View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Universidad Carlos III de Madrid e-Archivo

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Departamento de Tecnología Electrónica
Ingeniería Técnica Industrial: Electrónica Industrial

Proyecto Fin de Carrera

Diseño de una aplicación de reconocimiento de firmas basada en alineamiento temporal dinámico

Autor: Roberto Barahona Expósito

Director: Oscar Miguel Hurtado

Tutor: Raúl Sánchez Reíllo

Junio 2012

Agradecimientos

A mi familia, por el constante apoyo y por la confianza que siempre depositan en mí.

Gracias a todos.

Índice

1 Introducción

1.1 Contexto

Mediante un algoritmo en Matlab y una base de datos de firmas, se puede desarrollar un programa que sea capaz de realizar una comparación de estas de forma que sea posible detectar similitudes entre ellas hasta el punto de poder concluir si una firma pertenece o no a la misma persona.

Entonces, sería interesante implementar un algoritmo como este en un lenguaje tan utilizado en el mundo de la informática y en la industria en general como es Java, de esta forma sería más sencillo y práctico portar una versión del algoritmo a una gran variedad de sistemas informáticos.

En este proyecto se implementará y testeará un algoritmo de comparación de firmas en Java y se estudiará la posibilidad de utilizar este algoritmo dentro de una aplicación para terminales móviles y tablets, dándole de esta forma gran versatilidad al algoritmo. En vista de la gran aceptación por parte del consumidor del sistema operativo para móviles y tablets Android, así como de su gran crecimiento en el mercado y, sobre todo, de su compatibilidad con el lenguaje de programación Java, esta sería una buena opción por la de decantarse a la hora de programar una aplicación.

1.2 Motivación

El inicio de este proyecto se lleva a cabo con cuatro motivaciones:

En primer lugar el estudio de las técnicas biométricas, centrándonos en las de reconocimiento de firmas, así como el análisis de algoritmos para comparar firmas[1].

En segundo lugar el estudio del lenguaje de programación Java y sus posibilidades a la hora de implementar un algoritmo matemático, concretamente un algoritmo biométrico de comparación de firmas.

Otra motivación de este proyecto es el estudio de los principales sistemas operativos móviles y posibilidades a la hora de programar una aplicación biométrica

para alguno de ellos. Nos centraremos en el estudio del sistema operativo móvil Android.

La última de las motivaciones de este proyecto es implementar una aplicación biométrica con la que probar el algoritmo matemático previamente programado en Java y, de esta manera, dar una visión práctica al código programado.

1.3 Objetivos

Al comienzo de este proyecto se plantean una serie de objetivos para conseguir la obtención de los resultados esperados a priori.

El primero de ellos es el de implementar en java el algoritmo de comparación de firmas DTW.

En segundo lugar, aunque podríamos clasificarlo como un objetivo secundario, la redacción de un manual de introducción a la programación de aplicaciones para dispositivos móviles Android.

El tercer y último objetivo del presente proyecto es el de implementar una aplicación Android que albergue el programa java previamente codificado y que proporcione una interfaz de usuario que le permita firmas y de esta forma probar el algoritmo.

1.4 Organización del documento

El trabajo realizado y que se expone en este documento se divide en dos bloques, por un lado el estudio de un algoritmo de comparación de firmas y su implementación en leguaje de programación Java y por otro el estudio del lenguaje para programar para Android y la creación de una aplicación para dispositivos móviles que se base en el algoritmo ya creado.

Se empezará con una introducción teórica a biometría y firmas manuscritas, ya que el presente proyecto trata sobre biometría basada en firmas manuscritas. En primer lugar, se exponen unas breves referencias al origen, historia, características, aplicaciones y utilidad de cada uno de los métodos biométricos.

A continuación se introduce el algoritmo que se va a implementar, explicando su funcionamiento, su estructura y sus resultados.

Seguidamente, una vez se tiene claro el funcionamiento del algoritmo, así como su punto de partida y su salida, se procede a su implementación en lenguaje de programación Java. Así como su posterior depuración y testeo utilizando el entorno de desarrollo integrado Eclipse.

Llegados a este punto podemos dar por concluida la primera parte de este proyecto. Es ahora cuando se hace una introducción a la programación de aplicaciones para terminales móviles. Concretamente, y aprovechando su carácter de software libre, así como su gratuidad, de tratará con aplicaciones para el sistema Android. Se detallará paso a paso como preparar el entorno de trabajo para comenzar a programar y se aportará algún ejemplo práctico de este sistema relativamente nuevo y ya en auge.

Después de la introducción a la programación de aplicaciones y, basándonos en el código java previamente implementado, se codificará una aplicación que albergue el algoritmo biométrico anteriormente explicado y que permita guardar firmas a modo de patrón para, a partir de ese momento, reconocer al usuario cuando firme en el terminal. Esta aplicación servirá para darle cierto carácter práctico al algoritmo así como para testearlo en distintos dispositivos.

2 Biometría

2.1 Breve introducción a la biometría

Biometría es un término general utilizado alternativamente para describir una característica o un proceso.

Como una característica: la biometría es una característica biológica (anatómica y psicológica) y de comportamiento que se puede medir y que puede ser utilizada en el reconocimiento autómata.

Como un proceso: la biometría es un método automático de reconocimiento de individuos, basado en características biológicas (anatómicas y psicológicas) y de comportamiento que se pueden medir.

2.1.1 Historia

La biometría no se puso en práctica en las culturas occidentales hasta finales del siglo XIX, pero era utilizada en China desde al menos el siglo XIV. Un explorador y escritor que respondía al nombre de Joao de Barros escribió que los comerciantes chinos estampaban las impresiones y las huellas de la palma de las manos de los niños en papel con tinta. Los comerciantes hacían esto como método para distinguir entre los niños jóvenes.

En Occidente, la identificación confiaba simplemente en la "memoria fotográfica" hasta que Alphonse Bertillon, jefe del departamento fotográfico de la Policía de París, desarrolló el sistema antropométrico (también conocido más tarde como Bertillonage) en 1883. Éste era el primer sistema preciso, ampliamente utilizado científicamente para identificar a criminales y convirtió a la biométrica en un campo de estudio. Funcionaba midiendo de forma precisa ciertas longitudes y anchuras de la cabeza y del cuerpo, así como registrando marcas individuales como tatuajes y cicatrices. El sistema de Bertillon fue adoptado extensamente en occidente hasta que aparecieron defectos en el sistema - principalmente problemas con métodos distintos de medidas y cambios de medida. Después de esto, las fuerzas policiales occidentales comenzaron a usar la huella dactilar, esencialmente el mismo sistema visto en China cientos de años antes.

En estos últimos años la biométrica ha crecido desde usar simplemente la huella dactilar, a emplear muchos métodos distintos teniendo en cuenta varias medidas físicas y de comportamiento. Las aplicaciones de la biometría también han aumentado, desde sólo identificación hasta sistemas de seguridad y más. [2]

2.1.2 Modalidades biométricas

Reconocimiento es un término genérico, y no implica necesariamente verificación e identificación. Todos los sistemas biométricos realizan reconocimientos para" volver a conocer" a una persona que ya había sido registrada previamente.

Verificación es una tarea por la que el sistema biométrico intenta confirmar la identidad proclamada de un individuo mediante la comparación de una muestra con una o más planillas obtenidas previamente.

Identificación es la tarea mediante la cual el sistema biométrico intenta determinar la identidad de un individuo. Los datos biométricos son comparados contra todos los datos que se albergan en la base de datos. La identificación de "serie cerrada" se explica si la persona existe en la base de datos, la de "serie abierta" (también llamada lista de visión) donde no hay garantía de que la persona esté registrada en la base de datos. El sistema debe determinar si la persona está o no en la base de datos.

Las plantillas biométricas son la representación digital de una característica distintiva de un individuo, representan información extraída de una muestra biométrica. Las plantillas biométricas son lo que se compara en un sistema de reconocimiento biométrico. Las plantillas varían de acuerdo a las distintas modalidades biométricas y sus oferentes. A la característica distintiva antes citada se la denomina indicador biométrico y debe cumplir los siguientes principios:

- Universalidad: cualquier persona tiene que poseer dicha característica
- Unicidad: la existencia de dos personas con una característica idéntica tiene una probabilidad muy pequeña
- Permanencia: no cambia en el tiempo
- Cuantificación: puede ser medida en forma cuantitativa.

El rendimiento de una medida biométrica se define generalmente en términos de:

La tasa de falsa aceptación (FAR, *False Acceptance Rate*) es una estadística que muestra la actuación del biométrico, típicamente cuando opera en la tarea de verificación. En general cuanto más bajo sea el valor de la tasa de falsa aceptación, más alta será la precisión del sistema biométrico. En esta tasa se muestra el porcentaje de número de veces que el sistema produce una falsa aceptación. Es decir cuando un individuo es identificado como usuario de manera incorrecta. El valor depende de lo sensible del área o sistema a proteger y de la necesidad del usuario. A nivel de fabricantes la mayoría tienen esta tasa entre el 0.0001% y el 0.1%. La tasa dada normalmente asume intentos pasivos del impostor.

La tasa de falso rechazo (FRR, *False Reject Rate*) es la probabilidad de que un dispositivo rechace una persona autorizada. Comercialmente su valor varía entre el 0.00066% y el 1%.

El punto de intersección entre la tasa de falsa aceptación y la tasa de falso rechazo se conoce como la tasa de error igual (EER, *Equal Error Rate*), algunas veces se llama tasa de error cruzada (CER, *Crossover Error Rate*) y se puede ver en la figura 2.1. Es una estadística que muestra la actuación del biométrico, típicamente cuando opera en la tarea de verificación. En general cuanto más bajo sea el valor de la tasa de error igual, más alto será la precisión del sistema biométrico.

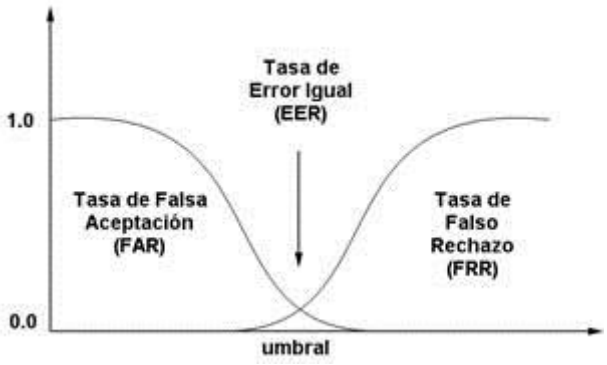

Figura 2.1- Rendimiento de una medida biométrica

Como se puede ver en la figura 2.2, los sistemas biométricos usan una combinación de factores corporales y de comportamiento, la clasificación de las técnicas biométricas facilita su estudio. La medición de las características físicas de un individuo es conocida como biometría estática. Los principales estudios y aplicaciones de la biometría estática están basados en la medición de huellas digitales, geometría de la mano, iris, forma de la cara, retina y venas del dorso de la mano. Existen

también, pero menos usadas, las técnicas biométricas basadas en forma de las orejas, temperatura corporal y forma del cuerpo.

Por otro lado, la medición de los rasgos del comportamiento de un individuo es conocida como biometría dinámica. Los principales estudios y aplicaciones de la biometría dinámica están basados en el patrón de voz, firma manuscrita, dinámica del tecleo, cadencia del paso y análisis gestual.

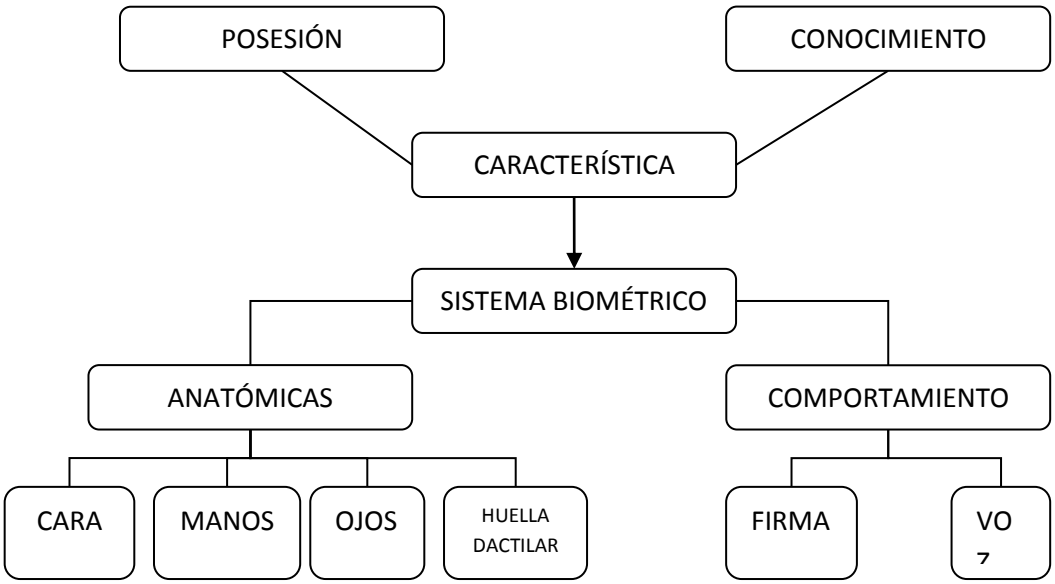

Figura 2.2- Clasificación de sistemas biométricos [3]

Sin tener en cuenta la clasificación anterior, las técnicas biométricas se pueden clasificar atendiendo a cual es la característica observada y, aunque la autenticación de usuarios mediante métodos biométricos es posible utilizando cualquier característica única y mesurable del individuo (esto incluye desde la forma de teclear ante un ordenador hasta los patrones de ciertas venas, pasando por el olor corporal), tradicionalmente ha estado basada en los seis grandes grupos que se citan a continuación:

· Reconocimiento de la huella dactilar.

· Reconocimiento de la cara.

· Reconocimiento de iris/retina.

· Geometría de dedos/mano.

· Autentificación de la voz.

· Reconocimiento de la firma.

Cada sistema biométrico utiliza un tipo de interfaz para recopilar la información sobre la persona que intenta acceder. Un software especializado procesará esa información en un conjunto de los datos que se pueden comparar con los modelos de los usuarios que se han introducido previamente al sistema. Si el programa encuentra un patrón que se corresponde con el usuario en la base de datos, se confirma la identidad de la persona y se concede el acceso.

En la tabla 1 se muestra una comparativa de los rasgos más generales de la clasificación antes descrita.

Se pueden citar también algunas tecnologías biométricas emergentes tales como: Reconocimiento de movimiento (capturando una secuencia de imágenes para analizar), escaneo de venas (captura imágenes del patrón del flujo sanguíneo) o medidor del pulso sanguíneo (sensores infrarrojos que miden el pulso de la sangre en el dedo).

