10 de la sección 2.2.4, a cada vector imagen Γ_i del conjunto original:

$$\Gamma_{iN} = \frac{\Gamma_i - \eta_i}{\sigma_i}\sigma_N + \eta_N$$

En la ecuación 2.10, se utilizan los valores siguientes para las nuevas estadísticas a las que se están igualando todas las imágenes:

$$\eta_N = 128$$

$$\sigma_N = 128$$

La selección de estos valores puede ser hecha de forma arbitraria porque el objetivo es simplemente *igualar* las estadísticas, a valores cualesquiera. Por supuesto, valores

extremos (0 y 255) de media y varianza no serían adecuados pues produce imágenes totalmente blancas o totalmente negras. Se selecciona, entonces, 128 para ambas variables puesto que es un valor medio entre 0 y 255. Las operaciones de este procedimiento se ejecutan en una línea del programa principal de entrenamiento, adjunto en el anexo A4.1.

3.2.2.4 Exhibición de las imágenes del conjunto original normalizadas

Tiene como objetivo la exhibición de las imágenes del conjunto original, normalizadas para la comprobación visual de que la normalización hecha en el anterior procedimiento se ha realizado correctamente.

Recibe como entrada a la matriz Γ de las imágenes normalizadas del conjunto de muestras, Γ_{i_N}, como vectores $N \times 1$, y como resultado, exhibe las imágenes de los 4 extremos de las 6 denominaciones del proyecto, para el mismo subconjunto de tipo de muestra predefinido por el usuario en el procedimiento *Exhibición de las imágenes del conjunto original*. Para esto, redimensiona cada uno de los vectores Γ_{i_N} a la dimensión original de las imágenes, 90×60, siguiendo el mismo método de ordenamiento del procedimiento *Formación del conjunto original de imágenes*. Un ejemplo de una de estas exhibiciones se muestra en la figura 3.9, en la cual se halla graficado el primer subconjunto de imágenes normalizadas de muestra, las rectas.

Figura 3.9. Exhibición del conjunto de imágenes normalizadas

Se implementa mediante la función de MATLAB definida por el usuario `MostrarImgNormalizadas`, cuyo código se puede revisar en el anexo A4.5. Utiliza la función de MATLAB `reshape` para redimensionar un vector a una forma matricial de imagen.

3.2.2.5 Sustracción de la imagen media del conjunto original

Tiene como objetivo hacer que el conjunto de imágenes originales tenga una media igual a cero, para facilitar las operaciones matriciales posteriores que requiere la matemática del PCA. Sigue el paso 2 del algoritmo del PCA (ver sección 2.4.3) para la realización de los pasos 3 y 4 del método de eigenimages (ver sección 2.4.4.1), que se está implementando en este entrenamiento. Cabe mencionar que este procedimiento es, en el contexto del PCA, la verdadera normalización de las imágenes. Sigue el diagrama de flujo de la figura 3.10.

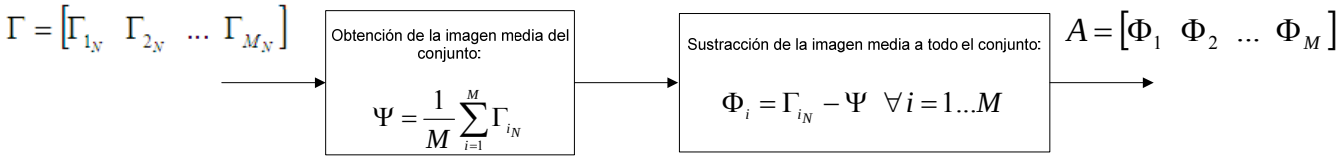


Figura 3.10. Sustracción de la imagen media del conjunto original

Recibe como entrada a la matriz $\Gamma = [\Gamma_{1_N} \Gamma_{2_N} \dots \Gamma_{M_N}]$ de las imágenes normalizadas del conjunto original de muestras, como vectores columna. Devuelve, como salida a la matriz $A = [\Phi_1 \Phi_2 \dots \Phi_M]$ que contiene, como vectores columna Φ_i, los resultados de la resta de cada imagen normalizada del conjunto original, como vector Γ_{i_N}, con la imagen media de dicho conjunto, Ψ, según indica la ecuación 2.44:

$$\Phi_i = \Gamma_{i_N} - \Psi \quad \forall i = 1...M$$

Para la producción de la imagen media, se aplica la ecuación 2.43:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_{i_N}$$

El nuevo conjunto obtenido contiene, como vectores columna, las imágenes Φ_i que están listas para participar en el proceso matemático de *eigenimages*. Este procedimiento

se implementa en líneas de código del programa principal de entrenamiento, adjunto en el anexo A4.1.

3.2.2.6 Exhibición de la imagen media del conjunto de imágenes normalizadas

Tiene como objetivo la exhibición de la imagen media del conjunto original, para la visualización de la imagen patrón que el sistema tratará de reconocer.

Recibe como entrada el vector de dimensión $N \times 1$ de la imagen media del conjunto de imágenes normalizadas, Ψ, obtenida en el procedimiento anterior. Hace un redimensionamiento de este vector a la dimensión original de las imágenes, 90×60, y exhibe esta imagen como resultado. En la figura 3.11 se muestra la imagen media del conjunto de imágenes de muestra, normalizadas, utilizado en el presente proyecto.

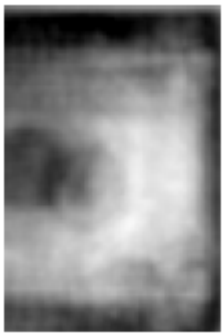


Figura 3.11. Imagen media del conjunto de imágenes original normalizado

Utiliza la función de MATLAB `reshape` para redimensionar un vector a una forma matricial de imagen a la imagen media, mediante líneas del código del programa principal, adjunto en el anexo A4.1.

3.2.2.7 Cálculo de la matriz equivalente a la matriz de covarianza, *L*

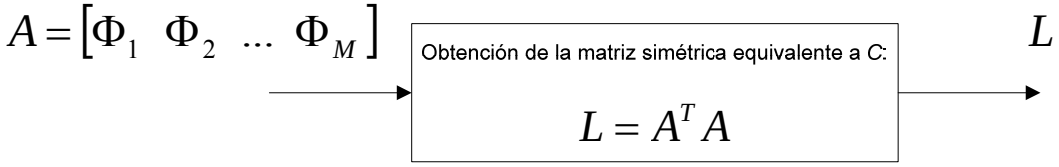


Figura 3.12. Cálculo de la matriz *L*

Tiene como objetivo el cálculo de la matriz *L*, definida en la teoría para la obtención de *eigenimages* de la sección 2.4.4.1 como una matriz simétrica equivalente a la matriz de covarianza C del conjunto de imágenes normalizadas en los anteriores procedimientos. El objetivo ulterior de este proceso es realizar el cálculo de los valores y vectores propios de la matriz de covarianza del conjunto de imágenes de muestra, *C*. Se obtiene *L* a través de la ecuación 2.48:

$$L = A^T A$$

En la figura 3.12, A^T es la transpuesta de la matriz de imágenes normalizadas, *A*.

El procedimiento recibe como entrada a la matriz *A*, que contiene como vectores columna a las imágenes normalizadas (en el contexto del PCA) del conjunto de muestras, obtenida en la sección *Sustracción de la imagen media del conjunto original*. La salida es la matriz simétrica *L*.

La matriz de covarianza del conjunto original de datos C debería tener dimensión $N \times N$, según la ecuación 2.45. En este proyecto, esa dimensión sería de 5400×5400. En cambio, la matriz *L* obtenida en este procedimiento es de dimensión $M \times M$ (en este proyecto, 168×168), lo que demuestra una reducción significativa que se reflejará en una facilidad para el cálculo de los valores y vectores propios de la matriz *C*, como lo establece la teoría de la sección 2.4.4.1

Se implementa el procedimiento en líneas de código del programa principal de entrenamiento adjunto en la sección A4.1.

3.2.2.8 Cálculo de los valores y vectores propios de la matriz *L*

Tiene como objetivo continuar el paso 6 del cálculo de *eigenimages* de la sección 2.4.4.1. Es un paso intermedio en el cálculo de los valores y vectores propios de la matriz de covarianza *C* del conjunto original de imágenes normalizadas, *A*, objetivo del entrenamiento.

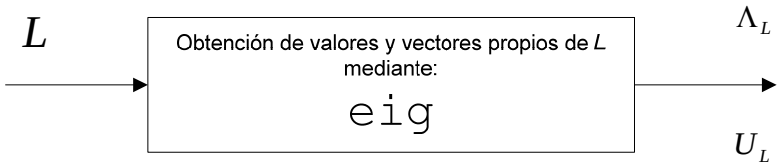


Figura 3.13. Cálculo los valores y vectores propios de *L*

Recibe, como entrada, a la matriz simétrica *L* calculada en el anterior procedimiento. Produce, a su salida, una matriz diagonal Λ_L con los valores propios de *L* y una matriz U_L con los correspondientes vectores propios de *L*.

Como se ha enunciado en la sección 2.4.2.3, que trata sobre los valores y vectores propios de una matriz, el cálculo se realiza mediante métodos matemáticos que son más o menos complejos dependiendo de la dimensión de la misma. Sin embargo, en la actualidad existen una gran cantidad de herramientas de *software* con librerías para calcular estas cantidades, mismas que siempre deberán relacionar a los valores propios con sus respectivos vectores propios. Así, en este proyecto, se utiliza la función de MATLAB, `eig`, que automáticamente devuelve en dos matrices los valores propios de la matriz *L* y sus respectivos vectores propios.

Así, este procedimiento produce como salida las dos matrices enunciadas en el paso 4 de la teoría del PCA de la sección 2.4.3, respectivamente en las ecuaciones 2.39 y 2.40: Λ_L y U_L. Se recuerda que cada vector propio ubicado en cada columna de la matriz U_L, está asociado al valor propio en la correspondiente columna de la matriz Λ_L.

Se implementa el procedimiento dentro de las líneas de código del programa principal de entrenamiento adjunto en la sección A4.1:

```
[vL lambdaL]=eig(L);%Vectores y valores propios de L
```

3.2.2.9 Ordenamiento descendente de los vectores y valores propios de *L*

Tiene como objetivo jerarquizar la importancia de los vectores propios obtenidos en el anterior procedimiento, según sus respectivos valores propios, de acuerdo a la teoría del paso 5 de la sección 2.4.3, del método del PCA, que indica que los vectores propios más importantes son aquellos cuyos valores propios asociados sean los de mayor valor absoluto. Sigue el diagrama de la figura 3.14.

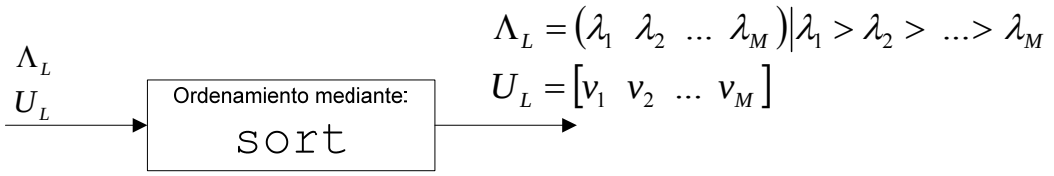


Figura 3.14. Ordenamiento de valores y vectores propios de *L*

Recibe como entrada las variables obtenidas en el anterior procedimiento, de la siguiente forma: la matriz diagonal de valores propios Λ_L y la matriz de vectores propios U_L, en la cual, cada vector propio se halla ubicado como columna. El índice de la columna de cada vector propio en U_L indica que corresponde al valor propio de la misma columna, en la matriz Λ_L. Con la herramienta informática `eig` usada en el anterior procedimiento, los valores propios no tienen ningún orden en particular.

Produce como salida, según la figura 3.14, el ahora *vector* Λ_L de valores propios ordenado de izquierda a derecha, con el valor propio de máximo valor absoluto a la izquierda; y la matriz U_L , pero ahora con los vectores propios de la matriz *L* ubicados por columnas, en la columna de índice correspondiente al de su valor propio en el anterior vector. Con esto, los vectores propios de la matriz *L* están ordenados de igual forma que los valores propios.

Para la realización de este procedimiento, se hace uso de la función de MATLAB `sort`, mediante las líneas de código del programa principal de entrenamiento, adjunto en el anexo A4.1.

3.2.2.10 Obtención de los valores y vectores propios de la matriz de covarianza del conjunto original, *C*

Tiene como objetivo la determinación de todos los componentes principales del conjunto de imágenes de muestra del presente programa de entrenamiento, que son los vectores propios de *C*, así como sus valores propios. Sigue la teoría de la sección 2.4.4.1, como parte del paso 6 del método de las *eigenimages*.

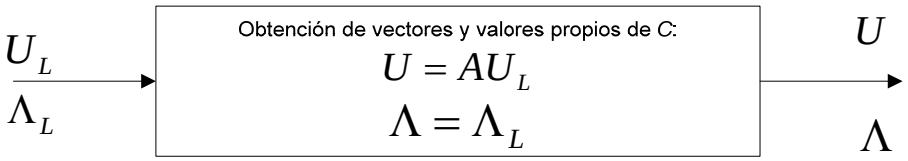


Figura 3.15. Obtención de los valores y vectores propios de *C*

Recibe como entradas a U_L, la matriz ordenada de vectores propios de la matriz *L*, a la matriz *A* del conjunto de muestras normalizadas, obtenida en la sección *Sustracción de la imagen media del conjunto original* y al vector ordenado de valores propios de *L*, Λ_L.

Produce como salida a la matriz U de vectores propios de la matriz C de covarianza del conjunto *A*. Lo hace mediante la operación matricial $U = AU_L$ mostrada en la figura 3.15 que implementa de por sí en cada vector propio la ecuación 2.49 de la teoría de las *eigenimages*:

$$u_i = Av_i$$

Ecuación en la cual se interpreta como v_i a cada vector propio de la matriz *L* y como u_i a cada vector propio de la matriz *C*.

Además, según la teoría de la sección 2.4.4.1, los valores propios de la matriz *C* son los mismos que los de la matriz *L*, por lo que no hay necesidad de realizar cálculo alguno para los mismos, y solo se manifiesta en la figura 3.15 que los valores propios de la matriz *C* se hallan en el vector Λ, que es el mismo que el vector Λ_L; es decir: $\Lambda = \Lambda_L$.

Con esto, ya se tienen a los verdaderos vectores y valores propios requeridos, los de la matriz de covarianza del conjunto de datos original, *C*. Cabe anotar aquí que el algoritmo implementado multiplica a todos los valores propios así obtenidos por un factor

de 10000. Esto servirá para tareas luego analizadas, en la implementación del código del sistema *online*, para ganar cifras de apreciación en cálculos.

3.2.2.11 Normalización de los vectores propios de la matriz de covarianza *C*

Tiene como objetivo cumplir el requerimiento de la matemática del PCA de que la matriz de vectores propios del conjunto de datos en análisis deba ser *ortonormal* para poder representar una transformación lineal de cambio de base, como se establece en la teoría de la sección 2.4.3. Dado que los vectores propios de una matriz son ortogonales [22], el requisito faltante es que sean normalizados. Se realiza para dar conclusión al paso 6 de la sección 2.4.4.1, siguiendo el método de las *eigenimages*. Sigue el diagrama de la figura 3.16.

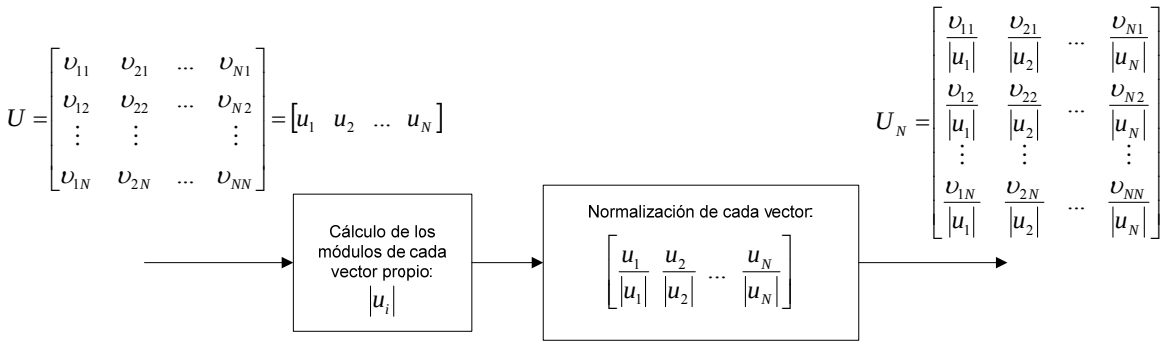


Figura 3.16. Obtención de los valores y vectores propios de *C*

Recibe como entrada a la matriz U de vectores propios del conjunto de imágenes obtenida en el anterior procedimiento. Produce como salida a la matriz normalizada U_N con todos los vectores propios de sus columnas normalizados. Se realiza mediante la aplicación de la ecuación 2.41:

$$U_N = \begin{bmatrix} \frac{\upsilon_{11}}{|u_1|} & \frac{\upsilon_{21}}{|u_2|} & \dots & \frac{\upsilon_{N1}}{|u_N|} \\ \frac{\upsilon_{12}}{|u_1|} & \frac{\upsilon_{22}}{|u_2|} & \dots & \frac{\upsilon_{N2}}{|u_N|} \\ \vdots & \vdots & & \vdots \\ \frac{\upsilon_{1N}}{|u_1|} & \frac{\upsilon_{2N}}{|u_2|} & \dots & \frac{\upsilon_{NN}}{|u_N|} \end{bmatrix}; donde: U = \begin{bmatrix} \upsilon_{11} & \upsilon_{21} & \dots & \upsilon_{N1} \\ \upsilon_{12} & \upsilon_{22} & \dots & \upsilon_{N2} \\ \vdots & \vdots & & \vdots \\ \upsilon_{1N} & \upsilon_{2N} & \dots & \upsilon_{NN} \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_N \end{bmatrix}$$

3.2.2.12 Reducción al espacio de *K* componentes principales

Tiene como objetivo la final determinación de los *componentes principales* del conjunto de imágenes de muestra del presente programa de entrenamiento, que son los *K* vectores propios normalizados que corresponden a los *K* valores propios más altos del conjunto de imágenes. Sigue el paso 7 del método de obtención de *eigenimages* de la sección 2.4.4.1, y se fundamenta en la teoría del paso 5 del PCA, de la sección 2.4.3. Sigue el diagrama de la figura 3.17.

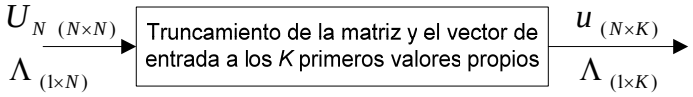


Figura 3.17. Reducción al espacio de *K* componentes principales

Recibe como entradas a la matriz U_N de vectores propios ortonormales y ordenados descendentemente del conjunto de imágenes, obtenida en el anterior procedimiento, que es de dimensión $N \times N$, y al vector Λ de los *N* valores propios ordenados descendentemente, del conjunto de imágenes obtenido en los anteriores procedimientos.

Produce como salidas a la matriz u, de dimensión $N \times K$ con los *K* vectores propios, seleccionados por el procedimiento como columnas, así como a la misma matriz Λ, pero ahora compuesta solo por los *K* valores propios más altos que se corresponden con los vectores propios de la matriz u producida.

Se realiza simplemente truncando la matriz y vector de entrada, en sus primeras *K* columnas, ya que, al estar ordenados descendentemente, de los procedimientos anteriores, al truncarlos se estará ya seleccionando los elementos correspondientes a los *K* valores propios más altos. El resultado es que los vectores y valores propios del conjunto final de datos tendrá menor dimensión que el original ($K < N$).

En este proyecto, se seleccionan, de todo el conjunto de valores y vectores propios de la matriz de covarianza del conjunto de datos, *C*, sólo los correspondientes a los primeros *K=24* valores propios más altos para lograr la reducción del espacio, de *N* a *K* dimensiones (5400 a 24, en este caso), que son los componentes principales del conjunto original. Esta

selección, como se ha dicho en la teoría, es empírica y, en el caso de este proyecto se ha hecho con el criterio de que existen: *K=4 extremos* $\times$ *6 denominaciones =24 especies*, que corresponden a 24 componentes principales que, al ser de una dimensión de *N*=5400, son en realidad imágenes. Por esto, se conoce a estas cantidades como *eigenimages*, ya que son vectores propios de la matriz de covarianza del conjunto original, por lo que son *"imágenes propias"*.

3.2.2.13 Exhibición de los *K* vectores propios como *eigenimages*

Tiene como objetivo la exhibición de los vectores propios obtenidos en el procedimiento anterior, como matrices de imágenes, para la observación de la apariencia de las *eigenimages* que determina el método de la sección 2.4.4.1.

Recibe como entrada a la matriz u de los *K* vectores propios ortonormales más relevantes, como vectores $N \times 1$, y como resultado, los exhibe en forma de imágenes. Para esto, redimensiona cada uno de los vectores $u_1, u_2, ..., u_K$ a la dimensión original de las imágenes, 90×60. La exhibición de las *eigenimages* de este proyecto se muestra en la figura 3.18.

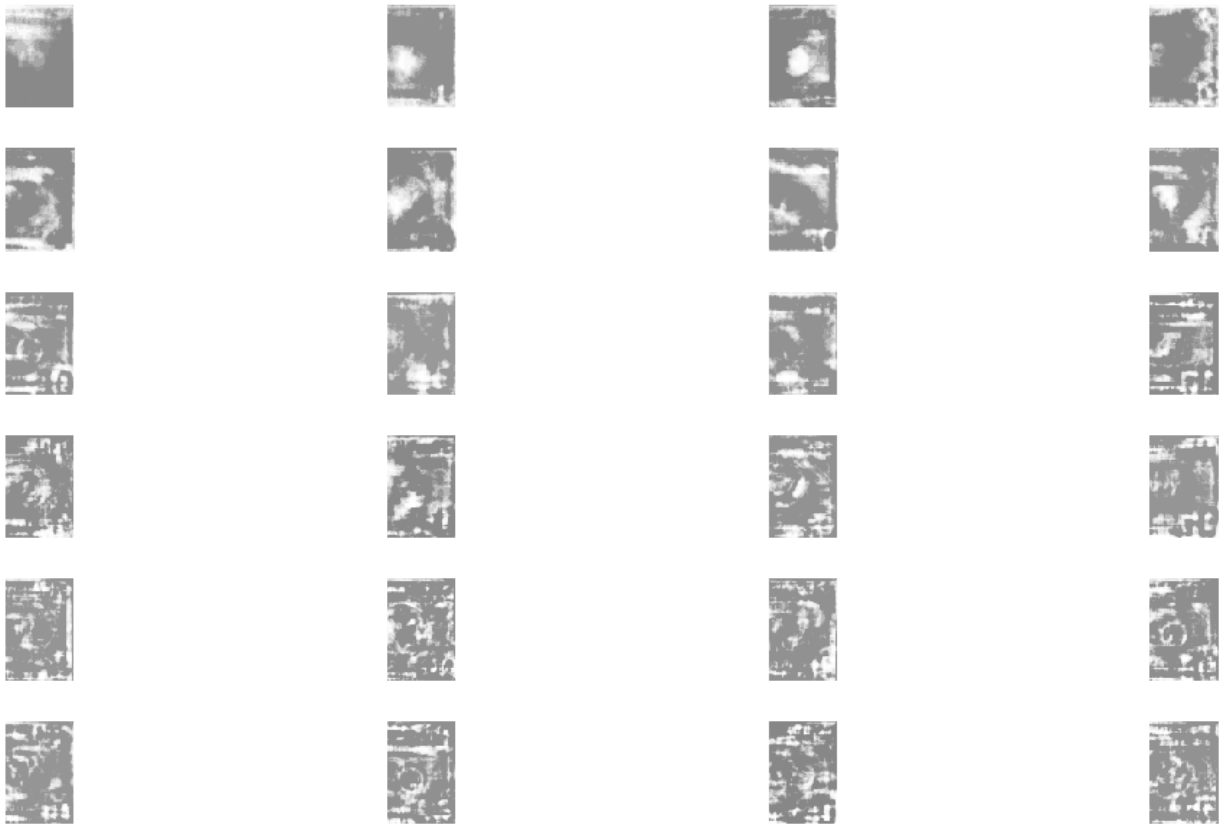


Figura 3.18. Los *K* vectores propios como *eigenimages*

En este proceso del programa se han utilizado varias herramientas de MATLAB, como `reshape`, `histeq` e `imshow` para redimensionar los vectores propios en la matriz u y exhibirlos como matrices de imágenes, al igual que se ha hecho con las imágenes originales. En la figura 3.18 se pueden apreciar las *K=24 eigenimages* con apariencias *"fantasmas",* comprobando lo establecido en la teoría de la sección 2.4.4.1.

Se debe recordar que estas *eigenimages* son además las *bases* del nuevo espacio reducido, *K-dimensional* en el cual se quería representar a las imágenes originales y a cualquiera de entrada del posterior sistema a implementarse en el dispositivo microprocesado. De aquí en adelante este espacio nuevo será conocido como *espacio de billetes*, ya que forman en u^T la matriz de cambio de base *P*, que es el objetivo del PCA, como se examina en la sección 2.4.3.

3.2.2.14 Proyección del conjunto original de imágenes al nuevo espacio, para formar el nuevo conjunto de pesos

Tiene como objetivo realizar la transformación lineal de cambio de base de las imágenes del conjunto original de muestras al nuevo espacio de billetes para producir un *conjunto de pesos* (vectores de dimensión reducida, *K*) que representan a cada una de las imágenes en el espacio original de dimensión *N*. Sigue los criterios dados en el último paso del método del PCA, resumidos en la expresión 2.33 y cumple el último paso del método de las *eigeninages* de la sección 2.4.4.1. El procedimiento sigue el diagrama de la figura 3.19.

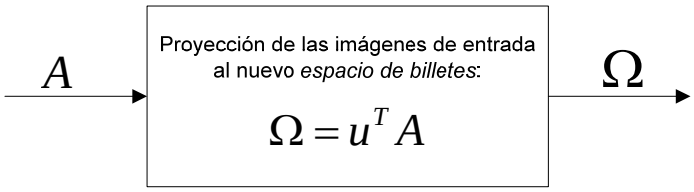


Figura 3.19. Proyección del conjunto de imágenes al nuevo *espacio de billetes*

Recibe como entrada a la matriz *P* de cambio de base que requiere el método del PCA, dada en la ecuación 2.42, que fue finalmente obtenida en el anterior procedimiento como la matriz u^T y que contiene a los componentes principales como vectores fila; también recibe a la matriz *A*, que contiene como vectores columna de dimensión $N \times 1$ a las imágenes normalizadas del conjunto de muestras.

Produce como salida una matriz Ω que contiene como vectores columna de dimensión $K \times 1$ a los pesos que representan a las imágenes originales, en el espacio de billetes.

El procedimiento proyecta cada imagen en el nuevo espacio de billetes mediante la aplicación de la expresión 2.50:

$$\Omega = u^T A$$

La operación en 2.50 es la forma matricial de un producto punto entre cada vector propio en u^T y cada una de las imágenes del conjunto de muestras en *A*. Los nuevos datos serán conocidos como *conjunto de pesos de las muestras*, ya que son *M=168* vectores de dimensión *K=24*, que representan a cada imagen del conjunto original de imágenes, en el nuevo espacio de billetes. La matriz Ω es de dimensión $K \times M$ ya que guarda a cada peso en forma de columna. Como se puede ver en el código de este procedimiento, en el anexo A4.1, en esta parte también se ha calculado la matriz de covarianza de este nuevo conjunto de imágenes, según los criterios del PCA y se ha observado que, efectivamente, es una matriz diagonal ordenada, como requería el método.

3.2.2.15 Almacenamientos para carga al teléfono celular

Finaliza el algoritmo de entrenamiento, proporcionando los datos que se requieren para el siguiente paso en el diseño de la presente solución, el reconocimiento *online* de una imagen de entrada al sistema microprocesado detallado en la figura 3.1, a través de la teoría del reconocimiento de una imagen de entrada mediante *eigenimages* de la sección 2.4.4.2.

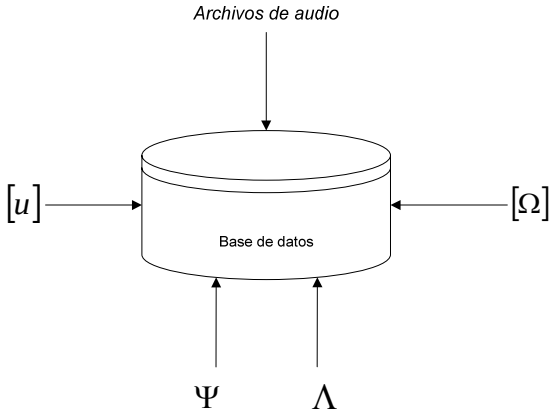


Figura 3.20. Conformación de la base de datos producto del programa de entrenamiento

Para esto se deben guardar a las variables que se desean obtener de este programa en forma de vectores. El propósito básico en el reconocimiento *online* de los billetes será procesar imágenes de entrada para obtener los pesos de esa imagen en el *espacio de billetes* creado en esta sección. Es lógico pensar, entonces, que la imagen de entrada deberá pasar por un proceso similar al que han tenido que pasar todas las imágenes del conjunto original de muestras en esta sección, es decir que la imagen de entrada (una vez que se haya pre procesado y determinado de la estética dictaminada en la sección 3.2.1.1) debe ser: normalizada, restada la imagen media del conjunto y, finalmente, proyectada al nuevo *espacio de billetes*; todo esto, con las expresiones usadas en esta sección. De aquí que, los datos que el sistema microprocesado encargado del reconocimiento debe tener cargados en su memoria, obtenidos de las variables del presente programa de MATLAB, son:

- La imagen media del conjunto de imágenes de muestra, determinada en el procedimiento *Sustracción de la imagen media del conjunto original,* en la variable Ψ, como un vector de dimensión *N* (5400 x 1). Se almacenará en el archivo *carga1.txt.* Este es el dato simbolizado por la variable vector Ψ en el esquema del sistema de la figura 3.1.
- Las *eigenimages*; es decir, los vectores propios normalizados de la matriz de covarianza del conjunto de datos original, *C*, obtenidos en el procedimiento *Reducción al espacio de K componentes principales,* en el cual, fueron organizados en la variable matriz u, en *K=24* vectores columna de dimensión *N* (5400 x 1). Se almacenará a cada uno de estos i vectores en un respectivo archivo *i.txt,* y a todos estos archivos, en una carpeta, llamada *"Eigen".* Este es el conjunto de datos simbolizado por la variable matriz $[u]$ en el esquema del sistema de la figura 3.1.

Además, según la teoría de la sección 2.4.4.2, para la determinación propiamente dicha de la denominación a la cual pertenece la imagen de entrada en el sistema *online* mediante la métrica de la distancia de *Mahalanobis*, se requiere que la memoria del sistema tenga cargados también los siguientes datos:

- Los valores propios, obtenidos en el procedimiento *Reducción al espacio de K componentes principales*, en la variable Λ, en un vector de dimensión *K* (24 x 1). Se almacenará en la continuación del archivo *carga1.txt*. Este es el dato simbolizado por la variable vector Λ en el esquema del sistema de la figura 3.1.

- Los pesos del conjunto de imágenes en el *espacio de billetes* obtenidos en el procedimiento *Proyección del conjunto original de imágenes al nuevo espacio, para formar el nuevo conjunto de pesos*; en la variable Ω, en *M=168* vectores de dimensión *K* (24 x 1). Se almacenará a cada uno de estos *i* vectores en un respectivo archivo *i.txt,* y a todos estos archivos, en una carpeta, llamada *"Omega"*. Este es el conjunto de datos simbolizado por la variable matriz $[\Omega]$ en el esquema del sistema de la figura 3.1.

Todos estos valores se guardan como datos enteros de 32 bits, de tipo *int32*, para ganar apreciación en los cálculos en el sistema microprocesado: el teléfono celular que se describirá en la siguiente sección. La implementación de los presentes procesos de almacenamiento, se realizan mediante la utilización de funciones de MATLAB para el manejo de archivos como `fopen`, `fprintf` y `fclose`. Este y todo código que no se haya mencionado de una forma explícita en la redacción de la sección 3.2, se halla en el código del programa principal del anexo A4.1, con sus respectivos comentarios para su mejor comprensión.

Como último paso, para completar la formación de la base de datos, se tiene que almacenar una cantidad de mensajes de audio. El formato seleccionado para estos archivos se discutirá en la sección 3.3. Se conforma el conjunto de datos *Archivos de audio* del esquema del sistema en la figura 3.1, con los archivos: *uno.mp3*, *cinco.mp3*, *diez.mp3*, *veinte.mp3*, *cincuenta.mp3*, *cien.mp3*, *adios.mp3*, *bienvenido.mp3*, *otra.mp3* y *reinicie.mp3*. Las razones para la existencia de estos mensajes se aclararán en los requerimientos de *software* de la sección 3.3.2.1.

Finalmente, toda la base de datos queda conformada dentro de un directorio llamado *Lectbill* cuyo contenido, con el respectivo espacio que ocupa en disco, es:

- Archivo *carga1.txt* (20 KB).
- Carpeta de archivos *Audio* (100 KB).
- Carpeta de archivos *Eigen* (772 KB).
- Carpeta de archivos *Omega* (672KB).

De tal forma, la base de datos producida ocupa, *1.52* MB de espacio en disco.

3.3 DESARROLLO DEL SISTEMA EN EL TELÉFONO MÓVIL

3.3.1 *Hardware*

3.3.1.1 Requerimientos de *hardware*

A continuación se describe los requerimientos mínimos de *hardware* para el prototipo diagramado en la figura 3.1 que abre el presente capítulo:

- **Memoria RAM:** El prototipo requiere *2 Megabytes* (MB) de memoria RAM. Esta cantidad se obtuvo de la función:

  ```
  void CCameraAppBaseContainer::mostrarRAM()
  ```

  Que fue una función implementada como parte del desarrollo de *software* en este proyecto que calcula la cantidad de memoria RAM utilizada por las variables del programa. Se puede hallar adjunto su código en el anexo A5.23.

- **Memoria interna:** Para determinar la cantidad memoria interna utilizada se debe tomar en cuenta dos aspectos:

  - **Memoria para la aplicación a instalarse:** Es la cantidad de memoria ocupada por el programa de aplicación obtenido como resultado del proyecto, instalado en el dispositivo celular. Esta cantidad es de *77 Kilobytes* (KB).

  - **Memoria para la base de datos del programa:** La memoria requerida por la base de datos del sistema es de *1.52 Megabytes* (MB), como se vio al final de la sección 3.2.2 del programa de entrenamiento. Este valor es la suma de los tamaños en disco de los ficheros relacionados con PCA (*eigenimages*, imagen media, valores propios y pesos) y la carpeta que contienen los mensajes de audio. Todos los archivos de esta base de datos están, en el dispositivo móvil, en el directorio *C:\\Data\\Lectbill*, cuya estructura se describe en la tabla 3.5. El formato seleccionado para los archivos de audio es MP3, debido al pequeño tamaño en disco que ocupan en comparación al formato WAV. Para el resto de ficheros, se escogió un formato de texto plano con codificación ANSI, el cual es compatible con las

funciones de lectura de archivos del lenguaje de programación seleccionado: *Symbian* C++ (ver sección 3.3.2.2).

Sumando ambos valores de memoria, se obtiene la cantidad total de memoria requerida por el prototipo: 1.52 MB + 77 KB = 1.6MB

- **Procesador (CPU):** El prototipo requiere de un procesador capaz de alcanzar al menos *1.04* DMIPS[4]/MHz *a 220* MHz, lo que da como resultado: *(220*MHz*)(1.04* DMIPS/MHz*) = 228* DMIPS.
  Se requiere esta cantidad de instrucciones por segundo ya que es la alcanzada por el procesador ARM926EJ-S [25] del teléfono móvil en el que se implementó con éxito el prototipo: *Nokia* N73 (ver sección 3.3.1.2). En procesadores de menor desempeño al utilizado, no se puede asegurar un procesamiento en tiempo real de la imagen.

- **Sistema de adquisición de imágenes:** Para determinar qué características debe reunir el sistema de adquisición de imágenes, que será la cámara digital del prototipo, es necesario tomar en cuenta que lo siguiente:

  - **Tipo de sensor:** Se requiere que la cámara posea un sensor a color que entregue la información según el modelo RGB. Esto es necesario ya que el sistema recibe como entrada una imagen en color RGB que posteriormente será transformada a gris. No se procesa la imagen RGB por la naturaleza del procesamiento de imágenes utilizado (*i.e.* PCA trabaja con imágenes en escala de grises) y por cuestiones de eficiencia computacional (*i.e.* una imagen RGB requiere el triple de tiempo de procesamiento).