La utilización de un sistema u otro dependerá de la aplicación para la que se quiera implementar, de sus necesidades, así como del entorno de la misma. [3]

Tecnología	Modo de trabajo	Fiabilidad	Facilidad de uso	Posibles incidencias	Costo	Aceptación usuario
Huella digital	Captura y compara patrones de la huella digital	Muy alta	Alta	Ausencia de miembro	Bajo	Alta
Geometría de la mano	Mide y compara dimensiones de la mano y los dedos	Baja	Alta	Edad, Ausencia de miembro	Bajo	Alta
Retina	Captura y compara patrones de la retina	Baja	Baja	Gafas	Alto	Baja
Iris	Captura y compara los patrones del iris	Baja	Baja	Luz	Muy Alto	Baja
Geometría facial	Captura y compara patrones faciales	Baja	Baja	Edad, Cabello, Luz	Medio	Baja
Voz	Captura y compara cadencia, pitch y tono de la voz	Alta	Media	Ruido, Temperatura, Meteorología	Alto	Media
Firma	Captura y compara ritmo, aceleración y presión de la firma	Alta	Media	Edad, cambios, analfabetismo	Alto	Media

Tabla 2.1- Comparativo de las tecnologías biométricas más comunes. [3]

2.2 Breve introducción a la firma manuscrita

Firma es, según la RAE, el nombre y apellido, o título, que una persona escribe de su propia mano en un documento, para darle autenticidad o para expresar que aprueba su contenido.

2.2.1 Historia

En la Antigua Roma, existía la Manufirmatio, que consistía en una ceremonia en que leído el documento por su autor o discípulo, se colocaba desenrollado y extendido sobre la mesa del escribano y después de pasar la mano abierta sobre el pergamino en actitud de jurar, pero sin hacerlo, se estampaba el nombre, signo, o una o tres cruces, por el autor o el funcionario en su nombre, haciéndolo seguidamente los testigos presenciales. Más que un requisito, la Manufirmatio era parte del espectáculo solemne para simbolizar autenticidad y compromiso.

En la Edad Media, se inscribía una cruz a la que se le añadían diversas letras y rasgos. Estos signos se utilizaban como firma. Debido a que no sabían leer ni escribir, los nobles remplazaron esta práctica con el uso de sellos.

La diferenciación entre “firmas” y “signos” hizo que se empezase a entender que aquellas eran, más que simples “signos”, la inscripción manuscrita del nombre o de los apellidos. En ese tiempo, pocas eran las personas que sabían leer y escribir, por lo que generalmente los particulares estampaban en los documentos que suscribían diversos signos o sellos, la extensión de la instrucción y el desenvolvimiento de las transacciones comerciales, hicieron que la firma fuera adquiriendo la importancia y uso que con el transcurso del tiempo se fue consgrando como un símbolo de identificación y de enlace entre el autor de lo escrito o estampado y su persona. [4]

2.2.2 Características

La función tradicional de una firma es testimonial, esta trata de dejar constancia de la procedencia del documento (su identidad) y de la intención del individuo con respecto a ese documento.

Por ejemplo, el papel de una firma en muchos contratos no es limitarse a la demostración de la identidad de la parte contratante, sino también proporcionar evidencia de que el individuo queda informado.

En muchos países, las firmas pueden ser presenciadas y grabadas en presencia de un notario público para dotarlas de fuerza legal adicional. En algunos países, las personas analfabetas marcan con una huella digital en documentos legales en lugar de utilizar una firma manuscrita.

La firma manuscrita o autógrafa tiene las siguientes características:

-Identificativa: Sirve para identificar quién es el autor del documento u obra.

-Declarativa: Significa la asunción del contenido del documento por el autor de la firma. Sobre todo cuando se trata de la conclusión de un contrato, la firma es el signo principal que representa la voluntad de obligarse a cumplir algo.

-Probatoria: Permite identificar si el autor de la firma es efectivamente aquél que ha sido identificado como tal en el acto de la propia firma.

Podemos además tener en cuenta una serie de elementos en las firmas manuscritas:

-**Elementos formales**, son aquellos elementos materiales de la firma que están en relación con los procedimientos utilizados para firmar y el grafismo mismo de la misma:

La firma como signo personal. La firma se presenta como un signo distintivo y personal, ya que debe ser puesta de puño y letra del firmante. Esta característica de la firma manuscrita puede ser eliminada y sustituida por otros medios en la firma electrónica.

El *animus signandi*. Es el elemento intencional o intelectual de la firma. Consiste en la voluntad de asumir el contenido de un documento, que no debe confundirse con la voluntad de contratar.

-**Elementos funcionales**. Tomando la noción de firma como el signo o conjunto de signos, podemos distinguir una doble función:

Identificadora. La firma asegura la relación jurídica entre el acto firmado y la persona que lo ha firmado. La identidad de la persona nos determina su

personalidad a efectos de atribución de los derechos y obligaciones. La firma manuscrita expresa la identidad, aceptación y autoría del firmante. No es un método de autenticación totalmente fiable. En el caso de que se reconozca la firma, el documento podría haber sido modificado en cuanto a su contenido -falsificado- y en el caso de que no exista la firma autógrafa puede ser que ya no exista otro modo de autenticación. En caso de duda o negación puede establecerse la correspondiente pericial caligráfica para su esclarecimiento.

Autenticación. El autor del acto expresa su consentimiento y hace propio el mensaje:

-Operación pasiva que no requiere del consentimiento, ni del conocimiento siquiera del sujeto identificado.

-Proceso activo por el cual alguien se identifica conscientemente en cuanto al contenido suscrito y se adhiere al mismo. [5]

2.2.3 Algoritmo DTW

El reconocimiento automático de firma dinámica suele abordarse mediante uno o varios algoritmos pertenecientes a los siguientes tipos de técnicas de clasificación de patrones:

- Métodos basados en alineamiento de características, que consisten en la comparación entre la muestra de entrada y un prototipo de referencia almacenado (denominado plantilla). El método perteneciente a esta categoría más empleado para el reconocimiento de firma dinámica es el Alineamiento Temporal Dinámico (DTW, *Dynamic Time Warping*).

- Métodos basados en modelos estadísticos: con estos métodos los patrones de referencia son usados para construir un modelo probabilístico/estadístico. Los modelos ocultos de Markov (HMM, *Hidden Markov Models*) son el algoritmo de referencia de este tipo en el campo de reconocimiento de firma dinámica.

- Métodos basados en fronteras de decisión: estos algoritmos se basan en la creación de fronteras entre clases a partir de la optimización de un determinado criterio de error entre los resultados buscados y los obtenidos. A este tipo pertenecen las redes neuronales, los árboles de decisión y las máquinas de soporte vectorial.

Las dos primeras categorías anteriores presentan la ventaja respecto a la tercera de no necesitar muestras negativas para la generación de los prototipos de referencia. Este hecho tiene especial importancia en firma ya que, incluso por implicaciones de tipo legal, no deberían utilizarse imitaciones de firmas para la creación de los prototipos de referencia.

De hecho será el primer método, el de alineamiento de características, el que se desarrolle y se implemente en el presente proyecto.

Las técnicas basadas en alineamiento de características miden la distorsión o deformación que es necesario realizar sobre una firma para alinearla con otra de referencia. En el caso de firmas dinámicas el alineamiento se realiza entre los puntos obtenidos al capturar la firma. Así pues, para alinear dos firmas de un mismo autor, se requerirá menos deformación que cuando se alinean dos firmas pertenecientes a personas distintas.

La técnica de alineamiento más utilizada en el reconocimiento de la firma dinámica es el alineamiento temporal dinámico (DTW, *Dynamic Time Warping*). El algoritmo DTW permite realizar un alineamiento óptimo entre dos secuencias de vectores de distinta longitud mediante programación dinámica. De dicho alineamiento se obtiene una medida de distancia entre los dos patrones temporales [6]

El Alineamiento Temporal Dinámico, puede considerarse actualmente como el método de referencia en el campo del reconocimiento biométrico basado en firma dinámica y será el DTW el método que se utilizará más adelante en este proyecto. Este conforma un sistema que tiene cuatro bloques:

- -**Sensor**: Es el dispositivo que capturará la firma dinámica del usuario, en este proyecto para los primeros test hará de sensor el *Android Virtual Device*, del que se hablará más adelante y para los

test finales del algoritmo será un terminal móvil real el que cumpla este papel.

- -**Características**: Una firma dinámica S es representada mediante una secuencia temporal finita de N vectores de características, donde N depende de la duración temporal real de la firma y de la frecuencia de muestreo del dispositivo de captura, en nuestro caso, sobre todo, de la capacidad de procesado del terminal móvil. En el sistema que se programará en este proyecto se incluirán entre las características la primera y la segunda derivada de las variables obtenidas del sensor.

- -**Normalización**: Se aplicaron dos métodos de normalización sobre el conjunto final de características basados en una normalización geométrica, para situar el origen de coordenadas en el centro geométrico de la firma, siguiendo la ecuación 1 y una normalización estadística(ecuaciones 2,3 y 4), con la que se pretende que su ponderación relativa dentro del vector de características sea la misma para todas.

$$f_k^{N1} = f_k^p - u_k^p$$

Ecuación 2.1-Situa los puntos en el origen

$$f_k^{N2} = (f_k - u_{k)/\sigma_k}$$

Ecuación 2.2- Ecuación de normalización

$$u_k = {(\sum_{t=1}^{N} f_{k,t})}/{N}$$

Ecuación 2.3- Media de los parámetros

$$\sigma_k = \sqrt{\left(\sum_{t=1}^{N}(f_t - u_k)^2\right) \Big/ (N-1)}$$

Ecuación 2.4- Desviación típica

- **-Comparador:** Se almacenan varias instancias de firma del cliente para tener en cuenta la variabilidad que pueda tener el mismo a la hora de repetir su propia firma, aunque trate de hacerla exactamente igual. Después de esto se calculan las distancias entre pares de firmas.

2.2.4 Base de datos MCyT

En la actualidad se necesitan bases de datos para testear sistemas de reconocimiento biométrico. En el proceso de desarrollar y evaluar aplicaciones biométricas surge precisamente la necesidad de este tipo de bases de datos. Existen algunas bases de datos públicas como MilDea, Biosecure o FVC200x Fingerprint databases. En este trabajo se ha elegido la base de datos bimodal MCYT para representar individuos y evaluar el funcionamiento del sistema.

La base de datos MCYT, es un proyecto promovido por el Biometric Research Laboratory de la Universidad Politécnica de Madrid. Esta base de datos incluye huellas dactilares además de firmas.

El ámbito de utilidad de la base de datos MCYT implica principalmente la evaluación en el diseño de sistemas de reconocimiento automático en aplicaciones civiles, comerciales y forenses, permitiendo el desarrollo y evaluación de algoritmos de reconocimiento biométrico basados en características biométricas simples, además de la fusión de ellas en un sistema de reconocimiento bimodal.

En la base de datos MCYT se obtienen por cada individuo 25 firmas verdaderas y 25 falsificaciones. Se incluye en la base de datos tanto la información estática (imagen de la firma escrita) como la dinámica (trayectoria, presión y azimut/altitud del bolígrafo).

3 Lenguajes de programación utilizados

3.1 Programación en Java

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. La memoria es gestionada mediante un recolector de basura.

Se ha escogido este lenguaje de programación por ser uno de los más utilizados en la industria de la programación hoy día, también por su similitud con C cosa que puede ayudar a minimizar la curva de aprendizaje en caso de conocer este lenguaje y finalmente por el uso del mismo por parte del sistema operativo que más está creciendo en la industria de la telefonía móvil, Android, gracias a esto se podrá portar una versión del algoritmo, en forma de aplicación, a este sistema.

3.1.1 Historia

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada the Green Project en Sun Microsystems en el año 1991. El equipo (Green Team), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

El lenguaje se denominó inicialmente Oak (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse Green tras descubrir que Oak era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a Java.

El término Java fue acuñado en una cafetería frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: James Gosling, Arthur Van Hoff, y Andy Bechtolsheim. Otros abogan por el siguiente acrónimo, Just Another Vague Acronym ("sólo otro acrónimo ambiguo más"). La hipótesis que más

fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana, de ahí que el icono de java sea una taza de café caliente. A pesar de todas estas teorías, el nombre fue sacado al parecer de una lista aleatoria de palabras.

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión de tres días entre John Gaga, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, de que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era Write Once, Run Anywhere (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar.

Java se ha convertido en un lenguaje con una implantación masiva en todos los entornos (personales y empresariales). El control que mantiene Sun sobre éste ha

generado reticencias en la comunidad de empresas con fuertes intereses en Java (IBM, Oracle) y obviamente en la comunidad de desarrolladores de software libre.

La evolución basada en un comité en el que participen todos los implicados no es suficiente y la comunidad demandaba desde hace tiempo la liberación de las APIs y bibliotecas básicas de la JDK.

En diciembre de 2006, Sun Microsystems comenzó el relanzamiento de su plataforma Java bajo la licencia GPL de GNU.

En abril de 2009 Oracle adquirió Sun Microsystems, lo que generó temor en la comunidad ante la posible mercantilización del lenguaje de programación a objetos más popular actualmente. Por ahora Oracle ha seguido manteniendo Java, siendo las versiones posteriores a la 6 bajo su control.

3.1.2 Características

Java es un lenguaje y una plataforma, para satisfacer diversas necesidades, Sun lo organizó en tres ediciones principales: Java SE, Java EE y Java ME.

Como lenguaje, Java tiene una sintaxis que proviene parcialmente de C y C++, tiene muchas similitudes con C, como palabras reservadas, tipos primitivos, muchos de los operadores, aunque Java fue diseñado para ser un lenguaje más seguro que C/C++, lo consigue en parte gracias a la omisión de ciertas características de estos lenguajes. Por ejemplo, Java no es compatible con los punteros (variables que contienen direcciones) y no nos permite sobrecargar los operadores.

Por otro lado Java es una plataforma para la ejecución de programas. En contraste con las que están compuestas de procesadores físicos y de sistemas operativos, la plataforma Java consiste en una máquina virtual y un entorno de ejecución asociado.

La máquina virtual es un procesador de software que presenta su propia serie de instrucciones. El entorno de ejecución asociado está compuesto de bibliotecas para el desarrollo de programas y la interacción con es sistema operativo interno.

El JRE (*Java Runtime Environment*, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o

plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (*Java Development Kit*) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK. [7]

3.1.3 Entorno de programación

Trabajar con las herramientas del JDK desde la línea de comandos no supone ningún problema para proyectos de pequeña envergadura. Sin embargo, esta práctica no es recomendable para trabajos de mayor envergadura que son difíciles de gestionar sin la ayuda de un entorno de desarrollo integrado (IDE).

Un IDE está compuesto de un gestor de proyecto para el tratamiento de archivos, un editor de texto para introducir y editar el código fuente, un depurador para la localización de *bugs* y otras características. Para la realización de este proyecto se contará con la ayuda del IDE Eclipse.

Eclipse es un IDE en código abierto para el desarrollo de programas en Java y en otros lenguajes como C, COBOL, PHP, Perl y Python.

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar aplicaciones. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Tools (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse)

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

3.2 Programación en Android

Para la realización de este proyecto se ha estudiado el lenguaje de programación Android. Debido a que es un sistema relativamente nuevo y emergente, aunque se ha abierto hueco en el mercado de los terminales móviles de manera muy rápida (en la figura 3.1 se puede ver una comparación de las activaciones de terminales Android respecto de la competencia), se ha pensado que era muy interesante redactar e incluir en el presente proyecto un manual de introducción a la programación Android.

Este manual, que se ha incluido en el Anexo A del presente documento, es un paso previo al punto 5, en el que se describirá la aplicación implementada con el algoritmo DTW que se viene tratando en el proyecto.

Table 2
Worldwide Smartphone Sales to End Users by Operating System in 3Q11 (Thousands of Units)

Operating System	3Q11 Units	3Q11 Market Share (%)	3Q10 Units	3Q10 Market Share (%)
Android	60,490.4	52.5	20,544.0	25.3
Symbian	19,500.1	16.9	29,480.1	36.3
iOS	17,295.3	15.0	13,484.4	16.6
Research In Motion	12,701.1	11.0	12,508.3	15.4
Bada	2,478.5	2.2	920.6	1.1
Microsoft	1,701.9	1.5	2,203.9	2.7
Others	1,018.1	0.9	1,991.3	2.5
Total	**115,185.4**	**100**	**81,132.6**	**100**

Source: Gartner (November 2011)

Figura 3.1- Tabla comparativa con altas de Android frente a otros sistemas móviles, obtenida de http://www.tuandroid.com

En nuestro proyecto vamos a programar para la versión 2.1 de Android, esto significa que se va a abarcar prácticamente la totalidad de los dispositivos que hoy día

están activados en el mundo. La figura 3.2 da una idea de como están repartidos los móviles Android según la versión del sistema que tienen instalada.

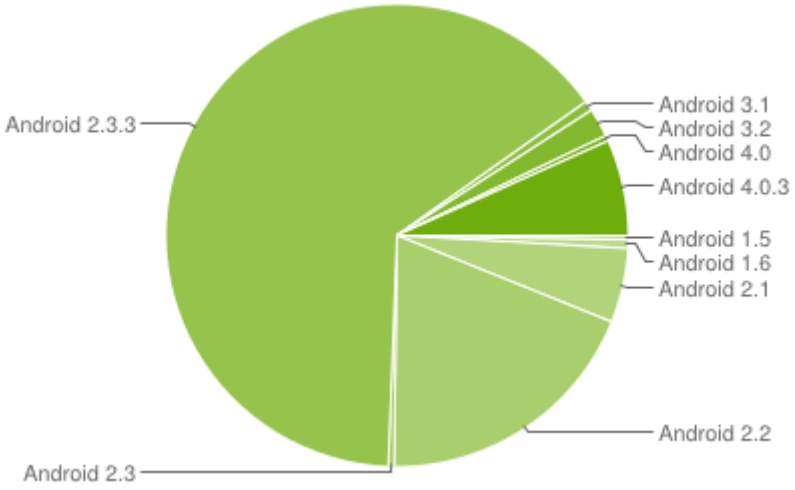

Figura 3.2 Proporción de la versiones de Android activas [8]

4 Implementación del algoritmo DTW en java

En este capítulo se va a detallar la implementación del algoritmo realizada en Java: se verán las principales clases y métodos del programa y se analizará el orden de ejecución del mismo para llegar al score final, que nos permitirá hacer una valoración de lo parecida que es una firma a un conjunto de firmas patrón.

4.1 Módulos

Para la implementación del algoritmo DTW en lenguaje de programación Java, se ha utilizado el entorno de desarrollo integrado Eclipse. Se ha dividido esta implementación en varios archivos .java que se corresponden con los diferentes módulos funcionales del programa, que se explicaran a continuación.

Como se puede ver en la figura 4.1, MainActivity.java que como su propio nombre indica es la actividad principal del programa, crea dos objetos, uno tipo Firma y otro de tipo Variable. Estos objetos, junto con el patrón que crea Alistamiento, se los pasa a Ejecución.java que ya se encarga de ejecutar tanto varios métodos que adaptaran la Firma a nuestras necesidades como el algoritmo DTW del que previamente se ha hablado y que más adelante se explicará en detalle.

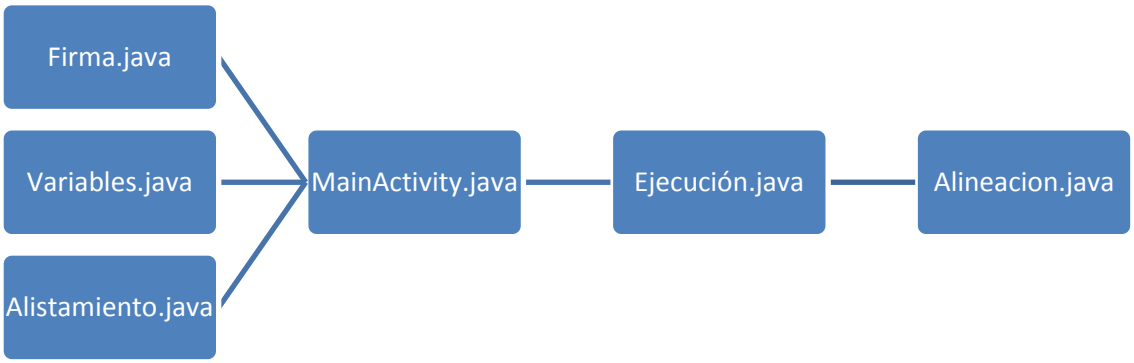

Figura 4.1- Estructura del programa

4.1.1 Explicación de los módulos

En primer lugar se van a explicar los tres objetos más importantes del programa, objetos que se utilizarán y manipularán a lo largo de todo el código, como muestra la parte superior de la figura 4.3, estos objetos son Firma, Variables y Patrón:

Firma.java

Este archivo contiene todo lo necesario para crear un objeto de tipo firma, será un objeto que contendrá todas las variables que necesitaremos para que una firma quede definida (posiciones 'x' e 'y' y presión 'p', con sus respectivas primera y segunda derivada, así como la variable tiempo 't'), para almacenar todo lo necesario se utilizarán las variables con más precisión que tiene Java, los double o variables de punto flotante de doble precisión cuyo tamaño es de 64 bits.

Dentro de esta clase hay también un método llamado imprimir_txt(). Este método crea un archivo de texto que contiene todos los valores de los parámetros que contenga en objeto firma en cuestión.

Variables.java

Este archivo contiene todo lo necesario para crear un objeto de tipo variables, será un objeto que contendrá todas las variables que nuestro programa necesite para caracterizar la firma, los patrones, o cualquier otro aspecto de la configuración del mismo.

Contiene algunas variables que es interesante explicar:

- **scores_norm:** este es un parámetro del algoritmo DTW, deberá tener valor '1', que indicará al algoritmo que debe calcular el 'score' final partiendo de las combinaciones de las firmas de entrada, calculando la media de ese 'score' y lo utiliza para modular los 'scores' de verificación o, por el contrario, valor '0', que no normaliza el 'score' final. (Por defecto vale 1)
- **features:** este es un array de booleanos que marca que parámetros tendrá en cuenta el algoritmo. Su estructura es la marcada por la figura 4.2.

x	y	p	t	dx	dy	dp	d(dx)	d(dy)	d(dp)

Figura 4.2- Array de booleanos *features*

- **num_puntos:** es el número de puntos al que se lleva cada una de las firmas, tanto las firmas patrón como la firma de prueba. (Por defecto vale 256)
- **num_patrones:** este parámetro nos indica el número de patrones que va a tener en cuenta nuestro algoritmo a la hora de hacer las sucesivas comparaciones en los cálculos. (Por defecto vale 3)

Patron.java

Este archivo contiene todo lo necesario para crear un objeto de tipo patrón, este objeto patrón contiene un array de 'firmas patrón' y crea un objeto Dp con los parámetros del grupo de 'firmas patron' (media, desviación típica, mínimo y máximo).

Después de conocer los tres objetos que conformaran nuestra firma, se explica el funcionamiento del archivo principal de la aplicación, el que se encarga de crear los tres objetos anteriores además de iniciar el algoritmo:

MainActivity.java

Este es el archivo principal de la aplicación, es la puerta de entrada a esta y se encarga de gestionar las llamadas al resto de archivos así como de crear gran parte de los principales objetos.

Lo primero que hace MatlabActivity.java, después de crear un objeto de variables, es leer la firma de prueba, en realidad leemos un archivo de firma .txt, pero nos sirve como si el usuario hubiera firmado y de esta manera podremos comprobar de forma más rápida el correcto funcionamiento del algoritmo. Este archivo procederá de la base de datos MCYT, y nuestro programa tendrá en cuenta los campos de coordenadas x e y, el tiempo y el campo presión, que se corresponden con las columnas número 1,2,3 y 7 de los ficheros de la base de datos respectivamente.

Cuando ya se tiene creado un objeto de tipo Firma, con los datos del .txt almacenados. Ahora se llama al método leer_patron(), que creará un array de objetos firma de la longitud correspondiente al número de patrones que queramos leer(este dato está almacenado en el objeto variables), lo que nos proporcionará las que llamaremos firmas patrón leídas de varios archivos .txt procedentes de la base de datos MCYT.

Con todos los objetos ya preparados, se llama al método evaluación(), en el que primero se llamará a Alistamiento para que nos devuelva el patrón creado, a partir de las firmas patrón, y posteriormente se llamará a ejecución.java.

A partir de los resultados de ejecución.java, se imprime un archivo .txt con el resultado del algoritmo, el patrón dps. Este patrón nos indicará la similitud relativa, según el algoritmo, entre el conjunto de firmas patrón y la firma de prueba.

Existen tres métodos, equiespaciado(), derivar() y normdatos(), que son una parte importante del programa, parte media de la Figura 4.3, y además la base del algoritmo DTW. Estos métodos son utilizados por Ejecucion.java y Alistamiento.java y a continuación se detalla el funcionamiento de los mismos:

Método equiespaciado()

El objetivo de este método es trabajar en todas las firmas con el mismo número de puntos y, ya de paso, que sea un número de puntos razonable de acuerdo a nuestro algoritmo.

El método esquiespaciado recibe un objeto de tipo firma y un entero procedente del objeto variables que contiene el número de puntos que queremos 'equiespaciar' y devuelve un objeto de tipo firma con el número de puntos deseado y 'equiespaciados' en el tiempo.

En primer lugar, el método fija el instante inicial y el instante final de nuestra medida temporal y traza entre estos dos puntos tantos otros como le hayamos pedido con la variable de entrada, todos ellos con una distancia siempre constante entre sí.

Después, partiendo de los valores de tiempo nuevos y de los antiguos, se interpola linealmente cada uno de los parámetros del objeto firma, reduciendo todos los parámetros al número de puntos deseado.

Método derivar()

El objetivo de este método es calcular las derivadas de nuestros parámetros.

Este método recibe un objeto firma y las variables. En primer lugar, se suavizan las coordenadas x e y además de la presión y a continuación se calculan la primera y la segunda derivada.

Finalmente el método devuelve un objeto de tipo firma, con tiempos ya equiespaciados y con el número de puntos deseados, gracias al anterior método, y con diez parámetros ya calculados: *x*, *y*, *t*, *p*, *x'*, *y'*, *p'*, *x''*, *y''* y *p''*.

Método normdatos()

El objetivo de este método es normalizar cada uno de los parámetros de un objeto firma que reciba, se calcula la media y la desviación típica teniendo en cuenta todos lo puntos de cada uno de los parámetros, y posteriormente se normaliza utilizando la ecuación que ya habíamos citado cuando se explicaron los fundamentos teóricos del algoritmo DTW (ecuación 2.2).

La parte que falta por analizar del programa no es sino el algoritmo en sí, que, aunque también estaría formado por los tres métodos anteriormente descritos, es desde las siguientes clases desde donde se llama a estos métodos:

Alistamiento.java

Esta clase es la encargada de crear un objeto patrón a partir de una serie de firmas patrón que recibe en forma de array de objetos firma. Se detalla el orden de ejecución del programa en el diagrama de la Figura 4.4.

Lo primero que hace es normalizar cada una de las firmas patrón mediante una llamada a los métodos equiespaciado(), derivar() y normdatos(). A continuación, se realizan las llamadas suficientes a Alineacion.java como para comparar dos a dos todas las firmas patrón. Alineación.java nos proporciona un parámetro dps por cada llamada, el programa va almacenando dichos valores en un array. Después de esto el programa da valor al patrón mediante el array de firmas patrón y el valor mínimo, medio, máximo y de desviación típica del array de dps creado.

Ejecucion.java

Esta clase es, en parte, parecida a Alistamiento.java, ya que la comparación entre firmas patrón que hace Alistamiento.java mediante Alineacion.java para conseguir sacar el objeto patrón, Ejecucion.java lo hace entre cada una de las firmas patrón y la firma de prueba, para verificar si es correcta. Después de esto, se calcula el score final, partiendo del score que aporta el patrón y, por otro lado, del que aporta la comparación del patrón con la firma de prueba.