  - **Resolución:** Generalmente, se entiende por resolución a la cantidad de píxeles que tiene una *imagen digital almacenada en la memoria interna* que ha sido fotografiada por la cámara fotográfica (*e.g.* 0.3 Megapixeles, 2 Megapixeles, 3.2 Megapixeles). Sin embargo, en el presente prototipo esta resolución no tiene importancia ya que lo que procesa el sistema no es la imagen digital almacenada en disco, sino cada cuadro o *frame* del

[4] *Dhrystone Million instructions per second* (*DMIPS*) cuantifica la cantidad de millones de instrucciones por segundo que ejecuta un procesador.

viewfinder[5] (ver sección 3.3.2.1). Por esto, la resolución a la que hace referencia este inciso es a la resolución del *viewfinder*. Debido a que la imagen del *viewfinder* se despliega a manera de vídeo en la pantalla del celular, la resolución de éste está limitado por la resolución de dicha pantalla. En este prototipo, se requiere que la resolución de la pantalla sea al menos *320 × 240 píxeles*. Se necesita dicha valor porque con resoluciones más bajas, se pierde información (cualitativamente las imágenes pierden calidad) y además, en la práctica, la resolución del *viewfinder* es menor que la de la pantalla (*e.g.* el *Nokia* N73 tiene una pantalla de resolución 320 × 240 mientras que la resolución del *viewfinder* alcanza solo 266 × 200). La resolución del *viewfinder* más baja en la que se probó con éxito el prototipo fue la de la cámara del *Nokia* E65: 240 × 180 en una pantalla de 320 × 240. Por lo general, el fabricante del celular no provee información de la resolución del *viewfinder* pero sí de la pantalla, por lo que los requerimientos mínimos harán referencia a esta última teniendo en cuenta que ambas resoluciones están relacionadas. En este proyecto la resolución del *viewfinder* fue determinada con código mediante las librerías de *Symbian* C++.

- **FPS en estado normal:** Es la cantidad de *frames* por segundo (FPS) que el sistema puede procesar sin ejecutar ninguna tarea de procesamiento extra, es decir, es el valor de FPS resultantes de sensar y desplegar en pantalla la imagen del *viewfinder*. Se puede inferir fácilmente que a mayor capacidad en DMIPS del procesador, mayor es el valor de FPS en estado normal. Por otra parte, a mayor resolución del *viewfinder*, menor es la cantidad de FPS en estado normal. Teniendo en cuenta esta definición, el presente prototipo requiere que el sistema procese al menos 15 FPS en estado normal. Este valor es necesario porque, de acuerdo a los resultados obtenidos en los teléfonos *Nokia* E65 y *Nokia* N73 (ver sección 4.5), al momento de ejecutar las funciones de procesamiento de imágenes, la velocidad cae a 7 FPS. Si la velocidad en estado normal fuese menor, la velocidad con carga disminuiría debajo de 7 FPS, lo cual comprometería el desempeño del sistema.

[5] Se entiende por *Viewfinder* al *buffer* de memoria que contiene la imagen que se actualiza periódicamente con la información entregada por los sensores de la cámara. Por lo general la imagen del *Viewfinder* es desplegada a manera de video en la pantalla del celular

- **Multimedia:** la característica multimedia que requiere el prototipo es la reproducción de archivos en formato MP3. Se prefirió utilizar este formato debido a que ocupan menos espacio en disco que su similar y comúnmente usado WAV. Además, MP3 es uno de los estándares más difundidos y fáciles de crear y editar a diferencia de otros como el formato WMA.

La tabla 3.1 resume los requerimientos de *hardware* aquí planteados del sistema.

Tabla 3.1. Requerimientos mínimos de *hardware* del sistema

<table>
<tr><th>Parámetro</th><th colspan="3">Requerimiento mínimo</th></tr>
<tr><td>Memoria RAM</td><td colspan="3">2 Megabytes</td></tr>
<tr><td>Memoria en disco</td><td colspan="3">1.6 Megabytes</td></tr>
<tr><td>Procesador</td><td colspan="3">1.04 DMIPS/MHz a 220 MHz</td></tr>
<tr><td rowspan="3">Sistema de adquisición de imágenes</td><td colspan="2">Sensor:</td><td>Color</td></tr>
<tr><td rowspan="2">Resolución:</td><td>Pantalla</td><td>320x240</td></tr>
<tr><td>Viewfinder</td><td>240x180</td></tr>
<tr><td>FPS en estado normal</td><td colspan="3">15 FPS</td></tr>
<tr><td>Multimedia</td><td colspan="3">MP3</td></tr>
</table>

3.3.1.2 Selección de la plataforma de *hardware*

De acuerdo a las entrevistas con la población no vidente, sus necesidades con respecto a un sistema de reconocimiento de denominación de dólares americanos se traducen en tres aspectos:

- *Portabilidad*: El sistema debe ser ligero y pequeño.
- *Penetrabilidad en el mercado*: El sistema debe estar basado en un dispositivo existente y fácil de encontrar en el mercado (*e.g.* palm, celular).
- *Facilidad de uso*: Que su modo de uso sea intuitivo, simple y transparente.

La plataforma de *hardware* que reúne estos 3 aspectos es un teléfono celular debido a su portabilidad, amplia aceptación en el mercado y facilidad de uso. Existen una infinidad de teléfonos celulares disponibles en el mercado, sin embargo, por las exigencias de *hardware* resumidas en la tabla 3.1, la plataforma escogida es la de los *teléfonos móviles inteligentes* o *smartphones*, los cuales poseen la capacidad de procesamiento, memoria y cámara digital adecuadas para manejar aplicaciones relativamente complejas como edición de documentos, *web browsers*, juegos 3D; al mismo tiempo que realizan las funciones de un teléfono móvil convencional, según requiere este proyecto.

Existen varias opciones de *hardware* en lo que concierne a teléfonos móviles inteligentes o *smarthphones* cuyo mercado global, en el segundo trimestre del 2009, estaba distribuido de la siguiente manera [26]:

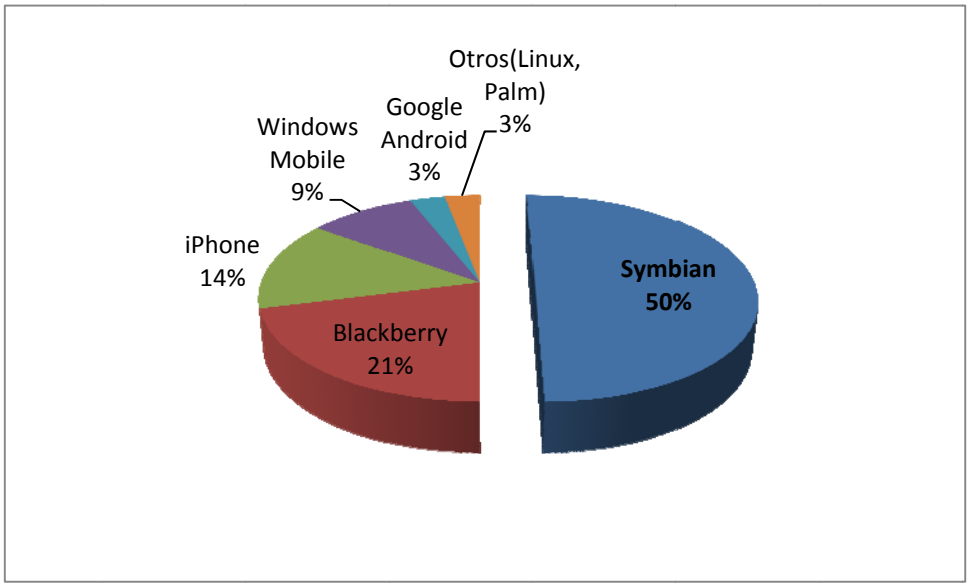


Figura 3.21. Participación en el mercado global de *smartphones* por sistema operativo

Como se observa en la figura 3.21, los teléfonos móviles inteligentes con sistema operativo *Symbian* acaparan la mitad del mercado. De otra parte, existen al menos 3 fabricantes de *smartphones Symbian*: *Samsung, LG* y *Nokia*; siendo este último el que tiene la participación más alta en el mercado con un 44 % [26].

Por la amplia aceptación de teléfonos *Nokia* en el mercado local, se decidió utilizar teléfonos móviles inteligentes *Nokia* con sistema operativo *Symbian* para la implementación del prototipo.

Existen varias versiones de *Symbian* como sistema operativo; en este proyecto se escogió la *versión 9.1*, más conocida como *Symbian S60, 3era edición, First Release*, ya que al momento del desarrollo de este proyecto, se disponía de 2 teléfonos *Symbian* con dicha versión donde se implementó con éxito el prototipo: *Nokia N73* y *Nokia E65*, cuyas características se detallan en la tabla 3.2, de acuerdo a los requerimientos mínimos.

Tabla 3.2. Características de los *smartphones* usados como plataforma de *hardware*

<table>
<tr><th>Parámetro</th><th colspan="3">Requerimiento mínimo</th><th>Nokia E65</th><th>Nokia N73</th></tr>
<tr><td>Memoria RAM</td><td colspan="3">2 MB</td><td>64 MB con 24 MB disponibles para aplicaciones de usuario</td><td>64 MB con 24 MB disponibles para aplicaciones de usuario</td></tr>
<tr><td>Memoria interna</td><td colspan="3">1.6 MB</td><td>42 MB</td><td>42 MB</td></tr>
<tr><td>Procesador</td><td colspan="3">1.04 DMIPS/MHz a 220 MHz</td><td>ARM926EJ-S, 1.04 DMIPS/MHz a 220 MHz</td><td>ARM926EJ-S, 1.04 DMIPS/MHz a 220 MHz</td></tr>
<tr><td rowspan="3">Sistema de adquisición de imágenes</td><td colspan="2">Sensor:</td><td>Color</td><td>Color</td><td>Color</td></tr>
<tr><td rowspan="2">Resolución:</td><td>Pantalla</td><td>320×240</td><td>320×240</td><td>320×240</td></tr>
<tr><td>Viewfinder</td><td>240×180</td><td>240×180</td><td>266×200</td></tr>
<tr><td>FPS en estado normal</td><td colspan="3">15 FPS</td><td>15 FPS</td><td>15 FPS</td></tr>
<tr><td>Multimedia</td><td colspan="3">MP3</td><td>MP3, WAV, otros</td><td>MP3, WAV, otros</td></tr>
</table>

Como se puede apreciar, las características de ambos dispositivos cumplen los requerimientos de *hardware* de la sección 3.3.1.1.

A continuación se listan los *smartphones Nokia Symbian S60,* 3º edición, *First Release,* de características similares a los utilizados como prototipo:

Nokia N77, Nokia E61i, Nokia E65, Nokia N93i, Nokia N91 8GB, Nokia E62, Nokia E50, Nokia 5500 Sport, Nokia N73, Nokia N93, Nokia N92, Nokia N71, Nokia N80, Nokia E60, Nokia E61, Nokia E70, Nokia 3250, Nokia N91.

Además de estos dispositivos, la aplicación prototipo puede correr en teléfonos cuya versión de *Symbian* es la 9.2 o más conocida como *Symbian S60 Tercera edición, Feature Pack 1*. Aunque no se pudo probar el *software* en ninguno de estos teléfonos, el prototipo debería funcionar adecuadamente ya que como se verá en la sección 3.3, el sistema está basado en un ejemplo de aplicación para acceder a la cámara provisto por *Nokia*, el cual es

compatible tanto con *Symbian 9.1* (versión de los teléfonos donde se instaló con éxito el prototipo) como *Symbian 9.2*. A continuación se listan los teléfonos *Nokia Symbian 9.2*:

Nokia E63, Nokia E71, Nokia E66, Nokia 6124 classic, Nokia N82, Nokia E51, Nokia N95-3 NAM, Nokia N95 8GB, Nokia N81 8GB, Nokia N81, Nokia 6121 classic, Nokia 6120 classic.

3.3.2 *Software*

3.3.2.1 Requerimientos de *software*

Una vez seleccionada la plataforma de *hardware* que cumple con los requisitos de portabilidad y penetrabilidad en el mercado sugeridos por los no videntes, para conseguir el requisito de facilidad de uso, el prototipo debe cumplir los siguientes requerimientos:

- **Modo de uso**: El prototipo debe trabajar con cuadros o *frames* entregados periódicamente por el *viewfinder* a manera de una secuencia de vídeo. Para el no vidente, esto significa que tiene que *filmar* el billete y no fotografiarlo, lo cual hace al sistema implementado simple de utilizar. La distancia entre el billete y el celular puede variar ya que el objetivo es que el no vidente *busque* al billete con el foco de la cámara. Una vez que el billete ha sido detectado, éste será reconocido. En el manual de usuario (Anexo A3) se detalla el método de utilización del presente sistema, el cual cumple los requerimientos expuestos en este inciso.

- **Mensajes de audio**: El sistema tiene que disponer de los siguientes mensajes de audio que ayudarán al no vidente como interfaz de usuario:

    - **Mensaje de bienvenida**: Este mensaje se debe reproducir cada vez que se inicia la aplicación para notificar al no vidente que ya puede colocar un billete frente al ojo de la cámara del celular.

    - **Mensaje "Pruebe con otro lado"**: Este mensaje notifica al usuario que el sistema, por cualquier razón (*e.g.* lado del billete rayado, sucio, alterado), no ha podido determinar la denominación de un billete. De este modo, el no vidente sabrá que tiene que intentar con otro lado o extremo del billete. En este proyecto los extremos de un billete se identifican de acuerdo a la figura 3.3.

- **Mensajes de denominación**: Notifican al usuario el valor reconocido del billete ingresado. Este mensaje debe reproducirse solo si 3 *frames* seguidos de la filmación han sido reconocidos como un billete de cierta denominación. Con esto se disminuye el riesgo de resultados falsos positivos, que son un anuncio de una denominación dada a pesar de que no se estaba filmando una imagen (ver sección 4.2).

- **Mensaje de reinicio**: Notifica al no vidente para que proceda a reiniciar la aplicación en caso de que ésta no haya sido inicializada correctamente o haya perdido el foco (*e.g.* cuando se ejecuta el salvapantallas).

- **Mensaje de despedida**: Notificará al usuario que la aplicación ha sido correctamente cerrada. Con esto el no vidente sabrá que la cámara ha sido desactivada con el consiguiente ahorro de batería que esto conlleva.

- **Tecla dedicada de inicio rápido:** El prototipo deberá tener una tecla dedicada de inicio rápido de la aplicación. Así, el no vidente no tendrá que desplazarse por ningún menú del celular para ejecutar el *software*.

- **Desempeño:** La tasa de aciertos del sistema deberá estar sobre el 90 %, el porcentaje de billetes no reconocidos bajo el 10 % y los falsos positivos (ver sección 4.2) bajo el 1 %. En cualquier circunstancia, será de mayor importancia reducir los falsos positivos aunque la tasa de acierto se vea comprometida.

- **Velocidad:** La velocidad, medida en FPS (*frames* o cuadros por segundo), deberá superar los 5 FPS. Este valor, escogido empíricamente al observar la velocidad de actualización del *viewfinder*, es el que se impone en este proyecto como mínima velocidad para que el sistema pueda ser considerado de tiempo real.

- **Condiciones de entorno:** El sistema será diseñado para condiciones *indoor*. Esto no significa que el prototipo no funcionará en condiciones *outdoor*, sino que el sistema tendrá menos porcentaje de acierto bajo estas condiciones.´

- **Base de datos:** La base de datos del sistema debe contener los archivos relacionados al entrenamiento *offline* del PCA (*eigenimages*, valores propios,

imagen media y pesos) generados por MATLAB y los mensajes de audio en formato MP3 que fueron determinados en la sección 3.2.

3.3.2.2 Selección del lenguaje de programación

En la plataforma de *hardware* escogida para el prototipo, teléfonos inteligentes con sistema operativo *Symbian*, existen 3 posibles lenguajes de programación:

Java ME: La plataforma J2ME es una familia de especificaciones que definen varias versiones minimizadas de la plataforma *Java 2* que pueden ser usadas para programar teléfonos celulares, PDAs, tarjetas inteligentes, etc. [27].

***Symbian* C++:** Poderoso lenguaje de programación basado en C++ para aplicaciones de propósito general diseñado exclusivamente para dispositivos móviles. *Symbian* C++ permite desarrollar aplicaciones a ser ejecutadas en tiempo real bajo un esquema multitarea en plataformas con recursos de *hardware* limitados [28].

***Python* S60:** Versión exclusiva para *smartphones Symbian* S60 del conocido lenguaje *Python*. Suele ser usado para desarrollar de manera rápida aplicaciones prototipo que eventualmente van a ser portadas a *Symbian* C++ [29].

Para comprender las ventajas y desventajas de los 3 lenguajes, la tabla 3.3 compara sus características.

De dicha tabla, se puede concluir que el lenguaje más óptimo para aplicaciones que requieren procesamiento en tiempo real, como es el caso del procesamiento de imágenes, es *Symbian* C++. Al ser una versión del conocido lenguaje de programación C++, *Symbian* C++ hereda todas sus características en cuanto a velocidad de ejecución, manejo de memoria y eficiencia; características que tanto J2ME y *Python* no reúnen por ser lenguajes interpretados. Además, *Symbian* C++ tiene una funcionalidad sin restricciones, a diferencia de los otros lenguajes, que permite acceder a todas las características del teléfono como la cámara digital, audio, lectura de datos, etc.

Tabla 3.3. Características de los lenguajes soportados por *Symbian* S60

	***Symbian* C++**	**J2Me**	***Python* S60**
Curva de Aprendizaje	Difícil	Promedio	Excelente
Depuradores	Excelentes	Excelentes	Promedio
Emuladores	Sí	Sí	Sí
IDE	*Carbide.c++*	*Eclipse*, *Netbeans*	*Eclipse*
Installer Packaging	sys	jar, jad	sys, *python runtime*
Costo de herramientas de desarrollo	Gratis	Gratis	Gratis
Interfaz gráfica	2D, 3D, *Open* GL	2D, 3D	2D
Funcionalidad	Sin restricciones	Varía de acuerdo a los JSR's de cada dispositivo	Parcial
Acceso a datos del dispositivo.	Completo	Varía de acuerdo a los JSR´s de cada dispositivo.	Parcial
Velocidad de ejecución	La mejor (lenguaje compilado)	Promedio usando java *bitecode*	Baja (lenguaje interpretado)
Soporte oficial y foros	Excelente	Excelente	Creciendo
Distribución y Licencias	Requiere pago anual de licencia de desarrollador[6]	Ninguna	Ninguna

Finalmente, a pesar de que *Symbian* C++ es un lenguaje muy parecido a C++, tiene también características propias que no posee el lenguaje ANSI C++. Para entender dichas características es recomendable la lectura de las siguientes particularidades de *Symbian* C++ explicadas detalladamente en la referencia [30]:

- Convención de nombres de clase (Clases C, T, R, M)
- Constructores de segunda fase
- Descriptores
- Manejo de archivos
- *Callback Functions*
- *Leaves* y Pánicos
- Declaración de variables en *Heap* y *Stack*
- Objetos Activos
- Reproducción de archivos de audio

[6] El IDE *Carbide.c++* (con emulador y depurador incluido) está disponible en descarga gratuita desde www.forum.nokia.com. Sin embargo, para distribuir la aplicación es necesario pagar una licencia anual de desarrollador para poder firmar el programa. Más información puede ser encontrada en www.symbiansigned.com.

3.4 DESCRIPCIÓN DEL PROGRAMA IMPLEMENTADO

En esta sección se describirá la lógica del programa implementado en lenguaje *Symbian* C++. Antes de programar las funciones de procesamiento de imágenes, es necesario inicializar el *hardware* de la cámara del celular, lo cual no es una tarea sencilla si se empieza de cero. Afortunadamente, *Nokia* ofrece, de manera libre, el código fuente de un programa (*Camera Example v2.2*) [7] que realiza las tareas de inicialización e implementa funciones básicas de la cámara. En este proyecto, se tomó dicho código como base del *software* de reconocimiento importándolo dentro de la herramienta de desarrollo de *software Carbide.c++* que es el IDE por defecto para programar teléfonos móviles bajo *Symbian* C++ que incluye características avanzadas como un emulador y la depuración *on device*. Esta última característica es muy útil para desarrollar aplicaciones que no pueden ser emuladas como es el caso de aquellas que usan la cámara. Para más información de cómo importar proyectos en *Carbide.c++,* se recomienda la referencia [31]; en cuanto a la depuración *on device*, se sugiere leer la información de la referencia [32].

Dentro del código de ejemplo que provee *Nokia,* el ámbito en el que se desarrollará todo el código descrito en las siguientes secciones, es el de la clase `CCameraAppBaseContainer`. Es decir que es en esta clase donde se declararán todas las variables, objetos, etc. Dentro de esta clase, la función miembro de donde se llamará a las funciones necesarias para el procesamiento del *frame*, es la función `DrawImageNow`, cuyo prototipo es:

```
void CCameraAppBaseContainer ::DrawImageNow(CFbsBitmap& aBitmap)
```

`DrawImageNow` toma como argumento a una referencia `aBitmap` de tipo `CFbsBitmap` (`CFbsBitmap` es una clase que almacena mapas de bits sin comprimir). El objeto `aBitmap`, es decir el objeto que contiene el *frame,* es automáticamente actualizado mediante la *callback function* (función que es llamada periódicamente por el sistema operativo):

```
CCameraCaptureEngine::ViewFinderFrameReady(CFbsBitmap& aFrame)
```

[7] http://www.forum.nokia.com/info/sw.nokia.com/id/2f492479-ac8c-4c3e-aa90-cc883e190d83/S60_Platform_Camera_Example_with_AutoFocus_Support_v2_2_en.zip.html

Cabe recalcar que en la función miembro anterior el argumento `aFrame` apunta a la misma dirección que `aBitmap`, por lo que si `aFrame` se actualiza periódicamente, `aBitmap` también lo hace con la misma información.

El objetivo de `DrawImageNow` es desplegar en pantalla lo que está almacenado en `aBitmap` haciendo uso de la función exportada `DrawNow` (función compilada a la que no se tiene acceso a su código fuente). Para concluir, toda función que procesará el *frame* actual almacenado en el objeto `aBitmap` deberá ser llamada desde:

```
void CCameraAppBaseContainer ::DrawImageNow(CFbsBitmap& aBitmap)
```

De otra parte, los objetos y variables necesarias tienen que ser declarados en el ámbito de `CCameraAppBaseContainer` e inicializados en su constructor de segunda fase.

3.4.1 Diagrama de flujo del sistema

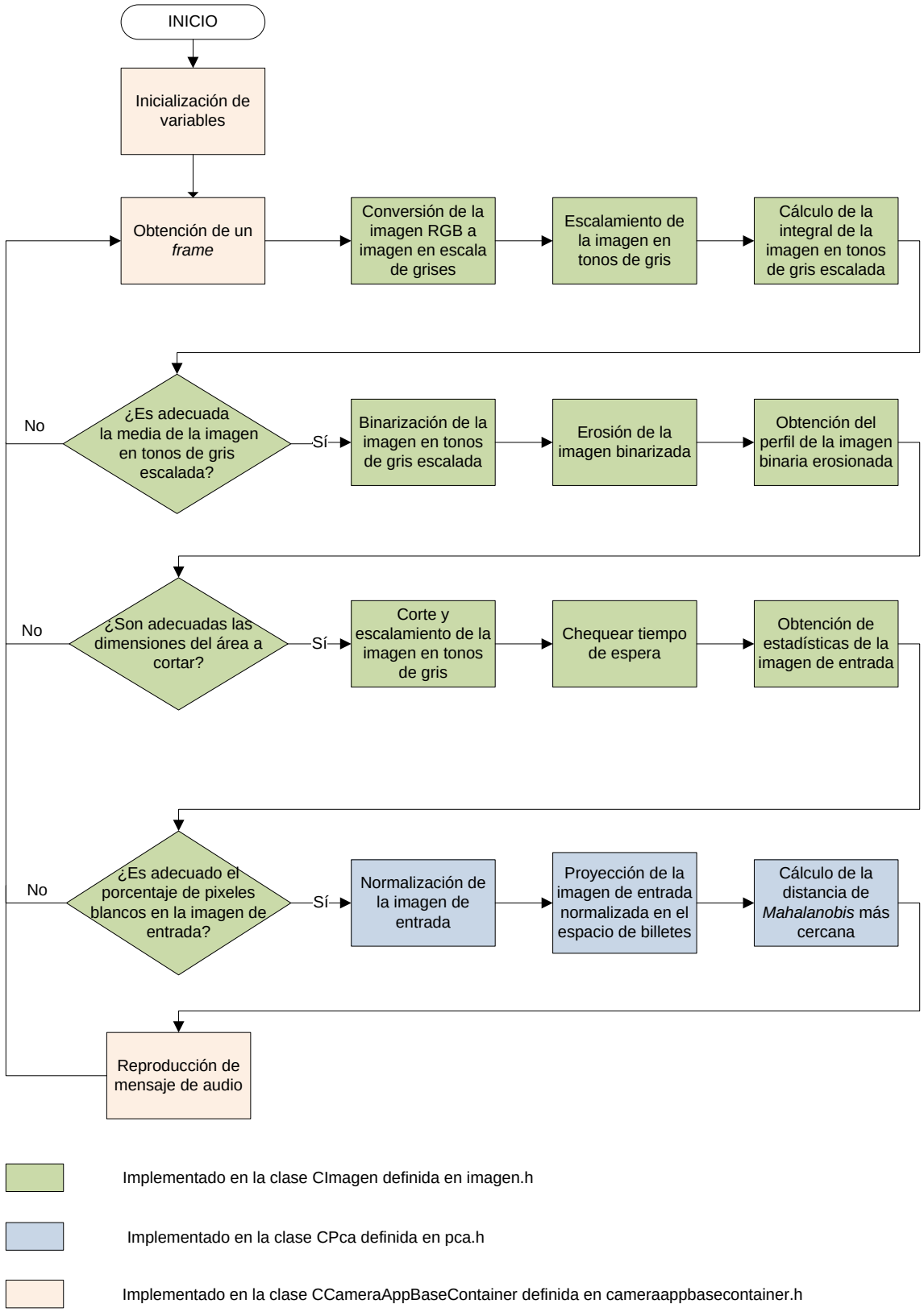


Figura 3.22. Diagrama de flujo del programa general en el teléfono celular

A continuación se hará una descripción de cada bloque. El código de este programa principal se adjunta en el anexo A5.6.

3.4.1.1 Inicialización de variables

La principal tarea de las inicializaciones es reservar memoria para los mapas de bits que se utilizarán. Todos los mapas de bits (ver tabla 3.4) son objetos de la clase `CFbsBitmap` que es una representación nativa usada por *Symbian* C++ para *bitmaps* sin compresión.

Tabla 3.4. Objetos de mapas de bits CFbsBitmap

Objeto *CFbsBitmap*	Inicializado en la clase:	Descripción	Bits por pixel
iGris	CImagen	Almacena la imagen resultante de la conversión de RGB a gris. Se la llamará imagen en tonos de gris o simplemente imagen gris	8
iGrisEsc	CImagen	Almacena la imagen resultante del escalamiento de la imagen gris. Se la llamará imagen gris escalada	8
iBin	CImagen	Almacena la imagen resultante de la binarización de la imagen gris escalada. Se la llamará imagen binaria	8
iEro	CImagen	Almacena la imagen resultante de erosionar la imagen binaria. Se la llamará imagen erosionada	8
iEscalada	CImagen	Almacena la imagen cortada y escalada, a partir de la imagen gris, que contendrá a la zona del billete que será reconocida por PCA. Se la llamará imagen de entrada	8
iIntegral	CImagen	Almacena la imagen resultante del proceso de integración. Se la llamará integral de imagen	32
iMuestra	CPca	Almacena una copia de iEscalada pero en una representación de 32 bits por pixel ya que en la clase CPca todas las operaciones son en 32 bits	32
iMedia	CPca	Almacena la imagen cargada desde *C:\\Data\\Lectbill\\carga1.txt*. Se la llamará imagen media	32
iEigenVectores	CPca	Almacena las *K eigenimages*. Se los carga desde el directorio *C:\\Data\\Lectbill\\eigen*	32
iPesos	CPca	Almacena los pesos del conjunto de entrenamiento. Se los carga desde el directorio *C:\\Data\\Lectbill\\Omega*	32
iEigenValores	CPca	Almacena los valores propios de cada *eigenimage*. Se los carga desde *C:\\Data\\Lectbill\\carga1.txt*	32

Es importante recordar que según la teoría de *Symbian* C++, los objetos de este tipo deben ser inicializados en el constructor de segunda fase de la clase a la cual pertenecen [30]. Los anexos A5.2, A5.3 y A5.4 presentan el código fuente de los constructores de segunda fase de las clases `CCameraAppBaseContainer`, `CImagen` y `CPca` respectivamente. Además de esto, cada objeto tendrá asociado dos variables tipo puntero que apuntan a la primera y última dirección de su respectivo *bitmap*. Estos dos punteros servirán a lo largo del programa para recorrer un *bitmap* pixel por pixel. La tabla 3.4, resume los objetos más importantes que servirán para el funcionamiento del *software* a lo largo de todo el programa

Como se lee de la tabla 3.4, a menudo se hace referencia al directorio *C:\\Data\\Lectbill.* Este directorio contiene los archivos generados por MATLAB durante el entrenamiento y los mensajes de audio que servirán como interfaz con el no vidente. Resumiendo lo descrito en dicha sección, la estructura del directorio *C:\\Data\\Lectbill,* es la siguiente:

Tabla 3.5. Estructura del directorio *Lectbill*

Directorio o Fichero	Variable de la base de datos	Descripción
Directorio audio	--	Contiene los mensajes de audio en formato mp3 que sirven de interfaz con el no vidente
Directorio Eigen	$[u]$	Contiene las *K eigenimages* almacenadas en *K* ficheros .txt.
Directorio Omega	$[\Omega]$	Contiene los pesos de las *M* muestras del conjunto de entrenamiento almacenados en *M* ficheros .txt
Fichero carga1.txt	Ψ, Λ	Almacena la imagen media y los valores propios de las *K eigenimages*.

En esta sección se presentó una breve descripción de los principales objetos que son inicializados desde el constructor de segunda fase de la clase `CCameraAppBaseContainer`:

```
void CCameraAppBaseContainer::ConstructL(const TRect& aRect)
```

Su código fuente es el presentado en el anexo A5.2. Inicializaciones de otros objetos mencionados en este inciso se detallan en los códigos adjuntos en los anexos A5.2 a A5.4. Las cargas de la información de los archivos se adjuntan en el anexo A5.5. Para más información sobre las demás variables inicializadas, se recomienda referirse a los códigos aquí referenciados.

3.4.1.2 Obtención del *frame* RGB

El objetivo de esta etapa es recuperar la información del *frame* y guardarla correctamente en un objeto de tipo *CFbsBitmap*. Con este objetivo, *Symbian* realiza una serie de comprobaciones para constatar si el *frame* está listo para ser desplegado en pantalla, a través de funciones exportadas (funciones en las cuales no se puede acceder a su código) cuyo funcionamiento escapa del enfoque de este proyecto. Como se explicó en la introducción de la presente sección (3.4), todas las funciones que procesarán el *frame* serán llamadas desde el ámbito de la función:

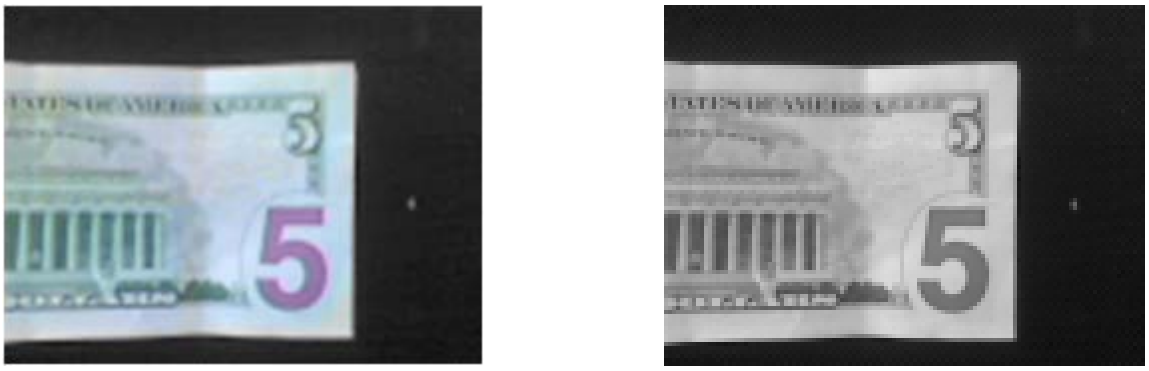
```
void CCameraAppBaseContainer ::DrawImageNow(CFbsBitmap& aBitmap)
```

El *frame* que se manipulará está almacenado en la variable `aBitmap.` Es importante aclarar que el *frame es* actualizado automáticamente una vez que se sale del ámbito de `DrawImageNow.`

3.4.1.3 Conversión de la imagen RGB a escala de grises

Al obtener cada nuevo *frame* como se describe al inicio de la presente sección, el primer paso es convertirlo a escala de grises. Para esto, se usa la ecuación 2.3 de la teoría del procesamiento de imágenes: $gris = \frac{R+G}{2}$. Las razones de usar esta última ecuación y no otras como las expuestas en las ecuaciones 2.1 y 2.2, se detallan en la sección 2.2.2. Es importante recalcar que *Symbian*, al ser una plataforma que funciona sobre procesadores ARM de 32 bits, almacena el *frame* RGB en palabras de 4 bytes (32 bits) de acuerdo al siguiente formato: 0x00RRGGBB, donde cada canal es representado por 1 byte. La función encargada del proceso de conversión es: `void CImagen::RgbAGris(CFbsBitmap &aImagenRGB),` cuyo código se halla en el anexo A5.7. El argumento `aImagenRGB` de dicha función es el *frame*

almacenado en el objeto `aBitmap` descrito anteriormente. La figura 3.23 muestra el resultado de la conversión descrita.



a) **b)**

Figura 3.23. Conversión de imagen RGB a gris: a) Imagen RGB b) Imagen gris

3.4.1.4 Escalamiento de la imagen en tonos de gris

Una vez que se ha obtenido la imagen del *frame* en escala de grises, se requiere de un procesamiento de bajo nivel de la imagen. Este proceso inicia por la reducción de la resolución espacial de la imagen gris a la mitad de su tamaño en filas y en columnas (lo que produciría una reducción final de un cuarto del área total original). Esta operación se justifica al notar que el propósito final de las operaciones de PDI en esta aplicación es la de separar, si es que existe, a la imagen del billete dentro de la imagen gris de entorno, para pasarla como entrada del PCA y que las operaciones previas no tienen por qué ocupar más capacidad de procesamiento que lo necesario. Es así que la imagen gris de entorno obtenida en el anterior inciso se reduce a un cuarto mediante el algoritmo de este proceso que utiliza la teoría de la sección 2.2.3, de escalamiento de imágenes. Esto se realiza mediante la función `void CImagen::EscalaGris()`, cuyo código se halla en el anexo A5.8.

Básicamente, lo que se hace en este procedimiento es utilizar los criterios de la sección 2.2.3 para la producción de una variable que se conocerá como *paso*, definida como la inversa del factor de escalamiento establecido en las ecuaciones 2.6 y 2.7. En este caso, en específico, dado que se desea la reducción a la mitad de la imagen gris, tanto en filas como en columnas, el factor de escalamiento para ambas es de $\frac{1}{2}$ y, por tanto, la variable paso es de 2. De la revisión de la teoría de escalamiento, se entiende que tener un

paso de 2 significa que la asignación de las intensidades de los píxeles de la malla original a la final se hará en pasos de dos columnas y dos filas.

De esta forma, el algoritmo se implementa en dos bucles que recorren la imagen gris original en pasos de dos columnas y luego en pasos de dos filas para asignar las intensidades hallada en los píxeles de esos índices a los de una nueva imagen, *la imagen gris escalada*, en la memoria de 8 bits del objeto de tipo *bitmap* `iGrisEsc`, como se aprecia en el código. El resultado de este proceso se puede observar en la figura 3.24, en la cual se muestra en una sola imagen a la gris original de resolución 266×200 y, sobrepuesta en la misma, la gris escalada mediante este código, de resolución 133×100.

Figura 3.24. Resultado del escalamiento de la imagen gris

3.4.1.5 Integración de la imagen gris escalada

La integral, que posteriormente será usada para la binarización, es aplicada a la imagen en tonos de gris escalada obtenida en la sección anterior. Se recomienda leer la sección 2.3.1 para comprender cómo la integral de la imagen reduce la complejidad computacional del proceso de umbralización. La función encargada de esta actividad es `void CImagen::Integra()` cuyo código se adjunta en el anexo es el A5.9.

3.4.1.6 Primera discriminación: Media de la imagen gris escalada

En este punto del programa, se introduce al lector la idea de que uno de los objetivos es el de aprovechar al máximo la capacidad de procesamiento disponible en el sistema y que los procedimientos utilizados para el PCA (posteriormente detallados), son de altos requerimientos en procesamiento. Es por esto que, según lo planteado en el diagrama de flujo general del programa, se hacen un total de tres procedimientos de decisión que

permitirán o no a la lógica del programa ingresar a las funciones implementadas para el procesamiento de alto nivel del PCA, a las que se llamarán *discriminaciones*.

La primera discriminación analiza a la imagen gris escalada, que representa todo el entorno que el *frame* de la cámara digital ha tomado, dentro del cual puede o no haber un objeto de interés; en este caso, un billete; por esto, se le llamará a partir de aquí, *imagen de entorno*. Es lógico pensar que si la imagen que la cámara está filmando es demasiado iluminada (*e.g.* si no se está filmando en ese instante al billete, sino una fuente de luz, un fondo muy claro, etc.) no debe ser procesada en el PCA, ya que hay poca probabilidad de reconocer la denominación de un billete. Además, como se ha planteado en la sección 2.4.4.3, el método de *eigenimages* presenta complicaciones con condiciones de iluminación, que se ven acentuadas si el entorno filmado es demasiado claro, por lo que no es deseable el procesamiento con el PCA de una escena con tales características, dado que ocasiona más probabilidad de que el método produzca resultados insatisfactorios.

Por estas razones, se halla un criterio para la determinación de una claridad demasiado alta en la imagen del entorno, en la media de esta imagen. La discriminación, entonces, consiste en devolver un resultado que indique si es prudente el paso al resto de procesos de la imagen del entorno, positivo si es que la media de la imagen es menor a un valor determinado en la constante `KMediaGrisAlta`, y uno negativo de otra forma. Se implementa la primera discriminación en la función *inline* del anexo A5.1.2 cuyo prototipo es: `TUint CImagen::ChkEst1 ().`

En esta función, se calcula la media de la imagen gris escalada con la ayuda de la integral de la imagen, hallada en el anterior inciso, con los criterios planteados en la sección 2.3.1.3 y la ecuación 2.19.

El valor `KMediaGrisAlta`, que es el criterio de la media máxima que debe tener una imagen para ser considerada adecuada para este proyecto, es un valor constante de *128*. Se llegó a este valor experimentalmente, de un proceso de toma de muestras de varias imágenes grises escaladas obtenidas con el teléfono celular que contenían en su entorno a un billete y, además, en las cuales el sistema funcionaba adecuadamente. Se calcularon las medias de todas estas imágenes y se llegó a la conclusión de que las imágenes grises que cumplían los requisitos, en promedio, tenían medias cercanas al entorno de 100; en cambio, imágenes con mucha iluminación en las cuales el sistema fallaba, tenían medias

superiores a 130. Cabe anotar que no se realizó una discriminación igual, pero restringiendo imágenes muy oscuras, debido a que el sistema llegó a funcionar en imágenes de entorno con medias de hasta 20, y la media observada en una oscuridad absoluta no se calculaba como 0 sino como 16, como mínimo; por lo que no había necesidad de tal discriminación de una media mínima. En la figura 3.25 se aprecia una imagen de entorno gris escalada que cumple la condición establecida en este criterio, así como una imagen que no lo hace.

(a) **(b)**

Figura 3.25. Imagen de entorno que: (a) Cumple la condición de la primera discriminación (b) No cumple la condición de la primera discriminación

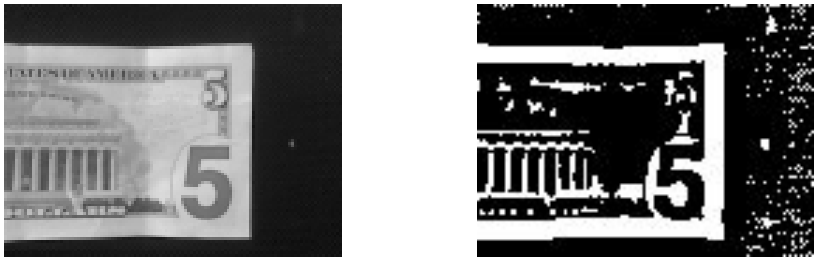
3.4.1.7 Binarización de la imagen gris escalada

El método de umbralización seleccionada es la *binarización adaptativa usando medias locales* debido, entre otras razones, a su robustez frente a cambios de iluminación (ver sección 2.3.1). Dichas medias locales son calculadas eficientemente usando la integral de la imagen como se describe en la sección 2.3.1.3, lo cual aumenta considerablemente la velocidad de procesamiento. La implementación de este proceso se lo realizó con los siguientes parámetros, los cuales fueron determinados de forma empírica:

- Tamaño de la ventana (variable `s` del anexo A5.10): Un octavo del ancho de la imagen gris escalada.
- Constante de ponderación de media local: Se utilizó un valor de 1.1 de acuerdo a la ecuación 2.12. Sin embargo, en la implementación del anexo A5.10, para evitar usar decimales en esta constante, se usa un artificio matemático:

$$\text{Constante de ponderación de media local} = \frac{100 - T}{100}\ ,\ para\ T = -10 \quad (3.1)$$

Con la ecuación 3.1, para *T=–10*, se obtiene una constante de ponderación de media local de valor 1.1, que es precisamente la que se requiere. Lo que se buscó con esta operación es que la imagen binarizada mantenga bien definido el borde blanco que rodea al billete ya que de esta manera se puede determinar dónde está la zona de interés como se describirá más adelante. Los parámetros anteriores son los que mejor cumplían este propósito. La figura 3.26 muestra el resultado de binarizar la imagen gris escalada obtenida en el anterior inciso mediante este método. Finalmente, la función que implementa este proceso es `void CImagen::UmbralAdap(),` cuyo código fuente se presenta en el anexo A5.10.



a) **b)**

Figura 3.26. Resultado del proceso de binarización adaptativa: a) Imagen Gris Escalada. b) Imagen Binarizada

3.4.1.8 Erosión de la imagen binaria

La única desventaja que presenta el anterior método de umbralización, en esta aplicación específica, es que, bajo ciertos fondos de filmación no ideales, presenta una gran cantidad de ruido sobre el fondo oscuro de la imagen binarizada. Dado que el siguiente paso que se desea dar es el de la separación del billete de la imagen de entorno, mediante el reconocimiento de su perfil, el ruido complica la tarea del reconocimiento de los bordes externos del billete sobre el fondo. Una de las prestaciones de la erosión en el PDI es justamente la de encargarse de este tipo de ruido, como se establece en la sección 2.3.2. Es así que aquí se implementa la erosión de la imagen binaria, obteniéndose una nueva versión de esta imagen, en la memoria de 8 bits del objeto tipo *bitmap* `iEro`. Como se establece en la teoría de la sección 2.3.2, se selecciona el elemento estructural más común para tareas de filtrado, allí definido y, siguiendo el criterio dado en la sección 2.3.2.3, se implementa la erosión mediante el uso de los 8 vecinos cercanos en cada píxel de la imagen binaria que se va recorriendo. Para esto, se sigue el algoritmo detallado en el diagrama de flujo de la figura 3.27.

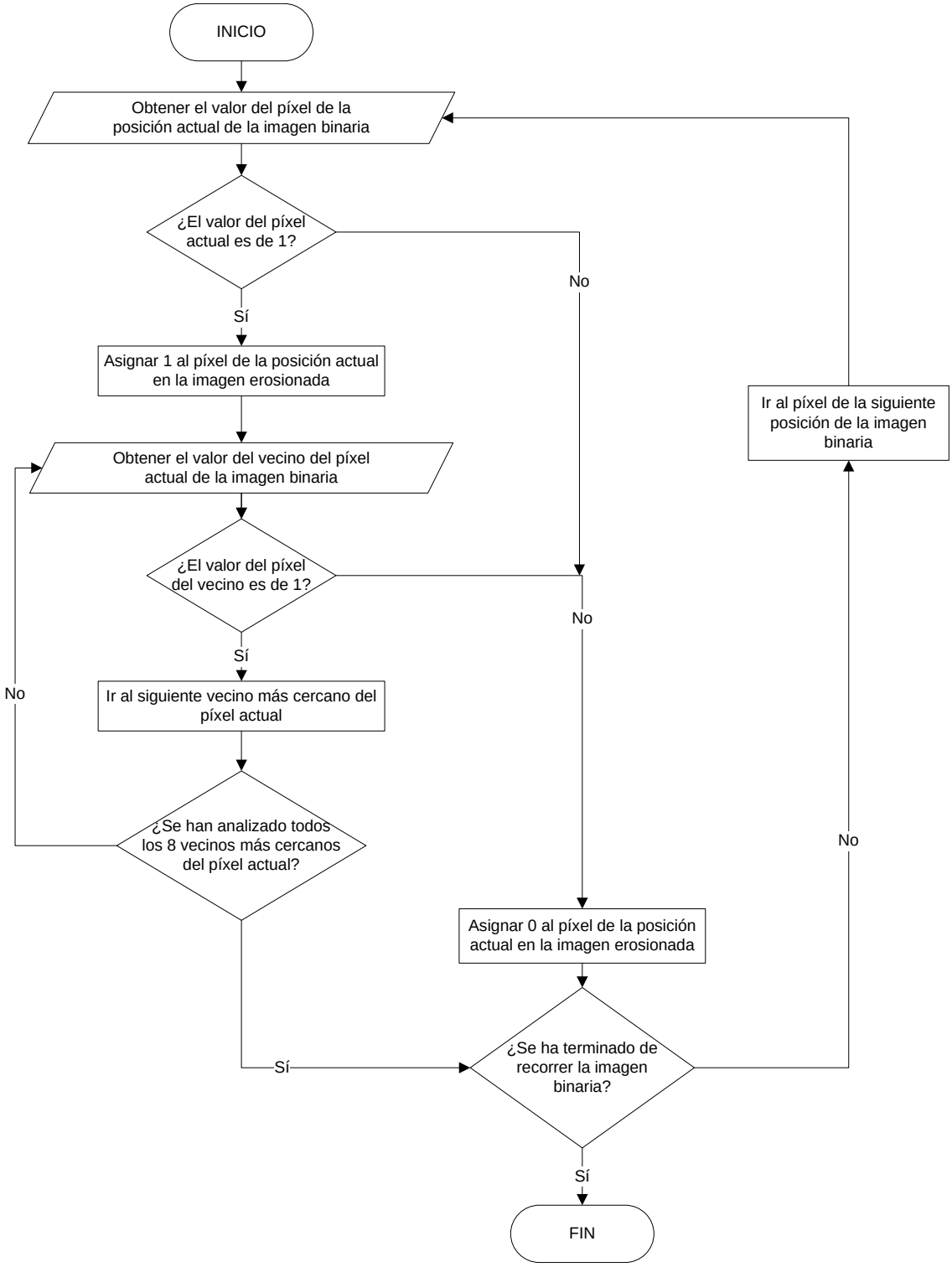


Figura 3.27. Diagrama de flujo para la erosión de la imagen binaria

Este algoritmo implementa el método de la sección 2.3.2.3 de la manera más eficiente posible. Su código en *Symbian* C++, en la función `void CImagen::Erosiona()`, se halla en el anexo A5.11. Los resultados de esta

operación se pueden notar en la figura 3.28, donde se muestra una imagen binaria y el resultado de su erosión.

(a) (b)

Figura 3.28. Resultado de la erosión de la imagen binaria del billete: (a) Imagen binaria (b) Imagen erosionada

3.4.1.9 Obtención del perfil de proyección de la imagen binaria erosionada

El objetivo de calcular el perfil de proyección (ver sección 2.3.3) es obtener la localización exacta del billete, es decir, los índices de las 2 filas (superior e inferior) y la columna de sus bordes internos. La figura 3.29 explica la definición de bordes internos y externos que utiliza el presente proyecto.

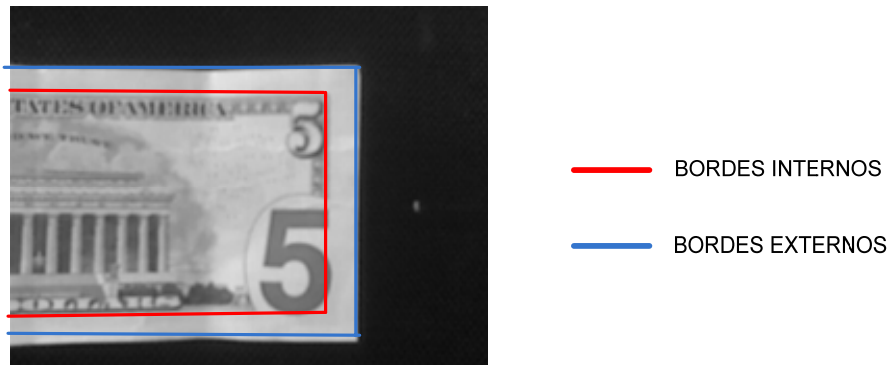


Figura 3.29. Bordes internos y externos de un billete

La localización del billete se puede dividir en 2 etapas:

- **Localización de bordes externos a partir del perfil de la imagen erosionada**. Para esto, se obtiene el perfil de proyección vertical y horizontal de la imagen erosionada teniendo en cuenta que los pixeles de interés son de color blanco y representan el borde del billete. Los dos índices horizontales de los *bordes externos* (superior e inferior) son aquellas filas donde su perfil de proyección vertical supera cierto umbral; en este caso el umbral escogido empíricamente es

25 % del ancho de la imagen. La figura 3.30 muestra un ejemplo de perfil vertical. Como se observa, pueden existir algunas filas que superen dicho umbral pero las únicas que interesan son la mínima y máxima fila que representarán los índices horizontales del borde externo inferior y externo superior respectivamente. Para evitar ambigüedades, cabe recalcar que el origen del sistema de coordenadas está ubicado en la esquina superior izquierda de la imagen. Por otra parte, un análisis similar se realiza para el único índice vertical del borde externo definiéndola como aquella columna donde se supere cierto umbral. En este caso el umbral escogido empíricamente es *50 % del alto de la imagen* (ver figura 3.30).

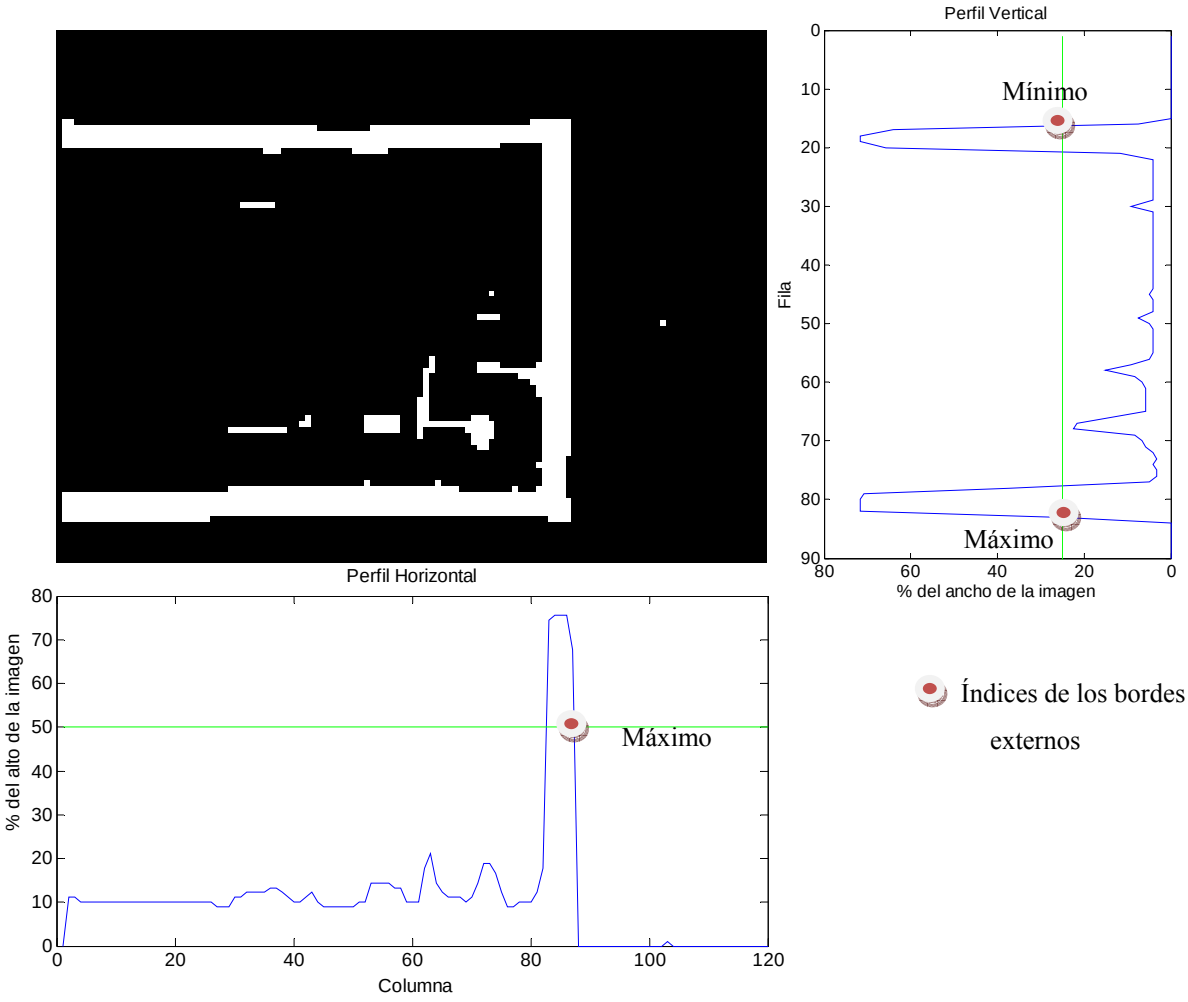


Figura 3.30. Determinación de los índices del borde externo del perfil de la imagen erosionada

De manera análoga a los índices horizontales, se observa que puede existir más de una columna que supere dicho umbral, por lo que para evitar ambigüedades, se debe escoger como índice vertical interno sólo a la máxima columna cumpla el criterio. En este punto, cabe aclarar que con el perfil obtenido de la imagen

erosionada, no se puede saber con precisión los índices de los bordes internos del billete ya que por lo general no se tendrá una imagen binaria tan bien definida como la de la figura 3.28(a). Esto se debe a que la erosión, además de eliminar el ruido, también reduce el ancho del borde blanco del billete (ver la sección 2.3.2.2). Por esta razón, los bordes externos sólo definen un área dentro de la cual será más fácil encontrar los bordes internos que son los que importan.

- **Localización de bordes internos a partir del perfil de la imagen binaria limitada por los bordes externos.** Una vez obtenidos los bordes externos, el objetivo es eliminar el borde blanco que caracteriza a un billete. En otras palabras, el objetivo es encontrar los bordes internos. Para esto, se debe considerar la imagen binaria limitada por los bordes externos como se muestra en la figura 3.31; de esta última imagen, es decir, de la imagen encerrada por los bordes externos, se obtiene el perfil de proyección.

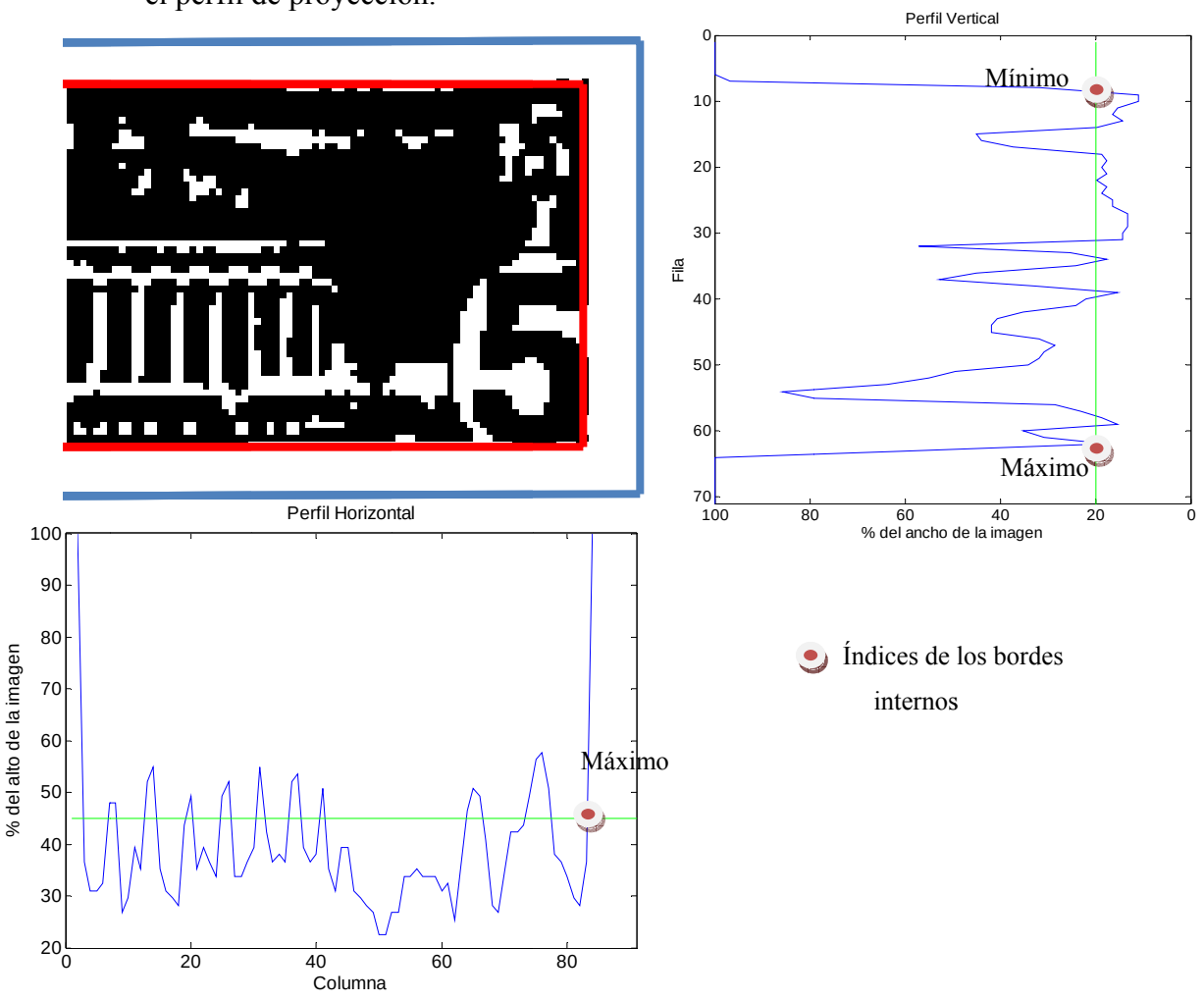


Figura 3.31. Determinación de los índices del borde interno del perfil de la imagen binaria

Los dos índices horizontales de los bordes internos (superior e inferior) son aquellas filas donde su perfil de proyección vertical baja de cierto umbral; en este caso el umbral escogido empíricamente es *20 % del ancho de la imagen limitada por los bordes externos*.

Como se observa en la figura 3.31, pueden existir algunas filas que sean menores a dicho umbral pero las únicas que interesan son la mínima y máxima fila que representarán los índices horizontales del borde interno inferior e interno superior respectivamente. Por otra parte, un análisis similar se realiza para el único índice vertical del borde interno definiéndolo como aquella columna donde su perfil baja de cierto umbral. En este caso el umbral escogido empíricamente es *45 % del alto de la imagen limitada por los bordes externos* (ver figura 3.31). De manera análoga a los índices horizontales, se observa que puede existir más de una columna que sea menor a dicho umbral, por lo que para evitar ambigüedades, se debe escoger como índice vertical interno sólo a la máxima columna que cumpla el criterio.

Para finalizar, se pone a disposición del lector el código que implementa el perfil de proyección y la búsqueda de los bordes internos y externos en el anexo A5.12. La función de *Symbian* C++ es `void CImagen::Perfil()`. Las variables miembro de la clase `CImagen` que almacenan los índices horizontales superior e inferior y el índice vertical son: `iFila_ver1`, `iFila_ver2` e `iColumna` respectivamente. Cabe anotar que, dado que el procesamiento que se ha dado hasta aquí ha sido aplicado a una versión escalada en la mitad de filas y columnas de la imagen gris que contenía originalmente al billete, se realiza al final del código expuesto una multiplicación por 2 de las últimas variables mencionadas, para restituir los índices del perfil interno del lugar de interés del billete en la imagen original, para proceder al corte de esta porción, como se verá en los siguientes incisos.

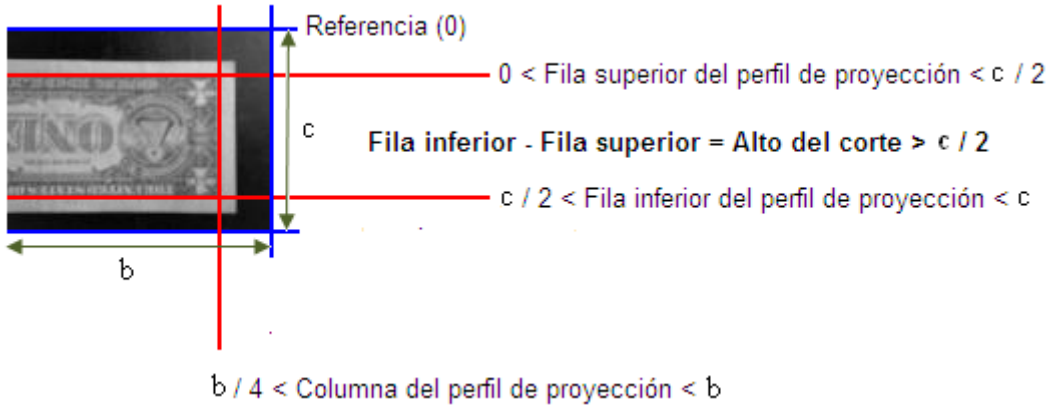
3.4.1.10 Segunda discriminación: Dimensiones adecuadas del área a cortarse

En este punto, cabe una segunda discriminación de la imagen que se está obteniendo, dado que se está a las puertas de la obtención de la imagen de entrada para el método de *eigenimages*, que sigue la teoría del PCA. Por la naturaleza del algoritmo implementado en el anterior paso, básicamente la de un conteo de píxeles, es posible que,

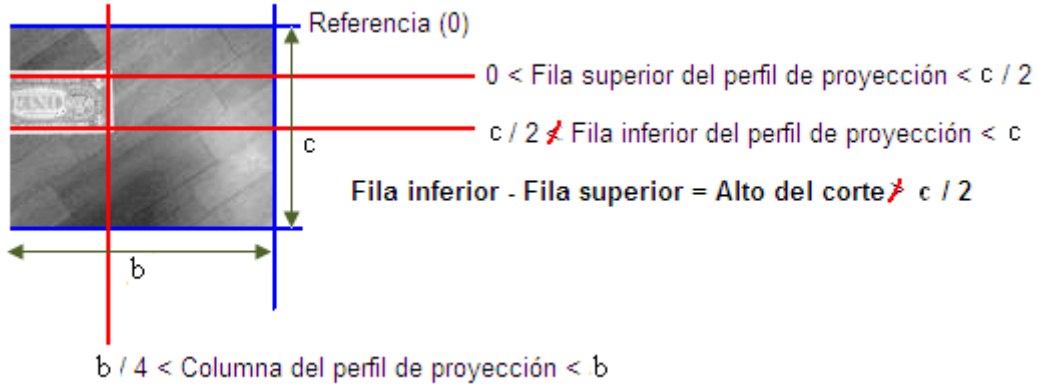
en un momento dado de la filmación, el método de umbralizado de la imagen haya producido una imagen binaria en la que no se reconozcan a los bordes por falta de píxeles necesarios de conteo, en cualquiera de sus fronteras. Es así que los índices antes obtenidos, que indican implícitamente la ubicación y las dimensiones de la imagen que se pretende cortar de la gris de entorno (y que se presume, es el billete), ayudan a determinar si la imagen que va a cortarse tiene dimensiones que sigan la lógica de contener un billete en su interior. De esta forma, el criterio para la discriminación se da por las dimensiones de la imagen que se va a cortar en el siguiente inciso, calculadas a partir de los índices de los vértices del perfil de proyección. Como se puede inferir, la altura del corte será la diferencia entre los índices de las filas superior e inferior y este deberá ser el suficiente como para contener el alto de un billete. Por su parte, la columna derecha del perfil debe ser de un valor tal que pueda proveer el espacio suficiente para que un porcentaje adecuado del billete entre por el lado izquierdo al *frame* en filmación. Estos criterios se expresan más específicamente en las siguientes condiciones escogidas experimentalmente:

- La columna del perfil debe ser mayor que un cuarto del ancho de la imagen de entorno.
- La columna del perfil debe ser menor que todo el ancho de la imagen de entorno.
- La fila inferior del perfil debe ser menor que todo el alto de la imagen de entorno.
- La fila inferior del perfil debe ser mayor que la mitad del alto de la imagen de entorno.
- La fila superior del perfil debe ser mayor que cero.
- La fila superior del perfil debe ser menor que la mitad del alto de la imagen de entorno.
- El alto de la imagen que se pretende cortar debe ser al menos el 50 % del alto de la imagen de entorno.

Para la verificación de la lógica de estos criterios, se muestran en la figura 3.32, dos imágenes de entorno de las cuales se han obtenido las filas y la columna de sus respectivos perfiles de proyección.



(a)



(b)

Figura 3.32. Imagen de entorno que: (a) Cumple las condiciones de la segunda discriminación (b) No cumple las condiciones de la segunda discriminación

La primera imagen de entorno contiene adecuadamente a un billete y, por tanto, cumple todas las anteriores condiciones; se puede inferir que una imagen cuyo perfil no contenga un objeto con la geometría parecida a la de un billete no cumplirá las condiciones. A pesar de que la segunda imagen contiene a un billete, este está muy alejado en la filmación, por lo que el corte del billete que se obtenga no producirá una imagen adecuada de entrada para el PCA; justamente, no cumple la última condición de la lista y deberá discriminarse del PCA. El procedimiento que implementa esta segunda discriminación, en código de *Symbian* C++, es la función *inline*, `TUint CImagen::ChkEst2 ()` cuyo código se presenta en el anexo A5.1.2.

3.4.1.11 Corte y escalamiento de la imagen gris

Una vez cumplida la condición anterior, se sabe que es más probable aún que la imagen enmarcada entre el perfil de proyección sea la de un billete. Por esto, se procede ya a la obtención de la muestra que servirá de entrada al método del PCA. Como se dijo en la anterior sección, la imagen que participará en el método, debe seguir un proceso similar al que siguen las imágenes de muestra del conjunto de datos del que se obtienen las *eigenimages*. Uno de estos requerimientos es que la imagen debe estar escalada a la misma resolución de las imágenes del conjunto (como se establece en la sección 2.4.4.1). Como se recordará, esta se seleccionó de 90 filas × 60 columnas. Como la imagen que se puede obtener con el perfil de proyección a partir de la imagen de entorno es de dimensiones variables, se presenta aquí la necesidad de usar una vez más el escalamiento de imágenes, cuya teoría se detalla en la sección 2.2.3.

El procedimiento presentado en este inciso se ocupa de realizar tanto el escalamiento mencionado como el corte de la imagen que será la entrada del PCA, en una misma función de *Symbian* C++, `void CImagen::Cortar()`. Esta función inicia estableciendo los valores para los factores de escalamiento para filas y columnas de la ecuación 2.6 y 2.7 respectivamente. Como éstas requieren de las dimensiones de las mallas original y final (de origen y de destino), se deben calcular las dimensiones de la imagen de origen a partir del perfil de proyección, mediante los siguientes criterios:

- Como se ha dicho antes, el alto de la imagen de origen será la diferencia entre las filas superior e inferior del perfil de proyección, y se guardará en una variable, *h*.
- El ancho de la imagen de origen debe ser obtenido con el criterio de que las dimensiones de esta imagen deberán *guardar la misma proporción de alto a ancho de la imagen de destino*. Se guardará en una variable, *a*.

El anterior criterio facilitará la operación de escalamiento de reducción que se pretende lograr. Según él, el ancho a cortarse de la imagen de origen se determina en base a la relación:

$$a = \frac{h}{p} \tag{3.2}$$

Donde p es la proporción de alto a ancho de la imagen de destino; en este caso, $p = \frac{90}{60}$. Al hacer esto, se puede ya determinar numéricamente el perímetro completo del corte que se llevará a cabo de la imagen de entorno, e identificar los vértices con las variables del programa; un ejemplo de este se aprecia en la figura 3.33, donde h es de 111 píxeles y, por tanto, $a = \frac{111}{p} = \frac{111}{\frac{90}{60}} = 74\ pixeles$.

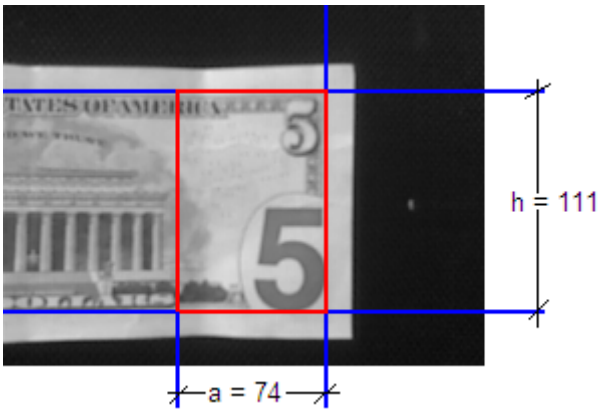


Figura 3.33. Porción determinada a cortarse de la imagen de entorno

Una vez determinada esta porción, la función procede a cortarla al mismo tiempo que la escala; esto lo hace usando el criterio de que el escalamiento no es más que una reasignación de los píxeles de la imagen de origen a la de destino de acuerdo al escalamiento requerido y a la interpolación implementada, como se enuncia en la sección 2.2.3. Para esto, se inicia con el establecimiento de los factores de escalamiento por filas y columnas. Se hace notar aquí que, a este punto, la proporción entre anchos y altos de las imágenes original y final, se mantiene, por el criterio tomado anteriormente:

$$\frac{h}{a} = p = \frac{h_f}{a_f} \tag{3.3}$$

Donde h_f y a_f son, respectivamente la altura y ancho de la imagen final, de destino. Como se puede verificar, de la relación 2.6 y 2.7, se cumple, entonces, la proporción:

$$S_x = \frac{h_f}{h} = \frac{a_f}{a} = S_y = S \tag{3.4}$$

Lo que quiere decir que los factores de escalamiento de filas y columnas son, en este caso, los mismos e iguales a *S*. Además, recordando la noción de *paso*, expuesta en el inciso de esta sección: *Escalamiento de la imagen gris*, se concluye que, en este caso, se debe ir recorriendo la porción antes seleccionada de la imagen original en pasos de $\frac{1}{S}$, tanto para filas como para columnas, para hacer posible el escalamiento, y asignando la intensidad de los píxeles que se vayan hallando en una nueva imagen, para hacer posible el corte. Como se ve en el código de la función, adjunto en el anexo A5.13, estas dos acciones se van realizando en un solo juego de bucles, logrando eficiencia en el algoritmo. En la figura 3.34 se puede apreciar el resultado del procedimiento, la obtención de la imagen de un billete, escalada a una resolución de ancho por alto de 60×90. Como se ve, la imagen final tiene la misma estética que todas las imágenes del conjunto original de muestras del programa de entrenamiento de la sección 3.2.2, mostradas en la figura 3.7.

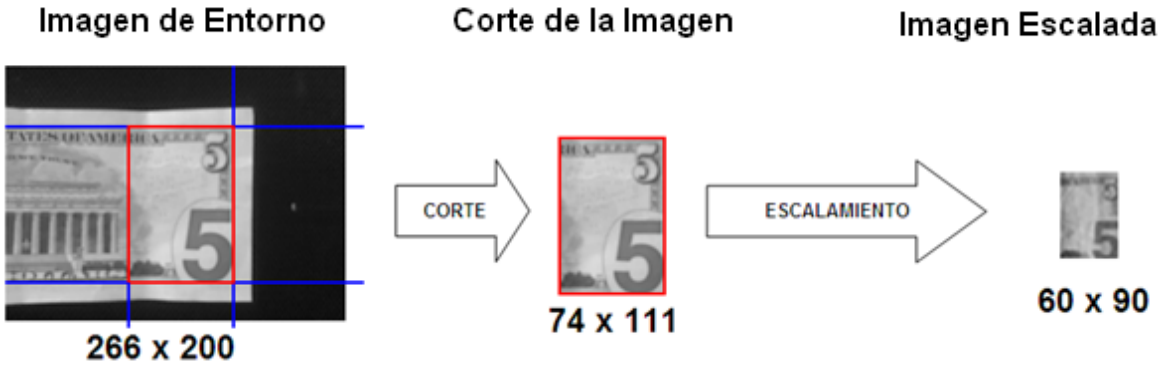


Figura 3.34. Resultado del corte y escalamiento de la porción de la imagen seleccionada

Lo que se ha logrado mediante este proceso, es una imagen que tiene las características necesarias para servir como entrada del método del PCA en el algoritmo de las *eigenimages*.

3.4.1.12 Obtención de las estadísticas de la imagen de entrada al PCA

La función `void CImagen::Estadist()`, cuyo código se encuentra en el anexo A5.14, es responsable de obtener la media y desviación estándar de la imagen de entrada, es decir, de la imagen que será procesada por PCA. La media y desviación estándar calculadas mediante las expresiones 2.19 y 2.22 respectivamente, como se verá en

las siguientes secciones, servirán para la normalización de la imagen así como para la tercera etapa de discriminación.

3.4.1.13 Tercera discriminación: Adecuado porcentaje de píxeles blancos en la imagen de entrada binaria

Llegado el programa hasta aquí, se establece la tercera y última discriminación de la imagen escalada que se ha obtenido, una oportunidad final de exclusión de la imagen del método del PCA en el caso de que tenga una probabilidad reducida de ser un billete. Para esto, se debe considerar que si es que la lógica del programa ha llegado a este punto, quiere decir que la información específica de discriminación que se tiene es que: (a) La imagen gris de entorno tiene una iluminación adecuada para contener un billete; y (b) Las dimensiones de la imagen que se cortó eran las apropiadas para contener un billete. No se ha considerado hasta aquí, la información que específicamente existe dentro de la porción antes cortada, que debería ser la de un billete. Para encontrar un criterio que permita determinar si es que la probabilidad de que esta información represente a un billete, se intentaron muchos métodos distintos, como el mismo recomendado por la referencia [17], así como la extracción de la media, sin llegar a poder plantearse una condición que permita, con precisión establecer que lo que hay adentro del corte es un billete, si no es ya el método de discriminación de *eigenimages* que justamente se quiere evitar en esta discriminación en el caso de no contener el corte a un billete. Los métodos que mejores resultados dieron, fueron los que se concluyeron de las siguientes observaciones y análisis:

- Al realizarse, experimentalmente, la toma de la desviación estándar de muchas imágenes escaladas de entrada en el método de reconocimiento del PCA, con buenos resultados, se estableció que las imágenes que representaban billetes tenían desviaciones estándar que se hallaban, de forma aproximada, en un rango de 10 a 50.
- El método de las *eigenimages*, por la teoría del PCA que lo fundamenta, presenta problemas al recibir como entrada la imagen de un fondo que se halla vacío.