Alineacion.java

Esta clase, que se corresponde con la parte inferior de la figura 3.3, utiliza a su vez las clases Calcular_d y Calcular_g para devolver un parámetro que marque la diferencia entre los dos objetos firma que recibe. Este parámetro es el último elemento de la Matriz que devuelve la clase Calcular_g. A continuación se definen ambas clases:

Calcular_d

Calcula una matriz de distancias, mediante un algoritmo basado en los cálculos del artículo de Hiroaki Sakoe y Seibi Chiba. [6]

Calcular_g

Esta clase parte de la matriz que devuelve Calcular_d y, mediante un algoritmo basado también en los cálculos del artículo de Sakoe y Chiba devuelve una segunda

matriz. De la matriz devuelta por esta clase, el elemento más importante será el último elemento de la última fila. A este elemento será al que llamemos score. [6]

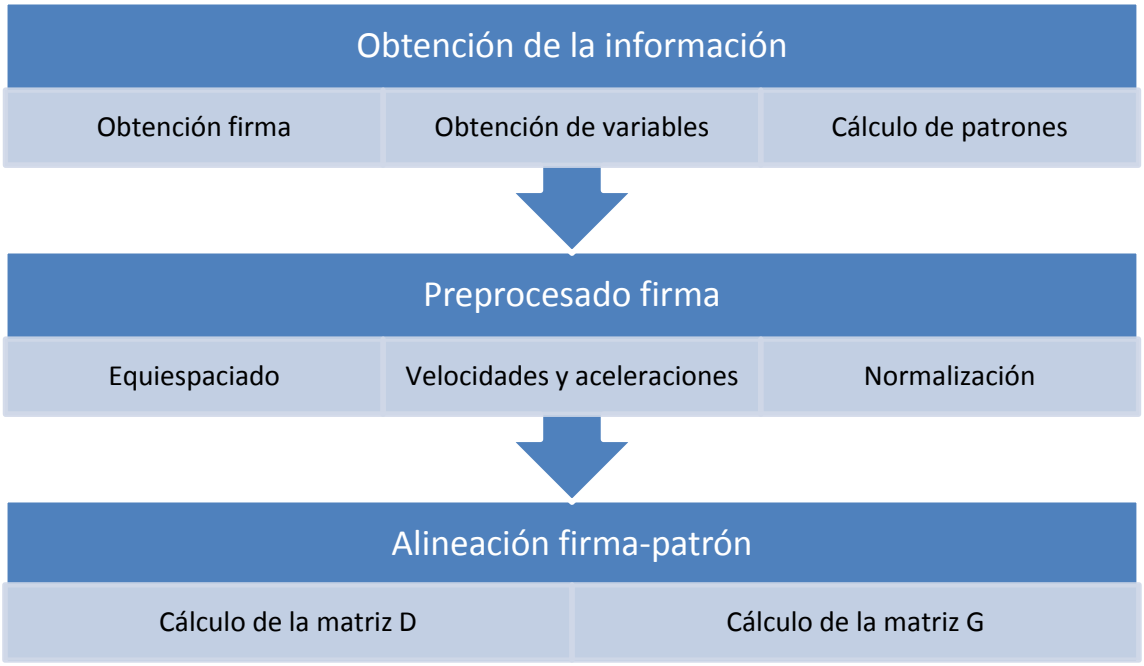

Figura 4.3- Esquema detalle del código

A lo largo del código se han implementado una serie de líneas y métodos cuya función es probar los resultados parciales que va dando el propio código. En los capítulos de resultados y de conclusiones se hablará de los resultados que han ofrecido estos métodos, pero será ahora cuando se comente su estructura, localización y función:

El primer ejemplo de código para pruebas lo tenemos en la propia clase Firma, que contiene un método que al ser llamado crea un archivo de texto que contiene los valores de los parámetros del objeto en el instante en que se realiza la llamada. Tenemos casos de llamadas a este método de apoyo en Alistamiento.java y Ejecucion.java, justo después de cada una de las llamadas a los métodos equiespaciado(), derivar() y normdatos().

Siguiendo el código, encontramos otro caso de método para pruebas en Ejecucion.java, después de la llamada a Alineacion.java, con las matrices D y G ya

creadas, se realiza una llamada a un método implementado en este mismo archivo y que se llama imprimir_matriz(). Se encarga, como su propio nombre indica, de crear un archivo de texto con cada uno de los valores de la matriz que recibe. Las matrices que imprimirá nuestra implementación serán la matriz d y la matriz g.

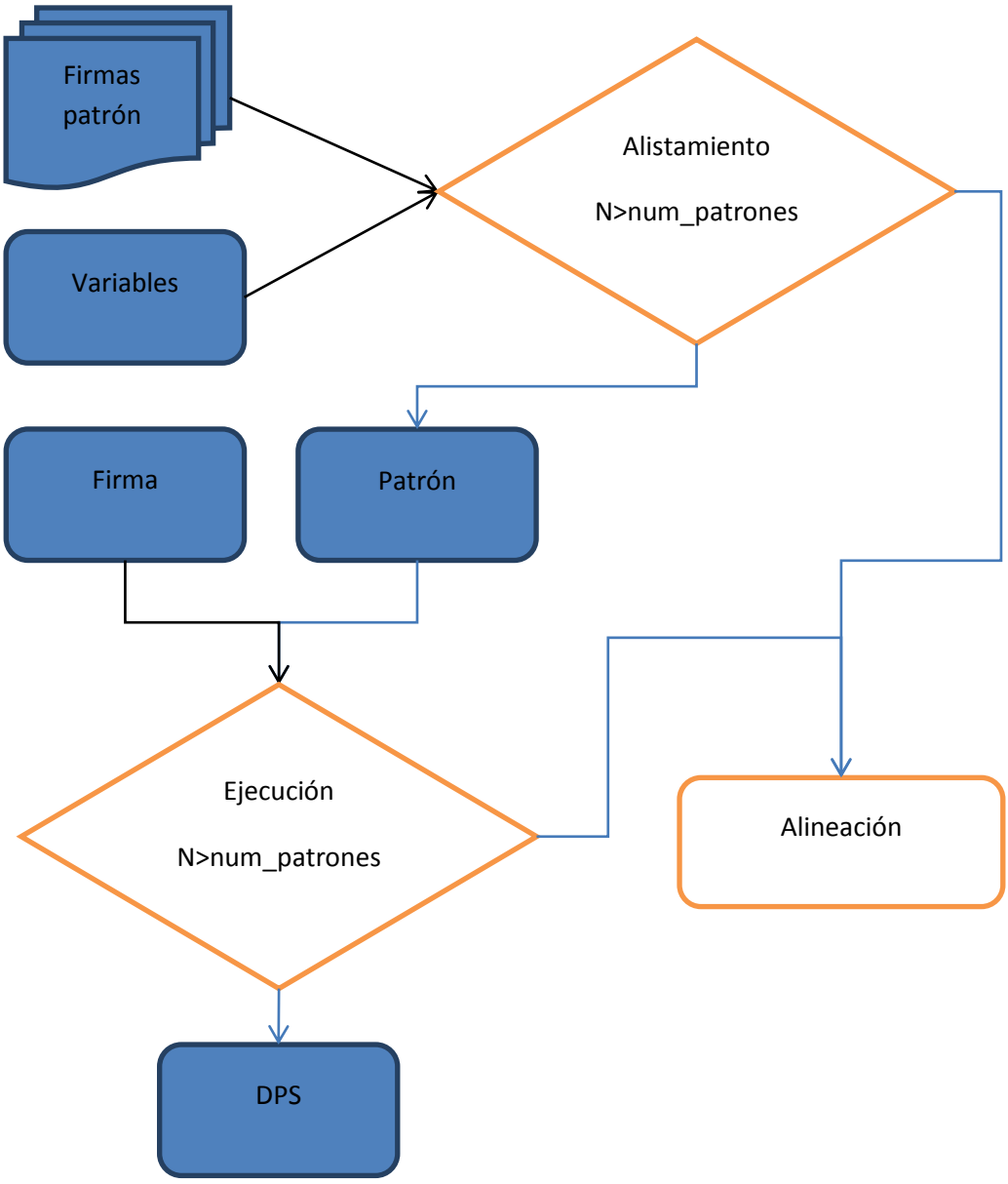

Figura 4.4- Diagrama de flujo del programa

5 Desarrollo de la aplicación

Al margen de las pruebas hechas con Eclipse del algoritmo DTW, desde el principio de este proyecto se estudió la posibilidad de probar el algoritmo Java mediante la implementación de una aplicación móvil, en este caso de una aplicación Android.

5.1 Visión general de la aplicación

Habiendo estudiado ya este lenguaje en el capítulo anterior, ahora pasaremos directamente a la codificación de la aplicación. Esta estará compuesta principalmente de un menú principal, un tablón de firma, que utilizaremos por partida doble, y el algoritmo diseñado en el capítulo 3.

Como se puede ver en la parte superior de la figura 5.1 el punto de partida de la aplicación es el menú principal, aquí el usuario tendrá tres opciones: firmar con el patrón que la aplicación tenga ya pre-cargado, crear un nuevo patrón firmando un número determinado de veces (almacenado en el objeto variables de nuestro código) o salir de la aplicación.

Si seguimos el flujo de la figura 5.1 hacia la opción “patrón”, llegamos hasta un tablón para firmar, el usuario guardará su firma tantas veces como número de patrones tenga el programa pre-configurado tratar. Internamente se estarán guardando los archivos .txt que posteriormente el algoritmo se encargará de leer. Una vez hecha la última firma patrón, la aplicación nos devuelve a la parte superior de la figura 5.1, el menú principal.

Si ahora descendemos de nuevo por el flujo de la figura 5.1 desde el menú principal, pero esta vez marcando la opción “firmar”, llegamos nuevamente a un entorno para firmar. En esta ocasión el entorno está directamente conectado con el algoritmo que hemos codificado en java en anteriores capítulos, y después de firmar y aceptar nuestra firma, el programa se iniciará tomando como fuentes los archivos .txt

que haya generado la opción “patrón” y como firma de prueba el archivo .txt que acaba de generar en este nuevo tablón.

Después de la ejecución el programa tendrá que valorar si la firma es o no correcta. Basándonos en el algoritmo programado, es tarea del programador decidir un nivel de precisión del dps final por encima del cual se considerarán falsas las firmas de prueba. Después de este paso, y siguiendo el diagrama de la figura 5.1, la aplicación nos dice si la firma es correcta, es decir, se ajusta a los patrones preestablecidos, o por el contrario es falsa.

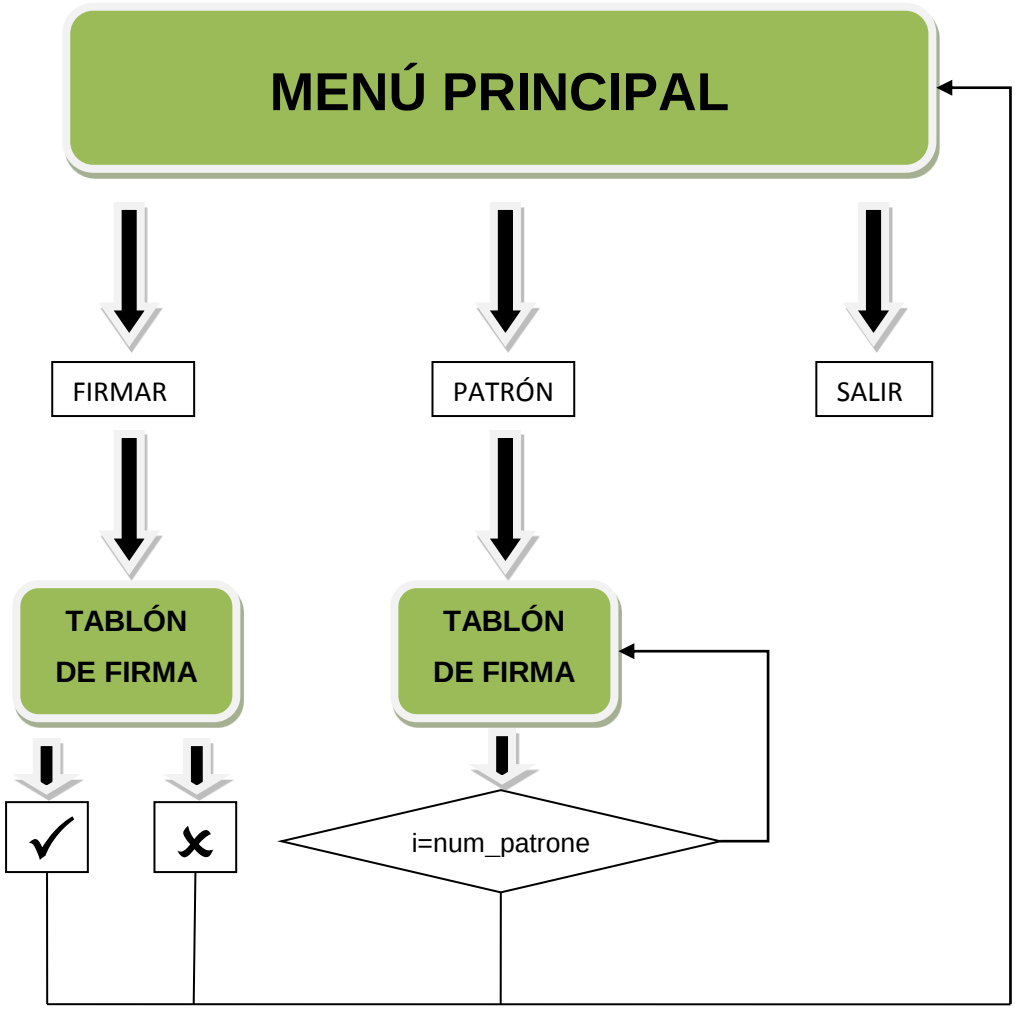

Figura 5.1- Diagrama del funcionamiento de la aplicación

5.2 Explicación de los módulos

La descripción de los módulos de la aplicación Android se realizará separando cada una de las Activity y tomando el algoritmo DTW, ya explicado en el punto 3, a modo de caja negra que toma una firma de prueba y varias firmas patrón como entrada y da un score, denominado dps, como salida.

MenuActivity.java

Esta es la activity que hace de menú principal y permitirá mediante tres botones el acceso a las otras dos actividades o abandonar la aplicación. Se puede ver como es la interfaz de esta activity en la figura 5.2.

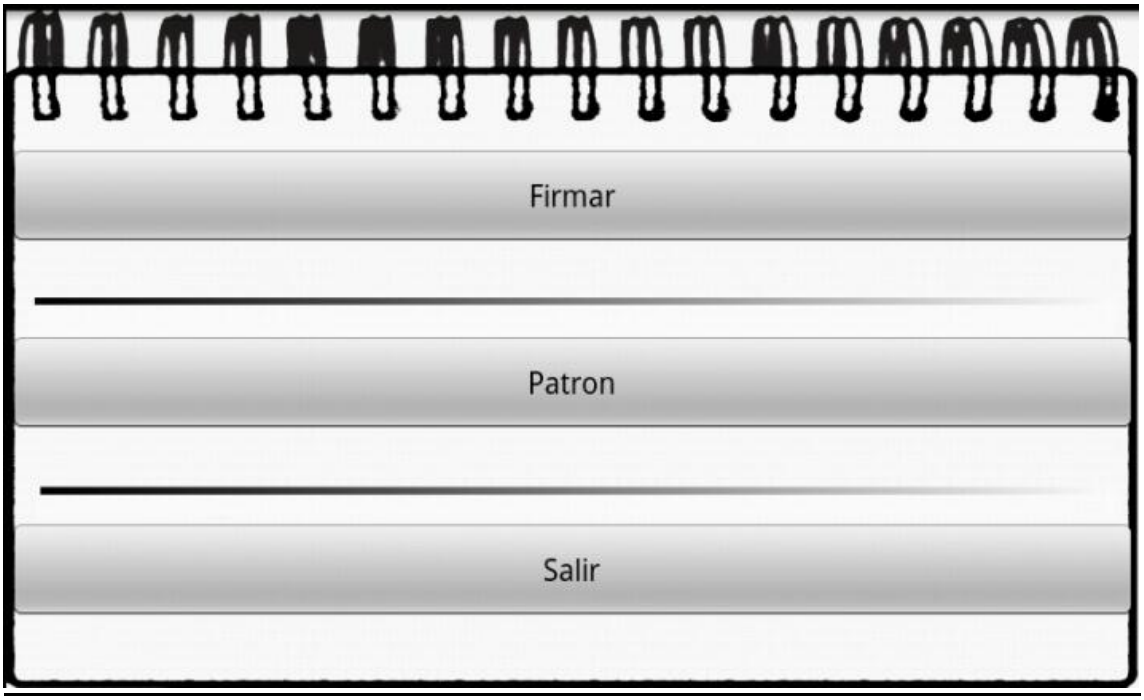

Figura 5.2- Imagen del menu principal de la aplicación

PatronActivity.java

Esta activity tiene por objetivo preparar un número de archivos de texto determinado por el objeto variables que serán el patrón que tomará el algoritmo.

Se apoya en una interfaz gráfica (figura 5.3) formada por un espacio para firmar y dos botones, el de la parte superior derecha permite confirmar la firma que se ha hecho, el de la derecha, por el contrario la deshecha y permite repetirla.

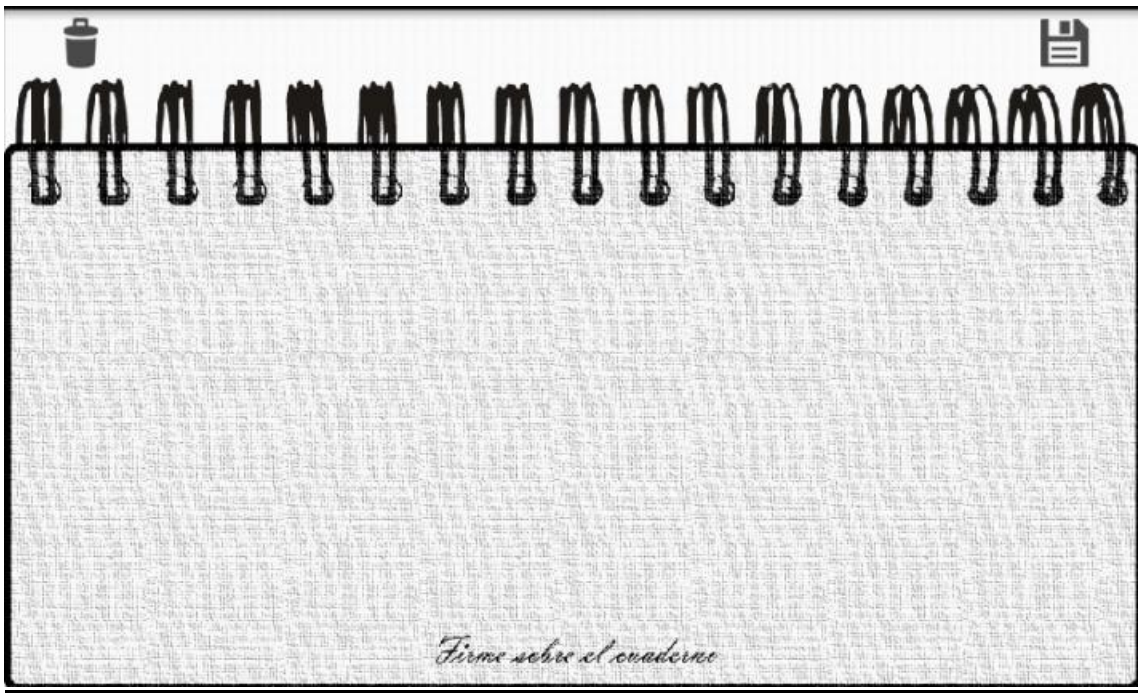

Figura 5.3- Entorno para recoger una firma

FirmarActivity.java

Esta activity es la encargada de utilizar el algoritmo que se ha programado en Java. En primer lugar parte de los ficheros de texto creados en la activity PatronActivity.java, a partir de ellos, de una firma que obtiene mediante su interfaz, que es un entorno de firma similar al de la activity antes explicada (figura 5.3) y de un objeto de variables predefinido por el programador, se llama al algoritmo DTW.

Ahora se explicará con mayor grado de detalle como se ha conseguido crear un panel de firma en Android, de modo que se puedan captar los puntos que va tocando el usuario y, a partir de ellos crear un fichero de texto con la firma.

En primer lugar se extiende la clase View de Android, aquí se aloja un canvas y se captan los movimientos del usuario mediante MotionEvent.ACTION_DOWN, MotionEvent.ACTION_MOVE y MotionEvent.ACTION_UP.

Todos los puntos captados con las llamadas que vayan recibiendo los métodos anteriores se irán guardando en un fichero .txt que conformará la firma o firma patrón dependiendo de a cuál de las dos activity no estemos refiriendo.

6 Resultados

Durante la ejecución de este proyecto se han ido haciendo numerosas pruebas con las que se ha llegado a diversos resultados. Estas pruebas se basan en la comparación de resultados entre el algoritmo que se ha programado en lenguaje Java, y el mismo algoritmo, del que se partía, codificado como función Matlab. Para ello, sabiendo que uno de los siguientes objetivos sería probar el algoritmo como aplicación Android, se ha utilizado un dispositivo Android, en este caso un dispositivo virtual (mediante Android Virtual Device), para realizar las pruebas.

Cada uno de los módulos que muestra la Figura 6.1 se ha ido probando paralelamente en Matlab y Eclipse, y no se ha procedido con la codificación del siguiente módulo hasta que la comparación no daba resultados similares (con un margen de error aceptable). Para ello se ha utilizado una pequeña muestra de las firmas contenidas en la base de datos MCyT.

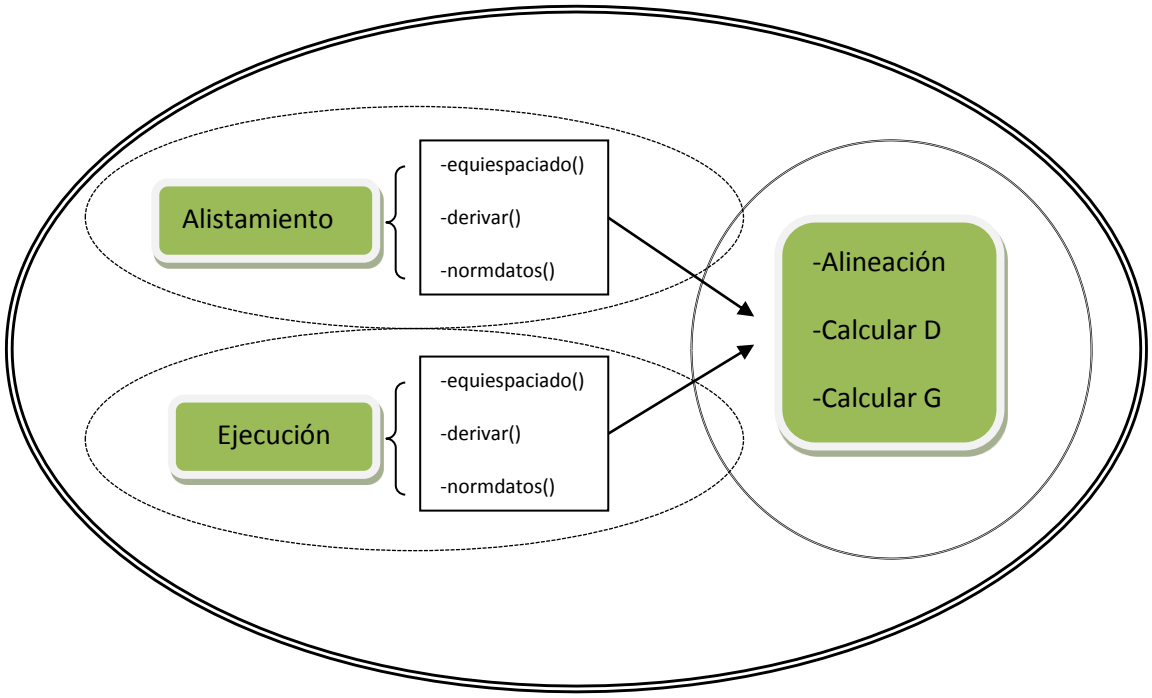

Figura 6.1- Pruebas hechas en el algoritmo

En primer lugar se ha realizado una comparación de los resultados que da Matlab después de pasar las firmas por los métodos equiespaciado(), derivar() y normdatos() frente a los resultados que da el programa en Java (módulos con

bordeados con línea discontinua en la figura 6.1). Este proceso dio algunas complicaciones a la hora de conseguir la máxima precisión en los resultados, pero finalmente se consiguieron unas diferencias del orden de, a lo sumo, 1*10e-12 unidades. Para realizar esta prueba se utilizaron cuatro ficheros de la base de datos MCyT: “U0001_S003_F_T1” (firma falsa entrenada), “U0001_S003_G_T0” (será el patrón), “U0001_S004_G_T0” (firma genuina) y “U0005_S002_G_T0” (firma falsa aleatoria).

La siguiente prueba que se hizo fue la de la función Alineacion.java, después de que los datos hubieran pasado ya por los tres métodos antes descritos. Esta prueba se correspondería con la parte que envuelve la línea simple continua de la Figura 6.1. Esta prueba consistió en hacer tres ensayos pasándole a Alineación.java la firma (normalizada) utilizada como patrón (U0001_S003_G_T0) y una de las otras firmas por cada uno de los tres ensayos (U0001_S003_F_T1, U0001_S004_G_T0 y U0005_S002_G_T0). Hay que decir que en realidad en esta prueba se hicieron seis ensayos, ya que por cada uno de los tres ensayos antes descritos se observaron los resultados con features activando solamente ‘x’ e ‘y’ y por otro lado con features activando ‘y’, ‘t’, ‘dy’ y ‘ap’. En la tabla 6.1 se ha hecho un resumen de los resultados de la prueba, ofreciendo como resultado la diferencia entre Matlab y el programa en Java de la última posición de la Matriz G. Esta es la posición que nos interesa para la prueba ya que además de ser la que mayor error acumulado tiene, es la que utilizaremos como score de la comparación.

Firma que se compara con el patrón	Features	Resultado (|Matlab-Java|)
U0001_S003_F_T1	‘x’ e ‘y’	1*10-6
U0001_S004_G_T0	‘x’ e ‘y’	1*10-6
U0005_S002_G_T0	‘x’ e ‘y’	1*10-6
U0001_S003_F_T1	‘y’, ‘t’, ‘dy’ y ‘ap’	1*10-4
U0001_S004_G_T0	‘y’, ‘t’, ‘dy’ y ‘ap’	1*10-4
U0005_S002_G_T0	‘y’, ‘t’, ‘dy’ y ‘ap’	0.0025

Tabla 6.1-Resultados de comparar el patrón (U0001_S003_G_T0) con diferentes firmas

Estudiando los resultados de la tabla 6.1 se puede decir que en el caso de los tres primeros son muy aceptables, de los tres siguientes podemos decir que aunque, a priori, pueda parecer que no son buenos resultados, en el programa real después de esta función se divide el score entre el score del patrón por lo que el error disminuye significativamente, a esto habría que añadir que en el caso de los tres últimos

resultados, se han seleccionado features aleatorias sin significado real, únicamente para probar el programa.

La última de las pruebas que se ha realizado ha sido la del algoritmo en su totalidad, comparando el score final que da Matlab con el que da Java para seis firmas de prueba frente a un patrón formado por tres firmas. Esta prueba se correspondería con la totalidad de la Figura 6.1 (Parte bordeada con línea doble y continua). Los archivos leídos para esta prueba (extraídos de la base de datos MCyT) han sido los siguientes:

-Patrón: U0001_S001_G_T0.txt, U0001_S002_G_T0.txt y U0001_S003_G_T0.txt

-Firmas: Genuinas: U0001_S004_G_T0.txt y U0001_S005_G_T0.txt

-Falsas entrenadas: U0001_S001_F_T1.txt y U0001_S002_F_T1.txt

-Falsas aleatorias: U0020_S001_G_T0.txt y U0030_S001_G_T0.txt

Los resultados de la comparación son, como se puede ver en la tabla 6.2, muy buenos, ya que existe una diferencia despreciable entre los resultados de Matlab y Java, esto hace que demos por buena esta versión del programa, entre otras cosas, porque cuando se fije un score a partir del cual el programa habrá de considerar como falsa una comparación, nunca se pondrá un número con tantos decimales como para notar la diferencia entre el algoritmo Matlab y el programado en Java.

6.1 Problemas encontrados

Durante la realización de las pruebas del código se encontraron algunos problemas. Unos por el hecho habitual del surgimiento de errores a la hora de programar, estos se fueron depurando poco a poco y otros por el problema que constituye el hecho de portar un código de un lenguaje a otro, más problemático aun si estamos portando un algoritmo matemático desde un entorno diseñado para las matemáticas como es Matlab a otro que no lo es tanto como Java, aunque hay que decir que este último posee muchas librerías matemáticas de las que nos hemos servido para ir implementando el algoritmo.

El primer problema que surgió en las pruebas fue a la hora de leer los archivos de texto de la base de datos MCyT, el programa en Java está diseñado para escribir

archivos de texto y leerlos hasta que encuentre un “null”, existía entonces un problema al leer los archivos de firma de MCyT y al final se llego a la conclusión de que era porque estos archivos no finalizaban de la misma forma que el programa en Java esperaba. Lo bueno de los ficheros de texto de la base de datos MCyT es que su primera línea indica el número de puntos que almacena el fichero, por lo que la solución para leer estos archivos de firma durante las pruebas fue leer esa primera línea y de esa manera interrumpir la lectura del archivo después de ese número de iteraciones.