De estas observaciones, se pudieron obtener los criterios de discriminación. La imagen escalada obtenida en los procesos anteriores, solo pasará como entrada al algoritmo del PCA, si:

- Tiene una desviación estándar que se halle en el rango de 10 a 50.
- El porcentaje de píxeles blancos que se hallan dentro del rectángulo de la imagen escalada sin bordes, en su versión umbralizada, es mayor a un porcentaje dado del área total de este rectángulo.

La primera condición se entiende de forma inmediata. La segunda se puede comprender a partir de las observaciones antes citadas. Si la imagen que se está filmando no contiene a un billete, sino algún otro objeto con la misma forma, como un pedazo de papel vacío, el corte al interior de los bordes internos que se obtenga, al ser umbralizado, se observará como el fondo; es decir, con casi una totalidad de píxeles negros rellenando su espacio. En cambio, si es un billete u otro objeto que no se presente vacío, la umbralización mostrará las impresiones internas dentro de ese espacio como píxeles blancos. Esto se puede ver claramente en la figura 3.35, en la que se muestran ejemplos de los umbralizados de estos dos tipos de imágenes.

(a) **(b)**

Figura 3.35. Umbralizado del corte de: (a) Un billete, que cumple con las condiciones de la tercera discriminación (b) Un papel vacío, que no cumple con las condiciones de la tercera discriminación

El procedimiento que implementa esta tercera discriminación, en código de *Symbian* C++, es la función `TUint CImagen::ChkEst3 ()`, definida en el anexo A5.15, en donde se puede ver que el porcentaje mínimo de píxeles blancos seleccionado en la implementación del proyecto, empíricamente se establece en la constante `KPorcen`, como *10*; lo que quiere decir que para que el corte de una imagen determinado, finalmente pase como entrada al método del PCA, además de tener su desviación estándar en un rango apropiado, debe tener, en su versión umbralizada, al menos un 10% del área total al interior de los bordes internos del corte de píxeles blancos. Si aún así una imagen que no

contiene un billete logra pasar como entrada al PCA, pasa a *eigenimages* el dominio de la discriminación de la imagen.

3.4.1.14 Chequeo del tiempo de espera por el reconocimiento

Este procedimiento, implementado en código de *Symbian* C++, en la función `void CCameraAppBaseContainer::ChkTiempo()`, adjunta en el anexo A5.16, se encarga de chequear si la espera del usuario por el reconocimiento de la imagen obtenida antes es demasiado larga. Se ejecuta inmediatamente después de haberse determinado que la imagen escalada a entrar en el método del PCA es apropiada. Utiliza las variables `iContador` e `iContadorMay`.

La variable `iContador` cuenta las veces que se tiene una imagen apropiada y aún no se ha reconocido como alguna denominación de billete; si llega a ser mayor a un número dado en la constante `KCuenta` (en este caso, de manera empírica *igualada a 20*), la función emitirá un mensaje de audio indicando al usuario que pruebe el sistema con otro lado del billete, reproduciendo el archivo *otra.mp3*, haciendo uso de la clase `CSoundPlayer` de *Symbian* C++, cuya teoría se puede revisar en [33].

La variable `iContadorMay` cuenta el número de veces que ha sido necesaria la emisión del anterior mensaje de audio; si llega a ser mayor a un número dado en la constante `KCueMay` (en este caso, de manera empírica igualada a 4), se emite otro mensaje, indicando al usuario que reinicie la aplicación, reproduciendo el archivo *reinicie.mp3*. El objetivo de esto es que si se ha llegado a este punto, en que bajo ninguna condición se reconoce a la imagen como un billete de alguna denominación, puede significar que la aplicación no se ha iniciado correctamente. Como se puede notar, las anteriores variables son contadores anidados.

Cabe anotar que los anteriores archivos de audio, así como otros que se detallarán en las siguientes secciones, deben cargarse al celular, previa la ejecución de la aplicación. En este prototipo, se cargan en el directorio: `C:\\Data\\Lectbill\\Audio\\`. Esta función se ha descrito antes de plantear los procedimientos del PCA, puesto que se debe ejecutar en cada pasada de los *frames* de la cámara. A continuación se detalla el procedimiento de reconocimiento de los billetes mediante el PCA, que hace que esta función tenga sentido.

3.4.1.15 Normalización de la imagen de entrada

Una vez que la zona de interés ha sido correctamente detectada y escalada con todo el procesamiento anterior, la imagen de entrada está lista para pasar por el primer paso del PCA: la normalización. Recordando un poco, la normalización es un proceso de bajo nivel necesario para disminuir los efectos de los cambios de iluminación del entorno. Para un análisis más profundo del concepto de normalización de una imagen y sus efectos, se recomienda referirse a la sección 2.2.4. En el presente proyecto, la normalización de la imagen de entrada se la hace mediante la ecuación 2.10. En dicha ecuación, los valores seleccionados de media deseada y desviación estándar deseada son $m_N = 128$, $\sigma_N = 128$ respectivamente:

$$g_N = \frac{g - m}{\sigma}\sigma_N + m_N$$

Estas constantes fueron escogidas para evitar que el efecto de truncar los valores normalizados haga perder exactitud en los cálculos posteriores como se explica en la sección 2.2.4. De otra parte, como este proceso puede devolver números mayores a 255 e incluso negativos, la imagen normalizada debe ser guardada en una variable entera de 32 bits con signo en lugar de la acostumbrada representación de 8 bits sin signo. La función que implementa este proceso es `void CPca::Normaliza(CImagen &im)`, cuyo código fuente se encuentra en el anexo A5.17. El argumento `im`, es una referencia necesaria para poder acceder a los datos miembros del objeto de clase `CImagen` mediante el concepto de amistad de clases, como se observa en las líneas de código de los archivos de cabecera de ambas clases, en el anexo A5.1.2 y A5.1.3. Algo importante de destacar es que la función `void CPca::Normaliza(CImagen &im)`, normaliza la imagen de entrada como se explica en la sección 2.2.4 y además resta de dicha imagen normalizada, la imagen media. Por lo tanto, lo que esta función calcula es $(\Gamma_{in} - \Psi)$ de acuerdo la ecuación 2.44, donde Γ_{in} es la imagen de entrada normalizada y Ψ es la imagen media obtenida durante el entrenamiento.

3.4.1.16 Proyección en el espacio de billetes

La proyección en el espacio de billetes es la parte *online* del proceso de *eigenimages* descrito en la sección 2.4.4.2. Para esto, se utiliza la ecuación 2.51:

$$\omega_i = u_i^T \left(\Gamma_{in} - \Psi \right) \ \forall i = 1,2,...,K$$

Donde:

- Γ_{in} es la imagen de entrada ya escalada y normalizada.
- Ψ es la imagen media previamente cargada desde el fichero generado por MATLAB durante el proceso de entrenamiento y ubicado en *C:\\Data\\Lectbill\\carga1.txt*. Para más información referirse a la sección 3.2.2.
- u_i son las 24 *eigenimages* generadas por MATLAB durante el proceso de entrenamiento y ubicados en *C:\\Data\\Lectbill\\Eigen*. En el entrenamiento, las *eigenimages* fueron escaladas multiplicándolas por 2^{20}; esto es necesario ya que en *Symbian* C++ no es recomendable trabajar con variables de tipo coma flotante [34]. Se escogió 2^{20} para el escalamiento porque como se sabe la multiplicación o división por un número múltiplo de potencia de 2 se puede implementar eficientemente con desplazamientos binarios a la izquierda y derecha respectivamente. Por ejemplo, $\frac{x}{2^{20}}$ es lo mismo que la operación binaria $x >> 20$. Para más información referirse a la sección 3.2.2.
- La constante *K*, en este caso es igual al número de *eigenimages* escogidas, es decir, 24.

Como se podrá inferir, lo que se requiere es implementar una función que realice el producto punto entre cada una de las *K eigenimages* y $\left(\Gamma_{in} - \Psi \right)$. Como resultado se obtendrá la proyección de la imagen de entrada en el *espacio de billetes* en forma de un vector $\Omega_{in}^T = \left[\omega_1 \quad \omega_2 \quad ... \quad \omega_K \right]$ de *K* coeficientes o pesos que posteriormente se compararán con las imágenes patrón, usando la distancia de *Mahalanobis*, para determinar a qué billete corresponde o si no corresponde a ninguno. De otra parte, se debe recordar

que como los datos fueron escalados multiplicándolos por 2^{20}, es necesario dividir por la misma cantidad (desplazamiento binario de 20 posiciones, $>>20$) los coeficientes obtenidos luego de la proyección.

Finalmente, la función que implementa este proceso es `void CPca::ProPunto()`, cuyo código fuente se encuentra en el anexo A5.18. Es importante recordar que la ecuación $(\Gamma_{in} - \Psi)$ es calculada en la función `void CPca::Normaliza(CImagen &im)` como se explicó en el apartado anterior.

3.4.1.17 Cálculo de la distancia de *Mahalanobis* más pequeña

El presente procedimiento concluye el método de reconocimiento de imágenes empleado en el presente proyecto y descrito en la sección 2.4.4.2. Una vez disponible la imagen de entrada en una representación vectorial de pesos en el espacio de billetes, en el cual también están representadas de igual forma las imágenes de muestra del conjunto de entrenamiento (determinadas en la sección 3.2.2), el reconocimiento, básicamente, se trata de establecer una métrica para la similitud entre la entrada y las muestras. Esta se implementará, como se ha mencionado en la sección 2.4.4.2, mediante la distancia de *Mahalanobis* (ver sección 2.4.2.4), con el uso de la función *de Symbian* C++, `TUint CPca::DistMaha()`, que obtiene *la distancia de Mahalanobis* entre el vector de pesos de la imagen de entrada y cada vector de pesos de las imágenes del conjunto de muestras, en el espacio de billetes y, además, devuelve la denominación a la cual pertenece la imagen de entrada en filmación. Su código se adjunta en el anexo A5.19.

Esta función inicia recorriendo, por un lado, cada uno de los vectores que representan a las imágenes del conjunto de entrenamiento, llamados pesos de muestras y los valores propios de este conjunto de datos en el espacio de billetes (ambos grupos, cargados en las inicializaciones del programa). Por otro lado, por cada pasada de estos grupos, se recorre una vez los pesos que componen al vector que representa a la imagen de entrada. En cada uno de estos recorridos se implementa la ecuación 2.32, que establece la *distancia de Mahalanobis* entre cada vector de pesos de cada muestra y el vector de pesos de la imagen de entrada en filmación, sin la extracción de la raíz cuadrada que sugiere, para mejorar la eficiencia del algoritmo (la radicación es una operación de altos recursos en *Symbian* C++). En esa ecuación, para los valores de las varianzas σ_i^2, se utilizan los valores propios del conjunto de datos en el espacio de billetes, ya que ambos son directamente

proporcionales [35]. Los valores de la distancia a cada vector de pesos del conjunto de muestras se almacenan en un vector de *distancias de Mahalanobis.* Además de esto, dentro de los bucles de estos procesos, se implementa un algoritmo para la determinación de la menor distancia que se ejecuta en cada una de las pasadas anteriormente descritas. Al terminar estas pasadas, en la variable `pos` (cuyo rango será de *M*, el número de muestras; en este caso, 168) se tendrá el número de la imagen de muestra cuyo vector de pesos dista menos del vector de pesos de la imagen de entrada en filmación. Recordando la organización del conjunto de muestras original, detallada en las primeras líneas de la sección 3.2.2, así como el paso del programa de entrenamiento: *Formación del conjunto original de imágenes*, de esa sección, se determina, a partir de la variable `pos`, a cuál de los 6 grupos de denominaciones debe pertenecer la imagen de entrada en filmación (del criterio de la menor distancia), mediante la línea de código:

```
grupo=(pos+1)/(4*KMuestrasLado)+(((pos+1) % (4*KMuestrasLado))>0);
```

Como se puede notar, el grupo al que pertenece la imagen de entrada determina inmediatamente su denominación, por lo que para la obtención de la misma, se utiliza a continuación una estructura de selección múltiple para asignar a la variable `billete`, la denominación correspondiente en orden ascendente de acuerdo al grupo al que pertenezca la imagen.

Sin embargo, la función aún no puede devolver este valor de denominación como resultado final del método de reconocimiento en base al PCA, por cuanto falta un paso más que asegure la calidad del resultado. En la explicación de las funciones de discriminación de la imagen de entrada en filmación, en todas sus etapas de PDI, antes de llegar al método del PCA, se ha dicho que el último criterio de discriminación le corresponde al mismo método del PCA; y es que, bajo la suposición de que a este ámbito del programa ha llegado una imagen de entrada que no es de un billete, el método, como hasta aquí se ha planteado, simplemente concluirá que la imagen de entrada es un billete de la denominación de la imagen de muestra a la cual su vector de pesos dio la menor distancia, dando un resultado *falso positivo* (sí reconoce algo que no es un billete). Para evitar esta ocurrencia, según lo establecido en la teoría de la sección 2.4.4.2, se ha determinado un *umbral* para la *distancia de Mahalanobis mínima* obtenida anteriormente. Así, esta distancia deberá ser menor al valor del umbral para que haya una certeza de que los pesos de la imagen de entrada en filmación hacen que la imagen esté lo suficientemente cerca del espacio de billetes (*i.e.* es un billete) y la denominación obtenida con el método pueda validarse.

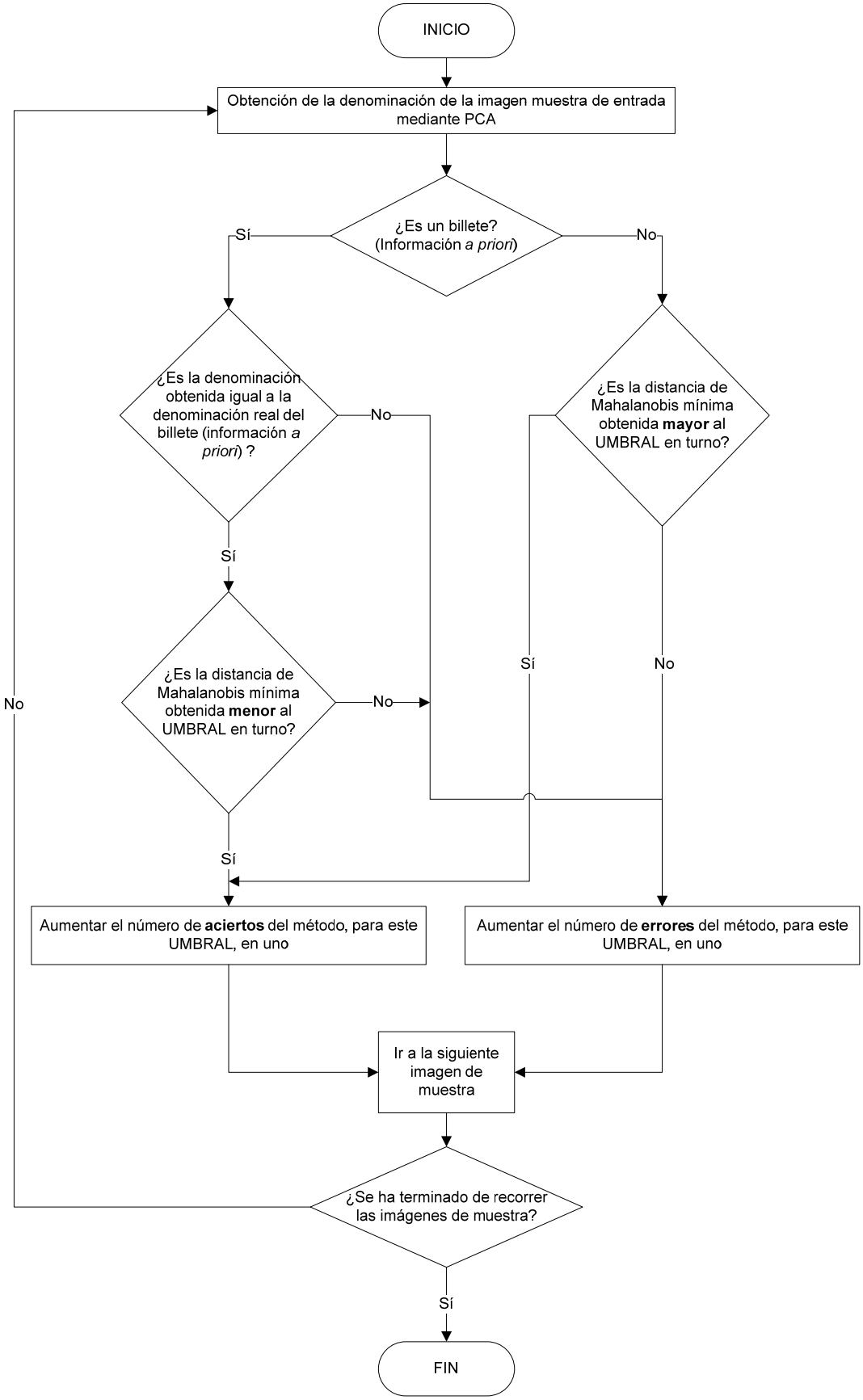


Figura 3.36. Diagrama de flujo para el establecimiento del umbral

El valor de dicho umbral se ha establecido mediante un procedimiento experimental. Se ha implementado, para esto, un programa en la herramienta MATLAB, complementario al de entrenamiento, cuyo código, basado en el diagrama de flujo de la figura 3.36, se puede revisar en el anexo A4.6.

Este programa hace uso de un conjunto de 85 muestras almacenadas en el directorio "Pruebas"; imágenes que, de forma aleatoria, representan billetes de diferentes denominaciones así como figuras que no son billetes (cortadas y escaladas de forma adecuada). El algoritmo recorre cada una de estas imágenes, implementando con ellas todo el procedimiento de reconocimiento explicado hasta aquí para determinar la denominación del supuesto billete que recibe como entrada. A continuación establece, para cada imagen, de forma discreta, el acierto o error del reconocimiento, haciendo uso de la información *a priori* hallada en el nombre del archivo de cada muestra de la carpeta "Pruebas", extraída en una variable de tipo estructura, propia de MATLAB, que indica de antemano si es o no un billete y su denominación, mediante la lógica del diagrama de flujo de la figura 3.36.

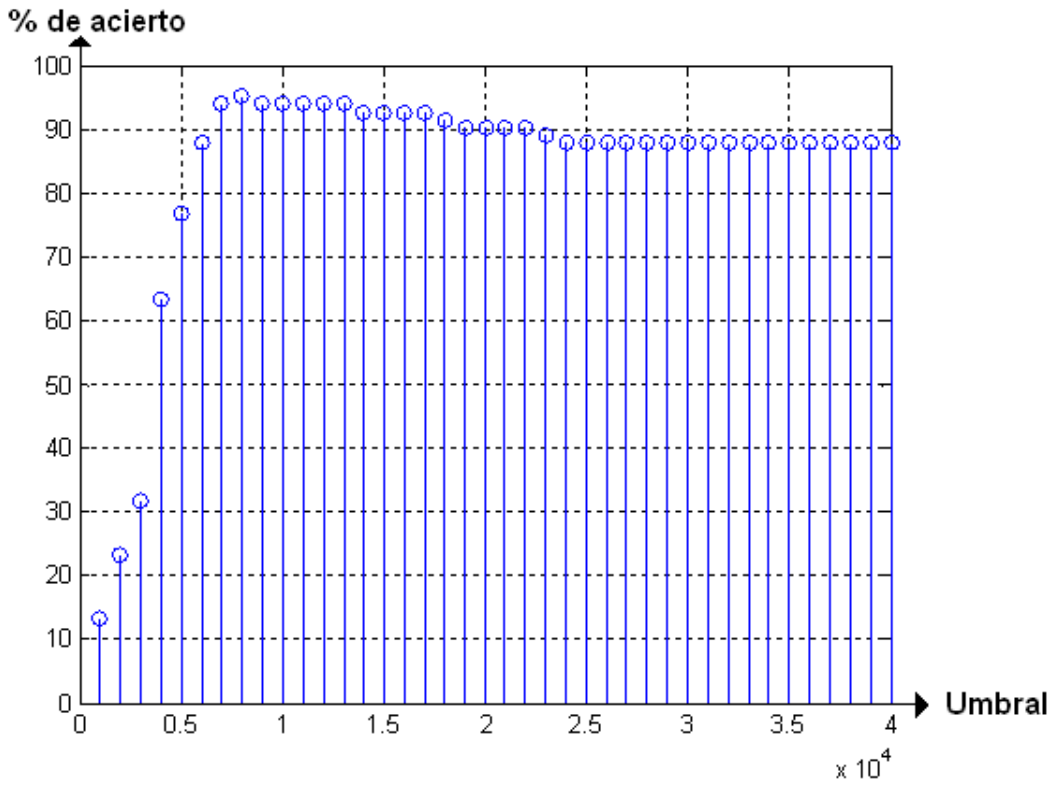


Figura 3.37. Porcentaje de acierto del método para distintos umbrales probados

El anterior algoritmo se realiza para distintos valores pruebas de umbral, en el rango de 1000 a 40000 (en base a las observaciones previas de *las distancias de Mahalanobis*

obtenidas para billetes). Para cada umbral, entonces, se puede obtener mediante esta lógica un porcentaje de aciertos que determina de una manera experimental una aproximación para el umbral que mejor porcentaje de éxitos tenga para este sistema. El resultado de este análisis se presenta en el gráfico de la figura 3.37.

De todo el anterior procedimiento, se estableció para este proyecto el valor del umbral en *8000*, a ser representado en el programa del teléfono celular en la constante `KUmbralDistM`.

Finalmente, la función `TUint CPca::DistMaha()`, devuelve el valor de la denominación encontrada para el billete de entrada en filmación solo si la distancia de *Mahalanobis* mínima obtenida para su determinación cumple la condición de ser menor a este valor de umbral. Retorna un valor de cero en caso contrario, como se ve en la línea de código de retorno del programa:

```
return (DistMin<KUmbralDistM) ? billete : 0;
```

3.4.1.18 Reproducción del mensaje de audio al usuario

Consiste en el procedimiento final del reconocimiento de la denominación de los billetes, la emisión del mensaje audible al usuario del sistema, implementado por la función de *Symbian* C++, `void CCameraAppBaseContainer::Aviso()`, adjunta en el anexo A5.20.

En el flujo del programa principal, mostrado en la figura 3.38, que se desarrolla en un ámbito de la clase `CCameraAppBaseContainer`, se define una variable de carácter global llamada `iDenominacion`, que, de inicio es igualada a cero. Al pasar la lógica del programa por el procedimiento del anterior inciso, el valor entero que devuelve la función `TUint CPca::DistMaha()`, se asigna a la variable `iDenominacion`, que, de esta forma, contendrá el valor de la denominación determinada por el método o un valor de cero, como se explicó. En base a este principio, la presente y última función implementa la lógica del diagrama de flujo de la figura 3.38.

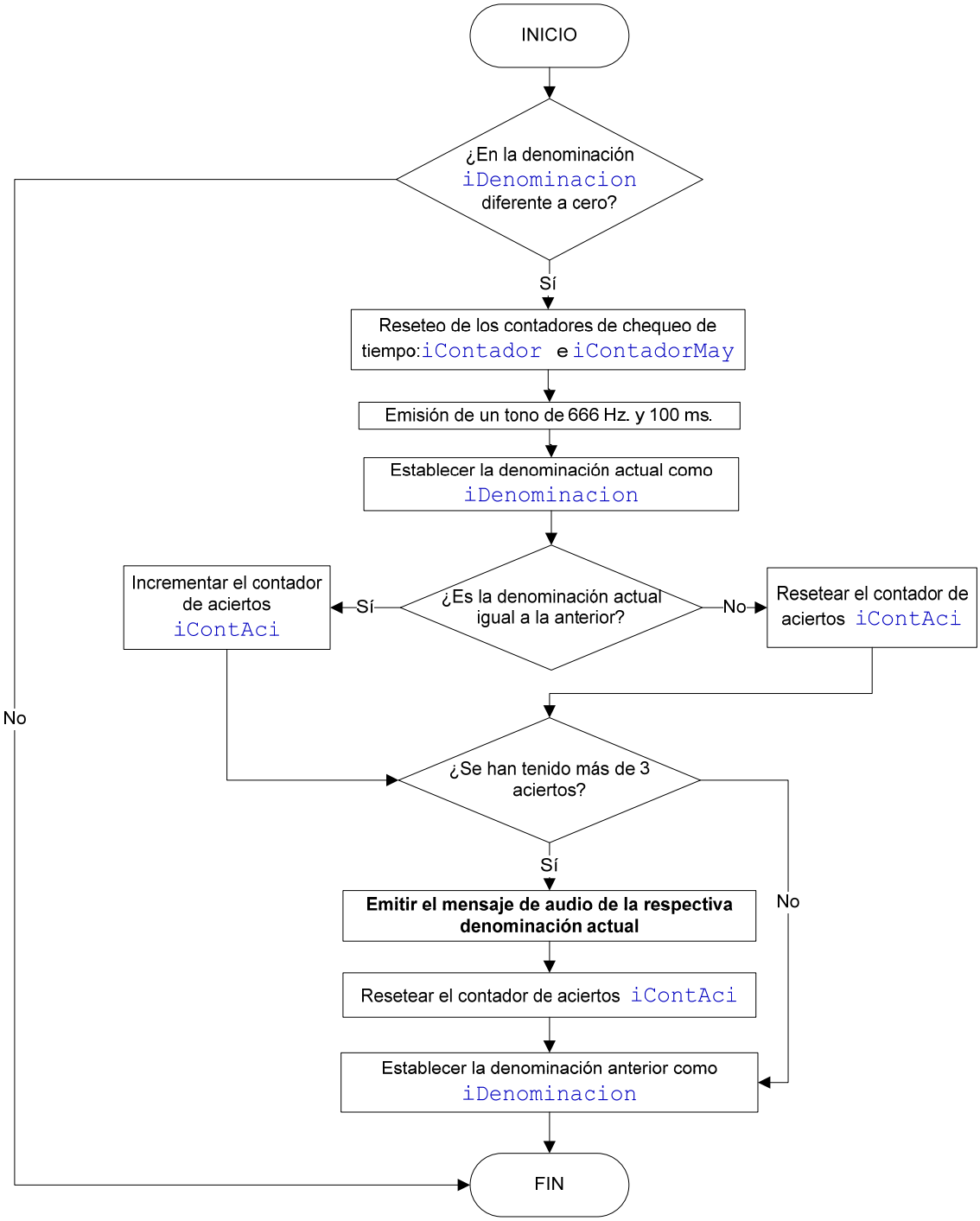


Figura 3.38. Diagrama de flujo del aviso al usuario

Este proceso se ejecutará en la pasada de cada *frame* procesado por el programa. En la lógica presentada, si se ha determinado un valor de denominación `iDenominacion`, se procede, primero, a la igualación a cero de los contadores explicados en el inciso: *Chequeo del tiempo de espera por el reconocimiento*, para indicar que la espera por la aplicación del método del PCA ha terminado, e iniciando un nuevo período. Inmediatamente, se procede a la emisión de un tono, propio del teléfono celular, de 666

Hz. de frecuencia y 100 ms. de duración, mediante la implementación de la clase `CToneAdapter` de *Symbian* C++, cuya teoría se puede revisar en [36]. Se hace esto con el objetivo de informar a la persona con discapacidad visual, mediante el sonido del tono, que la filmación que está realizando en el lugar que está y de la forma que lo está realizando está produciendo un resultado apropiado, por lo que sirve como *orientación* en el proceso de filmación del billete en reconocimiento.

El resto de la lógica se ocupa de establecer la condición de que, para que se pueda tener más certeza de que el método está produciendo un resultado preciso, debe al menos reconocer *3 veces* seguidas la misma denominación del billete que se está filmando. Esto se hace pues, si se llegara a dar el resultado erróneo por parte del método de confundir una denominación con otra, se prevé que este error se daría por un corto tiempo; en cambio, si el método recurrentemente comprueba la veracidad del resultado, al menos tres veces, se podrá decir con la mayor certeza posible en este proyecto que la denominación reconocida es correcta.

En el caso de que la anterior condición se cumpla, se procede por fin a la realización del proceso *Emitir el mensaje de audio de la respectiva denominación actual*, el cual es implementado por la función de Symbian C++, `void CCameraAppBaseContainer::Audio()`, adjunta en el anexo A5.21, que contiene una simple estructura de selección múltiple que evalúa la variable `iDenominacion` y, según su valor, finalmente emite el sonido correspondiente a la denominación al usuario, mediante la implementación de la clase `CSoundPlayer` de *Symbian* C++, cuya teoría se puede revisar en [33], para la reproducción del archivo correspondiente de cada denominación: *uno.mp3, cinco.mp3, diez.mp3*, etc. Como se ha dicho anteriormente, estos 6 archivos deben cargarse al celular, previa la ejecución de la aplicación; en este prototipo, en el directorio: `C:\\Data\\Lectbill\\Audio\\`, junto a los otros archivos de audio usados en los anteriores incisos.

Finalmente, los algoritmos descritos en esta sección son los que implementan la aplicación propuesta para la solución del problema planteada en este proyecto, a la cual se la ha denominado con el nombre de *LectBill*. Haciendo uso de las características del sistema detalladas tanto en su forma de funcionamiento, indicadas en este capítulo 3, como en la naturaleza teórica de sus procedimientos, detalladas en el capítulo 2; se ha procedido a la elaboración de un *manual de usuario* del sistema que se ha obtenido como producto

final, indicando, además, en el mismo la forma de uso e instalación de la aplicación *.sisx* que se produce mediante la plataforma de *software* utilizada y descrita en este capítulo. Se recomienda extensamente la lectura de este manual para aclarar toda duda sobre el funcionamiento en sí de la aplicación, documento que se adjunta como anexo A3.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

4.1 DESCRIPCIÓN DE LAS PRUEBAS REALIZADAS

Una vez terminado el desarrollo del prototipo del sistema, que finalizó con la producción de una aplicación de *software* denominada *LectBill*, para las plataformas que han sido descritas en el capítulo 3, se procede a la realización de un conjunto de pruebas de reconocimiento de la denominación de los dólares americanos de circulación más común en el entorno ecuatoriano, para establecer la eficacia del sistema.

La metodología seguida para la realización de las pruebas se basó en el paradigma de que la aplicación es dirigida a personas con discapacidad visual, razón por la cual surgió la necesidad de la preparación previa de un método para el uso de la aplicación. Esta necesidad produjo como resultado el *manual de usuario del sistema*, que describe en su literatura la forma en que una persona con discapacidad visual debe utilizar el sistema *LectBill*. De aquí que, siguiendo la metodología mencionada, fue necesaria la privación temporal del sentido de la vista para la realización de cada una de las pruebas individuales y, por tanto, la utilización de la guía de uso del sistema antes mencionada, misma que puede ser revisada en el anexo A3 de la presente memoria técnica.

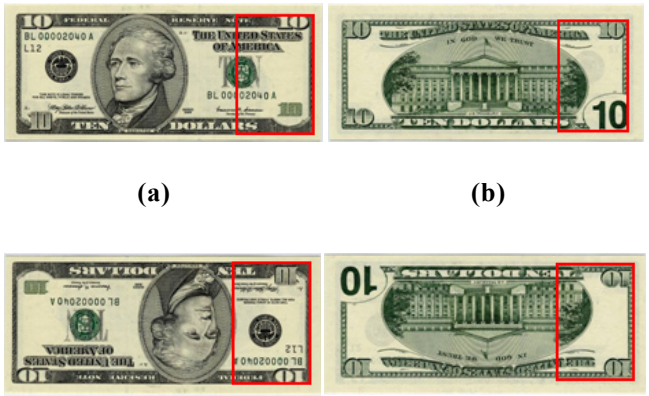
El trabajo de toma de pruebas se realizó haciendo uso de un banco de especímenes de billetes, adquirido especialmente para este propósito, que se distribuyeron de la siguiente forma:

- 116 billetes de un dólar.
- 36 billetes de cinco dólares.
- 44 billetes de diez dólares.
- 20 billetes de 20 dólares.

Se menciona, además, que se contó con un espécimen de cincuenta dólares y uno de cien dólares, para las pruebas del reconocimiento de estas denominaciones; sin embargo, no se incluyen a estas denominaciones en la descripción de las pruebas de este capítulo

porque no son de común circulación en el entorno de las personas con discapacidad visual y bastó con el reconocimiento de estos únicos especímenes.

Determinada la forma y los recursos para la realización de las pruebas, a continuación se describen los tres tipos de condiciones en las cuales se procedió a ejecutarlas; se realizó el intento de reconocimiento de la denominación de cada uno de los billetes, por cada uno de sus 4 extremos, mostrados en la figura 4.1.



(a) **(b)**

(c) **(d)**

Figura 4.1. Extremos considerados para la toma de imágenes en los reconocimientos de prueba: (a)Extremo uno (b) Extremo dos (c) Extremo tres (d) Extremo cuatro

Los reconocimientos fueron realizados para las condiciones físicas siguientes:

- **Condiciones ideales.** El billete se sitúa sobre un fondo de color negro. La iluminación al billete es directa y ajustada para asegurar la eliminación de la producción de sombras encima de su imagen. El entorno en estas condiciones es *indoor*; es decir, dentro de una edificación que provee todas estas facilidades.

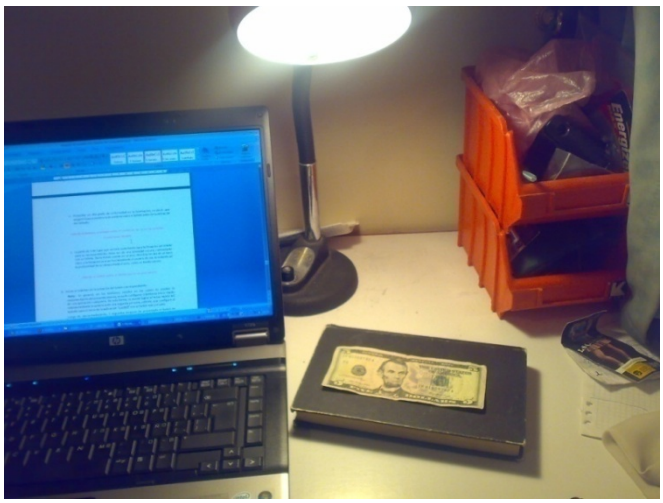


Figura 4.2. Condiciones ideales para el reconocimiento

- **Condiciones normales.** El billete es blandido por el usuario, que utiliza un fondo específico que no es el ideal (*e.g.* el piso de una habitación). El entorno es *indoor*; es decir, dentro de una edificación cualquiera, con la iluminación que provea en su momento, según sean horas de claridad o de oscuridad.

Figura 4.3. Condiciones normales para el reconocimiento

- **Condiciones extremas.** El billete es blandido por el usuario, que utiliza un fondo específico que no es el ideal. El entorno es *outdoor*; es decir, cualquier lugar fuera de una edificación en el cual la iluminación sea la natural.

Figura 4.4. Condiciones extremas para el reconocimiento

4.2 RESULTADOS OBTENIDOS

Previamente, se establece el criterio para la clasificación de los resultados a obtenerse mediante la realización de estas pruebas. Se clasifica al resultado de la prueba realizada como:

- **Acierto a la primera prueba.** Si con la primera implementación de la metodología para el uso del sistema, este emite un mensaje audible que corresponde a la denominación del billete en prueba.
- **Acierto a la segunda prueba.** Si ha fallado el primer intento de reconocimiento, pero con la segunda implementación de la metodología para el uso del sistema, este emite un mensaje audible que corresponde a la denominación del billete en prueba.
- **Sin resultado.** Si, tras haber realizado dos veces el proceso detallado para el uso del sistema, el mismo no ha producido mensaje alguno que indique la denominación de un billete.
- **Falso positivo.** Si tras cualquiera de los dos intentos de reconocimiento mediante la metodología para el uso del sistema se ha escuchado un mensaje audible, pero de una denominación distinta a la del billete en prueba. Se considera el caso más desfavorable de todos.

Para la recopilación de resultados exitosos, se utiliza el criterio que el sistema ha funcionado correctamente solo si ha producido un *acierto* hasta la segunda prueba como máximo. A continuación, se exhiben en distintas secciones los resultados obtenidos, tabulados según la condición de prueba y la denominación del billete en prueba:

4.2.1 Resultados en condiciones ideales

Tabla 4.1. Resultados del billete de $1 en condiciones ideales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	114	98.3	0	0.0	114	98.3	2	1.7	0.0	0.0	116
2	86	74.1	29	25.0	115	99.1	1	0.9	0.0	0.0	116
3	111	95.7	5	4.3	116	100.0	0	0.0	0.0	0.0	116
4	97	83.6	19	16.4	116	100.0	0	0.0	0.0	0.0	116
TOTAL:		**87.9**		**11.4**		**99.4**		**0.6**		**0.0**	

Tabla 4.2. Resultados del billete de $5 en condiciones ideales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	35	97.2	1	2.8	36	100.0	0	0.0	0.0	0.0	36
2	32	88.9	4	11.1	36	100.0	0	0.0	0.0	0.0	36
3	34	94.4	2	5.6	36	100.0	0	0.0	0.0	0.0	36
4	36	100.0	0	0.0	36	100.0	0	0.0	0.0	0.0	36
TOTAL:		**95.1**		**4.9**		**100.0**		**0.0**		**0.0**	

Tabla 4.3. Resultados del billete de $10 en condiciones ideales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	44	100.0	0	0.0	44	100.0	0	0.0	0.0	0.0	44
2	44	100.0	0	0.0	44	100.0	0	0.0	0.0	0.0	44
3	44	100.0	0	0.0	44	100.0	0	0.0	0.0	0.0	44
4	43	97.7	1	2.3	44	100.0	0	0.0	0.0	0.0	44
TOTAL:		**99.4**		**0.6**		**100.0**		**0.0**		**0.0**	

Tabla 4.4. Resultados del billete de $20 en condiciones ideales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	19	95.0	1	5.0	20	100.0	0	0.0	0.0	0.0	20
2	20	100.0	0	0.0	20	100.0	0	0.0	0.0	0.0	20
3	20	100.0	0	0.0	20	100.0	0	0.0	0.0	0.0	20
4	20	100.0	0	0.0	20	100.0	0	0.0	0.0	0.0	20
TOTAL:		**98.8**		**1.2**		**100.0**		**0.0**		**0.0**	

Tabla 4.5. Resultados totales en condiciones ideales

	Aciertos	Sin resultado	Falsos positivos
Porcentaje:	**99.838**	**0.162**	**0**

4.2.2 Resultados en condiciones normales

Tabla 4.6. Resultados del billete de $1 en condiciones normales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	107	92.2	5	4.3	112	96.6	4	3.4	0.0	0.0	116
2	40	34.5	76	65.5	116	100.0	0	0.0	0.0	0.0	116
3	116	100.0	0	0.0	116	100.0	0	0.0	0.0	0.0	116
4	71	61.2	45	38.8	116	100.0	0	0.0	0.0	0.0	116
TOTAL:		**72.0**		**27.1**		**99.1**		**0.9**		**0.0**	

Tabla 4.7. Resultados del billete de $5 en condiciones normales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	36	100.0	0	0.0	36	100.0	0	0.0	0.0	0.0	36
2	29	80.6	7	19.4	36	100.0	0	0.0	0.0	0.0	36
3	31	86.1	4	11.1	35	97.2	0	0.0	1.0	2.8	36
4	28	77.8	8	22.2	36	100.0	0	0.0	0.0	0.0	36
TOTAL:		**86.1**		**13.2**		**99.3**		**0.0**		**0.7**	

Tabla 4.8. Resultados del billete de $10 en condiciones normales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	42	95.5	2	4.5	44	100.0	0	0.0	0.0	0.0	44
2	42	95.5	1	2.3	43	97.7	1	2.3	0.0	0.0	44
3	44	100.0	0	0.0	44	100.0	0	0.0	0.0	0.0	44
4	43	97.7	1	2.3	44	100.0	0	0.0	0.0	0.0	44
TOTAL:		**97.2**		**2.2**		**99.4**		**0.6**		**0.0**	

Tabla 4.9. Resultados del billete de $20 en condiciones normales

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	18	90.0	2	10.0	20	100.0	0	0.0	0.0	0.0	20
2	19	95.0	0	0.0	19	95.0	1	5.0	0.0	0.0	20
3	18	90.0	2	10.0	20	100.0	0	0.0	0.0	0.0	20
4	20	100.0	0	0.0	20	100.0	0	0.0	0.0	0.0	20
TOTAL:		**93.8**		**5.0**		**98.8**		**1.2**		**0.0**	

Tabla 4.10. Resultados totales en condiciones normales

	Aciertos	Sin resultado	Falsos positivos
Porcentaje:	**99.158**	**0.670**	**0.172**

4.2.3 Resultados en condiciones extremas

Tabla 4.11. Resultados del billete de $1 en condiciones extremas

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	89	76.7	20	17.2	109	94.0	7	6.0	0.0	0.0	116
2	31	26.7	85	73.3	116	100.0	0	0.0	0.0	0.0	116
3	98	84.5	18	15.5	116	100.0	0	0.0	0.0	0.0	116
4	62	53.4	54	46.6	116	100.0	0	0.0	0.0	0.0	116
TOTAL:		**60.3**		**38.2**		**98.5**		**1.5**		**0.0**	

Tabla 4.12. Resultados del billete de $5 en condiciones extremas

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	33	91.7	1	2.8	34	94.4	2	5.6	0.0	0.0	36
2	29	80.6	5	13.9	34	94.4	2	5.6	0.0	0.0	36
3	30	83.3	4	11.1	34	94.4	1	2.8	1.0	2.8	36
4	30	83.3	4	11.1	34	94.4	2	5.6	0.0	0.0	36
TOTAL:		**84.7**		**9.7**		**94.4**		**4.9**		**0.7**	

Tabla 4.13. Resultados del billete de $10 en condiciones extremas

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	40	90.9	2	4.5	42	95.5	2	4.5	0.0	0.0	44
2	41	93.2	2	4.5	43	97.7	1	2.3	0.0	0.0	44
3	38	86.4	4	9.1	42	95.5	2	4.5	0.0	0.0	44
4	40	90.9	1	2.3	41	93.2	3	6.8	0.0	0.0	44
TOTAL:		**90.3**		**5.1**		**95.5**		**4.5**		**0.0**	

Tabla 4.14. Resultados del billete de $20 en condiciones extremas

Muestras del lado	Aciertos a la primera prueba		Aciertos a la segunda prueba		Aciertos totales		Sin resultado		Falsos positivos		Total de muestras de cada lado
	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	
1	19	95.0	1	5.0	20	100.0	0	0.0	0.0	0.0	20
2	14	70.0	4	20.0	18	90.0	2	10.0	0.0	0.0	20
3	19	95.0	0	0.0	19	95.0	1	5.0	0.0	0.0	20
4	14	70.0	3	15.0	17	85.0	3	15.0	0.0	0.0	20
TOTAL:		**82.5**		**10.0**		**92.5**		**7.5**		**0.0**	

Tabla 4.15. Resultados totales en condiciones extremas

	Aciertos	Sin resultado	Falsos positivos
Porcentaje:	**95.223**	**4.605**	**0.172**

4.2.4 Resultados Finales

Tabla 4.16. Resultados finales

Condiciones	% Aciertos	% Sin resultado	% Falsos positivos
Condiciones ideales	99.838	0.162	0.000
Condiciones normales	99.156	0.670	0.174
Condiciones extremas	95.223	4.605	0.172
Promedio:	**98.1**	**1.8**	**0.1**

4.3 ANÁLISIS DE LOS RESULTADOS

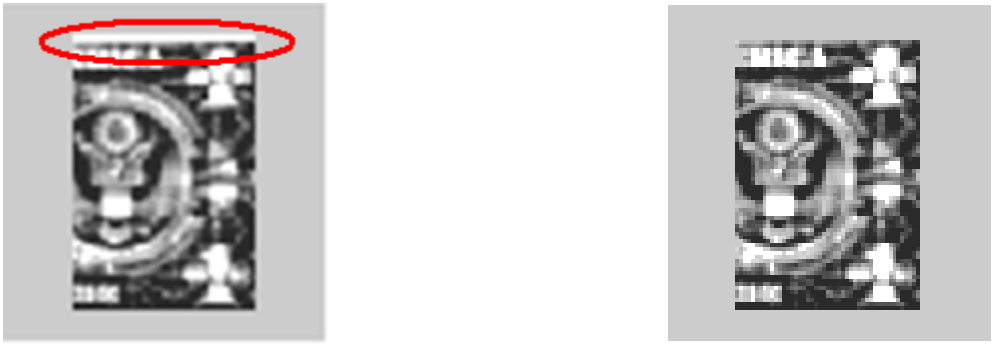
Los resultados obtenidos en la anterior sección proveen de información acerca de la eficacia del sistema, así como de su manejo; estos se pueden analizar desde distintos puntos de vista, en cada uno de los cuales establece una conclusión en particular sobre el sistema. Los más importantes son los siguientes:

Al estudiar los porcentajes totales de éxito del sistema para cada condición de funcionamiento, se puede apreciar que el sistema tiene un desempeño prácticamente infalible bajo condiciones ideales, y que a partir de allí, su eficacia se va coartando en un bajo porcentaje, según disminuye la calidad de las condiciones. Según esto, el porcentaje total de éxitos más alto se tiene en un número de 99.8% y el más bajo, en las condiciones extremas, en un valor de 95.2%. Además, se observa que la degradación de la calidad del método, de las condiciones ideales a las normales, es tan solo de un 0.6%. De esto, se puede concluir, que de forma general, el sistema tiene mejores oportunidades de dar un resultado favorable en condiciones puertas adentro; es decir, que para que su eficacia se garantice, se debe hacer funcionar al sistema dentro de alguna edificación. Por esto, se clasifica al sistema aquí realizado como *indoor*. Estas conclusiones son acordes a las predicciones de la teoría utilizada para el desarrollo del sistema, el método del PCA. Justamente, siguiendo estos preceptos se estructuró el método de utilización recomendando el ambiente más favorable de funcionamiento al usuario, como se aprecia en el manual de usuario en el anexo A3.

Un problema que se encontró durante las pruebas, fue el bajo porcentaje de acierto al primer intento de los extremos 2 y 4 del billete de 1 dólar (en algunos casos menos del 50 %). Luego de analizar varias imágenes, se concluyó que el problema eran las muestras del billete no ideales que presentaban una parte del borde de la imagen cortada, blanco (ver

figura 4.5a) a diferencia de las muestras ideales (ver figura 4.5b). Esto provocaba que las muestras no ideales generen distancias mínimas más altas (alrededor de 14000) que el umbral máximo (8000). Para solucionar este inconveniente, se siguió la siguiente lógica:

- Si la muestra es reconocida como el lado 2 o 4 del billete de 1 dólar, la distancia mínima se divide por 2 antes de compararla con el umbral máximo. Por ejemplo, si la muestra genera una distancia mínima de 12000, se divide por 2, lo cual resulta en una distancia de 6000 que es menor al umbral 8000. Con esto la denominación es reconocida como de 1 dólar.



a) **b)**

Figura 4.5. Lado 2 del billete de 1 dólar:
a) Muestra no ideal con borde blanco remarcable b) Muestra ideal sin borde blanco

Con respecto al tema de los falsos positivos, el resultado más desfavorable que se puede tener en un sistema de reconocimiento de dinero, se puede apreciar que el sistema tiene una baja ocurrencia de los mismos, en un porcentaje de 0.172 %. De hecho, bajo condiciones normales y extremas, presenta un alcance similar y en las ideales es inexistente. De cualquier forma, se concluye que las condiciones en las que el sistema funcione, deben tener la prestación de ser las más aproximadas a las ideales para la evasión de los falsos positivos.

Como comentario final, las pruebas que se catalogan como: *sin resultado*, indican, en su bajo porcentaje de ocurrencia, que el sistema no ha funcionado en los primeros dos intentos; no sugiriendo con esto que no ha funcionado en absoluto. A pesar de esto, se redacta en el método de uso del manual de usuario también la forma de afrontar problemas relacionados con el reconocimiento tardío o demorado de la denominación, así como en la programación detallada en la sección 3.4 se toma en cuenta esta eventualidad mediante la producción de los mensajes de audio correspondientes.

4.4 PRUEBAS CON LA POBLACIÓN NO VIDENTE

Las pruebas realizadas fueron orientadas a medir la facilidad de uso del sistema por parte de la población no vidente. Con este objetivo, se siguió el siguiente protocolo:

1. Se solicitó la ayuda de un grupo de 4 personas no videntes de la *Sociedad de Ciegos Luis Braille*[8] (misma institución donde se aplicaron las entrevistas iniciales mediante la guía del anexo A1).
2. Durante una etapa de entrenamiento, se los capacitó en el uso del sistema siguiendo los pasos del método de utilización descrito en el manual de usuario (ver anexo A3).
3. Una vez que el no vidente comprendió el modo de utilización del sistema, se le entregó un billete para que aplique el método aprendido.

Los resultados obtenidos de estas pruebas fueron satisfactorios. A pesar de haber practicado solo algunos minutos, los no videntes fueron capaces de utilizar el sistema de forma adecuada. Bajo estas circunstancias, es de gran utilidad que el presente prototipo procese *frames,* es decir vídeo, ya que los usuarios no tienen que preocuparse por fotografiar el billete, simplemente deben filmarlo. Además, los pitidos que el no vidente escucha cada vez que el billete ha sido correctamente reconocido, facilitan la tarea de reconocimiento.

4.5 VELOCIDAD DE PROCESAMIENTO

Para concluir las pruebas, se determinó la velocidad de procesamiento que la aplicación *LectBill* tiene en los dispositivos en los que se probó, detallados en la sección 3.3.1.3. Para esto, dentro de la programación del dispositivo, se implementó una función que utiliza los recursos de *Symbian* C++ para medir la velocidad de procesamiento en *frames* por segundo (FPS) que la aplicación es capaz de procesar. Esta función es `void CCameraAppBaseContainer::mostrarFps()` y se adjunta en el anexo A4.24.

La función implementada imprime en la pantalla el número de FPS que la aplicación procesa. Se observó que, al instante de un reconocimiento de la denominación de un billete, el número mínimo de FPS que la aplicación procesaba era de 7.

[8] Calle Fores N4132 y Espejo, Quito, Ecuador. Tlf. 2950969 ó 096313225 (Sr. Hernán Boada).

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- Un sistema, dirigido a la población con discapacidad visual, para el reconocimiento de la denominación de los dólares americanos de más común circulación en la República del Ecuador, con las características de fácil acceso y portabilidad, se puede implementar mediante el uso de la tecnología de los teléfonos celulares y la teoría del procesamiento digital de imágenes.

- El sistema desarrollado en el presente proyecto con la teoría del Procesamiento digital de imágenes, presenta un conjunto de limitaciones con respecto al entorno físico en el que se ejecuta. De todas ellas, las más considerables son: que el fondo de la imagen que contiene al objeto que se pretende identificar con el sistema debe ser contrastante con dicho objeto y que las condiciones de luz sobre la imagen que se pretende procesar, deben ser uniformes.

- La teoría del Análisis de componentes principales, detallada en el capítulo 2 de la presente redacción, es una técnica matemática con múltiples aplicaciones en el procesamiento y análisis de datos. El método de las *Eigenimages* es una de aquellas aplicaciones, utilizada para un rápido y preciso reconocimiento de imágenes en sistemas de visión por computador.

- Los teléfonos inteligentes bajo plataforma *Symbian*, mediante su lenguaje nativo *Symbian* C++, pueden ser utilizados en aplicaciones que requieran procesar imágenes en tiempo real. Esto quedó demostrado por la velocidad de procesamiento que el prototipo desarrollado alcanza (al menos 7 FPS).

- Como el presente prototipo ha sido desarrollado en *Symbian* C++, su portabilidad a otras plataformas de teléfonos inteligentes basadas en C++ como *Android*, *Windows Mobile*, *Linux* e inclusive *iPhone*, es totalmente viable, ya que el núcleo de la aplicación, es decir, la parte del código fuente que procesa la imagen, hace uso de tipos y estructuras típicas de cualquier versión de C++.

- *LectBill*, el sistema prototipo implementado en el presente proyecto para el reconocimiento de la denominación de los dólares americanos, tiene un porcentaje máximo de éxito de 99.838% en las condiciones ideales detalladas en la sección 4.1. Además, tiene la capacidad de desempeñar una exactitud de 99.158% en condiciones *indoor*.

- La realización de todo tipo de aplicaciones que se traduzcan en un aporte social, ya sea orientado a minorías o de un carácter más general, en teléfonos celulares móviles, es totalmente factible mediante la utilización de herramientas como las descritas en la sección 3.3 u otras disponibles en el vasto espacio de la *web*; mismas que pueden seleccionarse de forma adecuada posteriormente a un proceso de investigación de sus prestaciones y limitaciones.

5.2 RECOMENDACIONES

- Previa a la realización de un trabajo que involucre los problemas de un grupo específico, se recomienda la implementación de un adecuado método de investigación, como la encuesta, para el reconocimiento de las necesidades y requerimientos específicos de las personas pertenecientes a dicho grupo.

- A pesar de que la plataforma *Symbian* ofrece otros lenguajes como alternativa a *Symbian* C++ (*e.g. Python* y J2ME), estos no son recomendables cuando se desea desarrollar aplicaciones de procesamiento de imágenes en tiempo real o en general aplicaciones que requieran procesamiento intenso.

- Con respecto a las características avanzadas de *Carbide.c++*, que es el entorno integrado de desarrollo por defecto para desarrollar aplicaciones bajo *Symbian***,** se recomienda la lectura de la referencia [32] para comprender el funcionamiento de la depuración *on device* y otras de sus prestaciones.

- Para obtener información con respecto al funcionamiento específico del sistema implementado en este proyecto, *LectBill*, se recomienda la lectura del manual de usuario, adjunto en el anexo A3 de esta memoria técnica.

- Previo al uso del sistema *LectBill*, se recomienda al usuario, con discapacidad visual, entregar todo el tiempo necesario para un entrenamiento adecuado de reconocimientos de billetes con el uso del sistema; siempre apoyado por la lectura del manual de usuario, en el anexo A3 del presente escrito.

- Se recomienda el desarrollo del presente proyecto en otras plataformas de *software* para teléfonos móviles, como el sistema operativo *Symbian* desde su versión 9.2 en adelante o alguna otra basada en C++.

- Es ampliamente recomendada una continua búsqueda del mejoramiento de la presente aplicación, mediante desarrollos como la implementación de un aviso al usuario de que se halla ubicando correctamente al billete en la filmación, con un mensaje de vibración en lugar de un tono, para cubrir más área de las necesidades de poblaciones con discapacidades; la investigación de métodos distintos o similares al PCA, para el reconocimiento de imágenes, menos sensibles a la uniformidad de la iluminación de entorno; y cualesquiera otros procesos que solventen los problemas hallados en los métodos utilizados en el presente proyecto.

- Se recomienda extensamente la continuación del desarrollo de aplicaciones para sistemas microprocesados portátiles, como los teléfonos celulares y, particularmente, que se procure que los desarrollos contribuyan de alguna forma al mejoramiento de la calidad de vida de las minorías, como la población de personas con algún tipo de discapacidad.

ANEXOS

A1

GUÍA DE ENTREVISTA A PERSONAS NO VIDENTES

ESCUELA POLITÉCNICA DEL EJÉRCITO

GUÍA DE ENTREVISTA

DATOS INFORMATIVOS

Nombre de la institución: Sociedad de ciegos de Pichincha "Luis Braille"

Nombre del/la entrevistado/a: Persona no vidente

Cargo de los entrevistadores: Estudiantes

Lugar de la entrevista: Calle Flores N4132 y Espejo

Fecha: Lunes, 2 de marzo de 2009

Hora: 16h00

DESARROLLO

Presentación

Nosotros, Felipe Grijalva y Juan Carlos Rodríguez, estudiantes de la Escuela Politécnica del Ejército, de la carrera de Ingeniería Electrónica, en la presente entrevista estableceremos los criterios necesarios para el desarrollo de la implementación del prototipo de nuestro tema de proyecto de grado: "DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA EL RECONOCIMIENTO DE LA DENOMINACIÓN DE DÓLARES AMERICANOS, DIRIGIDO A PERSONAS CON DISCAPACIDAD VISUAL."

Para el efecto, se realizará una serie de entrevistas a 4 personas con discapacidad visual de la institución mencionada, con el objetivo de establecer las características que debe poseer el sistema que se pretende implementar con el fin de satisfacer de la mejor manera posible las necesidades específicas que se obtengan en base al análisis de los criterios de este grupo social.

Sr./Srta. Entrevistad@:

El objetivo de esta entrevista es conocer su experiencia con el problema del reconocimiento de la denominación de los billetes así como la forma en la que le gustaría que se dé solución a este problema.

Temario de preguntas de la entrevista

1. ¿De qué forma usted, en su diario vivir, reconoce la denominación de un billete?
2. ¿Qué problemas se le han presentado al realizar las transacciones con billetes necesarias, diariamente?
3. ¿Qué tan útil le parece la idea de disponer de un artefacto electrónico capaz de emitir un mensaje de audio que le informe la denominación del billete que usted le exponga?
4. De parecerle útil la idea, ¿de qué forma piensa que se tendría la mejor interacción de la persona no vidente con el artefacto?

Para la presente cuestión, se expone al entrevistado las opciones que se manejan como referencias, que son:

- *Una interacción con un aparato que absorba al billete al instante de su inserción en su ranura de entrada, al estilo de una máquina dispensadora.*
- *Una interacción con un artefacto que disponga de una plataforma en la cual se deposite al billete encima de la misma y se le tape, al estilo de un escáner o una copiadora.*
- *Algún otro dispositivo.*

5. ¿Qué características físicas adicionales cree que le darían un valor agregado al dispositivo?/¿Qué sugerencias adicionales puede plantear al respecto?

Agradecemos al entrevistado por su tiempo y las facilidades prestadas.

A2

IMÁGENES DE LOS TIPOS DE BILLETES DE MAYOR CIRCULACIÓN EN ECUADOR A DISCRIMINARSE MEDIANTE EL SISTEMA

Figura 1. Billete de 1 dólar americano.

Figura2. Billete de 5 dólares americanos (Series 1999 - 2003A)

Figura 3. Billete de 5 dólares americanos (Series 2006)

Figura 4. Billete de 10 dólares americanos (Series 2006)

Figura 5. Billete de 20 dólares americanos (Series 1996-2001)

Figura 6. Billete de 20 dólares americanos (Series 2006)

Figura 7. Billete de 50 dólares americanos (Series 2006)

Figura 8. Billete de 100 dólares americanos (1996-2003A)

A3

MANUAL DE USUARIO DEL SISTEMA

SISTEMA DE RECONOCIMIENTO DE LA DENOMINACIÓN DE DÓLARES NORTEAMERICANOS DIRIGIDO A PERSONAS CON DISCAPACIDAD VISUAL

"LectBill"

Versión: Prototipo

MANUAL DE USUARIO

El sistema de reconocimiento de la denominación de dólares americanos, "LectBill" está diseñado específicamente para personas no videntes o con alguna discapacidad visual. El presente manual de usuario está orientado a una persona vidente, que provea de asistencia al usuario del sistema, previa al uso del mismo.

ÍNDICE DE CONTENIDO

1. PRESENTACIÓN

El presente manual persigue el objetivo de la familiarización del usuario con discapacidad visual, con el sistema de reconocimiento de denominación de dólares americanos, *LectBill*. Mediante este documento, el lector (persona vidente) estará en la capacidad de asistir al usuario del sistema, para la correcta utilización del mismo. La parte más importante, referente al método de utilización del sistema, se anexa en una versión audible para el uso de la persona con discapacidad visual. Se recuerda que la presente versión es de prototipo, por lo que presenta limitaciones propias del diseño.

2. REQUERIMIENTOS

LectBill es una aplicación de *software* diseñada para trabajar en un teléfono celular móvil. En la presente versión, de prototipo, el dispositivo móvil, debe cumplir con los siguientes requerimientos mínimos:

- **Cámara de vídeo**
- **Sistema operativo [9]:** *Symbian* OS
  - **Versión:** 9.1, 9.2
  - **Plataforma:** S60 3º edición
  - ***Feature Pack*:** *Initial Release* o *Feature Pack* 1
- **Memoria RAM necesaria:** 2 MB
- **Capacidad en disco:** 1.6 MB
- **Multimedia:** MP3

Para la instalación de la aplicación, se requiere además una PC con conectividad *Bluetooth* 2.0, ó el *software* de comunicación del dispositivo móvil a una PC, según el tipo de conectividad de cada modelo de celular (para el fabricante *Nokia*, el *software* de comunicación es el N*okia PC Suite*).

[9] Requisito fundamental

3. ESPECIFICACIONES DE LOS EQUIPOS

El presente prototipo del sistema, fue probado con total éxito en dos dispositivos móviles que cumplen con los requerimientos de la sección anterior. Se presentan en esta sección a manera de ejemplo de los requerimientos de funcionamiento del sistema. Los modelos de estos dispositivos son:

Nokia N73

- **Pantalla:** 320 x 240 pixeles
- **Cámara de vídeo:** 3.2 Megapíxeles con grabación de vídeo y auto contraste
- **Sistema operativo:** *Symbian* OS 9.1 + Plataforma S60, 3º edición, *Initial Release*
- **CPU:** ARM 9 @ 220 MHz
- **Memoria RAM:** 64 MB (24 MB disponibles)
- **Capacidad de disco:** 42 MB
- **Multimedia:** MP3, WAV, otros.

Nokia E65

- **Pantalla:** 320 x 240 píxeles
- **Cámara de vídeo:** 2 Megapíxeles con grabación de vídeo y auto contraste
- **Sistema operativo:** *Symbian* OS 9.1 + Plataforma S60, 3º edición, *Initial Release*
- **CPU:** ARM 9@ 220 MHz
- **Memoria RAM:** 64 MB (24 MB disponibles)
- **Capacidad en disco:** 42 MB.
- **Multimedia:** MP3, WAV, otros.

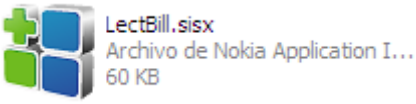
Adicionalmente, se presenta a continuación una lista de los modelos de teléfonos celulares que cumplen con los requerimientos básicos listados en la sección 2, clasificados según su sistema operativo:

- ***Symbian* OS 9.1, S60, 3era edición, *Initial Release:*** Nokia N77, Nokia E61i, Nokia E65, Nokia N93i, Nokia N91 8GB, Nokia E62, Nokia E50, Nokia 5500 Sport, Nokia N73, Nokia N93, Nokia N92, Nokia N71, Nokia N80, Nokia E60, Nokia E61, Nokia E70, Nokia 3250, Nokia N91.
- ***Symbian* OS 9.2, S60, 3era edición, *Feature Pack 1:*** Nokia E63, Nokia E71, Nokia E66, Nokia 6124 classic, Nokia N82, Nokia E51, Nokia N95-3 NAM, Nokia N95 8GB, Nokia N81 8GB, Nokia N81, Nokia 6121 *classic*, Nokia 6120 *classic*.

4. CONTENIDO DEL SISTEMA

La versión del prototipo del sistema que es entregada a manera de cortesía a la institución que corresponda, consta del siguiente contenido:

- Un disco compacto con los archivos:



  - **Archivo *"LectBill.sisx"***: Archivo de la aplicación a instalarse en el dispositivo móvil.

  - **Carpeta *"Lectbill"***: Carpeta con los archivos necesarios para el funcionamiento de la aplicación.
  - El presente manual de usuario en formato digital.

Nota.- En este punto, cabe aclarar que, para que una aplicación desarrollada para *Symbian* OS sea válida y se pueda utilizar, debe estar firmada (Consultar: https://www.symbiansigned.com). Como la presente versión es un prototipo, no se

requirió un pago por la firma de la aplicación realizada y la aplicación debería funcionar en cualquier dispositivo con los requerimientos fijados en las secciones anteriores. Sin embargo, existe la posibilidad de que no sea funcional en determinados dispositivos aún si cumplen los requerimientos, por el tema de la firma digital. Este puede ser un caso de imposibilidad de la realización de los incisos 4 y 5 del presente manual de usuario.

5. GUÍA DE INSTALACIÓN [10]

1. Introducir el CD de la aplicación en una PC y ubicar su contenido.
2. Encender el dispositivo móvil y establecer la comunicación pertinente con la computadora personal:

   EN UN DISPOSITIVO CON CONECTIVIDAD BLUETOOTH

   - Ir al menú principal.
   - Entrar a “Herramientas”.
   - Entrar a *Bluetooth*. En el menú que se despliega:
     - Activar al dispositivo *Bluetooth*, si no está activado.
     - Escoger la pestaña “Dispositivos vinculados”.
   - Seleccionar a la PC con la que se está trabajando, y, en “Opciones”, seleccionar “Conectar”.

   EN NOKIA PC SUITE

   - Posterior a la instalación del Nokia PC Suite, se conecta al dispositivo a la PC con el cable de comunicación incluido con el mismo.

3. Instalar la aplicación “*LectBill.sisx*”:

   EN UN DISPOSITIVO CON CONECTIVIDAD BLUETOOTH

   <u>En la PC:</u>

[10] La presente guía de instalación está hecha para los dispositivos móviles con los que se probaron el prototipo, descritos en la sección 3. Para otros dispositivos móviles, se pueden emular estos pasos, si es que no son exactamente los mismos.

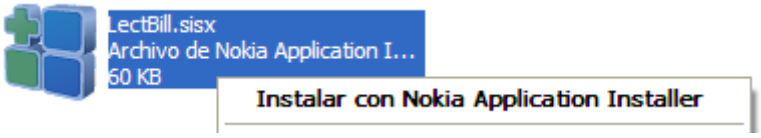
- Click derecho en el archivo "*LectBill.sisx*".
- Seleccionar: "Enviar a" → "*Bluetooth*" → Nombre del dispositivo móvil (*e.g*. Nokia N73).

En el teléfono móvil:

- Se recibe como mensaje la aplicación enviada. Aceptar el mismo.
- Se pide permiso para instalar la aplicación → Aceptar.
- Se pregunta el lugar de la instalación → Seleccionar la memoria del teléfono.
- Aceptar toda petición hecha.
- La aplicación queda instalada en la memoria del teléfono.

EN NOKIA PC SUITE

Posterior a la instalación del Nokia PC Suite, al dar cick derecho sobre el archivo, se tiene la opción de instalación con las herramientas de este *software*:



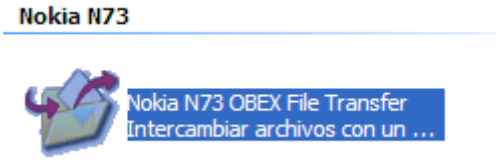
- Se siguen los pasos en el teléfono móvil detallados para el dispositivo con *Bluetooth*.

4. Copiar la carpeta "*Lectbill*" con los archivos necesarios para la aplicación en el dispositivo móvil, en la ubicación del teléfono:

C:\Data

EN UN DISPOSITIVO CON CONECTIVIDAD BLUETOOTH

- Doble *click* en la pestaña de *Bluetooth* en la barra de herramientas.
- Seleccionar, de los dispositivos conectados a la PC, el dispositivo móvil conectado con *Bluetooth* (*e.g.* Nokia N73):



- Aceptar cada solicitud de *Bluetooth*, hecha por la PC, en el teléfono móvil, donde requiere confirmación frecuente.
- Entrar a ***"C:\" → "Data"*** y copiar en esta ubicación a la carpeta "*Lectbill*" con todo su contenido.

EN NOKIA PC SUITE

- Posterior a la instalación del Nokia PC Suite, mediante el ícono de esta aplicación, se puede acceder a los directorios del teléfono. Acceder a la misma ubicación detallada arriba y copiar la carpeta "*Lectbill*" con todo su contenido.

Al culminar estos pasos, la aplicación ha quedado completamente instalada en el dispositivo móvil.

6. CARACTERÍSTICAS DE TRABAJO

Un paso más, previo a la utilización del sistema, es el de conocer las fundamentales características de trabajo del mismo, con el objetivo de brindar al sistema las condiciones propicias para que pueda funcionar al máximo de su capacidad. Dentro de estas, se hallan las funcionalidades del sistema, así como las limitaciones. Las características principales de trabajo de *LectBill*, son las siguientes:

1. *LectBill* es un sistema que se inscribe en la rama de *visión por computador*, que utiliza la cámara de vídeo del dispositivo móvil en el cual se halla instalado, para la realización de un proceso de reconocimiento de la imagen del billete que se encuentra filmando. Cuadro a cuadro de la filmación, el sistema procesa la imagen que tiene delante, y se mantiene inactivo hasta el reconocimiento de un billete en su foco, momento en el cual, implementa un algoritmo para el establecimiento de su denominación.

2. La imagen que filma la cámara deberá ser una de características aceptables para el correcto desempeño del método. Estas son, principalmente:
    - La imagen que se está filmando debe tener un fondo oscuro, que contraste con la tonalidad de los billetes, que es generalmente clara. Esto es muy importante, ya que el sistema no podrá ubicar al billete de no cumplirse este requerimiento y, por tanto, no podrá funcionar. La figura muestra una imagen tomada con un fondo ideal para el funcionamiento:

Figura 1. Billete en fondo contrastante ideal

    - Para el reconocimiento, el sistema no requiere la imagen del billete completo, sino solamente la de un extremo del mismo (cualquiera de los cuatro extremos), por lo que, de cualquier forma que el usuario tome el billete para su filmación, debe ubicar en foco la parte extrema derecha del billete. El ejemplo de lo que aquí se considera un extremo se muestra en la siguiente figura:

Figura 2. Billete remarcando la zona que se debe procurar filmar

    - La imagen del billete al momento de su reconocimiento debe ser, en la medida de lo posible, plana; por esto, es conveniente aplanar al billete antes

del uso del sistema. La siguiente imagen muestra un ejemplo de billete que el sistema no podrá reconocer.

Figura 3. Billete sin aplanar correctamente

- El algoritmo de reconocimiento presenta problemas con condiciones particulares de iluminación, por lo que es importante que el lugar en el que se haga funcionar al sistema tenga condiciones de luz uniformes; esto es, que presente pocas probabilidades de introducir sombras encima de la imagen del billete que está siendo filmada. De preferencia, la aplicación deberá ser utilizada de forma "*indoor*" (puertas adentro).

Figura 4. Billete en condiciones de iluminación uniforme

Figura 5. Billete en malas condiciones de iluminación no uniforme

3. El sistema implementado *NO discrimina billetes de dólares norteamericanos falsos de verdaderos*; por lo que no se debe utilizar con esta expectativa.

4. Como prerrequisito final, el usuario de este sistema, deberá tener un proceso de entrenamiento de su utilización, asistido por la persona que se halla en la lectura del presente manual. Este proceso será detallado en la siguiente sección.

7. MÉTODO DE UTILIZACIÓN [11]

El sistema implementado, como todo sistema dirigido a personas no videntes, presenta sus específicas capacidades, competencias y limitaciones por lo que, previa la utilización del mismo, es necesario un entrenamiento previo al usuario, en el cual adquiera las destrezas básicas para lograr el máximo desempeño del sistema en uso. De esta forma es necesario, en primer lugar, el establecimiento de un método de utilización recomendado por los autores del sistema, con los pasos básicos para la utilización del mismo.

1. Situarse físicamente en un espacio que brinde al sistema las condiciones necesarias para su mejor funcionamiento. El usuario requerirá la mayor cantidad posible de información previa que se tenga acerca de este sitio. En general, las condiciones que debe proveer el lugar seleccionado, son las siguientes:

    - De preferencia, ser un sitio "*indoor*"; es decir, dentro de alguna edificación, o "bajo techo".

[11] Esta sección presenta el conjunto de instrucciones de utilización del sistema, dirigidas directamente al usuario con discapacidad visual, y se anexa también en versión audible, para este fin.

Figura 6. Ejemplo de condiciones indoor

2. Presentar un alto grado de uniformidad en la iluminación; es decir, que asegure la poca existencia de sombras sobre el billete antes de la utilización del método.

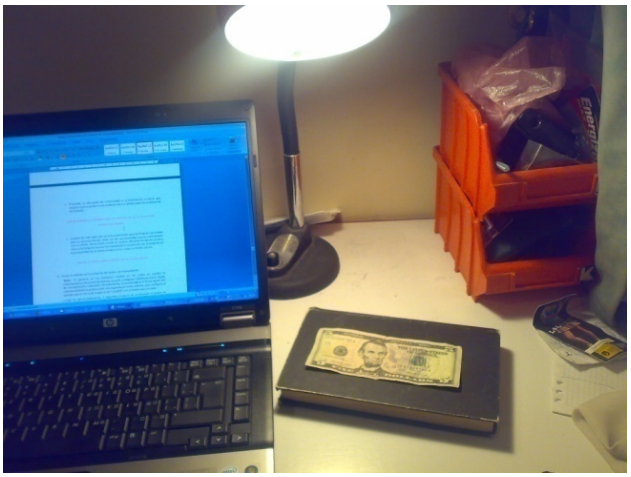


Figura 7. Billete en condiciones ideales de iluminación indoor

3. La parte de este lugar que servirá como fondo para la filmación del billete para su reconocimiento, debe ser de una tonalidad oscura, contrastante con un billete. Dicho fondo puede ser el piso si la filmación va a ser hecha de pie, brindando así la profundidad de la cámara hasta el piso, como un fondo oscuro. En este caso, el piso no debe ser de un tono claro.

Figura 8. Billete sobre fondo de piso oscuro

4. Iniciar el sistema con la pulsación del botón correspondiente.

   Nota.- *En general, en los teléfonos móviles en los cuales es posible la implementación del presente sistema, se pude configurar el botón de inicio rápido de una aplicación cualquiera. Una segunda persona, que sea vidente, puede configurar al aparato para el inicio de la aplicación "LectBill" con el botón más cercano.*

5. Luego de, aproximadamente, 5 segundos después de presionado el botón de inicio, se escuchará un mensaje en los parlantes del dispositivo, que dice: "Bienvenido"; entonces, el sistema ha iniciado.

6. Tomar al billete del cual se pretende reconocer su denominación y, en la medida de lo posible, aplanarlo para que, antes del reconocimiento, no presente curvaturas, ya que el sistema requiere una imagen plana. Seleccionar el extremo del billete más plano. Este será el extremo a ser filmado.

7. Tomar con la mano izquierda al billete por el extremo opuesto al seleccionado en el paso anterior, procurando mantener el billete suspendido en el aire de una forma plana.

Figura 9. Billete suspendido de una forma ideal

8. Con la mano derecha, palpar el lente de la cámara de vídeo del dispositivo móvil, a fin de conocer el foco de filmación. Luego, alinear de la mejor manera al dispositivo móvil con el billete, enfrentando cara a cara el lente de la cámara con la parte del billete seleccionada para su filmación.

Figura 10. Manera correcta manera de alinear al billete con la cámara

9. Una vez que se ha asegurado la alineación del dispositivo con el billete, y manteniendo la misma, elevar al dispositivo aproximadamente 15 cm. hacia arriba del billete, hasta escuchar un pitido intermitente, que indica que se está detectando al billete. Luego de aproximadamente 5 pitidos, se escuchará un mensaje desde los parlantes del dispositivo con la denominación del billete que se tiene en reconocimiento. En caso de que no se escuche el pitido, y dado que se hayan cumplido todos los pasos anteriores, significa que la imagen del billete necesaria para el reconocimiento no está en el foco de la cámara de una forma recta y se debe seguir intentando el enfoque del mismo, repitiendo los pasos 6 a 9

.