Firma	Features	Resultados		
		Matlab	Java	Diferencia
U0001_S004_G_T0	‘x’ e ‘y’	0.981342432532678	0.9813424325326776	4.4409e-016
U0001_S005_G_T0	‘x’ e ‘y’	1.071612075230812	1.071612075230812	0
U0001_S001_F_T1	‘x’ e ‘y’	3.141904262191634	3.1419042621916335	4.4409e-016
U0001_S002_F_T1	‘x’ e ‘y’	2.876976950706125	2.876976950706125	0
U0020_S001_G_T0	‘x’ e ‘y’	4.110431416634752	4.110431416634752	0
U0030_S001_G_T0	‘x’ e ‘y’	4.472699483362021	4.472699483362021	0
U0001_S004_G_T0	‘x’, ‘y’, ‘vx’ y ‘vy’	0.850558099892467	0.850558099892467	0
U0001_S005_G_T0	‘x’, ‘y’, ‘vx’ y ‘vy’	0.957829510075056	0.9578295100750558	2.2204e-016
U0001_S001_F_T1	‘x’, ‘y’, ‘vx’ y ‘vy’	2.450122237150615	2.4501222371506146	4.4409e-016
U0001_S002_F_T1	‘x’, ‘y’, ‘vx’ y ‘vy’	2.554620070455044	2.554620070455046	2.2204e-015
U0020_S001_G_T0	‘x’, ‘y’, ‘vx’ y ‘vy’	3.819491306894910	3.8194913068949123	2.2204e-015
U0030_S001_G_T0	‘x’, ‘y’, ‘vx’ y ‘vy’	3.971824817757114	3.9718248177571165	2.6645e-015

Tabla 6.2- Resultados de comparar el score final de Java frente a Matlab

Otro problema del proyecto tuvo lugar durante las pruebas del programa, concretamente al intentar hacer pruebas sobre un dispositivo físico, real y no el

terminal virtual Android (AVD). Si tratamos de probar la versión final de la aplicación no hay ningún problema, pero si lo que queremos es una prueba parcial del algoritmo, es decir, pasarle como se ha comentado anteriormente al programa nuestros propios archivos de texto (de la base MCyT), es entonces cuando surge un importante problema, ya que, Eclipse no nos permite, al navegar por un dispositivo real, guardar archivos a nuestro antojo en las carpetas de las que leen las aplicaciones (esto si nos lo permite hacer con un terminal virtual) por lo que no podremos guardar los archivos de la base de datos MCyT y por consiguiente, no podremos hacer una prueba general con todas las firmas de la base de datos MCyT para conseguir un score óptimo como se había planificado en un principio. Podría pensarse en llevar a cabo esta tarea con la aplicación en un terminal virtual, pero hay que decir que la velocidad de ejecución del programa en el terminal virtual es muy baja y sería una prueba demasiado larga. La ejecución del algoritmo dentro de la aplicación, utilizando un terminal virtual del programa AVD, lanzado desde eclipse y en un equipo con el procesador de doble nucleo T8100 de Intel y 3GB de memoria ram, llevaba entre 30 segundos y 1 minuto de tiempo dependiendo de la prueba.

7 Conclusiones y líneas futuras

Si evaluamos todo el proceso del proyecto, así como los resultados obtenidos, se puede afirmar que se han cumplido los objetivos propuestos. Se ha conseguido afinar lo suficiente un programa, que realiza un algoritmo, en Java para que de resultados con una diferencia despreciable respecto del mismo programa en un entorno matemático como es Matlab.

Posteriormente se ha estudiado el lenguaje de programación móvil Android y se ha visto que era posible crear una aplicación para probar el algoritmo creado en java en un terminal móvil real. Como parte de este estudio se ha desarrollado el manual de introducción a la programación Android incluido en el Anexo A. Este era el segundo objetivo que se había planteado.

Por último se ha implementado la aplicación, gracias a este último paso se ha podido probar el correcto funcionamiento del algoritmo en sus resultados, por lo que se ha cumplido el último objetivo propuesto al principio de este proyecto. Aun así, el comportamiento de la aplicación no es, aún después de depurarla, el que se había previsto. Se ha llegado a la conclusión de que la pantalla de un terminal móvil, al menos del terminal en el que se han hecho las pruebas, combinado con la imprecisión que aporta el hecho de firmar con el dedo, nunca va a parecerse a los dispositivos para firma electrónica para los que está preparado este algoritmo. El funcionamiento con formas simples, ajustando el filtro para el score en torno a 1.2 es satisfactorio, ya sean formas únicas o múltiples, juntas o separadas. Pero en cuando vamos realizando formas más complejas la tolerancia de las firmas es muy difícil de ajustar, puesto que o acepta firmas descaradamente falsas o no aprueba como válidas ni las más parecidas réplicas. Hay que decir que el funcionamiento se puede catalogar como malo únicamente con firmas muy complejas.

Como futuros trabajos aplicables a completar este proyecto queda pendiente el hacer un testeo global de la base de datos MCyT con nuestro algoritmo en Java. Se ha investigado sobre los problemas que se habían tenido para llevar a cabo esta tarea y se ha descubierto que sería posible realizar esta prueba cn un teléfono que permita ser usuario root.

También se ha pensado que puede ser interesante probar el algoritmo en una aplicación diseñada para otros sistemas de dispositivos móviles, por ejemplo para iOs.

Esto permitiría comprobar si, por ejemplo, uno u otro capta mayor número de puntos al hacer una firma semejante.

8 Bibliografía

[1] Óscar Miguel Hurtado, Online Signature Verification Algorithms and Development of Signature International Standards

[2] http://es.wikipedia.org/wiki/Biometr%C3%ADa (Fecha de último acceso 30/07/2012)

[3] http://www.biometrics.gov/ (Fecha de último acceso 30/07/2012)

[4] http://www.alambre.info/2003/12/08/origenes-de-la-firma-autografa/ (Fecha de último acceso 30/07/2012)

[5] REYES KRAFFT, Alfredo. La firma electrónica y las entidades de certificación. Ed. Porrúa. México 2004. (http://www.razonypalabra.org.mx/libros/libros/firma.pdf)

[6] Hiroaki Sakoe y Seibi Chiba, Dynamic Programming Algorithm Optimization for Spoken Word Recognition.

[7] http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n) (Fecha de último acceso 30/07/2012)

[8]http://developer.android.com/intl/es/index.html (Fecha de último acceso 30/07/2012)

[9] http://en.wikipedia.org/wiki/Signature (Fecha de último acceso 30/07/2012)

[10] http://www.biometria.gov.ar/acerca-de-la-biometria/preguntas-frecuentes.aspx (Fecha de último acceso 30/07/2012)

[11]Tapiador Mateos, M. & Sigüenza Pizarro, J.A. (2005). Tecnologías biométricas aplicadas a la seguridad. RA-MA, 1st edn.

[12] Dimauro, et al, Recent Advancements in Automatic Signature Verification (2004)

[13]Jeff Friesen, Java for Android developers (2011)

[14] Hello, Android 2nd Edition (ed burnete).

[15] Joan Ribas Lequerica, Desarrollo de aplicaciones para Android, Anaya

[16] Sayed Hashimi, Komatineni y MacLean, Pro Android 2.

Anexo A

En este anexo se presenta el manual de introducción a la programación Android que se ha redactado para que haya un mayor acercamiento por parte del lector a este lenguaje de programación emergente y relativamente nuevo.

1 Que es Android

Android es un sistema operativo basado en Linux para dispositivos móviles, tales como teléfonos inteligentes o tablets. Fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005. Es el principal producto de la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio.

Android, al contrario que otros sistemas operativos para dispositivos móviles como iOS o Windows Phone, se desarrolla de forma abierta y se puede acceder tanto al código fuente como al listado de incidencias donde se pueden ver problemas aún no resueltos y reportar problemas nuevos.

Programar Android no es una tarea fácil, ya que utiliza el lenguaje de programación Java como base y, apoyándose en este, (no todas las librerías de java estarán disponibles para programar en Android) sigue una serie de directrices para adecuarse lo máximo posible a los terminales móviles y tablets a los que está enfocado este sistema operativo.

Durante la lectura de este manual de introducción a Android y, sobre todo, cuando ya se haya cogido suficiente experiencia como para programar nuevas aplicaciones, es muy recomendable seguir en todo momento, a modo de guía y de manual de apoyo y consulta, la página web oficial de Android, donde viene totalmente detallado y ejemplificado cada uno de los componentes e instrucciones del lenguaje de Google.

A continuación se resumen una serie de componentes de interés, que son la base de la programación Android:

Activity

Las actividades (activity) representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana en cualquier otro lenguaje visual.

View

Los objetos view son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, análoga por ejemplo a los controles de Java o .NET. De inicio, Android pone a nuestra disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados.

Service

Los servicios son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (activity) si se necesita en algún momento la interacción con del usuario.

Content Provider

Un content provider es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los content provider que se hayan definido.

Broadcast Receiver

Un broadcast receiver es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: “Batería baja”, “SMS recibido”, “Tarjeta SD insertada”,…) o por otras aplicaciones (cualquier aplicación puede generar mensajes (intents, en terminología Android) broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).

Widget

Los widgets son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (home screen) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal. También se denomina Widget en el ámbito de la programación a muchos de los elementos que podemos poner en nuestra interfaz gráfica (botones, cuadros de texto...)

Intent

Un intent es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

2 Instalar Android en Windows

Para programar terminales Android primero tenemos que asegurarnos de tener el JDK(Java Development Kit) instalado en nuestro ordenador, ya que no es suficiente con el habitual JRE(Java Runtime Environment) que suele estar en nuestros sistemas, para descargarlo entraremos en la web de Oracle (http://www.oracle.com/), vamos a Downloads>Java for Developers y después seleccionamos Download JDK. Después de esto instalaremos un entorno de programación y finalmente el SDK(Software Development Kit) de Android, así como los plugins necesarios para la interacción entre el entorno y el SDK.

2.1 Eclipse

Para programar y depurar nuestros códigos utilizaremos el entorno de desarrollo java Eclipse. Pueden utilizarse otros entornos de desarrollo, pero se ha elegido este por ser muy completo, gratuito y sobre todo porque es el que utilizan y para el que dan soporte los desarrolladores de Google que han creado y que actualizan Android.

Se recomienda descargar Eclipse de su página web oficial (http://www.eclipse.org/downloads/), eligiendo la versión “Eclipse IDE for java developers”, y seguimos las instrucciones de descarga e instalación del programa. La última versión disponible en el momento de redactar este proyecto es la 3.6.

2.2 Android

Lo siguiente que tenemos que descargar es el SDK de Android. Lo haremos desde la página web oficial (http://developer.android.com/sdk/index.html). Durante la instalación se comprobará si tenemos instalada la versión del JDK adecuada, A continuación se nos solicitará un directorio de instalación, que debemos anotar y recordarlo, ya que lo utilizaremos más adelante en la configuración del ADT (Android Development Tools) que es el plug-in para Eclipse que nos proporcionará las

herramientas necesarias para programar, compilar, ejecutar y depurar nuestras aplicaciones Android.

2.3 Instalación y configuración del ADT Plug-in

El último paso para preparar nuestro ordenador para programar en Android, es instalar el ADT plug-in (Android Development Toolkit). Este software nos permitirá compilar y ejecutar desde eclipse nuestras aplicaciones Android. En el momento de la redacción de este manual (Junio 2012), la última versión disponible del ADT es la 20.0.0. Este plug-in, así como el SDK, van actualizándose periódicamente, ya que los terminales Android van incorporando regularmente nuevas funcionalidades que necesitan un soporte a la hora de programar. Para instalar el plug-in hay que seguir estos pasos (se explica para la versión de Eclipse “Helios SR2”):

1-Iniciar Eclipse, seleccionar la opción Help > Install new software

2-Hacer click en el botón Add...

3-En el formulario que aparece ponemos “ADT Plugin” en el campo “Name” y la siguiente URL en el campo “Location”:

https://dl-ssl.google.com/android/eclipse/

4-Hacer click en el botón Ok.

5-Marcar la casilla que está a la izquierda de “Developer Tools” y hacer click en el botón Next.

6-En la siguiente ventana aparecerá una lista de las herramientas que se van a descargar, hacer click en el botón Next.

7-Leer y aceptar los acuerdos de licencia y hacer click en el botón Finish.

8-Tras la instalación elegiremos la opción de reiniciar Eclipse

Tras reiniciar Eclipse tendremos que configurar el Plug-in de manera que pueda “comunicarse” con el SDK de Android anteriormente instalado. Para ello seguiremos estos pasos:

1-Seleccionar Windows > Prefences

2-Sleccionar Android del panel de la izquierada

3- En el apartado SDK location, hacer click en el botón Browse e introducir aquí la dirección en la que hayamos instalado el SDK de Android (debemos buscar la carpeta android-sdk-windows).

4-Hacer click en el botón Apply y después en Ok.

2.4 Descargar otros componentes y plataformas

Existe una herramienta llamada “Android SDK and AVD manager” que sirve para descargar más componentes del conjunto de utilidades para la programación en android, así como para manejar las distintas plataformas android sobre las que queremos trabajar(diferentes versiones o configuraciones de android de modo que se pueda probar una misma aplicación en distintas configuraciones). Esta herramienta es parte del SDK que ya se instaló y que puede ser ejecutada desde la propia barra de herramientas de Eclipse.

Actualmente se tiene instalado el SDK pero no se tiene disponible ninguna plataforma Android por lo que si queremos programar usando el emulador es obligatorio usar esta utilidad y descargar al menos una plataforma sobre la que trabajar.

Entramos en Eclipse y ejecutamos la utilidad desde el icono de la barra de herramientas (“Android SDK and AVD manager”). De todas las entradas que podemos ver a la izquierda pulsamos “Available Packages”. El programa se conectará a unos servidores externos para obtener informacion sobre los paquetes disponibles en ese momento. Una vez tenga disponible dicha información, la presentará en la lista de la derecha junto con un cuadro de selección para permitir su descarga. Los paquetes básicos que seleccionaremos para trabajar serán los siguientes(es posible que haya versiones más recientes cuando el lector esté realizando la descarga):

-Android SDK platform-tools revision 1

-SDK platform Android 2.3, API 9

-Google APIs by google inc. Android API 9

-Google Usb driver

Al aceptar las licencias y pulsar sobre el botón install, se descargarán los componentes seleccionados e instalarán en los correspondientes directorios.

2.5 Crear un AVD

La instalación llevada a cabo en el apartado 4.2.2 incluía una aplicación que crea y controla terminales android virtuales, AVD(Android Virtual Device) manager. Esto nos permitirá probar nuestras aplicaciones sin necesidad de tener un terminal Android real. Para crear un AVD seguiremos estos pasos:

1-En Eclipse, pulsamos Window > Android SDK and AVD Manager

2-Hacer click en el botón New en la parte derecha de la ventana que ha aparecido.

3-Escribir un nombre para nuestro terminal en el campo Name, una versión de Android en el campo Target (En posteriores pruebas de este manual trabajaremos con la versión Android 2.1-update1) , escribir 128 MiB por ejemplo en el campo SD Card/ Size y seleccionar en Skin/ Built-in un valor de la lista desplegable, por ejemplo HVGA.

4-Hacer click en el botón Create AVD.

3 Instrucciones básicas

La mejor forma de aprender a programar y aprender las distintas instrucciones y facetas de un lenguaje es precisamente programando, por ello vamos a empezar a analizar las distintas posibilidades y directrices de la programación Android con un sencillo programa que muestre el mensaje “Hello World, HelloAndroid!” por pantalla.

3.1Hola Mundo (Código)

Creando el proyecto

-Primero creamos un nuevo proyecto en Eclipse para ello seleccionamos File > New > Project y seleccionamos Android > Android Project, ahora click en el botón Next.

-Ahora tendremos que rellenar los campos del formulario que aparece con los valores de la figura 1.

Project name: HelloAndroid
Application name: Hello, Android
Package name: com.example.helloandroid
Create Activity: HelloAndroid

Figura 1- Valores de un proyecto nuevo

Y también marcar “Create new project in workspace” y la casilla que hay junto a “Android 2.1” en el recuadro Build Target. Después de esto, hacer click en el botón Finish.

Explorador de paquetes del proyecto

En la izquierda de la pantalla, tras haber creado el proyecto, aparecerá un explorador para navegar por los distintos ficheros que conforman este. Los que más nos van a interesar son los que contenga la carpeta “src” (archivos .java), los que contenga la carpeta “res” (resources, como layouts, imagenes, cadenas de caracteres) y el archivo AndroidManifest.xml.

Los archivos .java serán la base de nuestro código y desde dónde tendremos que llamar y utilizar el resto de los recursos que creemos o de los que dispongamos.

Los archivos que contiene la carpeta “res” son recursos asociados al proyecto, son archivos de formato .xml. Los más importantes son los que contiene la carpeta layout ya que definirán el aspecto de cada una de las Activiy.

Por último AndroidManifest.xml contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono, …), sus componentes (pantallas, mensajes, …), o los permisos necesarios para su ejecución, este archivo contiene una lista de cada una de las Activity que forman nuestro programa. Este archivo es especialmente importante por lo que se recomienda su lectura. Se puede encontrar toda la información importante acerca del AndroidManifest en la siguiente URL:

http://developer.android.com/guide/topics/manifest/manifest-intro.html

HelloAndroid.java

En las primeras lineas de este programa, cuyo código se puede ver en la figura 2, se define el paquete al que asignar la clase, los imports necesarios y la definición de la clase. Como es una Activity la clase extiende la clase básica (Activity). Dentro de la clase Activity hay varios métodos que podemos sobrescribir. En el método onCreate(), que es llamado a la hora de crear la Activiy, es donde se encuentra nuestro código.

La primera linea del método onCreate(), super.onCreate(savedInstanceState), llama a la clase superior y la segunda, setContentView, indica el archivo de tipo xml que se utilizará como View principal de esta Activity (en este caso, será main.xml). Con esto lo que se hace es mostrar por pantalla la interfaz gráfica definida en el

archivo main.xml. En la siguiente URL se puede ver en detalle la estructura y funcionamiento de las Activity:

http://developer.android.com/guide/topics/fundamentals/activities.html

```
package com.example.helloandroid;


            import android.app.Activity;
            import android.os.Bundle;


            public class HelloAndroid extends Activity {
             /** Called when the activity is first created. */
             @Override
             public void onCreate(Bundle savedInstanceState) {
                  super.onCreate(savedInstanceState);
                  setContentView(R.layout.main);
            }
            }
```

Figura 2- Código de HelloAndroid.java

Main.xml

La figura 3 muestra el código de lo que se mostrará por pantalla con la llamada desde el archivo HelloAndroid.java. Las View que puedan aparecer en la pantalla (cuadros de texto o TextView, botones...) se organizan en Android mediante layouts. Hay múltiples tipos que se puede consultar en la página web de Android (http://developer.android.com/resources/tutorials/views/index.html). En este caso se utiliza un LinearLayout que organiza los elementos uno detrás de otro. Elegimos la orientación vertical (android:orientation="vertical") y queremos ocupar toda la pantalla

a lo ancho, para lo que se asignará el siguiente valor al ancho del layout: android:layout_width="fill_parent" y a lo alto, android:layout_height="fill_parent".

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout_width="fill_parent"

android:layout_height="fill_parent"

>

      <TextView

      android:layout_width="fill_parent"

      android:layout_height="wrap_content"

      android:text="@string/hello"

      />

</LinearLayout>
```

Figura 3- Codigo de Main.xml

Una vez programado el tipo de layout que queremos introducimos una View de tipo texto (TextView) dentro de ella. Siguiendo las recomendaciones de la documentación de Android, los textos se almacenan en un archivo .xml de strings que utilizaremos para todos los mensajes de texto que pongamos por pantalla en nuestros programas, y hacemos una llamada a string/hello. Todos los detalles de el archivo de Strings se pueden consultar en el apartado correspondiente de la guía online de Android:

http://developer.android.com/guide/topics/resources/string-resource.html

strings.xml

Como se puede ver en la figura 4, string/hello tiene el mensaje: “Hello, Android”, y de este modo será como tendremos que añadir etiquetas o textos cada vez que queramos escribir algo, puede parecer más práctico escribir directamente en el

layout el texto que queramos, pero android nos permite hacer varias carpetas “values” con distintos idiomas (values-XX, donde XX son las letras asignadas por la ISO 639-1 al idioma en concreto) de esta manera el propio terminal detectará el idioma en el que está operando y ejecutará los strings de esa lengua. Se puede consultar la ISO antes citada en la siguiente URL:

http://en.wikipedia.org/wiki/List_of_ISO_639-2_codes

```
<resources>
    <string name="hello">Hello World, HelloAndroid!</string>
    <string name="app_name">Hello, Android</string>
    </resources>
```

Figura 4- Archivo strings.xml

3.2 Simulación

Para simular una aplicación tenemos que seleccionar Run > Run y después escoger “Android Application”.

Ahora el plug-in de eclipse creará una nueva configuración para la aplicación y lanzará el simulador (AVD) para que podamos ver nuestro programa funcionando(en la mayoría de los casos hay que desbloquear antes el terminal virtual, como si de un teléfono real se tratara). A modo de recomendación hay que decir que un buen “truco” para agilizar nuestras simulaciones es no cerrar el terminal virtual cada vez que lo utilicemos, no es necesario, y el tiempo en arrancar de este suele ser relativamente elevado. El manual online de Android proporciona buena información acerca de la simulación y los dispositivos virtuales:

http://developer.android.com/guide/developing/devices/index.html

4-Ejemplo Aplicación: Interface de Usuario

4.1- Interfaz gráfica

Después de entender el ejemplo básico de programación “Hola mundo” ahora vamos a dar un paso más creando una nueva interfaz de usuario, que consistirá en mostrar una vista con cinco botones por pantalla. Cada uno de ellos iniciará una nueva Activity mostrando otra pantalla. Habrá también un sexto botón que cierre nuestra aplicación.

En primer lugar vamos a analizar el código .xml de la interfaz gráfica del menú (Anexo B). Nuestro principal objetivo es mostrar los seis botones por pantalla.

Como se puede ver creamos un LinearLayout que va a contener todo lo que hay en nuestra pantalla.

Por un lado el LinearLayout “principal”, contendrá un RelativeLayout, para colocar el botón de “Salir” en la esquina superior derecha y por otro lado un TableLayout para el resto de los botones. El RelativeLayout nos permite, a diferencia del LinearLayout, colocar los diferentes elementos que queramos en cualquier posición de la pantalla y no uno detrás de otro por necesidad.

En el primero de los botones(que está dentro del RelativeLayout), como se puede ver en el Anexo B, colocamos un elemento de tipo Button. Este tipo de elementos al igual que todos los widgets o views tiene una amplia variedad de opciones de personalización y de comandos específicos que se pueden ver en la guía online de la web oficial de Android que hemos ido siguiendo durante todo el manual. En este caso nos interesa destacar la opción “android:text="@string/exit" que recurre al archivo de strings, que ya hemos utilizado previamente, para colocar un texto en el botón.

Por su parte android:layout_alignParentRight="true" nos sirve para colocar el botón en la parte derecha de la pantalla. Con este código el botón saldría colocado por pantalla según indica la figura 5.

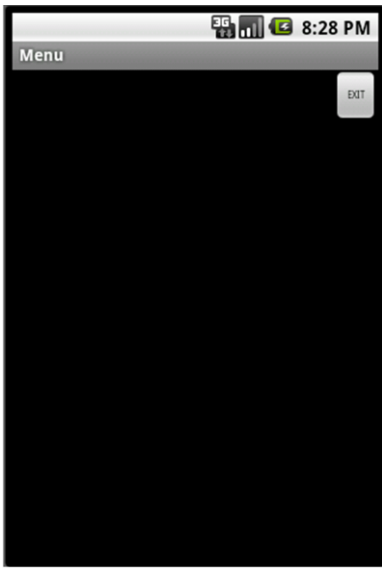

Figura 5- Posición del botón SALIR

A continuación del RelativeLayout, hemos situado un TableLayout. Este tipo de Layout coloca a los elementos que contiene en filas y columnas.Se pretende hacer una tabla de 5x2. En cada celda de la primera columna se colocará un botón, y en las celas correspondientes de la segunda columna, un texto que explique, de forma resumida, la función del botón a su izquierda.

Como podemos ver en la figura 6 creamos un elemento de tipo TextView para colocar la palabra Menú al frente del mismo. El resto de comandos se refieren a colocación y formato del elemento. Recomendamos, para un mejor aprendizaje de las distintas opciones, consultar el API correspondiente, en este caso puede ser muy útil el siguiente apartado del manual online de Android:

http://developer.android.com/resources/tutorials/views/index.html

Se puede ver en la figura 6 que se ha introducido un elemento de tipo View, que utilizamos como linea horizontal que separe la palabra Menú y cada uno de los botones. Esta linea se dibuja haciendo el elemento de una altura muy pequeña (android:layout_height="2dip") y posteriormente coloreándolo, seleccionando un color de fondo mediante la etiqueta “background”.

```
      <TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:stretchColumns="1"
            android:gravity="center">
            <TextView
            android:text="@string/main_tittle"
            android:textSize="22sp"
            android:gravity="center"
            android:paddingBottom="8dip" />
            <View android:layout_height="2dip"
            android:background="#FF909090" />
```

Figura 4.6- Inicio TableLayout

A continuación y como vemos en el código del anexo B, se coloca el primer botón del Menú. Para ello utilizamos el comando <TableRow> para indicarle a TableLayout que estamos colocando una fila nueva y aquí colocamos dos elementos, un botón y un TextView como anteriormente hemos explicado. Como se puede ver en el código del anexo B, utilizamos el comando layout_column para indicar a cada uno de los botones que se tienen que alinear formando una columna, la columna 1.

El aspecto final de nuestro menú debe ser el que se muestra en la figura 7, aunque el usuario puede "jugar" con las instrucciones de tamaño, posición y colores para crear una interfaz totalmente diferente a la que se presenta como ejemplo.

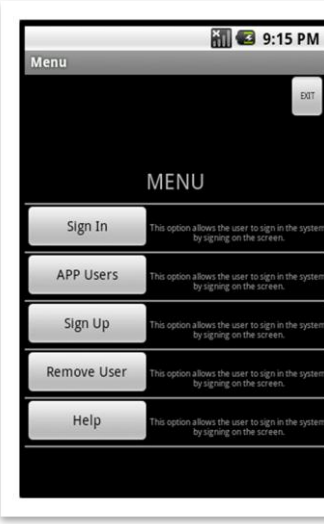

Figura 7- Menú final

Como se ha podido ver durante todo el código del Anexo 1 en los archivos de Layout se sigue la estructura XML. Y hay que saber que dentro de cada Layout se pueden meter los elementos que se quiera. Una buena técnica sobre todo si se quieren sacar muchos elementos por pantalla es anidar Layouts como se ha hecho en el programa de ejemplo. Cabe destacar como muy importante la instrucción android:id="@+id/nombre", Mediante esta instrucción se permite asociar un id único al elemento en el que está anidada con el “nombre” indicado en la propia instrucción. De esta forma nos podremos referir a este elemento desde el código de nuestro archivo .java con sólo indicar el nombre (android busca el id que el mismo entorno ha creado y almacenado por su cuenta y de manera automática, sin que esto lo tenga que hacer el programador).

En el programa que estamos creando, cada uno de los botones creará una nueva activity y abrirá en consecuencia una nueva ventana. Estas ventanas carecerán de contenido alguno, ya que nuestro único objetivo es aprender las funcionalidades básicas de android, así como de los layout.

Como ya hemos explicado antes, cada una de las activity de un programa Android lleva asociado un archivo .java y cada uno de estos, en nuestro programa será una nueva vista(view) definida en un archivo .xml. Así pues, a parte del archivo main.xml crearemos un archivo .xml por cada uno de los cinco botones. Estos archivos no serán tan complejos como el explicado en este punto. De hecho, simplemente tendremos que crearlos y dejarlos tal como están por defecto (muestran por pantalla una ventana negra). En el ejemplo los nombres de estos archivos son: appusers.xml, help.xml, removeuser.xml, signin.xml y signup.xml.

4.2 Parte funcional de nuestro programa (Main.java)

Ahora que tenemos construida la interfaz gráfica, tendremos que configurar el menú para que cada uno de los botones acceda a una nueva activity y para que el botón exit cierre el programa. Esto lo haremos desde el archivo main.java que contendrá el código de la Activity principal, dicho código se encuentra en el Anexo C.

Después de importar las librerías que vamos a utilizar a lo largo del código comienza la clase correspondiente a la Activity principal. El primer método que implementamos es OnCreate (figura 8). En este método, mediante setContentView, le indicamos a Android que la vista principal de la Activity es la definida por el fichero main.xml , después creamos una instancia por cada uno de los seis elementos de tipo button, para indicarle el método que será el que "escucha" cuando se haga click. Si no se define este método, no pasará nada cuando se haga click.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button signin = (Button) findViewById(R.id.signin);
        signin.setOnClickListener(this);
        Button appusers = (Button) findViewById(R.id.app_users);
        appusers.setOnClickListener(this);
        Button signup = (Button) findViewById(R.id.signup);
        signup.setOnClickListener(this);
        Button removeuser = (Button) findViewById(R.id.remove_user);
        removeuser.setOnClickListener(this);
        Button help = (Button) findViewById(R.id.help);
        help.setOnClickListener(this);
        Button exit = (Button) findViewById(R.id.exit);
        exit.setOnClickListener(this);
 }
```

Figura 8- Método onCreate de Main.java

El siguiente método que tendremos que programar será onClick(figura 9), que controlará la acción que ocurra cada vez que pulsemos uno de nuestros botones. El método onClick está definido en la interfaz onClickListener y para que onClick pueda funcionar correctamnte lo hemos implementado previamente (public class Menu extends Activity implements OnClickListener). Creamos un switch que detecte según la id cual es el botón que se ha pulsado y repetimos el mismo procedimiento para cada uno de los case (botones): Primero se crea un intent para pasarle a la actividad y después se inicia mediante este último la actividad con “StartActivity(intent)”.

```
public void onClick(View v) {
      switch (v.getId()) {
      case R.id.signin:
       Intent i = new Intent(this, signin.class);
       startActivity(i);
       break;
      case R.id.app_users:
       Intent i2 = new Intent(this, appusers.class);
       startActivity(i2);
       break;
      case R.id.signup:
        Intent i3 = new Intent(this, signup.class);
        startActivity(i3);
        break;
      case R.id.remove_user:
         Intent i4 = new Intent(this, removeuser.class);
         startActivity(i4);
          break;
      case R.id.help:
         Intent i5 = new Intent(this, help.class);
         startActivity(i5);
          break;
      case R.id.exit:
        finish();}}
```

Figura 9- Método onClick de Main.java

Cada uno de los botones iniciará una actividad. Si nos fijamos en los distintos case del switch, mediante el intent correspondiente le decimos a cada uno de los botones que activity (class, ya que las activity no son más que clases) debe iniciar . Para que funcione el programa correctamente debemos crear cinco archivos de tipo java con los nombres correpondientes (signin.java, appusers.java, signup.java, removeuser.java y help.java).

Para crear un archivo de tipo .java abriremos la carpeta src del "Package explorer" situado a la izquierda y después haremos click con el botón derecho del ratón sobre el nombre de nuestro Package, ahora seleccionamos new>file, y ponemos el nombre que queramos en el apartado File name, siempre acabado en la extensión .java (Ej: prueba.java) y a continuación pulsamos el botón Finish.

El código básico que tiene que tener un archivo .java en nuestro programa es el que muestra la figura 10. Hay que asegurarse de llamar a la Activity con el mismo nombre que hemos llamado al archivo .java (en nuestro ejemplo ha sido prueba).

```
package PACKAGE_NAME;

import android.app.Activity;

import android.os.Bundle;


public class prueba extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.main);

}

}
```

Figura 10- Modelo para archivos .java

Como lo único que queremos es aprender a crear un programa básico no es necesario que modifiquemos demasiado estos cinco archivos, lo único que tenemos que conseguir será que llamen a un archivo de tipo .xml que nosotros ya hemos creado. Por ejemplo signup.java quedaría como se muestra en la figura 11.

```
package org.rbe;


import android.app.Activity;
import android.os.Bundle;
public class signup extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.signup);
}
}
```

Figura 11- Modelo de los demás archivos .java

5-Depuración

Una de las técnicas más útiles a la hora de depurar y/o realizar el seguimiento de aplicaciones sobre cualquier plataforma es la creación de logs de ejecución. Android por supuesto no se queda atrás y nos proporciona también su propio servicio y API de logging a través de la clase android.util.Log.

En Android, todos los mensajes de log llevarán asociada la siguiente información:

-Fecha/Hora del mensaje.

-Criticidad. Nivel de gravedad del mensaje.

-PID. Código interno del proceso que ha introducido el mensaje.

-Tag. Etiqueta identificativa del mensaje.

-Mensaje. El texto completo del mensaje

De forma similar a como ocurre con otros frameworks de logging, en Android los mensajes de log se van a clasificar por su criticidad, existiendo así varias categorías (ordenadas de mayor a menor criticidad):

1 Error

2 Warning

3 Info

4 Debug

5 Verbose

Para cada uno de estos tipos de mensaje existe un método estático independiente que permite añadirlo al log de la aplicación. Así, para cada una de las categorías anteriores tenemos disponibles los métodos e(), w(), i(), d() y v() respectivamente.

Cada uno de estos métodos recibe como parámetros la etiqueta (tag) y el texto en sí del mensaje que se desea mostrar. Como etiqueta de los mensajes, aunque es un campo al que podemos pasar cualquier valor, suele utilizarse el nombre de la aplicación o de la actividad concreta que genera el mensaje. Esto nos permitirá más

tarde crear filtros personalizados para identificar y poder visualizar únicamente los mensajes de log que nos interesan, entre todos los generados por Android [que son muchos] durante la ejecución de la aplicación.

Hagamos un mini programa de ejemplo para ver cómo funciona esto (figura 4.12). El programa será tan simple como añadir varios mensajes de log dentro del mismo onCreate de la actividad principal y ver qué ocurre.

```
public class LogsAndroid extends Activity {

private static final String LOGTAG = "LogsAndroid";

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

Log.e(LOGTAG, "Mensaje de error");

Log.w(LOGTAG, "Mensaje de warning");

Log.i(LOGTAG, "Mensaje de información");

Log.d(LOGTAG, "Mensaje de depuración");

Log.v(LOGTAG, "Mensaje de verbose");

}

}
```

Figura 4.12- Programa depuración

Si ejecutamos la aplicación anterior en el emulador veremos cómo se abre la pantalla principal que crea Eclipse por defecto y aparentemente no ocurre nada más. ¿Dónde podemos ver los mensajes que hemos añadido al log? Pues para ver los mensajes de log nos tenemos que ir a la perspectiva de Eclipse llamada DDMS. Una vez en esta perspectiva, podemos acceder a los mensajes de log en la parte inferior de la pantalla, en una vista llamada LogCat. En esta ventana se muestran todos los mensajes de log que genera Android durante la ejecución de la aplicación, que son

muchos, pero si buscamos un poco en la lista encontraremos los generados por nuestra aplicación, tal como se muestra en la figura 4.13.

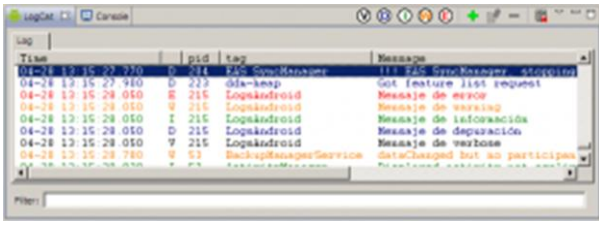

Figura 4.13-Mensajes de Log

Como se puede observar, para cada mensaje se muestra toda la información que indicamos al principio del artículo, además de estar diferenciados por un color distinto según su criticidad.

En este caso de ejemplo, los mensajes mostrados son pocos y fáciles de localizar en el log, pero para una aplicación real, el número de estos mensajes puede ser mucho mayor y aparecer intercalados caóticamente entre los demás mensajes de Android. Para estos casos, la ventada de LogCat ofrece una serie de funcionalidades para facilitar la visualización y búsqueda de determinados mensajes.

Por ejemplo, podemos restringir la lista para que sólo muestre mensajes con una determinada criticidad mínima. Esto se consigue pulsando alguno de los 5 primeros botones que se observan en la parte superior derecha de la ventana de log. Así, si por ejemplo pulsamos sobre el botón de la categoría Info (en verde), en la lista sólo se mostrarán los mensajes con criticidad Error, Warning e Info.

Otro método de filtrado más interesante es la definición de filtros personalizados (botón “+” verde), donde podemos filtrar la lista para mostrar únicamente los mensajes con un PID o Tag determinado. Si hemos utilizado como etiqueta de los mensajes el nombre de nuestra aplicación o de nuestras actividades esto nos proporcionará una forma sencilla de visualizar sólo los mensajes generados por nuestra aplicación.

Así, para nuestro ejemplo, podríamos crear un filtro indicando como Tag la cadena “LogsAndroid”, tal como se muestra en la figura 4.14.

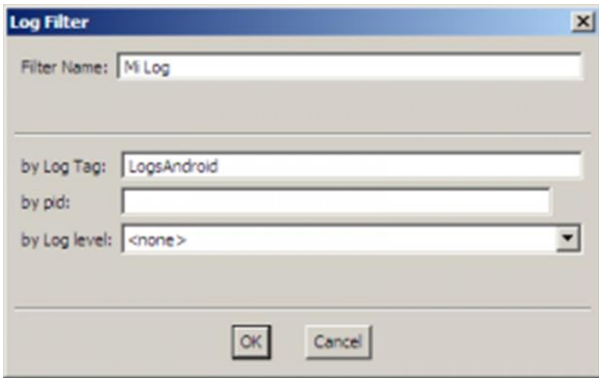

Figura 4.14- Log filter

Esto creará una nueva ventana de log con el nombre que hayamos especificado en el filtro, donde sólo aparecerán nuestros 5 mensajes de log de ejemplo (figura 4.15).

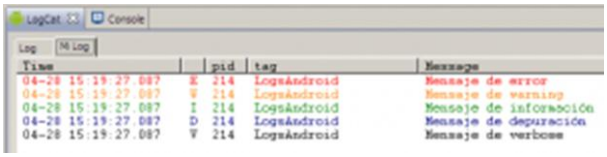

Figura 4.15- Log con filtro

Por último, cabe mencionar que existe una variante de cada uno de los métodos de la clase Log que recibe un parámetro más en el que podemos pasar un objeto de tipo excepción. Con esto conseguimos que, además del mensaje de log indicado, se muestre también la traza de error generada con la excepción.

Veamos esto con un ejemplo, y para ello vamos a forzar un error de división por cero, vamos a capturar la excepción y vamos a generar un mensaje de log con la variante indicada(figura 4.16).

```
try
{
int a = 1/0;
}
catch(Exception ex)
{
Log.e(LOGTAG, "División por cero!", ex);
}
```

Figura 4.16- División por cero

Si volvemos a ejecutar la aplicación y vemos el log generado, podemos comprobar cómo se muestra la traza de error correspondiente generada con la excepción(figura 5.6).

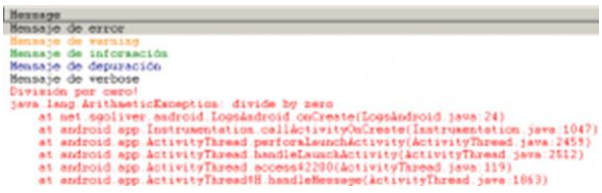

Figura 4.17- Log

En definitiva, podemos comprobar como la generación de mensajes de log puede ser una herramienta sencilla pero muy efectiva a la hora de depurar aplicaciones que no se ajustan mucho a la depuración paso a paso, o simplemente para generar trazas de ejecución de nuestras aplicaciones para comprobar de una forma sencilla cómo se comportan.

Anexo B

En este anexo se presenta el código de main.xml al que se hace referencia dentro del manual de introducción a la programación Android del Anexo A.

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content">

<Button  android:id="@+id/exit"
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:text="@string/exit"
android:textSize="8dp"
android:layout_alignParentRight="true"
android:height="1dp"
android:width="40dp"  />
</RelativeLayout>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:stretchColumns="1"
      android:gravity="center">
```

```
    <TextView
     android:text="@string/main_tittle"
     android:textSize="22sp"
     android:gravity="center"
     android:paddingBottom="8dip" />
    <View android:layout_height="2dip"
    android:background="#FF909090" />
    <TableRow>
     <Button  android:id="@+id/signin"
     android:layout_column="1"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/signin" />
     <TextView android:layout_column="2"
     android:text="@string/signin_text"
     android:textSize="9sp"
     android:gravity="center" />
    </TableRow>
<View android:layout_height="2dip"
    android:background="#FF909090" />
    <TableRow>
     <Button  android:id="@+id/app_users"
     android:layout_column="1"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/app_users" />
     <TextView android:layout_column="2"
     android:text="@string/appusers_text"
     android:textSize="9sp"
     android:gravity="center" />
    </TableRow>
<View android:layout_height="2dip"
```

```
    android:background="#FF909090" />
    <TableRow>
     <Button  android:id="@+id/signup"
     android:layout_column="1"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/signup" />
     <TextView android:layout_column="2"
     android:text="@string/signup_text"
     android:textSize="9sp"
     android:gravity="center" />
    </TableRow>
 <View android:layout_height="2dip"
    android:background="#FF909090" />
    <TableRow>
     <Button  android:id="@+id/remove_user"
     android:layout_column="1"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/remove_user" />
     <TextView android:layout_column="2"
     android:text="@string/removeuser_text"
     android:textSize="9sp"
     android:gravity="center" />
    </TableRow>
 <View android:layout_height="2dip"
    android:background="#FF909090" />
    <TableRow>
     <Button  android:id="@+id/help"
     android:layout_column="1"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
```

```
        android:text="@string/help" />
        <TextView android:layout_column="2"
        android:text="@string/help_text"
        android:textSize="9sp"
        android:gravity="center" />
       </TableRow>
    <View android:layout_height="2dip"
       android:background="#FF909090" />

</TableLayout>
</LinearLayout>
```

Anexo C

En este anexo se presenta el código de menú.java al que se hace referencia dentro del manual de introducción a la programación Android del Anexo A.

menu.java

```
package org.rbe;

import android.app.Activity;

import android.os.Bundle;

import android.content.Intent;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.*;


public class Menu extends Activity implements OnClickListener {

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);


        Button signin = (Button) findViewById(R.id.signin);

        signin.setOnClickListener(this);

        Button appusers = (Button) findViewById(R.id.app_users);

        appusers.setOnClickListener(this);

        Button signup = (Button) findViewById(R.id.signup);

        signup.setOnClickListener(this);

        Button removeuser = (Button) findViewById(R.id.remove_user);

        removeuser.setOnClickListener(this);

        Button help = (Button) findViewById(R.id.help);

        help.setOnClickListener(this);

        Button exit = (Button) findViewById(R.id.exit);

        exit.setOnClickListener(this);

    }
```

```
    public void onClick(View v) {
      switch (v.getId()) {
        case R.id.signin:
         Intent i = new Intent(this, signin.class);
         startActivity(i);
         break;
        case R.id.app_users:
                Intent i2 = new Intent(this, appusers.class);
         startActivity(i2);
               break;
        case R.id.signup:
                Intent i3 = new Intent(this, signup.class);
                startActivity(i3);
               break;
        case R.id.remove_user:
            Intent i4 = new Intent(this, removeuser.class);
            startActivity(i4);
              break;
        case R.id.help:
            Intent i5 = new Intent(this, help.class);
            startActivity(i5);
              break;
        case R.id.exit:
               finish();
                     break;
      }
      }
}
```

Anexo D

En este anexo se presenta el código de strings.xml al que se hace referencia dentro del manual de introducción a la programación Android del Anexo A.

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Menu!</string>
    <string name="app_name">Menu</string>
    <string name="sign_title">SIGN ACTIVITY</string>
    <string name="appusers_title">APP USERS</string>
    <string name="signup_title">SIGN UP</string>
    <string name="removeuser_title">REMOVE USER</string>
    <string name="help_title">HELP</string>
    <string name="main_tittle">MENU</string>
    <string name="signin">Sign In</string>
    <string name="app_users">APP Users</string>
    <string name="signup">Sign Up</string>
    <string name="remove_user">Remove User</string>
    <string name="help">Help</string>
    <string name="version">V. 1.0</string>
    <string name="exit">EXIT</string>
    <string name="appusers_text">appusers information</string>
    <string name="help_text">help information</string>
    <string name="signup_text">signup informatio</string>
    <string name="removeuser_text">remove user information</string>
    <string name="signin_text">This option allows the user to sign in
the system\n by signing on the screen.</string>
    <!-- Strings de la Activity Signup -->
    <string name="name_signup">Name</string>
    <string name="surname_signup">Surname</string>
```

```
    <string name="password_signup">Password</string>
    <string name="button1_signup">Save user without signing </string>
    <string name="button2_signup">Sign</string>
</resources>
```