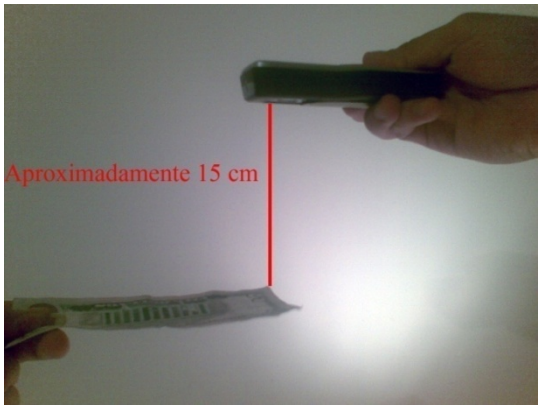


Figura 10. Alineación y distancia correcta del billete a la cámara para su reconocimiento

10. Una vez finalizado el reconocimiento con éxito, seguir con el próximo reconocimiento requerido o terminar la aplicación con el botón de salida (puede ser configurado como el mismo de inicio del sistema). Luego de, aproximadamente, 2 segundos después de presionado el botón de salida, se escuchará un mensaje en los parlantes del dispositivo, que dice: “Adios”, para el conocimiento del usuario de que el sistema se ha cerrado.

8. SOLUCIÓN DE PROBLEMAS

¿Qué hacer si se ha pasado mucho tiempo sin lograr resultados?

La primera posibilidad es que no se esté captando al billete, por una de las razones planteadas en la sección 6. Lo recomendable en este caso es seguir las recomendaciones de la sección 7: cambiar el sitio en el que se usa la aplicación, seguir el método de ubicación correctamente, etc.

La segunda posibilidad es que se la aplicación no esté en funcionamiento, porque se detuvo o no se inició correctamente. En este caso, se debe tratar de reiniciar la aplicación saliendo de la misma con el botón de salida y volviendo a entrar con el botón de inicio.

¿Qué hacer si se escuchan más de 10 pitidos en forma intermitente, sin finalización, sin que el billete sea reconocido?

Usualmente, se debe retirar el billete del foco de la cámara una vez que se han escuchado alrededor de 5 pitidos, para que el mensaje de audio con la denominación tenga espacio en memoria y sea reproducida.

¿Qué hacer si se escucha el mensaje *"Pruebe con otro lado"* más de 3 veces?

La primera opción será efectivamente probar otro lado u otro extremo del billete, tomando en cuenta todas las consideraciones del método de utilización en la sección 6.

Otra posibilidad es que el sistema no se haya iniciado correctamente o se haya detenido, por lo que se debe reiniciar la aplicación e intentar reconocer de nuevo el billete. De hecho, si se tienen varios mensajes de esta naturaleza sin que se haya reconocido al billete, la aplicación misma emitirá un mensaje que dice: "Reinicie la aplicación".

Si a pesar de esto se continúa recibiendo este mensaje más de 3 veces implica que el fondo sobre el cual el billete está siendo filmado no es lo suficientemente contrastante o existe iluminación inadecuada. Se puede intentar cambiar de sitio para que el fondo y las condiciones de iluminación sean favorables.

9. REALIZADORES

El presente sistema fue desarrollado como proyecto de grado, respectivamente, de las carreras de Ingeniería Electrónica en Telecomunicaciones y en Automatización y Control del departamento de Eléctrica y Electrónica de la Escuela Politécnica del Ejército, por:

Felipe Grijalva – 092761367 / felipegrijalva@gmail.com

Juan Carlos Rodríguez – 095204793 / jnkrodro@gmail.com

Durante el período comprendido entre los meses de marzo y octubre del año 2009, en San Francisco de Quito - República del Ecuador.

A4

CÓDIGOS FUENTE DE LOS PROGRAMAS EN MATLAB

A4.1 Programa principal de entrenamiento

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%PROGRAMA DE ENTRENAMIENTO PARA "EIGENIMAGES"%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Recibe:
% - Un conjunto de M imágenes de muestra de los billetes en la carpeta
% 'Muestras', para formar el set de entrenamiento.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Produce los siguientes datos para carga en el teléfono celular:
% - Los píxeles de la imagen media del set de muestras normalizadas.
% - El conjunto de vectores propios, 'u' o 'eigenimages'.
% - Sus respectivos valores propios.
% - Los vectores de pesos resultantes de la proyección de cada imagen
% muestra
% en el "espacio de billetes": 'omega'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Exhibe:
% - Un subconjunto de las imágenes de muestra, así como las mismas
% imágenes normalizadas.
% - La imagen media del set de muestras normalizadas.
% - Un subconjunto de las 'eigenimages' reconstruidas de los
% vectores propios, 'u', obtenidos.
% - Las distancias de Malahanobis entre las muestras de testeo y cada
% 'eigenimage' del conjunto de vectores propios, 'u'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%ESPE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-Felipe Grijalva
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%-J.C. Rodríguez
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Septiembre, 2009
clear all;
close all;
imtool close all;
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%DECLARACIÓN DE CONSTANTES NECESARIAS%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nb=6;%Número de billetes a reconocerse.
nl=4;%Número usual de lados de cada billete a reconocerse (cada esquina).
m=90;%Alto o número de filas del espacio original de imágenes.
n=60;%Ancho o número de columnas del espacio original de imágenes.
N=m*n;%Número de píxeles de cada imagen del espacio original de imágenes.
direc='./Muestras/'; %Directorio de donde se leerá las muestras
                     %de entrenamiento.
ext='.pgm'; %Extensión de los archivos de entrenamiento.
M=length(dir(strcat(direc,'*',ext)));%Número de muestras del set de
                                     %entrenamiento
                                     %(es igual al numero de imágenes en
                                     %la carpeta de 'direc'.)
mb=7;%Número máximo de tipos de muestras a tomarse por lado de cada
     %billete. Forma mb subocnjuntos de tipos de muestra.
plotset=1;  %Determina uno de los mb subconjuntos de tipos de imágenes
            %(rectas, inclinadas, etc.); qle que se graficará.
            %Toma valores de 1 hasta mb.
K=24;%Número final de bases que se desea usar en el PCA.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%CREACIÓN DEL ESPACIO ORIGINAL DE IMÁGENES EN R^N%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%CREACIÓN DE MATRIZ DE IMÁGENES EN EL ESPACIO ORIGINAL DEL SET
%-Lee las imágenes de muestra que formarán en set de entrenamiento en una
% matriz cuyas columnas son las imágenes originales en dimensión Nx1.
%-Obtiene una estructura de información sobre cada una de las
%muestras obtenidas en gammainfo.
[gamma gammainfo]=LeerImagenes(direc,ext,N,M,m,n,nb,mb,nl);

%EXHIBICIÓN DEL SET DE IMÁGENES ORIGINALES
%Imprime en una figura el subconjunto de muestras indicado por plotset
%de las imágenes originales:
MostrarImg(gammainfo,direc,ext,M,nb,plotset,nl);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%PCA (PRINCIPAL COMPONENT ANALYSIS: PASOS%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%NORMALIZACIÓN DE LAS IMÁGENES DEL ESPACIO ORIGINAL
%Nuevas media y desviación estándar seleccionadas:
umedia=128;
ustd=128;   %De preferencia, se usan múltiplos de exponentes de 2
            %y solo usar desplazamientos en Symbian C++.
%Normaliza:
gamma=double(gamma);%Manejo de datos tipo double.
gamma=((gamma-repmat(mean(gamma),N,1))*ustd)./repmat(std(gamma),N,1) + umedia;

%EXHIBICIÓN DEL SET DE IMÁGENES NORMALIZADAS
%Imprime en una figura el subconjunto de muestras indicado por plotset
%de las imágenes, ahora normalizadas:
MostrarImgNormalizadas(gamma,direc,ext,M,m,n,nb,mb,nl,plotset);

%RESTA DE LA MEDIA DE CADA IMAGEN ORIGINAL
%Siguiendo el método de PCA:
immedia=mean(gamma');%Vector de medias de cada dimensión.
immedia=immedia';%Imagen media de cada imagen original, en vector Nx1.
fi=double(gamma)-repmat(immedia,1,M);%Set de imágenes restadas la
                                     %media == SET DE DATOS DEFINITIVO.

%Exhibición de la imagen media.
figure(3);
img_media=reshape(immedia,n,m);%Redimensiona a matrices de m x n píxeles.
img_media=img_media';%Por la naturaleza de MATLAB.
imshow(uint8(img_media));%Exhibe la imagen media.
title('IMAGEN MEDIA DEL SET NORMALIZADO');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%VALORES Y VECTORES PROPIOS DE LA MATRIZ DE COVARIANZA DE LOS DATOS%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%MATRIZ DE COVARIANZA DEL SET DE DATOS NORMALIZADOS
%El método trata de diagonalizar la matriz de covarianza del set de datos.
%Se empieza estableciendo la matriz equivalente L, M x M,para la obtención
%de los valores propios de la mtriz de covarianza original, C, que serán
%los mismos que los de la matriz  L.
A=fi;%El nombre dado por la teoría para la matriz del set datos.
L=A'*A;%Nueva matriz MxM de la cual se obtendrán los vectores propios.
[vL lambdaL ]=eig(L);%Vectores y valores propios de L

%ORDENAMIENTO DE LOS VECTORES PROPIOS DE L DE ACUERDO A SUS VALORES (ORD. DESC.)
%Se ordenan los vectores propios obtenidos de acuerdo al orden descendente
%de sus respectivos valores propios.
[lambdaL indices]=sort(sum(lambdaL),'descend');%Ordena desc. a los valores
                                               %propios.
vL=vL(:,indices);%Vectores propios ordenados de mayor a menor.
```

```
%VECTORES Y VALORES PROPIOS DE LA MATRIZ C, DEL ESPACIO ORIGINAL
% = Eigenvectores = Eigenimages.
u=A*vL;%Matriz de vectores propios de C como columnas.
lambdaC=(lambdaL/max(lambdaL))*10000;%Escalamiento de los valores propios
%de L, que son los mismos de C,para su envío al celular.
u=u./repmat(sqrt(sum(u.^2)),N,1);%NORMALIZACIÓN de los vect. propios.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%REDUCCIÓN DEL ESPACIO DE LOS COMPONENTES PRINCIPALES A R ^K%%%%%%%%%%
%%%%%%%%%%%%%%%%%(A sus K componentes más importantes)%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u=u(:,1:K);%Los K vectores propios más representativos, correspondientes a
lambdaC=lambdaC(1:K);%los K valores propios más representativos.
%Con esto, se han obtenido las bases de un espacio nuevo al cual se
%proyectarán las imágenes originales. Estas componentes principales,
%son: R ^ K. Bases=Vectores propios=Componentes principales=Eigenimages.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%MUESTRA Y ALMACENAMIENTO DE 'EIGEN - FACES'%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(4);
for i=1:size(u,2)%A través de las columnas de la matriz (M)
    %Almacenamiento:
    ruta=strcat('./Lectbill/Eigen/',int2str(i),'.txt');%Crea la ruta de
                                                       %cada eigenimage.
    fid=fopen(ruta,'w');%Crea y abre el archivo para escritura.
    fprintf(fid,'%d\n',int32(u(:,i)*(2^20)));%Realiza la grabación de las
                                             %eigenimages escaladas x 2^20.
    fclose(fid);%Cierra el archivo manipulado.
    %Muestra:
    imagen=reshape(u(:,i),n,m);%Redimensiona a matrices m x n.
    imagen=imagen';%Necesario por la naturaleza de MATLAB.
    imagen=histeq(imagen,255);%Reforma el histograma a 256 tonos (Hace imágenes.)
    subplot(nb,nl,i);%Selecciona una porción de la figura para plotear c/imagen.
    imshow(imagen);%Plotea la imagen en turno
    if i==2 %Para centrar el título
         title('EIGENIMAGES (VECTORES PROPIOS)','fontsize',18)
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%PESOS (COEFICIENTES) DE CADA IMAGEN DEL SET ORIGINAL EN EL NUEVO ESPACIO%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
omega=(u')*fi;%Dado que u', que contiene a los vectores propios por filas,
%es la nueva matriz de cambio de base del espacio original, fi, a uno nuevo
%, omega, de columnas iguales al número de muestras.

%Matriz de covarianza de este nuevo espacio:
Comega=omega*omega'
%Debería ser diagonal, para covarianza entre las dimensiones de 0, lo que
%da poca redundancia y máxima importancia y comprueba al PCA.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%GRABACION DE LA IMAGEN MEDIA, LOS VALORES PROPIOS Y LOS PESOS%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%CARGA DE LA IMAGEN MEDIA Y LOS K VALORES PROPIOS EN EL ARCHIVO carga1.txt:
vapropios=lambdaC';%Los valores propios en una columna.
envio=[int32(immedia);int32(vapropios)];%Vector de envío de la imagen media
                                        %y los K valores propios.
fid=fopen('./Lectbill/carga1.txt','w');%Crea y abre archivo para escritura.
fprintf(fid,'%d\n',envio);%Carga al archivo.
fclose(fid);%Cierra el archivo.
```

```
%CARGA DE LOS K PESOS DE LAS M MUESTRAS EN EL NUEVO ESPACIO,
%EN LA CARPETA OMEGA:
for i=1:M%A través de las muestras que se tienen.
    ruta=strcat('./Lectbill/Omega/',int2str(i),'.txt');%Crea la ruta de
                                                        %cada peso.
    fid=fopen(ruta,'w');%Crea y abre el archivo para escritura.
    fprintf(fid,'%d\n',int32(omega(:,i)));%Realiza la grabación de
                                          %cada peso.
    fclose(fid);%Cierra el archivo manipulado.
end
```

A4.2 Función de lectura del conjunto original de imágenes

```
function [gamma gammainfo]=LeerImagenes(direc,ext,N,M,m,n,nb,mb,nl)
%Función de lectura del set original de imágenes.
%ENTRADAS:
% direc: Directorio de donde se leera las muestras de entrenamiento.
%           Ejemplo: dir='./Muestras/'. Notar que se incluye '/' al final
%           de la ruta del directorio.
% ext: Extensión de los archivos de entrenamiento.
%           Ejemplo: ext='.pgm'
% N: Número de píxeles de cada imagen  original.
% M: Número de muestras del set de entrenamiento.
% m: Alto o número de filas de las imágenes originales.
% n: Ancho o número de columnas de las imágenes originales.
% nb: Número de billetes a reconocerse.
% mb: Número de muestras por lado de cada billete.
% nl: Número usual de lados por billete (4).
%SALIDAS
% - gamma: Matriz de NxM que contiene las imagenes como vectores Nx1,
% por columnas.
% - gammainfo: Matriz de estrcuturas con información de las imágenes
% obtenidas de la función etiquetar.m.

%LECTURA DE LOS NOMBRES DE LOS ARCHIVOS (Tienen información sobre su
%procedencia).
nombre=dir(strcat(direc,'*',ext));%Lee los nombres de los archivos
                                  %en una estructura.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%El ordenamiento de las imágenes se realizará así:
%- Grupos de columnas en orden ascendente de denominación. En c/u de estos:
%- Grupos de columnas en orden ascendente según el extremos del billete
% tomado. En c/u de estos:
%- Grupos de columnas en orden ascendente según el número del tipo.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%VARIABLES NECESARIAS PARA EL ORDENAMIENTO:
cont=zeros(nb,nl,mb);%Contador de las muestras obtenidas,
                     %para su ordenamiento: nb x nl x mb.
for i=1:M%Recorre todos los archivos, contando.
    etaux=etiquetar(nombre(i).name);%Obtiene etiquetas de cada muestra.
                                    %Ver la función
    cont(etaux.ordinal,etaux.lado,etaux.nset)=cont(etaux.ordinal,etaux.lado,etaux.nset)+1;
                                  %Cuenta en una matriz, según los parámetros.
end
contden=sum(sum(cont,2),3);%Vector que contiene el número de billetes
                           %de cada denominación.
vecden=cumsum(contden);%Vector de offsets de ubicación de imágenes
                       %en función de su denominación.
contlad=sum(cont,3);%Vectores contadores del número de muestras de
                    %cada fila, para cada denominación.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%Bucle de ordenamiento del arreglo en la obtención de gamma%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gamma=zeros(N,M);%Matriz de imágenes del set de entrenamiento.
for i=1:M %Va recorriendo todos los archivos.
    %Apertura de cada imagen:
    ruta=strcat(direc,nombre(i).name);%Crea la ruta para abrir cada imagen.
    imagen=imread(ruta);%Lee la imagen en ese archivo.
    [h1 w1]=size(imagen);%Obtiene las dimensiones de la imagen.
    if isequal([h1 w1],[m n])==0   imagen=imresize(imagen,[m n]); end
                    %Redimensiona si la imagen no tiene dimensiones m por n.
    et=etiquetar(nombre(i).name);%Etiqueta el billete:Ver función etiquetar
    %Establecimiento de la ubicación de la imagen en la matriz gamma:
    if et.ordinal==1 %Si se está en la primera denominación ($1).
        offsden=0;%Para evitar el acceso a índice 0.
    else
        offsden=vecden(et.ordinal-1);%Offset en denominación para
                                     %la ubicación.
    end
    veclad=contlad(et.ordinal,:);%Obtiene el vector de contadores de filas
                                  %para esta denominación actual.
    veclad=cumsum(veclad);%Vector de offsets de ubicación de imágenes en
                          %función de cada lado de la denominación en turno.
    if et.lado==1 %Si se tiene la muestra del primer lado.
        offslad=0;%Para evitar el acceso a índice 0.
    else
        offslad=veclad(et.lado-1);%Offset en lado.
    end
    imagen=imagen';%Para la toma por filas de la imagen.
    columna=offsden+offslad+et.nset;%Columna de la matriz gamma en la cual
              %debe ubicarse esta imagen, según el ordenamiento planteado.
    gamma(:,columna)=imagen(:);%Se va creando la matriz gamma de imágenes
                               %en el espacio original, como vectores Nx1.
    gammainfo(columna)=et;%Va creando la estructura con información útil
                          %sobre la  presente imagen.
end
```

A4.3 Función de etiquetado de imágenes

```
function etstruct=etiquetar(nombre)
%Función de etiquetado de imágenes.
%Etiqueta un billete en función de su nombre y guarda la información dentro
%de una estructura(etstruct) con la siguiente informacion:

tipo=nombre(1);%Determina de qué tipo es la imagen que se tiene:
% 'b'.- Una imagen muestra para el set de entrenamiento.
% 't'.- Una imagen muestra para el testeo de la aplicación.
% 'f'.- Una imagen muestra que no es un billete.

%DETERMINACIÓN DE LA DENOMINACIÓN Y EL LADO DE CADA BILLETE
if (nombre(3)=='.') %Denominación de un guarismo.
    deno=str2num(nombre(2));%Denominación: $1,5,10,20,50 ó 100.
    lado=str2num(nombre(4));%Lado o extremo: 1,2,3 ó 4.
    if (nombre(7)=='.')%Solo de un guarismo, todos los campos.
        nset=str2num(nombre(6));%Número de set al que pertenece: 1 a 7.
    else
        nset=str2num(nombre(6:7));
    end
else%Denominación de dos guarismos.
    deno=str2num(nombre(2:3));
    lado=str2num(nombre(5));
    if (nombre(8)=='.')
        nset=str2num(nombre(7));%nset de un guarismo.
    else
        nset=str2num(nombre(7:8));%nset de dos guarismos.
    end
end
```

```
if (nombre(2:4)=='100')%La única denominación de tres guarismos
    deno=100;
    lado=str2num(nombre(6));
    if (nombre(9)=='.')
        nset=str2num(nombre(8));
    else
        nset=str2num(nombre(8:9));
    end
end

%ASIGNACIÓN DE UN ORDINAL PARA DETERMINAR EL GRUPO SEGÚN CADA DENOMINACIÓN
switch deno
    case 1
        ord=1;%ordinal del billete, $1 == 1
    case 5
        ord=2;%ordinal del billete, $5 == 2
    case 10
        ord=3;%ordinal del billete, $10 == 3
    case 20
        ord=4;%ordinal del billete, $20 == 4
    case 50
        ord=5;%ordinal del billete, $50 == 5
    case 100
        ord=6;%ordinal del billete, $100 == 6
    otherwise
        ord=0;
end
if (nombre(1)=='f') %Si es una muestra falsa, que no es billete
    deno=0;
    ord=0;
end
%Creación de la estructura:
 etstruct =
struct('tipo',tipo,'ordinal',ord,'denominacion',deno,'lado',lado,'nset',nset);
end
```

A4.4 Función de exhibición de las imágenes del conjunto original

```
function MostrarImg(gammainfo,direc,ext,M,nb,plotset,nl)
%Función de exhibición de las imágenes del set original.
figure(1);
for i=1:M %Va recorriendo todos los archivos .
    %Gráfico del set de imágenes originales, gamma.
    et=gammainfo(i);%Obtiene la información de la imagen ordenada en curso.

ruta=strcat(direc,et.tipo,num2str(et.denominacion),'.',num2str(et.lado),'.',num2s
tr(plotset),ext);
         %Crea la ruta para abrir cada imagen (con el plotset determinado).
    imagen=imread(ruta);%Lee la imagen en el archivo de esa ruta.
    if (et.nset==plotset) %Dibuja solo el subconjunto de muestra indicado
                          %por plotset(solo rectos, solo inclinados, etc.),
                          %de cada lado.
        subplot(nb,nl,nl*(et.ordinal-1)+et.lado);
                          %Selecciona una porción de la figura para plotear
                          %c/imagen del set determinado.
        imshow(imagen);%Plotea la imagen en turno.
    end
    if i==2 %Para centrar el título
       title('ESPACIO ORIGINAL DE IMÁGENES','fontsize',18);
    end
end
```

A4.5 Función de exhibición de las imágenes normalizadas del conjunto original

```
function MostrarImgNormalizadas(gamma_norm,direc,ext,M,m,n,nb,mb,nl,plotset)
%Función de exhibición de las imágenes normalizadas del set original.

%Variables necesarias para el establecimiento de la muestra apropiada:
nombre=dir(strcat(direc,'*',ext));
cont=zeros(nb,nl,mb);%Contador de las muestras obtenidas,
                                %para su ordenamiento: nb x nl x mb.
for i=1:M%Recorre todos los archivos, contando.
    etaux=etiquetar(nombre(i).name);%Obtiene etiquetas de cada muestra.

cont(etaux.ordinal,etaux.lado,etaux.nset)=cont(etaux.ordinal,etaux.lado,etaux.nse
t)+1;%Cuenta en una matriz, según los parámetros.
end

contden=sum(sum(cont,2),3);%Vector que contiene el número de billetes de
                           %cada denominación.
vecden=cumsum(contden);%Vector de offsets de ubicación de imágenes en
                       %función de su denominación.
contlad=sum(cont,3);%Vectores contadores del número de muestras de cada
                    %fila  para cada denominación.

%Muestra el set de imágenes normalizadas:
figure(2);
for i=1:M %Va recorriendo todos los archivos .
    %Establecimiento de la imagen normalizada a dibujarse.
    et=etiquetar(nombre(i).name);%Etiqueta el billete.
    if et.ordinal==1 %Si se está en la primera denominación ($1).
        offsden=0;%Para evitar el acceso a índice 0.
    else
        offsden=vecden(et.ordinal-1);%Offset en denominación para la
                                     %ubicación.
    end
    veclad=contlad(et.ordinal,:);%Obtiene el vector de contadores de filas
                                 %para esta denominación actual.
    veclad=cumsum(veclad);%Vector de offsets de ubicación de imágenes en
                          %función de cada lado de la denominación en turno.
    if et.lado==1 %Si se tiene la muestra del primer lado.
        offslad=0;%Para evitar el acceso a índice 0.
    else
        offslad=veclad(et.lado-1);%Offset en lado.
    end
    ind=offsden+offslad+plotset;%Columna de la matriz gamma_norm a
                    %ubicarse, tomándose siempre la muestra de tipo plotset.

    %Muestra de la imagen del subconjunto seleccionado por setplot:
    imagen=reshape(uint8(gamma_norm(:,ind)),n,m);%En MATLAB, a la inversa.
    imagen=imagen';%Idem.
    imagen=histeq(imagen,255);%Reforma el histograma a 256 tonos.
    subplot(nb,nl,nl*(et.ordinal-1)+et.lado);
                %Selecciona una porción de la figura para plotear c/imagen.
    imshow(imagen);%Plotea la imagen en turno.
    if i==2 %Para centrar el título.
         title('IMÁGENES NORMALIZADAS','fontsize',18)
    end
end
```

A4.6 Programa para la determinación experimental del umbral de la distancia de Mahalanobis

```
%%%%%%%PROGRAMA PARA LA DETERMINACIÓN DEL UMBRAL DE LAS DISTANCIAS%%%%%%%%%
%Recibe: - Un conjunto de P imágenes de testeO, en la carpeta: 'Pruebas'.
%Calcula: - El umbral para la distancia de Malahanobis a implementarse.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Felipe Grijalva, J.C. Rodríguez, Sept 2009
%NOTA: Ejecutar después de haber ejecutado el programa de entrenamiento.
%%%%%%%%%%%%%REDEFINICIÓN (solo)DE CONSTANTES NECESARIAS%%%%%%%%%%%%%%%%%%%%%
direc='./Pruebas/'; %Directorio de donde se leerá las muestras de testeo.
P=length(dir(strcat(direc,'*',ext)));%Número de muestras del set de testeo
                    %(es igual al numero de imágenes en la carpeta de 'direc').
%%%%%%%%%%%%%%%%%%%%DETERMINACIÓN DEL MEJOR UMBRAL%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Lectura de los archivos
nombre=dir(strcat(direc,'*',ext));%Lee los nombres de los archivos en una struct.
umbrales=[];%Guardará los umbrales en turno probados.
exitos=[];%Guardará el número de éxitos en cada umbral.
for umbral=1000:1000:40000 %Se prueba para todos estos umbrales.
    aciertos=0;
    errores=0;%Reseteo de contadores
    for i=1:P %Va recorriendo todos los archivos de Pruebas.
        et=etiquetar(nombre(i).name);%Etiqueta la muestra en turnoen l struct et.
        %APERTURA DE CADA IMAGEN DE PRUEBA:
        ruta=strcat(direc,nombre(i).name);%Crea la ruta para abrir c/imagen
        imagen=imread(ruta);%Lee la imagen en ese archivo.
        imagen=double(imagen);%Datos tipo double.
        [h1 w1]=size(imagen);%Obtiene las dimensiones de la imagen.
        if isequal([h1 w1],[m n])==0   imagen=imresize(imagen,[m n]); end
                  %Redimensiona si la imagen no tiene dimensiones m por n.
        imagen=imagen';%Toma por filas.
        vector=double(imagen(:));%Imagen de entrada como vector de 1xN.
        %ESTABLECIMIENTO DE LA DENOMINACIÓN DEL BILLETE, SEGÚN EL UMBRAL
        EN TURNO, Aplicando PCA a cada imagen de prueba:
        %NORMALIZACIÓN:
        me=mean(vector);%Media.
        st=std(vector);%Desviación estándar.
        vector=(vector-me)*ustd/st+umedia;%Normaliza.
        vector = vector-immedia; %RESTA DE LA IMAGEN MEDIA:
        %PESOS DE LA IMAGEN EN EL NUEVO SET DE EIGEN VECTORES:
        omegain=(u')*vector;%Pesos de la imagen de entrada actual.
        % DISTANCIA DE MAHALANOBIS:
        maha=sum(((repmat(omegain,1,M)-omega).^2)./repmat(lambdaC',1,M));
        [distmin indmin]=min(maha);
        %RECONOCIMIENTO DEL BILLETE:
        grupo=floor(indmin/(nl*mb))+(rem(indmin,nl*mb)>0);%Dice la denominación.
        if et.ordinal ~= 0 %Si es que sí es un billete, (lo indica la et.)
            %Se determina si se ha reconocido bien su denominación Y está
            %por debajo del umbral en curso:
            if distmin < umbral & et.ordinal == grupo
                aciertos=aciertos+1; %Es un acierto.
            else
                errores=errores+1;%Es un error(de denominación o de umbral)
            end
        else %No es un billete, es una imagen falsa, entonces:
            %Debería estar la distancia por encima del umbral en curso:
            if distmin > umbral
                aciertos=aciertos+1;%Es un acierto del método.
            else
                errores=errores+1;%Es un error (falso positivo).
            end
        end
    end
    paciertos=(aciertos*100)/(aciertos+errores);%Obtiene el % de aciertos.
    %Conformación de los vectores:
    umbrales=[umbrales; umbral];%Va acumulando el umbral en curso.
    exitos=[exitos; paciertos];%Va ac. el % de exitos para este umbral.
end
```

```
%Exhibición de los umbrales
figure(8);
stem(umbrales,exitos);grid;title('Porcentaje de éxitos - Umbral');
%Ordenación descendente para averiguar los mejores umbrales:
[exitosord indices]=sort(exitos,'descend');
umbralesord=umbrales(indices);
%Exhibición de los mejores umbrales:
resultado=[umbralesord exitosord]
```

A5

CÓDIGOS DE LOS PROGRAMAS EN *SYMBIAN* C++

A5.1 Código de los archivos de cabecera más importantes

A5.1.1 Archivo de cabecera de la clase `CCameraAppBaseContainer`

```
/*
 * Copyright © 2008 Nokia Corporation.
 */

#ifndef __CAMERAAPP_BASE_CONTAINER_H__
#define __CAMERAAPP_BASE_CONTAINER_H__

#include <coecntrl.h>
#include <akntabobserver.h>
#include <aknutils.h>

// Constantes
const TInt KCuenta=20;//Número de veces máxima de plazo para que se tenga éxito
//en el reconocimiento del billete (Ver ChkTiempo()).
const TInt KCueMay=4;//Número máximo de veces que sucede que se vence
//el anterior plazo, para pedir al usuario una reiniciación de la aplicación.
const TInt KUmbralCuenta=3;//Número de veces que se debe tener éxito en el
//reconocimiento del billete para considerarlo acierto (Ver Aviso()).

//Declaraciones forward
class CFbsBitmap;
class CWsBitmap;
class CFbsBitGc;
class CFbsBitmapDevice;
class CCameraAppController;
class CAknNavigationDecorator;
//Clases declaradas por los autores:
class CImagen;//Clase para el PDI del frame de entrada
class CPca;//Clase para la aplicación del PCA a la muestra
class CToneAdapter;//Clase para reproducción de tonos
class CSoundPlayer;//Calse para reproducción de audio multimedia
class CSoundAux;//Idem

/**
 * Container control class.
 */
class CCameraAppBaseContainer : public CCoeControl
    {
    public: // Constructors and destructor
        **
         * Symbian OS second phase constructor.
         *
         * @param aRect Frame rectangle for container.
         */
        void ConstructL(const TRect& aRect);

        /**
         * Constructor.
         *
         * @param aController Camera controller
         */
        CCameraAppBaseContainer(CCameraAppController*& aController);

        /**
         * Destructor.
         */
        virtual ~CCameraAppBaseContainer();
```

```
public: // New functions
    /**
     * Draws the bitmap image immediately.
     *
     * @param aBitmap bitmap to be displayed
     */
    void DrawImageNow(CFbsBitmap& aBitmap);

    /**
     * Redraws the offscreen bitmap.
     * EN ESTE ÁMBITO SE PROGRAMA LA APLICACIÓN LectBill
     */
    void ReDrawOffScreenBitmap();


public:  // Functions from base classes
    void SetController(CCameraAppController*& aController);

    /**
     * Shows an error message on the screen.
     *
     * @param aMsg the message to be displayed
     */
    void ShowErrorMessage( const TDesC &aMsg );

    /**
     * From CCoeControl.
     * Handles changes to the control's resources.
     *
     * @param aType A message UID value
     */
    void HandleResourceChange( TInt aType );

protected: // New protected functions
    /**
     * Draws a bitmap onto the real screen.
     */
    void DrawImage(CWindowGc& aGc, const TRect& aRect) const;


private: // New function
    /**
     *  Creates the offscreen bitmap.
     */
    void CreateOffScreenBitmapL();

private: // Functions from base classes

    /**
     * From CoeControl.
     * Called when this control's rect changes.
     */
    void SizeChanged();

    /**
     * From CoeControl.
     * Identify the help context so that the framework can look up
     * the corresponding help topic.
     *
     * @param aContext The help context
     */
    void GetHelpContext(TCoeHelpContext& aContext) const;
```

```
        /**
         * From CCoeControl.
         */
        void Draw(const TRect& aRect) const;

    public:
        TBool iOffScreenBitmapCreated;
        //Funciones declaradas por los autores:
        void Aviso();//Indica el billete que se está observando en pantalla
        void Audio();//Emite el sonido .wav del billete reconocido
        void DelaySeg(TInt aTiempo);//Da un retardo de aTiempo segundos
        void ChkTiempo();//Chequea si la espera del usuario es demasiado larga

    protected: //Data
       CCameraAppController*&      iController;
        CFbsBitmap*                 iBitmap;
        CWsBitmap*                  iOffScreenBitmap;
        CFbsBitGc*                  iFbsBitGc;
        CFbsBitmapDevice*           iBmpDevice;

        //Objetos declarados por los autores:
       CImagen* img;//Objeto para el PDI del frame de entrada
       CPca* eigenimage;//Objeto para el PCA de la muestra de entrada
       CToneAdapter* iToneAdapter;//Objeto para reproducción de tonos
       CSoundPlayer* iSoundPlayer;//Objetos para reproducción de audios generales
       CSoundAux* iAdios;//Otro objeto de audio para mensaje de despedida
       //Variables declaradas por los autores:
        TInt iDenominacion;//Denominación hallada por PCA
        TInt iContador;//Contador para medir el tiempo de demora del método
        TInt iContadorMay;//Contador mayor: Las veces que debe haber plazo
        //máximo para pedir reiniciación.
        TInt iContAci;//Cuenta de aciertos
        TInt iAnterior;//Valor anterior de la denom. para la cuenta
        TInt iActual;//Valor actual de la denominación
       //Extras
        TInt iMUID; //machine UID, identifica el modelo de celular
   };

#endif // __CAMERAAPP_BASE_CONTAINER_H__
```

A5.1.2 Archivo de cabecera de la clase `CImagen`

```
/*
 * imagen.h
 *
 *  Created on: 24-ago-2009
 *      Author: J.C. Rodríguez
 */

#ifndef IMAGEN_H_
#define IMAGEN_H_

#include <coecntrl.h>
#include <akntabobserver.h>
#include <aknutils.h>

//Constantes
const TInt KUmbral = 150;//Umbral para binarización global
const TInt KAnchoEscalado=60;//Width escalado
const TInt KAltoEscalado=90;//Height escalado, para el corte de in al PCA
const TReal KProporcion=(TReal)KAltoEscalado/(TReal)KAnchoEscalado;
                                //Proporción alto a ancho para escalado
const TInt KDec=100; //Factor que se multiplica para simular decimales.
const TReal KFactorDecEscalado=(TReal)KDec/(TReal)KAltoEscalado;//Para el corte
const TUint KMediaGrisAlta=128;//Máxima media aceptable de la imagen gris
                               //para el método PCA.
const TInt KDvcEstBaja=10;//Mínima desviación estándar para la discriminación
                          //de los cortes de billetes.
```

```
const TInt KDvcEstAlta=50;//Máxima desviación estándar para la discriminación
                          //de los cortes de billetes.
const TInt KPorcen=10;//% mínimo de píxeles blancos en la imagen umbralizada
class CPca;//Declaración forward de la clase que será amiga de CImagen

class CImagen: public CBase
    {
    private:
       /*
        * VARIABLES MIEMBRO DE LA CLASE IMAGEN
        */
       //Imágenes a partir de la original RGB:
       CFbsBitmap* iGris;//Versión en escala de grises de 256 tonos
       CFbsBitmap* iGrisEsc;//Imagen gris escalada a 1/4 de su dimensión
                            //para preprocesamiento.
       CFbsBitmap* iBin;//Versión binarizada de la gris escalada
       CFbsBitmap* iEro;//Versión erosionada de la imagen binaria
       CFbsBitmap* iEscalada; //Imagen corte de muestra escalada a
                              //KAnchoEscaladoxKAnchoAltoEscalado
       CFbsBitmap* iIntegral; //Integral de la imagen gris escalada

//Características comunes de la imagen patrón:
       Int iAncho;//Width
       TInt iAlto;//Height
       TInt iAnchoEsc;//Width de la imagen gris escalada
       TInt iAltoEsc;//Height de la imagen gris escalada
       TInt iNum_pixeles;//Número de píxeles
       TInt offset_8vecino[8];//Offset para los 8vecinos cercanos
       TInt iMedia;//Contendrá la media de la imagen gris corte
       TInt iDvcEst;//Contendrá la desviación estándar de la img. gris corte
       TInt iKTHor;//Umbral para bordes horizontales del perfil del billete
       TInt iKTVer;//Umbral para borde vertical del perfil del billete
       //Direcciones de memoria de las imágenes:
       TUint8* iPinigris;//Puntero inicial de la imagen gris
       TUint8* iPfingris;//Puntero final de la imagen gris
       TUint8* iPinigrisesc;//Ptro. inicial de la versión gris escalada
       TUint8* iPfingrisesc;//Ptro. final de la versión gris escalada
       TUint8* iPinibin;//Puntero inicial de la imagen binaria
       TUint8* iPfinbin;//Puntero final de la imagen binaria
       TUint8* iPiniero;//Puntero inicial de la imagen erosionada
       TUint8* iPfinero;//Puntero final de la imagen erosionada
       TUint8* iPiniesc;//Puntero inicial de la imagen corte escalada
       TUint8* iPfinesc;//Puntero final de la imagen corte escalada
       TUint32* iPiniint;//Puntero inicial de la integral de la imagen
       TUint32* iPfinint;//Puntero final de la integral de la imagen
       //Índices y constantes comunes:
       TUint8* Pt;//Puntero para recorrer una imagen cualquiera
       TInt iFila_ver1;//Índice de la fila superior del billete
       TInt iFila_ver2;//Índice de la fila inferior del bilelte
       TInt iColumna;//Índice de la columna extrema derecha del billete
       TInt i;
       TInt j;//Contadores de propósito general
       TInt iPixBin;//Número de píxeles binarios en el corte umbralizado
    public:
       /*
        * FUNCIONES MIEMBRO DE LA CLASE IMAGEN
        */
       friend class CPca;//Declaración de amistad de la clase CPca
       //Constructores y destructores:
       CImagen();//Constructor (Para inicializar variables)
       ~CImagen();//Destructor (Para liberar memoria)
       void ConstructL(TSize size);//Constructor de 2º fase
                                   //(Para inicializar memoria)
        //Funciones de procesamiento de la imagen:
        void RgbAGris(CFbsBitmap &aImagenRGB);//Obtiene la imagen gris
        void EscalaGris();//Obtiene la img. gris escalada para preprocesamiento
        void Integra();//Obtiene la integral de la imagen gris escalada
        void UmbralAdap();//Binariza la imagen gris escalada
```

```
        void Erosiona();//Erosiona la img. binaria para reconocer borde externo
        void Estadist();//Obtiene las características estadísticas de la img.
        void Perfil();//Obtiene los vértices en donde se halla el billete
        void Cortar();//Corta la porción de interés del billete (y escala)
        //Funciones para discriminar el ingreso de la img. a funciones de PCA:
        /*
         * ChkEst1():
         * Retorna TRUE si la media de la imagen gris escalada tiene
         * probabilidad de contener un billete (MediaGris < MediaGrisMax).
         */
        TUint ChkEst1(){return
(((*iPfinint)/(iAnchoEsc*iAltoEsc))<KMediaGrisAlta)  ? 1:0;}
        /*
         * ChkEst2():
         * Retorna TRUE si los vértices para el corte son los adecuados y
         * lógicos para una muestra de billete.
         */
        TUint ChkEst2(){return((iColumna > (iAncho/4))
                        &&  (iColumna < iAncho )
                        &&  (iFila_ver2 < iAlto)
                        &&  (iFila_ver2 > (iAlto/2))
                        &&  (iFila_ver1 <= (iAlto/2))
                        &&  ((iFila_ver2 - iFila_ver1) >= iAlto/2)
                        &&  (iFila_ver1 > 0)) ? 1:0;}
        TUint ChkEst3();//Retorna TRUE si el % de píxeles blancos dentro de la
        //imagen binaria contenedora del corte es el adecuado Y si la
        //desviación estándar del corte al PCA es el adecuado.
    };

#endif /* IMAGEN_H_ */
```

A5.1.3 Archivo de cabecera de la clase `CPca`

```
/*
 * pca.h
 *
 *  Created on: 01-sep-2009
 *      Author: J.C. Rodríguez
 */

#ifndef PCA_H_
#define PCA_H_

#include <coecntrl.h>
#include <akntabobserver.h>
#include <aknutils.h>
#include "imagen.h"

//Constantes:
const TInt KAncho=60;//Width.- Ancho de la imagen de entrada
const TInt KAlto=90;//Height.- Alto de la imagen de entrada
const TInt KNumPixeles = KAncho * KAlto;//Número de píxeles
const TInt KNumBases=24;//Número de bases a usarse seleccionadas
                        //(Número de 'Eigenimages en la carpeta Eigen).
const TInt KNumMuestras=168;//Número total de muestras seleccionadas que forma
//el set de imágenes (Número de vectores omega en la carpeta Omega)
const TInt KMuestrasLado=7;//Número máximo de muestras a tomarse por cada lado
//de cada billete (Determina el # de subconjuntos de muestras)
const TInt KDeci=100; //Factor que se multiplica para simular decimales
                      //(Usado para operaciones).
const TInt KMedia=128;//Media para la normalización de la imagen
                      //(Debe ser igual a la usada en el ENTREMAMIENTO).
const TInt KDvcEst=128;//Desviación estándar para la normalización de la imagen
                       //(Debe ser igual a la usada en el ENTREMAMIENTO).
const TUint KUmbralDistM = 8000;//Umbral mínimo de la distancia de Mahalanobis
                                //Obtenido del programa de ENTRENAMIENTO).
const TUint32 KTUint32Max = 2100000000;//Máximo entero unsigned de 32 bits
                                       //(Usado para operaciones).
```

```
class CImagen;//Declaración forward de la clase a la q se accederá con amistad

class CPca: public CBase
    {
    private:
       /*
        * VARIABLES MIEMBRO DE LA CLASE PCA (Principal Component Analysis)
        */
       //Eigenimages a usarse en el método:
       CFbsBitmap* iMuestra;//Entrada, corte del billete, normalizada
       CFbsBitmap* iMedia;//Imagen media de las muestras
       CFbsBitmap* iEigenVectores;//Vector de Eigenvectores bases: u
       CFbsBitmap* iPesos;//Vector de pesos de las muestras
       CFbsBitmap* iPesosMuestra; //Vector de pesos de la img. de entrada
       CFbsBitmap* iEigenValores; //Valores propios de loes Eigenvectores
       //Direcciones de memoria de los objetos Bitmaps:
       TUint32* iPtIniMuestra;//Puntero inicial de la imagen de entrada
       TUint32* iPtFinMuestra;//Puntero final de la imagen de entrada
       TUint32* iPtIniMedia;//Puntero inicial de la imagen media
       TUint32* iPtFinMedia;//Puntero final de la imagen media
       TUint32* iPtIniEv;//Puntero inicial de los Eigenvectores
       TUint32* iPtFinEv;//Puntero final de los Eigenvectores
       TUint32* iPtIniPe;//Puntero inicial de los pesos de las muestras
       TUint32* iPtFinPe;//Puntero final de los pesos de las muestras
       TUint32* iPtIniPm;//Puntero inicial de los pesos de la entrada
       TUint32* iPtFinPm;//Puntero final de los pesos de la entrada
       TUint32* iPtIniVap;//Puntero inicial de los valores propios
       TUint32* iPtFinVap;//Puntero final de los valores propios
       //Otras variables:
       TUint32 DistM[KNumMuestras];//Resultados, distancias de Mahalanobis
                                   //de la muestra a cada peso.
       //Índices y constantes comunes:
       TUint32* Pt;//Puntero para recorrer una imagen cualquiera
       TInt i;
       TInt j;//Índices enteros de peopósito general

    public:
       /*
        * FUNCIONES MIEMBRO DE LA CLASE IMAGEN
        */
        //Constructores y destructores:
       CPca();//Constructor (Para inicializar variables)
        ~CPca();//Destructor (Para liberar memoria)
        void ConstructL();//Constructor de 2º fase (Para inicializar memoria)
        //Funciones:
        void Carga();//Carga Imagen media y Valores Propios desde la carpeta
                    //Lectbill, en el constructor.
        void Carga_Eigen(TInt k);//Carga los vectores propios (Eigenvectores)
                                   //uno por uno, desde la carpeta Eigen.
        void Carga_Omega(TInt k);//Carga los pesos de las muestras (Omegas)
                                   //uno por uno, desde la carpeta Omega.
        void Normaliza(CImagen &im);//Normaliza a una media y dvc. est.
                                      //la imagen de entrada.
        void ProPunto();//Obtiene los pesos de la muestra a través del producto
                        //punto con los eigenvectores.
        TUint DistMaha();//Obtiene el vector de distancias de Mahalanobis a
                        //cada vector de pesos del set de muestras.
    };


#endif /* PCA_H_ */
```

A5.2 Constructores y destructor de la clase `CCameraAppBaseContainer`

```
/*
-----------------------------------------------------------------------------
Constructor. Initializes member variables.
-----------------------------------------------------------------------------
*/
CCameraAppBaseContainer::CCameraAppBaseContainer(
  CCameraAppController*& aController )
: iController( aController ),
  iBitmap( 0 ),
  iOffScreenBitmap( 0 ),
  iFbsBitGc( 0 ),
  iBmpDevice( 0 )
        {
        iController->SetAppContainer( this );
        //INICIO
        //Inicializacion de variables:
        iContador=0;
        iContadorMay=0;
        iContAci=0;
        iActual=0;
        iAnterior=0;//Contadores de tiempo y éxitos del método
        //FIN
        }

/*
-----------------------------------------------------------------------------
Symbian OS 2nd phase constructor: CONSTRUCTOR DE SEGUNDA FASE
-----------------------------------------------------------------------------
*/
void CCameraAppBaseContainer::ConstructL(const TRect& aRect)
      {
      CreateWindowL();
      SetRect(aRect);
      ActivateL();
      // This should be called after the windows has been activated
      CreateOffScreenBitmapL();

      //INICIO
      //CREACIONES DE LOS OBJETOS A USARSE:
      //Objeto Imagen:
      img = new (ELeave) CImagen;//Asigna memoria en RAM
      //TInt mUID;//Machine UID
      HAL::Get(HAL::EMachineUid, iMUID);//Obtiene el modelo del celular en iMUID
      TSize sizevf(266,200);//Crea tamaño del viewfinder:266x200 (Del Nokia N73)
      if (iMUID==0x20000604)//Si el modelo es Nokia E65:
            {
            sizevf.iWidth=240;
            sizevf.iHeight=180;//Reconfigura al tamaño viewfinder del Nokia E65
            }
      img->ConstructL(sizevf);//Construye el obj. del tamaño del viewfinder
                              //arriba determinado.
      //Objeto PCA:
      eigenimage= new (ELeave) CPca;//Asigna memoria en RAM
      eigenimage->ConstructL();//Ejecuta el constructor de 2º fase
      //Objeto para reproducción de tono:
      iToneAdapter = CToneAdapter::NewL(666,TTimeIntervalMicroSeconds(100000));
                                          //Crea un objeto tono de 666 Hz. y 100 ms.
      //Objetos para reproducción de los audios:
iSoundPlayer=CSoundPlayer::NewL(_L("C:\\Data\\Lectbill\\Audio\\bienvenido.mp3"));
      //Construye el objeto tipo sonido de propósito general, reproduciendo
     //un sonido de bienvenida a la aplicación.
iAdios=CSoundAux::NewL(_L("C:\\Data\\Lectbill\\Audio\\adios.mp3"));
            //Crea un objeto solo para el sonido de despedida.
      //FIN
      }
```

```
/*
-------------------------------------------------------------------------------
Destructor. Frees reserved resources.
-------------------------------------------------------------------------------
*/
CCameraAppBaseContainer::~CCameraAppBaseContainer()
    {
    //Sonido de despedida de la aplicación:
    iAdios->Play();//Objeto especialmente creado para el sonido de despedida
    DelaySeg(2);//Retardo de 2 s.
    //Liberación de recursos utilizados:
       delete iOffScreenBitmap;
    delete iFbsBitGc;
    delete iBmpDevice;
    //Objetos definidos por los autores:
    delete img;//Borra el objeto tipo CImage
    delete eigenimage;//Borra el objeto tipo CPca
    delete iToneAdapter;//Borra el objeto del tono
    delete iSoundPlayer;//Borra el objeto de audio
    delete iAdios;//Borra el objeto de sonido de despedida
    }
```

A5.3 Constructores y destructor de la clase `CImagen`

```
/*
-------------------------------------------------------------------------------
Constructor. Inicializa variables miembro que no requieren memoria dinámica.
-------------------------------------------------------------------------------
*/
CImagen::CImagen()
: iGris(0),
iBin(0),
iEscalada(0),
iAncho(0),
iAlto(0),
iAnchoEsc(0),
iAltoEsc(0),
iNum_pixeles(0),
iMedia(0),
iDvcEst(0),
iKTHor(0),
iKTVer(0),
iPinigris(0),
iPfingris(0),
iPinibin(0),
iPfinbin(0),
Pt(0),
iFila_ver1(0),
iFila_ver2(0),
iColumna(0),
i(0),
j(0)

            {
            //Declaraciones adicionales sin req. de memoria
            }
```

```
/*
-------------------------------------------------------------------------------
Symbian OS : Constructor de 2º fase. Todo aquello que requiera de la memoria.
-------------------------------------------------------------------------------
*/
void CImagen::ConstructL(TSize size)
	{
	//Características comunes:
	iAncho=size.iWidth;//Número de columnas
	iAlto=size.iHeight;//Número de filas
	iNum_pixeles = iAncho * iAlto;//Número de píxeles
	//Imagen gris:
	iGris = new (ELeave) CFbsBitmap();//Memoria para la img. gris
	iGris->Create(size, EGray256);//Creación del objeto Bitmap gris
	//Imagen gris escalada:
	iAnchoEsc=iAncho/2;//Ancho para la imagen gris escalada
	iAltoEsc=iAlto/2;//Alto para la imagen gris escalada
	iGrisEsc = new (ELeave) CFbsBitmap();//Memoria para la img. gris
	iGrisEsc->Create(TSize(iAnchoEsc,iAltoEsc), EGray256);
					//Creación del objeto Bitmap gris
	//Imagen binaria:
	iBin = new (ELeave) CFbsBitmap();//Memoria para la img. binaria
	iBin->Create(TSize(iAnchoEsc,iAltoEsc), EGray256);
					//Creación del objeto Bitmap binaria
	//Imagen erosionada:
	iEro = new (ELeave) CFbsBitmap();//Memoria para la img. erosionada
	iEro->Create(TSize(iAnchoEsc,iAltoEsc), EGray256);
					//Creación del objeto Bitmap erosionada
	//Imagen escalada:
	iEscalada = new (ELeave) CFbsBitmap();//Memoria para la img. escalada
	iEscalada->Create(TSize(KAnchoEscalado,KAltoEscalado), EGray256);
					//Creación del objeto Bitmap escalada
	//Integral de la imagen gris escalada:
	iIntegral = new (ELeave) CFbsBitmap();//Memoria para la integral
	iIntegral->Create(TSize(iAnchoEsc,iAltoEsc), EColor16MU);
					//Creación del objeto Bitmap integral

	//Punteros de acceso a memoria:
	iPinigris = (TUint8*)iGris->DataAddress();
			//Dirección del 1º píxel de la img. gris
	iPfingris = iPinigris + iNum_pixeles - 1;
			//Dirección del píxel final de la img. gris
	iPinigrisesc = (TUint8*)iGrisEsc->DataAddress();
			//Dirección del 1º píxel de la img. gris esc.
	iPfingrisesc = iPinigrisesc + iAltoEsc*iAnchoEsc- 1;
			//Dirección del píxel final de la img. gris esc.
	iPinibin = (TUint8*)iBin->DataAddress();
			//Dirección del 1º píxel de la img. bin
	iPfinbin = iPinibin + iAnchoEsc*iAltoEsc - 1;
			//Dirección del píxel final de la img. bin
	iPiniero = (TUint8*)iEro->DataAddress();
			//Dirección del 1º píxel de la img. eros.
	iPfinero = iPiniero + iAnchoEsc*iAltoEsc - 1;
			//Dirección del píxel final de la img. eros.
	iPiniesc=(TUint8*)iEscalada->DataAddress();
			//Dirección del píxel final de la img. esc.
	iPfinesc = iPiniesc + KAnchoEscalado * KAltoEscalado - 1;
	iPiniint = (TUint32*)iIntegral->DataAddress();
			//Dirección del 1º píxel de la integral
	iPfinint = iPiniint + iAnchoEsc*iAltoEsc - 1;
			//Dirección del último píxel de la integral

	//Inicialización con ceros de los contenedores de las imágenes:
	for (Pt=iPiniero;Pt<=iPfinero;++Pt)*Pt=0;

	//Otras inicializaciones:
	//Umbrales para el reconocimiento del perfil externo del billete:
	iKTHor=iAnchoEsc/4;//25% del ancho de la imagen gris escalada
```

```
        iKTVer=iAltoEsc/2;//50% del alto de la imagen gris escalara
        //Offsets de los 8 vecinos cercanos del píxel p:
        /*
         *                        abc                             012
         *                        dpe          ===                3p4
         *                        fgh                             567
         */
        offset_8vecino[0] = -iAnchoEsc-1;//a
        offset_8vecino[1] = -iAnchoEsc;//b
        offset_8vecino[2] = -iAnchoEsc+1;//c
        offset_8vecino[3] = -1;//d
        offset_8vecino[4] = 1;//e
        offset_8vecino[5] = iAnchoEsc-1;//f
        offset_8vecino[6] = iAnchoEsc;//g
        offset_8vecino[7] = iAnchoEsc+1;//h
´
        }
/*
-------------------------------------------------------------------------------
Destructor. Libera recursos.
-------------------------------------------------------------------------------
*/
CImagen::~CImagen()
    {
    delete iGris;
    delete iGrisEsc;
    delete iEro;
    delete iIntegral;
    delete iBin;
    delete iEscalada;
    }
```

A5.4 Constructores y destructor de la clase `CPca`

```
/*
-------------------------------------------------------------------------------
Constructor. Inicializa variables miembro que no requieren memoria dinámica.
-------------------------------------------------------------------------------
*/
CPca::CPca()
:iMuestra(0),
iMedia(0),
iEigenVectores(0),
iPesos(0),
iPesosMuestra(0),
iEigenValores(0),
iPtIniMuestra(0),
iPtFinMuestra(0),
iPtIniMedia(0),
iPtFinMedia(0),
iPtIniEv(0),
iPtFinEv(0),
iPtIniPe(0),
iPtFinPe(0),
iPtIniPm(0),
iPtFinPm(0),
iPtIniVap(0),
iPtFinVap(0),
Pt(0),
i(0),
j(0)
{
            //Declaraciones adicionales sin req. de memoria

}
```

```
/*
-------------------------------------------------------------------------------
Symbian OS : Constructor de 2º fase. Todo aquello que requiera de la memoria.
-------------------------------------------------------------------------------
*/
void CPca::ConstructL()
	{
	//Imagen de entrada:
	iMuestra = new (ELeave) CFbsBitmap();//Memoria para la img. muestra,
					//copia del corte obtenido en CImagen.
	iMuestra->Create(TSize(KAncho,KAlto), EColor16MU);
					//Creación del objeto Bitmap
	//Imagen media:
	iMedia = new (ELeave) CFbsBitmap();//Memoria para la img. media
	iMedia->Create(TSize(KAncho,KAlto), EColor16MU);//Creación del objeto
	//Vector de vectores propios:
	iEigenVectores = new (ELeave) CFbsBitmap();//Memoria para el vector
					//lleno de los eigenvectores.
	iEigenVectores->Create(TSize(KNumBases*KNumPixeles,1), EColor16MU);
			//Creación del objeto Bitmap: KNumPixeles x KNumBases
	//Vector de pesos de las muestras del set original:
	iPesos = new (ELeave) CFbsBitmap();//Memoria para el vector de pesos
	iPesos->Create(TSize(KNumBases*KNumMuestras,1), EColor16MU);
	//Vector de pesos de la imagen del celular:
	iPesosMuestra = new (ELeave) CFbsBitmap();
					//Memoria para el vector de pesos muestra
	iPesosMuestra->Create(TSize(KNumBases,1), EColor16MU);
//Vector de valores propios:
	iEigenValores = new (ELeave) CFbsBitmap();
				//Memoria para el vector de valores propios
	iEigenValores->Create(TSize(KNumBases,1), EColor16MU);
	//Direcciones de memoria de los objetos Bitmaps:
	//Imagen muestra
	iPtIniMuestra = iMuestra->DataAddress();
	iPtFinMuestra = iPtIniMuestra + KAncho*KAlto - 1;
	//Imagen media
	iPtIniMedia = iMedia->DataAddress();
	iPtFinMedia = iPtIniMedia + KAncho*KAlto - 1;
	//Eigen - vectores
	iPtIniEv = iEigenVectores->DataAddress();
	iPtFinEv = iPtIniEv + KNumBases*KAncho*KAlto - 1;
	//Pesos
	iPtIniPe = iPesos->DataAddress();
	iPtFinPe = iPtIniPe + KNumBases*KNumMuestras - 1;
	//Pesos muestra
	iPtIniPm = iPesosMuestra->DataAddress();
	iPtFinPm = iPtIniPm + KNumBases*KNumMuestras - 1;
	//Eigen - valores
	iPtIniVap = iEigenValores->DataAddress();
	iPtFinVap = iPtIniVap + KNumBases - 1;
	/*
	 * CARGA DESDE EL ARCHIVO ANEXO DE LOS ELEMENTOS NECESARIOS
	 * PARA LA OPERACIÓN DE RECONOCIMIENTO.
	 */
	Carga();//Carga de la imagen media y de los valores propios
	//Carga de los archivos de los eigen - vectores:
	for(j=0;j<KNumBases;j++)
		{
		Carga_Eigen(j);//Carga cada eigen-vector
		}
	//Carga de los pesos de las muestras Omega:
	for(j=0;j<KNumMuestras;j++)
		{
		Carga_Omega(j);//Carga cada vector de peso de c/muestra del set
		}
	}
```

```
/*
-------------------------------------------------------------------------------
Destructor. Libera recursos.
-------------------------------------------------------------------------------
*/
CPca::~CPca()
    {
    delete iMuestra;
    delete iMedia;
    delete iEigenVectores;
    delete iPesos;
    delete iPesosMuestra;
    delete iEigenValores;
    }
```

A5.5 Funciones de carga de la información de los archivos necesarios

```
/*
-------------------------------------------------------------------------------
Carga. Carga la imagen media y los valores propios.
-------------------------------------------------------------------------------
*/
void CPca::Carga()
      {
      TInt Size;//Guardará el tamaño del archivo en Bytes
      //Construcción de la ruta con buffer <256>:
      _LIT( KChivo,"C:\\Data\\Lectbill\\carga1.txt");//Ruta
      RFs fsSession;//Objeto de la sesión para manejo de archivos
      //Manejo de errores:
      User::LeaveIfError(fsSession.Connect());
      CleanupClosePushL(fsSession);
      //Objeto archivo:
      RFile file;//Objeto de tipo archivo
      TInt err = file.Open(fsSession,KChivo,EFileRead);//Abre el archivo
      if (err==KErrNotFound) // Si el archivo no existe, lo crea
            err=file.Create(fsSession,KChivo,EFileStream);
      if(err==KErrNone)//Si no hay problemas y halla el archivo
            {
            //Necesario para almacenar la lectura del archivo:
            file.Size(Size);//Obtiene el tamaño del archivo en Bytes
            HBufC8* buffer = HBufC8::NewLC(Size);//Descriptor heap a guardar los
                                                 //datos del archivo.
            TPtr8 ptr= buffer->Des();
            //Puntero que apunta a los datos del archivo, del tamaño size
            //Para cada dato individual del archivo:
            const TUint8 *pun;//Puntero de cada dato individual
            pun=buffer->Ptr();//Obtención del puntero
            file.Read(ptr);//LECTURA DE todo EL ARCHIVO
            /*
             * OBTENCIÓN DE LA IMAGEN MEDIA Y LOS VALORES PROPIOS
             */
            TInt32 vNum;
            //Dato con signo de 32 bits para guardar lo q se va leyendo
            TLex8 vLex(pun);//Objeto TLex de 8 bits a partir del puntero q leyó
            TUint32* PtMedia = iPtIniMedia;//Recorre la imagen media
            TUint32* PtValores = iPtIniVap;
                  //Recorre el vector de valores propios
            //Obtención de la imagen media:
            for(i=0;i<KNumPixeles;i++)
                  {
                  vLex.Val(vNum);//Lee c/dato del archivo como Tint32 en vNum
                  *PtMedia = vNum;//Almacena en la imagen media
                  vLex.Inc();//Incrementa el puntero de lectura del archivo
                  PtMedia++;//Incrementa el puntero de grabación
                  }
```

```
		//Obtención de los valores propios:
		for(i=0;i<KNumBases;i++)
			{
			vLex.Val(vNum);//Lee c/dato como Tint32 en vNum
			*PtValores = vNum;//Almacena en el vector de eigen-valores
			vLex.Inc();//Incrementa el puntero de lectura del archivo
			PtValores++;//Incrementa el puntero de grabación
			}
		file.Close();//Cierre del archivo
		CleanupStack::PopAndDestroy(buffer);//Manejo de errores
		}
	fsSession.Close();//Cierre de sesión
	CleanupStack::PopAndDestroy(&fsSession); // destruye la sesión
	}
/*
-------------------------------------------------------------------------------
Carga_Eigen. Carga los vectores propios ('Eigenimages'), uno por uno.
-------------------------------------------------------------------------------
*/
void CPca::Carga_Eigen(TInt k)
	{
	TInt Size;//Guardará el tamaño del archivo en Bytes
	//Construcción de la ruta con buffer:
	_LIT(KChivo,"C:\\Data\\Lectbill\\Eigen\\%d.txt");//Ruta en KCHivo
	TBuf<40> mibuf;//Nuevo buffer modificado con la longitud de la ruta
	mibuf.Format(KChivo,k+1);//Ubica el k+1 en turno en el %d de la ruta
	RFs fsSession;//Objeto de la sesión para manejo de archivos
	User::LeaveIfError(fsSession.Connect());//Manejo de errores
	CleanupClosePushL(fsSession);
	RFile file;//Objeto, //Objeto archivo
	TInt err = file.Open(fsSession,mibuf,EFileRead);//Abre el archivo en mibuf
	if (err==KErrNotFound) // Si el archivo no existe, lo crea
		err=file.Create(fsSession,mibuf,EFileStream);
	if(err==KErrNone)//Si no hay problemas y halla el archivo
		{
		//Necesario para almacenar la lectura del archivo:
		file.Size(Size);//Obtiene el tamaño del archivo en Bytes
		HBufC8* buffer = HBufC8::NewLC(Size);//Descriptor heap a guardar los
                                                     //datos del archivo.
		TPtr8 ptr= buffer->Des();//Puntero q apunta a los datos del archivo,
                                         //del tamaño size.
		//Para cada dato individual del archivo
		const TUint8 *pun;//Puntero de cada dato individual
		pun=buffer->Ptr();//Obtención del puntero inicial en pun
		file.Read(ptr);//LECTURA DE todo EL ARCHIVO
		/*
		 * OBTENCIÓN DEL EIGEN VECTOR EN TURNO
		 */
		TInt32 vNum;
		//Dato con signo de 32 bits para guardar lo q se va leyendo
		TLex8 vLex(pun);//Objeto TLex de 8 bits a partir del puntero q leyó
		TUint32* PtEigen;//Recorre el vector de Eigen - vectores
		//Obtención del Eigen vector:
		PtEigen = iPtIniEv + k*KNumPixeles;//Se incrementa el puntero
		//de acuerdo a k, el índice del vect. en turno.
		for(i=0;i<KNumPixeles;i++)//Cada vector propio es una imagen
			{
			vLex.Val(vNum);//Lee c/dato como Tint32 en vNum
			*PtEigen = vNum;//Almacena en el vector de eigenvectores
			vLex.Inc();//Incrementa el puntero de lectura del archivo
			PtEigen++;//Incrementa el puntero de grabación
			}
		file.Close();//Cierre del archivo
		CleanupStack::PopAndDestroy(buffer);//Manejo de errores
		}
	fsSession.Close();//Cierre de sesión
	CleanupStack::PopAndDestroy(&fsSession); // destruye la sesión
	}
```

```
/*
-----------------------------------------------------------------------------
Carga_Omega. Carga los pesos de las imágenes del conjunto de muestras,
uno por uno.
-----------------------------------------------------------------------------
*/
void CPca::Carga_Omega(TInt k)
	{
	TInt Size;//Guardará el tamaño del archivo en Bytes
	//Construcción de la ruta con buffer:
	_LIT(KChivo,"C:\\Data\\Lectbill\\Omega\\%d.txt");//Ruta
	TBuf<40> mibuf;//Nuevo buffer modificado con la longitud de la ruta
	mibuf.Format(KChivo,k+1);//Ubica el k+1 en turno en el %d de la ruta
	RFs fsSession;//Objeto de la sesión para manejo de archivos
	//Manejo de errores:
	User::LeaveIfError(fsSession.Connect());
	CleanupClosePushL(fsSession);
	//Objeto archivo:
	RFile file;//Objeto
	TInt err = file.Open(fsSession,mibuf,EFileRead);//Abre el archivo
	if (err==KErrNotFound) // Si el archivo no existe, lo crea
		err=file.Create(fsSession,mibuf,EFileStream);
	if(err==KErrNone)//Si no hay problemas y halla el archivo
		{
		//Necesario para almacenar la lectura del archivo
		file.Size(Size);//Obtiene el tamaño del archivo en Bytes
		HBufC8* buffer = HBufC8::NewLC(Size);//Descriptor heap a guardar
		                                     //los datos del archivo
		TPtr8 ptr= buffer->Des();
		//Puntero que apunta a los datos del archivo, del tamaño size.
		//Para cada dato individual del archivo:
		const TUint8 *pun;//Puntero de cada dato individual
		pun=buffer->Ptr();//Obtención del puntero
		file.Read(ptr);//LECTURA DE todo EL ARCHIVO
		/*
		 * OBTENCIÓN DEL VECTOR DE PESO EN TURNO
		 */
		TInt32 vNum;
		//Dato con signo de 32 bits para guardar lo q se va leyendo
		TLex8 vLex(pun);//Objeto TLex de 8 bits a partir del puntero q leyó
		TUint32* PtPesos;//Recorre el vector de pesos de las muestras
		//Obtención del vector de pesos:
		PtPesos = iPtIniPe + k*KNumBases;//Se incrementa el puntero de
		//acuerdo a k, el índice del vector de pesos en turno.
		for(i=0;i<KNumBases;i++)//Cada peso es de longitud del número de bases
			{
			vLex.Val(vNum);//Lee c/dato como Tint32 en vNum
			*PtPesos = vNum;//Almacena en el vector de vectores de pesos
			vLex.Inc();//Incrementa el puntero de lectura del archivo
			PtPesos++;//Incrementa el puntero de grabación
			}
		file.Close();//Cierre del archivo
		CleanupStack::PopAndDestroy(buffer);//Manejo de errores
		}
	fsSession.Close();//Cierre de sesión
	CleanupStack::PopAndDestroy(&fsSession); // destruye la sesión
	}
```

A5.6 Programa principal en `DrawImageNow(CFbsBitmap& aBitmap)`

```
/*
-----------------------------------------------------------------------------
Gets from Controller the image to draw and calls DrawNow()
-----------------------------------------------------------------------------
*/
void CCameraAppBaseContainer::DrawImageNow(CFbsBitmap& aBitmap)
	{
	if (iOffScreenBitmapCreated && IsActivated() && IsReadyToDraw())
		{
		iBitmap = &aBitmap;
		/*
	**************************************************************
	**********************PROYECTO DE GRADO************************
	***********Reconocimiento de la denominación de billetes**********
	**************Dirigido a personas no videntes*******************
	*************************************************************ESPE
	*********************************************************Grijalva F.
	********************************************************Rodríguez J.C.
	*******************************************************Octubre, 2009
	*****************************************************Quito, Ecuador
*/
	//INICIO
	iDenominacion=0;//Por defecto,la denominación del billete es 0
	img->RgbAGris(aBitmap);//Crea la imagen gris
	img->EscalaGris();//Escala la imagen gris a 1/4 de sus dimensiones
	img->Integra();//Saca la integral a la imagen gris escalada
	if(img->ChkEst1())//1º Discriminación: Media de la imagen gris
		{
		img->UmbralAdap();//Crea la imagen binarizada a partir
				  //de la gris escalada.
		img->Erosiona();//Erosiona la imagen binaria
		img->Perfil();//Obtiene los vértices de corte del billete
		if(img->ChkEst2())//2º Discriminación: Índices del corte
			{
			img->Cortar();//Corta la muestra del billete
			img->Estadist();//Obtiene las estadísticas d la entrada
			if(img->ChkEst3())//3º Discrim.: % de píx y dsv. est.
				{
//Ahora, la probabilidad de estar frente a un billete es alta y se //analiza, su
denominación:
				ChkTiempo();//Chequea si la espera del método al
				//usuario es demasiado larga
				eigenimage->Normaliza(*img);//Normaliza el corte
				//de entrada a KMedia y KVarianza de pca.h
				eigenimage->ProPunto();//Obtiene la proyección //del
				corte actual en el espacio de billetes.
				iDenominacion=eigenimage->DistMaha();
				//Reconoce un billete y Actualiza iDenominacion.
				}
			}
		}
		Aviso();//Mensaje auditivo si es billet
		//FIN
	/*
	**************************************************************
	**********************PROYECTO DE GRADO************************
	***********Reconocimiento de la denominación de billetes**********
	**************Dirigido a personas no videntes*******************
	*************************************************************ESPE
	*********************************************************Grijalva F.
	********************************************************Rodríguez J.C.
	*******************************************************Octubre, 2009
	*****************************************************Quito, Ecuador
	 */
		DrawNow();
		}
```

A5.7 Conversión de la imagen RGB a escala de grises

```
/*
--------------------------------------------------------------------------------
RgbAGris. Obtiene la imagen gris.
--------------------------------------------------------------------------------
*/

void CImagen::RgbAGris(CFbsBitmap &aImagenRGB)

	{

	TUint32 *Prgb32 = aImagenRGB.DataAddress();//Puntero de la imagen RGB
	for (Pt=iPinigris;Pt<=iPfingris;++Pt)
		{
		//Obtención de la imagen en escala de grises: (R+G)/2
		*Pt=((((*Prgb32 & 0x00FF0000)>>16)
				+((*Prgb32 & 0x0000FF00)>>8))>>1);
		Prgb32++;//Aumenta el puntero de la img. RGB
		}

	}
```

A5.8 Escalamiento de la imagen en tonos de gris

```
/*
--------------------------------------------------------------------------------
EscalaGris. Crea la versión escalada de la imagen gris del viewfinder
--------------------------------------------------------------------------------
*/

void  CImagen::EscalaGris()

	{

	TInt ci=0,cj=0;//Índice de la fila escalado al vecino más cercano
	//cj es el indice de la columna escalado al vecino mas cercano
	TInt p=2;//Paso (del factor de escalamiento para filas y columnas)

	TUint8* PtGrisEsc=iPinigrisesc;//Recorre la imagen gris escalada
	TUint8* PtGris=iPinigris;//Recorre la imagen gris original

	for (i=0;i<iAltoEsc;i++)
		{
		cj=0;//Índice nuevo por columnas
		for(j=0;j<iAnchoEsc;j++)
			{
			*PtGrisEsc=PtGris[cj];
			cj+=p;//Incrementa en el paso en las columnas
			PtGrisEsc++;//Al sguiente píxel de destino
			}
		ci+=p;//Incrementa el paso en las filas
		PtGris=iPinigris+ci*iAncho;	//A la siguiente fila
		}

}
```

A5.9 Integración de la imagen gris escalada

```
/*
-----------------------------------------------------------------------------
Integra. Obtiene la integral de la imagen gris escalada.
-----------------------------------------------------------------------------
*/
void CImagen::Integra()
	{
	TUint32* PtInt=iPiniint;//Recorre la integral de la imagen
	TUint8* PtGris=iPinigrisesc;//Recorre la imagen gris escalada
	TInt suma=0;//Sumatoria de píxeles
	TInt indice;//Índice del píxel actual

	for (i=0; i<iAnchoEsc; i++)
		{
		suma = 0;//Resetea la suma de la columna a 0
		for (j=0; j<iAltoEsc; j++)
			{
			indice = j*iAnchoEsc+i;//A la siguiente fila
			suma += PtGris[indice];//Suma cada píxel
			if (i==0)
				{
				PtInt[indice]=suma;//En la primer columna, solo es la suma
				}
			else
				{
				PtInt[indice] = PtInt[indice-1]+suma;//La anterior y esta
				}
			}
		}

	}
```

A5.10 Binarización de la imagen gris escalada

```
/*
-----------------------------------------------------------------------------
UmbralAdap. Umbral Adaptativo mediante la integral de la imagen gris escalada.
(Bradley Derek)
-----------------------------------------------------------------------------
*/

void CImagen::UmbralAdap()

	{

	TInt indice;
	TInt s=iAnchoEsc/8;//Ancho y alto de la región de vecindad

	TInt s2 = s/2;//Mitad de la región
	TInt x1, y1, x2, y2;//Índices de la región

	TInt area;//Área de la región actual
	TInt suma;//Sumatoria de píxeles

	TInt T=-10;//Porcentaje de comparación para la umbralización (%)

	TUint32* PtInt=iPiniint;//Recorre la integral de la imagen gris escalada
	TUint8* PtGris=iPinigrisesc;//Recorre la imagen gris escalada
	TUint8* PtBin=iPinibin;//Recorre la imagen binaria
```

```
	for (i=0; i<iAnchoEsc; i++)
		{
		for (j=0; j<iAltoEsc; j++)
			{
			indice = j*iAnchoEsc+i;//Salta a la siguiente fila
			// Setea una región sxs con centro en el píxel actual
			x1=i-s2; x2=i+s2;
			y1=j-s2; y2=j+s2;
			// Chequea si se está en el borde
			if (x1 < 0) x1 = 0;
			if (x2 >= iAnchoEsc) x2 = iAnchoEsc-1;
			if (y1 < 0) y1 = 0;
			if (y2 >= iAltoEsc) y2 = iAltoEsc-1;

			area = (x2-x1)*(y2-y1);//Área de la región

			//Integral de esta región:
			//I(x,y)=Int(x2,y2)-Int(x1,y2)-Int(x2,y1)+Int(x1,x1)
			suma=PtInt[y2*iAnchoEsc+x2]-
			PtInt[y1*iAnchoEsc+x2]-
			PtInt[y2*iAnchoEsc+x1]+
			PtInt[y1*iAnchoEsc+x1];

			//Condición
			PtBin[indice]= (PtGris[indice]*area*100 < suma*(100-T)) ? 0:1;
			}
		}

	}
```

A5.11 Erosión de la imagen binaria

```
/*
-----------------------------------------------------------------------------
Erosiona. Obtiene la versión erosionada de la imagen binaria.
-----------------------------------------------------------------------------
*/
void  CImagen::Erosiona()
	{
	/* El elemento estructural B es una matriz de unos de 3x3
	 * con el origen O en el centro.*/
	TUint8* PtEro=iPiniero;//Recorre el objeto para img. erosionada
	TInt8 vecino=0;//Inicia el valor del vecino
	for (Pt=iPinibin;Pt<=iPfinbin;++Pt)
		{
		if((*Pt))//Solo si es distinto a cero
			{
			*PtEro = 1;//Por defecto
			for(i=0;i<8;i++)
				{//AND de los 8 vecinos, si están en el rango, sino,
				//directamente 0, por la naturaleza de B.
				vecino=*(Pt + offset_8vecino[i]);//Obtiene c/vecino
				if(vecino==0)
					{
					*PtEro=0;//Si algún vecino es cero,
					break;// sale inmediatamente
					}
				}
			}
		else
			{
			*PtEro=0;//La operación AND será forzosamente cero
			}
		PtEro++;
		}

	}
```

A5.12 Obtención del perfil de proyección de la imagen binaria erosionada

```
/*
-------------------------------------------------------------------------
Perfil. Obtiene el perfil externo e interno del billete.
-------------------------------------------------------------------------
*/

void CImagen::Perfil()

	{

	TInt acum;//Acumulador para cuenta de unos

	iFila_ver1=0;//Vértice de la fila superior, x1

	iFila_ver2=0;//Vértice de la fila inferior,x2

	iColumna=0;//Vértice de la columna, y2

	//PERFIL EXTERNO: El algortimo se aplica a la imagen erosionada
	//y2:Busca el borde externo vertical derecho del billete
	acum=0;
	for(j=0;j<iAnchoEsc;j++)
			{

			Pt=iPiniero+iAnchoEsc-1-j;//Va decrementándose en columnas
			for(i=0;i<iAltoEsc;i++)
				{

				acum+=(*Pt);//Cuenta solo los unos
				if(acum > iKTVer)//Si ya cumplió el número de unos:
//~50% de iAltoEsc (Altura de funcionamiento ~ 15 cm.)
					{

					iColumna=iAnchoEsc-j;//Guarda y2 y

					break;//Sale de este bucle
					}
				Pt+=iAnchoEsc;//Al píxel de la siguiente fila
				}
			if(iColumna!=0) break;
			//Sale de este otro bucle(ya se obtuvo y2)
			acum=0;//Se resetea el acumulador
			}
	//x1:Busca el borde externo superior horizontal del billete
	acum=0;
	Pt=iPiniero;//Se sitúa al inicio de la imagen
	for(i=0;i<iAltoEsc;i++)
		{
		for(j=iColumna-1;j>=0;j--)//Solo el número de veces de las columnas
//ya figuradas, desde la derecha
			{
			acum+=Pt[j];//Cuenta unos
			if(acum > iKTHor)//Si ya cumplió el número de uno: ~25% de
//iAnchoEsc (Altura de funcionamiento ~ 15 cm.)
				{
				iFila_ver1=i;//Guarda x1 y
				break;//Sale de este bucle
				}
			}
		if(iFila_ver1!=0) break;//Sale de este otro bucle (ya se obtuvo x1)
		acum=0;//Se resetea el acumulador
		Pt+=iAnchoEsc;//A la siguiente fila
		}
```

```
	//x2:Busca el borde externo inferior horizontal del billete
	Pt=iPfinero-iAnchoEsc+1;//Se sitúa en la última fila de la imagen
	acum=0;
	for(i=0;i<iAltoEsc;i++)
		{
		for(j=0;j<iAnchoEsc;j++)
			{
			acum+=Pt[j];
			if(acum > iKTHor)//Si ya cumplió el número de uno: ~25% de
//iAnchoEsc (Altura de funcionamiento ~ 15 cm.)
				{
				iFila_ver2=iAltoEsc-i;//Guarda x2 y
				break;//Sale de este bucle
				}
			}
		acum=0;//Se resetea el acumulador
		if(iFila_ver2) break;//Sale de este otro bucle (ya se obtuvo x2)
		Pt-=iAnchoEsc;//A la fila anterior
		}


	//PERFIL INTERNO
	//Redimensionamiento de los perfiles para eliminar el borde blanco
	//del billete en la imagen binaria.
	//Algortimo aplicado a la imagen binaria. //Ajusta la columna  y2
		acum=0;
	for(j=0;j<iColumna;j++)
		{

		Pt=iPinibin+iFila_ver1*iAnchoEsc+iColumna-j-2;//Se da un margen de 2
//píxeles, y hacia la columna izq.
		for(i=iFila_ver1;i<iFila_ver2;i++)
			{
			acum+= *Pt;//Suma los blancos del borde (unos)
			Pt+=iAnchoEsc;//Hasta terminar con esta columna
			}

		if(acum < (75*iKTVer)/100)//Si baja del 75% del borde
			{
			iColumna-=(j+2);//Nueva columna, compensando el margen
			break;
			}
		acum=0;
		}

	//Ajusta la fila x1
	acum=0;
	//A recorrer desde el 35% del alto del billete y más 2 píxeles abajo de
//margen:

	Pt=iPinibin+iFila_ver1*iAnchoEsc+iColumna-(35*(iFila_ver2-
iFila_ver1))/100+2*iAnchoEsc;
	for(i=iFila_ver1;i<iFila_ver2;i++)
		{
		for(j=0;j<(35*(iFila_ver2-iFila_ver1))/100;j++) acum+=Pt[j];
	//Hasta el 35%
		if(acum < (60*iKTHor)/100)//Baja del 75% de los píxeles del borde.
			{
			iFila_ver1=i+2;//Nueva fila, compensando el margen
			break;
			}
		acum=0;
		Pt+=iAnchoEsc;
		}
```

```
	//Ajusta la fila x2
	acum=0;
	//A recorrer desde el 35% del alto del billete y menos 2 píxeles abajo de
//margen:
	Pt=iPinibin+iFila_ver2*iAnchoEsc+iColumna-(35*(iFila_ver2-
iFila_ver1))/100-2*iAnchoEsc;
	for(i=0;i<iFila_ver2-iFila_ver1;i++)
		{
		for(j=0;j<(35*(iFila_ver2-iFila_ver1))/100;j++)acum+=Pt[j];
		if(acum < (60*iKTHor)/100)//Baja del 75% de los píxeles del borde
			{
			iFila_ver2-=(i+2);//Nueva fila, compensando el margen
			break;
			}
		acum=0;
		Pt-=iAnchoEsc;
		}
	//Redimensionamiento para la imagen real (del doble de dimensión)
	iFila_ver1*=2;
	iFila_ver2*=2;
	iColumna*=2; //Para cortar en la imagen original

	}
```

A5.13 Corte y escalamiento de la imagen gris

```
/*
-------------------------------------------------------------------------------
Cortar. Corta y escala el corte del billete para establecerse la muestra.
-------------------------------------------------------------------------------
*/
void CImagen::Cortar()

	{

	/*DETERMINACIÓN DE LAS DIMENSIONES DE LA MALLA ORIGINAL (corte de la gris)
	Se determina el ancho a cortarse en base a la relación: a = h/proporción,
	donde la proporción es el alto al ancho de la imagen.
	El objetivo es producir un corte de la imagen gris que esté en igual
	proporción de ancho y alto de una nueva imagen, una versión escalada
	de este corte, que será la imagen a usarse  en el método de
	reconocimiento,
	de resolución: KAnchoEscalado x KAltoEscalado;
	de donde se determina: porporcion =  KAltoEscalado / KAnchoEscalado.
	*/
	TInt a=0, h=0;//Variables para ancho (a) y alto(h) de la imagen original
	TReal aaux=0, haux=0;//Variables reales, auxiliares para las divisiones
	                                  //por las proporciones

	h = iFila_ver2-iFila_ver1;//Altura de la imagen corte original
	/*
	Las siguentes operaciones convierten a h en un entero que,
	a diferencia del original, será exactamente divisible para la proporción
	de corte, para asegurar que el ancho a obtenerse sea un número entero.
	*/
	haux= ((TReal)h/KProporcion) * KProporcion;
		//h=piso(h/proporcion)*proporcion
	h= (TInt)haux;
//piso(h), lo que lo hace entero y ya es exactamente divisible
//para la proporción.
	//De esta forma, se aseguró que a sea entero
	aaux = (TReal)h / (TReal)KProporcion;//Ancho de la imagen corte, entero
	a=TInt(aaux);//Ya era entero, pero se asegura su formato en casting
```

```
/*ALGORITMO DE ESCALAMIENTO DE LA MALLA ORIGINAL EN LA NUEVA
Realiza la operación de escalamiento de la imagen de corte, cuyas
dimensiones fueron antes determinadas.
Como se ve, el corte y escalamiento se realizarán aquí,
en un sólo juego de bucles.
*/
TInt ci=0;//Índice de columnna seleccionada de la malla original
            //como la de vecino más cercano
TInt cj=0;//Índice de fila seleccionada de la malla original
        //como la de vecino más cercano
TInt ind=0;//Índice real de columna escalada (ci,cj / KDec) (KDec=100)
//KDec se utiliza para emular decimales, cuando se vaya haciendo la
//operación de paso.
//Punteros:
TUint8* Pinicorte=iPinigris + iAncho * (iFila_ver1 - 1) + 1 + (iColumna - a);
TUint8* Pcorte;
//Pcorte es el puntero que se moverá en la zona de la imagen gris,
//cortando la malla original.
//Pinicorte es la referencia del inicio del corte de la imagen original:
//el píxel superior izquierdo de la zona a recortarse.
Pt=iPiniesc; //Se inicializa el puntero, que se moverá en la malla de la
             //imagen escalada

//Factores de escalamiento:
/*
Como la proporción entre anchos y altos originales y finales se
mantiene: h/a = hfinal/afinal, también se cumple la proporción:
Sx = hfinal/h = afinal/a = Sy; por lo que los factores de escalamiento
de filas y columnas son los mismos: Sx = Sy   = S.
Además, este escalamiento en este caso es de reducción,
por lo que: 0 <S <1; Se define un "paso" como:
p = Sx^(-1) = Sy^(-1) ; que cumple: 1 < p < inf.
Estos pasos son en los que se debe ir recorriendo la malla original
para la asignación de cada píxel de esta, en la malla final.
*/
TReal paux=KFactorDecEscalado*h;//Auxiliar real
//Este es el paso: p = (hf/h)^(-1) = (h/hf) , pero de otra forma:
//KFactorDecEscalado = 100 / hf; por lo que: p =(h*100/hf):
//Esto se hace por seguridad: si p < 1,
//se emulan sus decimales haciéndo p x 100
TInt p=(TInt)paux;//Se hace al paso entero para trabajar en el bucle

//Finalmente, el bucle de asignación:
Pcorte=Pinicorte;//Se establece el puntero de origen
//en el inicio de la imagen a cortarse.
for (j=0;j<KAltoEscalado;j++)//Recorre la imagen escalada
      {
      ci=0;//Se reinicia el índice a seleccionar en columnas
      for(i=0;i<KAnchoEscalado;i++)
            {
            ind=ci/KDec;//Se restituye al paso establecido
                  //para establecer el índice seleccionado en columnas
            *Pt=Pcorte[ind];//Se asigna el píxel del índice establecido
                        //a la nueva imagen
            ci+=p;//Se da el paso de selección en columnas
            Pt++;//Se incrementa el puntero de la imagen destino
            }
      cj+=p;//Se da el paso de selección en filas
      Pcorte=Pinicorte+(cj/KDec)*(iAncho);//Salta a la siguiente fila
                              //seleccionada por el paso, restituido.
      }

}
```

A5.14 Obtención de las estadísticas de la imagen de entrada al PCA

```
/*
-----------------------------------------------------------------------------
Estadist. Obtiene las características estadísticas de la imagen de entrada
(cortada)
-----------------------------------------------------------------------------
*/
void  CImagen::Estadist()
	{
	TReal DvcEstR;//Contenedor real de sigma
	TInt nump=KAltoEscalado*KAnchoEscalado;
	//Número de píxeles de la imagen a obtener sus estimadores
	TUint8* Ptini=iPiniesc;
	//Puntero indicando el inicio de la img. a obtener sus estimadores.
	TUint8* Ptfin=iPfinesc;
	//Puntero indicando el fin de la img. a obtener sus estimadores.
	iMedia=0;//Media
	iDvcEst=0;//Sigma o desviación estándar

	for (Pt=Ptini;Pt<=Ptfin;++Pt)//Sobre la imagen gris
		{
		iMedia+=(*Pt);//Acumulador de la media
		iDvcEst+=(*Pt) * (*Pt);//Acumulador de la varianza
		}
	iMedia/=nump;//Media = sum(1,N,pix)
	iDvcEst/=nump;//desvest=sqrt(sum(1,N,pix^2)/N-media^2)
	iDvcEst-=iMedia * iMedia;//desvest=sqrt(sum(1,N,pix^2)/N-media^2)
	DvcEstR=(TReal)iDvcEst;
	Math::Sqrt(DvcEstR,DvcEstR);// Obtiene la raíz cuadrada
	iDvcEst = (TInt) DvcEstR;//Desviación estándar
	}
```

A5.15 Tercera discriminación: Adecuado porcentaje de píxeles blancos en la imagen de entrada binaria

```
/*
-----------------------------------------------------------------------------
ChkEst3(). Devuelve 1 si la imagen erosionada tiene un porcentaje suficiente
de blancos como para no ser una imagen vacía
(que daría problemas de falsos positivos con papeles).
-----------------------------------------------------------------------------
*/
TUint CImagen::ChkEst3()
	{
	TInt e=3;
//Espesor de tolerancia para la toma del nuevo rectángulo a analizarse:
	TInt f1=iFila_ver1/2 + e;
//Fila 1 escalada al tamaño de la imagen umbralizada
	TInt f2=iFila_ver2/2 - e;
//Fila 2 escalada al tamaño de la imagen umbralizada
	TInt c=iColumna/2 - e;
//Columna escalada al tamaño de la imagen umbralizada
	TInt area=c*(f2-f1);//Área del nuevo rectángulo a analizarse
	TInt sumpix=0;//Sumatoria de píxeles iguales a uno en el rectángulo
	TInt porcen=0;//Porcentaje de píxeles blancos en el rectángulo (%)

	Pt=iPinibin+f1*iAnchoEsc;//Sitúa al puntero al inicio del rectángulo
	for(i=f1;i<f2;i++)//Se recorre todo el rectángulo
		{
		for(j=0;j<c;j++)
			{
			if(Pt[j])sumpix++;//Acumula el número de píxeles blancos
			}
		Pt+=iAnchoEsc;//A la siguiente fila
		}
```

```
	porcen=(sumpix*100)/area;
	//Porcentaje de blancos en relación al área cortada del rectángulo

	//Criterio de discriminación:
	//Sólo se considerará una imagen con probabilidad de ser billete, si:
	//El procentaje de blancos en el umbralizado es > a un KPorcen y
	//Si la desviación estándar está entre
//KFvcEstBaja y KDvcEstAlta (cálculos de las muestras):

	return ((iDvcEst<KDvcEstAlta && iDvcEst>KDvcEstBaja) && (porcen>KPorcen)) ? 1:0;
	}
```

A5.16 Chequeo del tiempo de espera por el reconocimiento

```
/*
-----------------------------------------------------------------------------
ChkTiempo(). Chequea si la espera del método al usuario es demasiado larga
-----------------------------------------------------------------------------
 */
void CCameraAppBaseContainer::ChkTiempo()

	{

		iContador++;//Cuenta las veces que se tiene una imagen adecuada para
					//la aplicación de PCA (Ya que llegó aquí).
		if(iContador > KCuenta)
			//Si más d 20 veces se reconoció una img. válida
			{
			iContador=0;//Inicia otra cuenta
			if(iContadorMay <= KCueMay)//Solo si aún no se ha superado la
						//tolerancia mayor.

	iSoundPlayer>Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\otra.mp3"));

			/*
			* Heurísticamente, se llega a la conclusión de que cada pasada por
			* esta bifurcación, conlleva 0.12 s., por lo que la constante
			* KCuenta debe ser elegida sabiéndose que se obtendrá un tiempo
			* de espera de:
			* T = 0.12 * KCuenta
			* para que el método identifique la denominación de un billete
			*/

			//Con la actual KCuenta, T = 15 s.
			iContadorMay++;//Se incrementa el contador
					//mayor par pedir reinicio
			if(iContadorMay > KCueMay)//Si fueron las veces tolerables,
					//pide reinicio del sistema.
				{
				if(!iSoundPlayer->ChkFlag())
					iSoundPlayer-
>Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\reinicie.mp3"));
			//Al no encerariContadorMay, se asegura que el sonido de
			//reinicio no pare hasta que se reinicie la aplicación

				}

			}

	}
```

A5.17 Normalización de la imagen de entrada

```
/*
-----------------------------------------------------------------------------
Normaliza. Normalización de la imagen muestra a una media y desv. est. dadas
en una imagen muestra de 32 bits, y resta de la imagen media del set.
-----------------------------------------------------------------------------
*/
void CPca::Normaliza(CImagen &im)
	{
	/*
	 A cada píxel de la muestra se pasa a p:
	  p=[(p_muestra-u_muestra)/sigma_muestra*sigma_nuevo+u_nueva]-p_imgmedia
	 Se desarrolla y se simplifica a las constantes fuera del for:
	 p=p_muestra*K1+K2-p_imgmedia
	 Donde:
	 K1=sigma_nueva/sigma_muestra
	 K2=u_nueva-(u_muestra*sigma_nueva)/sigma_muestra
	*/
	TUint8* Ptfuente = im.iPiniesc;//Imagen escalada de 8 bits
	TUint32* Ptmedia = iPtIniMedia;//Recorre la imagen media
	TReal K2r = KMedia - (im.iMedia * KDvcEst)/im.iDvcEst;//K2 real
	TInt K2=(TInt)K2r;//Versión en tipo entero
	for (Pt=iPtIniMuestra;Pt<=iPtFinMuestra;++Pt)
		{
		 (*Pt) = ((*Ptfuente) * KDvcEst)/im.iDvcEst + K2 - *Ptmedia;
		Ptfuente++;
		Ptmedia++;
		}
	//Esta imagen es la que va al producto punto
	}
```

A5.18 Proyección en el espacio de billetes

```
/*
-----------------------------------------------------------------------------
ProPunto. Obtiene los pesos de la muestra a través del producto punto con los
eigen - vectores.
-----------------------------------------------------------------------------
*/

void CPca::ProPunto()

	{
	Pt = iPtIniMuestra;//Recorre la muestra
	TUint32* PtDestino = iPtIniPm;//Recorre los pesos muestra resultado
	TUint32* PtFuente = iPtIniEv;//Recorre los eigen - vectores

	for (i=0;i<KNumBases;i++)
		{
		Pt = iPtIniMuestra;//Vuelve a recorrer la muestra desde el inicio
		(*PtDestino)=0;//Inicializa el acumulador
		for(j=0;j<KNumPixeles;j++)
			{
			//El casting es necesario
			(*PtDestino)+=((TInt)((*Pt)*(*PtFuente))>>10);
					//Pues fueron escalados por 2^20
			Pt++;
			PtFuente++;
			}
		(*PtDestino)=((TInt)(*PtDestino))>>10;
				//Pues fueron escalados por 2^20
		PtDestino++;
		//Aumenta el puntero del destino a otro peso de la muestra
		}//Se ha construido el vector de pesos de muestra iPesosMuestra
	}
```

A5.19 Cálculo de la distancia de *Mahalanobis* más pequeña

```
/*
-----------------------------------------------------------------------------
DistMaha. Obtiene la distancia de Mahalanobis entre el vector de pesos
obtenido de la muestra y cada vector de peso de las bases de eigen-vectores.
-----------------------------------------------------------------------------
*/
TUint CPca::DistMaha()
	{
	TUint32* PtBases = iPtIniPe;//Recorre los pesos de los eigen-vectores
	TUint32* PtValores = iPtIniVap;//Recorre los valores propios
	TUint32 DistMin = KTUint32Max;//Inicialización con un valor muy alto
	TUint pos=0;//Posición o índice de la distancia menor
	TUint grupo=0;//Grupo al que pertenece la muestra (1 a 7)
	TUint lado=0;//Lado del billete que se reconoció
	TUint billete=0;//Denominación del billete reconocido

	for (i=0;i<KNumMuestras;i++)
		{
		Pt = iPtIniPm;//Vuelve a recorrer la muestra desde el inicio
		PtValores = iPtIniVap;//Vuelve a recorrer los valores propios
		DistM[i]=0;//Inicializa el acumulador de cada respuesta
		for(j=0;j<KNumBases;j++)
			{
			//El casting es necesario.
			DistM[i]+=(TUint32)((((*Pt)-(*PtBases))*((*Pt)-
(*PtBases)))/(*PtValores));
			Pt++;
			PtBases++;
			PtValores++;
			}
		//Algoritmo para la obtención de la menor distancia
		if(DistM[i]<DistMin)
			{
			DistMin=DistM[i];
			pos=i;
			}
		}
//Se ha construido un vector de distancias a cada peso de eigen-faces

	//Reconocimiento de la denominación del billete
	grupo=(pos+1)/(4*KMuestrasLado)+(((pos+1) % (4*KMuestrasLado))>0);
		//A cuál de los 6 grupos pertenece
	lado=((pos+1)/KMuestrasLado)+(((pos+1) % (KMuestrasLado))>0);
		//De qué lado es la muestra del billete más parecida a esta muestra
	switch(grupo)
		{
		case 1:
			{
			billete=1;
			if(lado==2 || lado==4)
			//Si es el 2º o 4º lado del billete de $1 (es el "más único")
				DistMin/=2;
			//Se asegura que se reconozca como billete de $1
			break;
			}
		case 2:
			{
			billete=5;
			break;
			}
		case 3:
			{
			billete=10;
			break;
			}
```

```
		case 4:
			{
			billete=20;
			break;
			}
		case 5:
			{
			billete=50;
			break;
			}
		case 6:
			{
			billete=100;
			break;
			}
		default:
			{
			billete=0; //Si no es billete
			}
		}
	return (DistMin<KUmbralDistM) ? billete : 0;
	//Si es menor al umbral de distancias determinado en el ENTRENAMIENTO
	}
```

A5.20 Función para aviso al usuario

```
/*
-----------------------------------------------------------------------------
Aviso. Anuncia el audio de la denominación del billete que se ha determinado o
hace nada si no se ha hecho.
-----------------------------------------------------------------------------
*/
void CCameraAppBaseContainer::Aviso()

	{

	if(iDenominacion!=0) //Se detectó un billete
		{
		iContador=0;//Reseteo del contador de cambio de lado
		iContadorMay=0;//Reseteo del contador de reinicio de aplicación
		iToneAdapter->PlayL();
			//BEEP:Sonido indicador de dónde está el billete
		iActual=iDenominacion;//Es el billete actual
		if(iActual == iAnterior)iContAci++;
				//Se aumenta el contador de aciertos
		else iContAci=0;//Se resetea si es que halla otro valor
		if(iContAci >= KUmbralCuenta)//Se confirma que es un billete de la
					//denominación repetidamente detectada.
			{
			Audio();//Se emite el mensaje auditivo con la denominación
				//del billete
			iContAci=0;//Se espera una nueva operación
			}
		iAnterior=iDenominacion;//Se recuerda el anterior valor
		}
	else //Si no es un billete, no hace nada

		{

		}

	}
```

A5.21 Reproducción de archivos de audio

```
/*
-------------------------------------------------------------------------------
Audio. Emite el sonido del billete reconocido
-------------------------------------------------------------------------------
*/
void CCameraAppBaseContainer::Audio()
	{
	switch(iDenominacion)
		{
		case 1:
			{
			if(!iSoundPlayer->ChkFlag())
			//Solo entra a cambiar el audio si el anterior audio
			//eventual terminó de reproducirse(indicado por la bandera)
	iSoundPlayer->Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\uno.mp3"));
//Cambia el archivo a reproducirse indicando la nueva ruta

			break;
			}
		case 5:
			{
			if(!iSoundPlayer->ChkFlag())
	iSoundPlayer->Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\cinco.mp3"));

			break;
			}
		case 10:
			{
			if(!iSoundPlayer->ChkFlag())
	iSoundPlayer->Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\diez.mp3"));

			break;
			}
		case 20:
			{
			if(!iSoundPlayer->ChkFlag())
	iSoundPlayer->Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\veinte.mp3"));

			break;
			}
		case 50:
			{
			if(!iSoundPlayer->ChkFlag())
iSoundPlayer->Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\cincuenta.mp3"));

			break;
			}
		case 100:
			{
			if(!iSoundPlayer->ChkFlag())
	iSoundPlayer->Reconfigurar(_L("C:\\Data\\Lectbill\\Audio\\cien.mp3"));

			break;
			}
		default:
			{
			}
		}

	}
```

A5.22 Función de retardo

```
/*
-------------------------------------------------------------------------------
DelaySeg(aTiempo). Produce un Delay de t segundos. Basado en la función anterior.
-------------------------------------------------------------------------------
 */
void CCameraAppBaseContainer::DelaySeg(TInt aTiempo)

	{

	TInt k,h, lim;//lim es el límite superior del primer for

	lim=188679*aTiempo;//Da el límite superior para producir t segundos.

	for(k=0;k<lim;k++)
		{
		for(h=0;h<100;h++)
			{
		/*
		* El tiempo medio de instrucción del procesador,
		* medido heurísticamente, es de:
		* ts =  5.3000e-008 s.
		* En base a esto, un Delay se realiza con este doble bucle,
		* de un tiempo: T = t1*t2*ts
		* Donde: t1:límite de k; t2: límite de h
		* **********************AHORA**************************
		* Sabiéndose que para un delay de 1 seg., se requiere que
		* k tenga un límite superior de 188679; se escalan de esa forma
		* los demás tiempos para dar un delay de t segundos.
		* Por tanto, el tiempo de dalay está determinado, en segundos, por:
		* T = lim*100*ts = t*188679*100*5.3000e-008 [s.]
		*/

			}

		}

	}
```

A5.23 Función para medir la memoria RAM usada y disponible de la aplicación

```
/*
-------------------------------------------------------------------------------
Muestra RAM disponible (L) y usada (U) por el programa
-------------------------------------------------------------------------------
*/
void CCameraAppBaseContainer::mostrarRAM()

	{

	if (contador==0)
		{
		TInt RAMusada,RAMtotal,RAMlibreActual;
		TPoint pos(1,70);
		TBuf<15> fpsBuf;

		HAL::Get(HAL::EMemoryRAM, RAMtotal);
		HAL::Get(HAL::EMemoryRAMFree, RAMlibreActual);
		RAMusada=(RAMlibreAlIniciar-RAMlibreActual);
		RAMlibreActual=RAMlibreActual/1024/1024*10;
		RAMusada=RAMusada/1024/1024*10;//Se transforma a Megabytes
```

```
		/*
		 * Si el celular es E65,imprime en otro lado
		 * porque la orientación de la imagen es vertical:
		*/
		if (mUID==0x20000604)
			{
			pos.iX=70;
			pos.iY=15;
			_LIT(KFps,"L: %d U: %d");
			fpsBuf.Format( KFps, RAMlibreActual,RAMusada);
			}
		else
			{
			_LIT(KFps,"L: %d\n\nU: %d");
			fpsBuf.Format( KFps, RAMlibreActual,RAMusada);
			}
		iFbsBitGc->UseFont(CEikonEnv::Static()->DenseFont());
		iFbsBitGc->SetPenColor( KRgbWhite);
		//iFbsBitGc->Clear();
		/*
		 *Se descomenta siempre y cuando mostrarFps() limpie la pantalla
		 * y sea llamada antes.
		*/
		iFbsBitGc->DrawText( fpsBuf, pos);
		}
	}
```

A5.24 Función para medir los *frames* por segundo de la aplicación

```
/*
-----------------------------------------------------------------------------
Muestra los FPS (Frames por segundo).
-----------------------------------------------------------------------------
*/
void CCameraAppBaseContainer::mostrarFps()
	{
	//Variables necesarias
	iFps = (TInt)(10000/(User::NTickCount() - iTimeMs));
//Divide para 10: tomar en cuenta
	iTimeMs = User::NTickCount();
	contador++;
	//Cuenta
	if (contador==10)
		{
		_LIT(KFps,"%d");
		TBuf<5> fpsBuf;
		TPoint pos(1,50);
		/*
		 * Si el celular es E65,imprime en otro lado
		 * porque la orientación de la imagen es vertical:
		*/
		if (mUID==0x20000604)
			{
			pos.iX=5;
			pos.iY=15;
			}
		fpsBuf.Format( KFps, iFps);
		iFbsBitGc->UseFont(CEikonEnv::Static()->DenseFont());
		iFbsBitGc->SetPenColor( KRgbWhite);
		iFbsBitGc->Clear();
		iFbsBitGc->DrawText( fpsBuf, pos);
		contador=0;
		}

	}
```

REFERENCIAS BIBLIOGRÁFICAS

[1] CONADIS, "Personas registradas en el CONADIS", http://www.conadis.gov.ec/provincias.php, consultado el 30 de Octubre de 2009.

[2] CONADIS, "Distribución de las personas con discapacidad por tipo de deficiencia", http://www.conadis.gov.ec/estadisticas.htm, consultado el 30 de octubre de 2009.

[3] CONADIS-INEC, *Ecuador: La Discapacidad en Cifras. Análisis de Resultados de la Encuesta Nacional de Discapacidades,* primera edición, s.e, Ecuador, 2005, pp. 107-108.

[4] GONZALEZ, Rafael y WOODS, Richard, *Digital Image Processing,* segunda edición, editorial Prentice Hall, New Jersey, 2002.

[5] PAJARES, Gonzalo y otros, *Imágenes Digitales. Procesamiento práctico con Java,* segunda edición, editorial AlfaOmega, Madrid, 2004.

[6] MathWorks, "Mathworks Technical Solutions", http://www.mathworks.es/support/solutions/en/data/1-1ASCU/index.html?product=IP&solution=1-1ASCU, 26 de Junio de 2009, consultado el 14 de Noviembre de 2009.

[7] ROHS, Michael, "Recognition of 2-dimensional visual codes with the Nokia 7650", www.mics.ch/SumIntU03/BGfeller.pdf, 2003, consultado el 14 de Noviembre de 2009.

[8] BART, Evgeniy y ULLMAN, Shimon**,** "Image normalization by mutual information", http://www.comp.leeds.ac.uk/bmvc2008/proceedings/2004/papers/paper_125.pdf, 2004, consultado el 14 de Noviembre de 2009.

[9] OTSU, Nobuyuki, "A threshold selection method from gray-level histograms", *IEEE Trans. Systems, Man, and Cybernetics,* Vol. 9, No. 1, pp. 62-66, 1979.

[10] WELLNER, Pierre, "Adaptative thresholding for the digitaldesk", *EuroPARC,* pp. 93-110, 1993.

[11] SAUVOLA, J. y PIETIKAINEN, M, **"**Adaptive document image binarization", *Pattern Recognition*, Vol. 33, No. 1, pp. 225-236, 2000.

[12] SHAFAIT, Faisal, KEYSERS, Daniel y BREUEL, Thomas**.** "Efficient Implementation of Local Adaptive Thresholding Techniques using Integral Images", pubs.iupr.org/DATA/2007-IUPR-11Sep_1129.pdf , 2007, consultado el 14 de Noviembre de 2009.

[13] VIOLA, P. y JONES, M. J, "Robust real-time face detection", *Int. Journal of Computer Vision*, Vol. 57, No. 2, pp. 137-154, 2004.

[14] BRADLEY, Dereck y ROTH, Gerhard. "Adaptive Thresholding Using the Integral Image", http://people.scs.carleton.ca/~roth/iit-publications-iti/docs/gerh-50002.pdf, 2007, consultado el 14 de Noviembre de 2009.

[15] ZRAMDINI, Abdelwahab y INGOLD, Rolf, "Optical font recognition from projection profiles", http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume6/issue3/ofr.pdf, 1993, consultado el 14 de Noviembre de 2009.

[16] RATH, Toni y MANMATHA, R, "Word Image Matching Using Dynamic Time Warping", ciir.cs.umass.edu/pubfiles/mm-38.pdf, 2003, consultado el 14 de Noviembre de 2009.

[17] TURK, M. y PENTLAND, A, **"**Face recognition using eigenfaces", *Journal of Cognitive Neuroscience*, Vol. 3, No. 1, pp. 71-86, 1991.

[18] SMITH, Lindsay, "A tutorial on Principal Components Analysis", www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf, 26 de Febrero de 2002, consultado el 14 de Noviembre de 2009.

[19] SHLENS, Jonathon. "A Tutorial on Principal Component Analysis", www.snl.salk.edu/~shlens/pub/notes/pca.pdf, 22 de Abril de 2009, consultado el 14 de Noviembre de 2009.

[20] ACAN, **"**Diccionario de Lengua Española", http://www.elcastellano.org/cgi-bin/diff.pl?palabra=media, 1996, consultado el 14 de Noviembre de 2009.

[21] WESTON, Harley, "A note on standard deviation", http://mathcentral.uregina.ca/RR/database/RR.09.95/weston2.html, 1995, consultado el 14 de Noviembre de 2009.

[22] GARCÍA, Joe, *PROBLEMAS Y EJERCICIOS DE MATEMÁTICAS SUPERIORES: Algebra lineal con MATLAB*, IV tomo, 1º edición, s.e, Quito - Ecuador, Octubre de 2005.

[23] ESCOBEDO, María Teresa y SALAS, Jorge, "Mahalanobis y las aplicaciones de su distancia estadística", http://dialnet.unirioja.es/servlet/dcfichero_articulo?codigo=2881069&orden=0, 2008, consultado el 10 de Noviembre de 2009.

[24] SIROVICH, L. y KIRBY, M, "Low-dimensional procedure for the characterization of human faces", *Journal of the Optical Society of America A*, Vol. 4, No. 3, pp. 519-524, 1987.

[25] ARM, "Upgrading ARM926EJ-S Software Systems to the ARM1176JZ-S", http://www.arm.com/miscPDFs/23896.pdf, 2008, consultado el 14 de diciembre de 2009.

[26] Canalys, "Canalys: iPhone outsold all Windows Mobile phones in Q2 2009, http://www.appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html, 21 de Agosto de 2009, consultado el 17 de Noviembre de 2009.

[27] FITZEK, Frank y REICHERT, Frank, *Mobile Phone Programming and its Application to Wireless Networking*, segunda edición, editorial Springer, Dordrecht-Holanda, 2007.

[28] STICHBURY, Jo, *Symbian OS Explained. Effective C++ Programming for Smartphones*, primera edición, editorial Wiley, Southern Gate-Inglaterra, 2004.

[29] SCHEIBLE, Jurgen y TUULOS, Ville. *Mobile Python. Rapid Prototyping of Applications.* segunda edición, Editorial Wiley, Southern Gate-Inglaterra, 2007.

[30] HARRISON, Richard, *Symbian OS C++ for Mobile Phones. Programming with extended functionality and Advanced Features*, primera edición, editorial Wiley, Southern Gate-Inglaterra, 2002.

[31] Nokia, "Carbide C++ Tutorial 1 – Helloworld", http://wiki.forum.nokia.com/index.php/Carbide.c%2B%2B_Tutorial_1_-_Helloworld, 23 de Septiembre de 2009, consultado el 17 de Noviembre de 2009.

[32] BUSTARRET, Eric, "Carbide C++: Setting up On Target Debugging", http://www.newlc.com/en/Carbide-c-Setting-up-On-Target.html, 26 de Octubre de 2006, consultado el 17 de Noviembre de 2009.

[33] BUSTARRET, Eric, "Playing a WAV file", http://www.newlc.com/en/Playing-a-WAV-file.html, 4 de marzo de 2003, consultado el 14 de diciembre de 2009.

[34] Noki, "S60 Platform: Comparison of ANSI C++ and *Symbian* C++", http://sw.nokia.com/id/36f953bf-59eb-4f41-9e60-f51f494c5d14/S60_Platform_Comparison_of_ANSI_Cpp_and_Symbian_Cpp_v2_0_en.pdf, 31 de Mayo de 2006, consultado el 14 de Noviembre de 2009.

[35] BROWN, James, "What is an eigenvalue?", http://jalt.org/test/bro_10.htm, abril de 2001, consultado el 17 de diciembre de 2009.

[36] POULTON, Gareth, "Using Sound with *Symbian*", http://www.newlc.com/Using-Sound-with-Symbian.html, 6 de octubre de 2003, consultado el 14 de diciembre de 2009.

FECHA DE ENTREGA

El proyecto fue entregado al Departamento de Eléctrica y Electrónica y reposa en la Escuela Politécnica del Ejército desde:

Sangolquí, a _____

ELABORADO POR:

_____	_____
FELIPE LEONEL GRIJALVA ARÉVALO	JUAN CARLOS RODRÍGUEZ GUERRA
CI: 1710847441	CI: 1721234902

AUTORIDADES:

_____

Ing. GONZALO OLMEDO

COORDINADOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

_____

Ing. VICTOR PROAÑO

COORDINADOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL