Universidad de Valladolid

E.T.S INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática de Sistemas

Aplicación Android para la creación y reconocimiento de patrones de desbloqueo libres

Autor:
David Polvorosa Hoyos

Tutor:
Carlos Enrique Vivaracho Pascual

Dedicado a mi familia, que me
ha apoyado durante toda
la realización de este proyecto

Índice general

Capítulo 1

Introducción y objetivos

A día de hoy, los teléfonos inteligentes han invadido el mercado de las telecomunicaciones. Esto ha sido gracias a que en esencia, este tipo de terminales funcionan como un ordenador personal. Con ellos, se puede navegar, mirar el correo, chatear... de una forma tan simple y rápida. Como con nuestro ordenador, todos queremos que nuestro dispositivo esté protegido de accesos no autorizados a nuestros datos (aunque al principio, no lo creamos en nuestros móviles se guardan múltiples datos como contactos, correos...). Para conseguir esto, los móviles incluyen sistemas de acceso mediante código numérico o patrón como el de la figura 1.1:

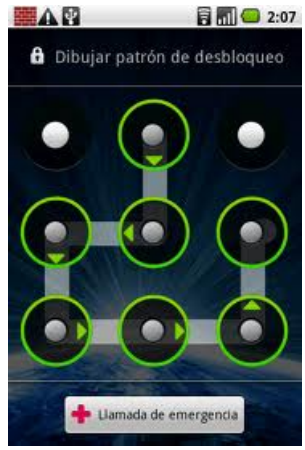


Figura 1.1: Pantalla de desbloqueo convencional de un smartphone

Un sistema de código numérico, nos garantiza cierta seguridad, pero se puede "romper" con ataques similares a los de ordenadores como el de fuerza bruta. Por ejemplo, teniendo una clave de 5 cifras, son 10.000 los intentos que el atacante tendría que realizar como máximo para lograr entrar en nuestro terminal. Esto para el cómputo de un ordenador supone un

coste de tiempo pequeño. Por otra parte, los patrones definidos por puntos ofrecen un mayor número de posibilidades, pero si alguien por la razón que sea, ve cual es nuestra clave, reproducirlo es bastante simple. Esto es debido a que sólo importa la secuencia de puntos de los (usualmente) 9 puntos que tiene. Como vemos, los actuales sistemas de desbloqueo en los móviles pueden ser mejorados.

Debido a ello, ya han aparecido en el mercado sistemas de este tipo como por ejemplo SigmaUnlock [2] que permite acceder a tu móvil a través de un sistema de realización de patrón libre. Esta aplicación utiliza la dinámica del patrón para añadir seguridad a la realización de este. El presente proyecto va en la misma dirección. Partiendo de la experiencia del GIR ECA-SIMM de la UVA en reconocimiento de firma manuscrita, se intenta desarrollar un sistema propio, del cual el presente proyecto es el primer paso.

1.1. Objetivos del trabajo

El **objetivo principal** de este proyecto es el de construir una aplicación experimental para el reconocimiento de patrón libre en móviles.

Este objetivo se concreta en los siguientes objetivos específicos:

- La respuesta de la aplicación debe ser en tiempo real, es decir, habrá que tener en cuenta las limitaciones de los móviles a la hora de plantear el sistema de reconocimiento.
- El sistema tomará muestras para crear el modelo del patrón para su reconocimiento de acuerdo a las especificaciones del cliente y desarrollador, en este, caso del sistema de reconocimiento.
- Como versión inicial, la aplicación debe ser fácil de modificar y/o ampliar. El resultado final será indicar, para varios niveles de fiabilidad, si el patrón introducido ha sido aceptado o rechazado.

Es importante comentar en este punto que no es un objetivo del presente proyecto el sistema de reconocimiento, ni el sistema de cálculo de umbrales de decisión los cuales son desarrollados por el grupo ECA-SIMM, siendo responsabilidad del desarrollador solamente su integración en la aplicación. Aún así, por el interés de esta parte, se añadirá a la presente memoria una sección donde se mostrarán de manera resumida los experimentos realizados, y resultados de reconocimiento obtenidos. Indicar, ya para acabar, que el autor del presente proyecto ha participado activamente en la adquisición de un pequeño corpus de patrones necesario para la obtención del sistema de reconocimiento. Este corpus es el utilizado en las pruebas del capítulo 5.

1.2. Organización de la memoria

Para que el lector, se haga una idea de que apartados se tratarán en esta memoria, se realiza en esta sección un pequeño resumen de cada una de las partes del trabajo.

1. **Introducción y objetivos**. En este apartado, se explicará el contexto en el cual se realiza este proyecto. También se detallarán los objetivos propuestos en este trabajo.

2. **Conceptos teóricos previos**. En este lugar de la memoria, se explicarán algunos conceptos técnicos, los cuales serán necesarios para comprender el resto de la memoria.

3. **Análisis y diseño**. En esta sección, se detalla la parte de análisis y diseño del proceso de ingeniería de software. En el mismo, se detallarán requisitos de la aplicación, casos de uso, diseño de la solución...

4. **Implementación y pruebas**. Tras haber realizado el análisis y el diseño de la aplicación, en este apartado se detalla el proceso de implementación junto con las baterías de pruebas realizadas para verificar su correcto funcionamiento.

5. **Pruebas del reconocedor**. En este apartado, se detallan las pruebas realizadas para obtener la mayor exactitud de reconocimiento de patrón, intentando que el rendimiento sea el óptimo para un terminal móvil

6. **Conclusiones**. Por último, en esta sección, se mostrarán las conclusiones obtenidas del proyecto. Además también se detallarán posibles mejoras de la misma y vertientes de trabajo futuro.

7. **Anexos**. Se añadirá toda la información adicional en los anexos, como el funcionamiento detallado de fastDTW o el seguimiento del proyecto junto con su planificación temporal.

Capítulo 2

Conceptos teóricos previos

2.1. El patrón móvil

Llamamos patrón móvil o patrón libre a la figura realizada sobre la pantalla de un dispositivo móvil de manera libre por el usuario, sin la limitación de seguir unos puntos fijos, como en los sistemas actuales de desbloqueo. El usuario libremente puede escoger la forma de este patrón. Durante la realización del patrón sobre la pantalla se captura y almacena la siguiente información de cada punto que lo compone:

- **X** : Coordenada del punto en el eje horizontal de las X.
- **Y** : Coordenada del punto en el eje horizontal de las Y.
- **T** (tiempo) : Tiempo ocurrido entre toma de un punto y otro punto. En el caso del primer punto, esta variable tomará el valor 0 siempre
- **P** (pulsacion) : Vale 1 al comienzo de cada trazo y 0 en el resto de puntos del trazo. Nos permite conocer el número de trazos del patrón y dónde comienza y acaba cada uno, tanto en coordenadas X e Y, como en tiempo.

Un ejemplo de patrón sería el que se muestra en la figura 2.1

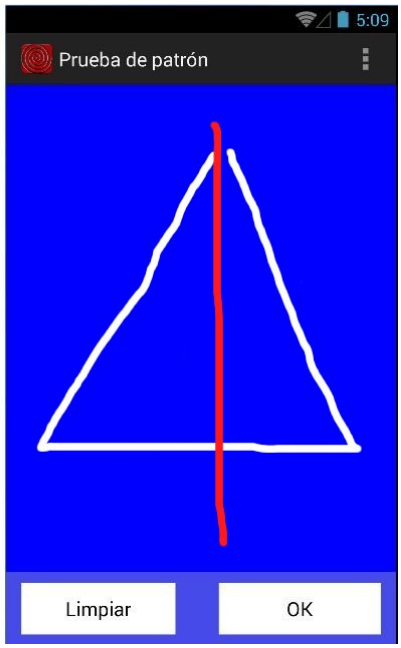


Figura 2.1: Patrón móvil compuesto por dos trazos

Como se puede ver, se captura tanto la forma del patrón (coordenadas X e Y), como su dinámica (evolución temporal de las coordenadas y trazos del patrón). Ambas características son propias del usuario, permitiendo, creemos, su identificación a partir del patrón realizado. Frente a imitaciones (cuando el atacante conoce el patrón), la dinámica añade un nivel de seguridad adicional en la identificación al sólo uso de la forma, como en los sistemas actuales de desbloqueo. Por contra, también añade una problema adicional al reconocimiento: la variabilidad en la realización del patrón por parte del propio usuario (como se verá en el apartado de pruebas del reconocedor). Este es el reto a abordar en su reconocimiento. Veremos en el siguiente apartado una breve descripción del sistema usado.

Con respecto a la variabilidad intrausuario, sería interesante contar con un sistema de predicción de patrones con error alto en su reconocimiento, bien porque sean demasiado simples, o bien porque impliquen demasiada variabilidad en su realización. Esto permitiría avisar al usuario para su modificación. Es un tema de investigación completamente abierto actualmente. el presente proyecto hemos implementado una alternativa que será mostrada en el capítulo 5.

2.2. El sistema de reconocimiento

El sistema de reconocimiento que realizamos para este proyecto se compone de cuatro partes, las cuales se detallan a continuación.

2.2.1. Captura

A la hora de realizar la captura, se guardarán las cuatro variables indicadas en la sección 2.1. Como facilidad para el usuario,el patrón se puede realizar en cualquier parte de la pantalla, por ello antes de pasar a la siguiente fase, es necesario normalizarle frente a traslación, rotación y tamaño.

La primera normalización se hace trasladando el centro de masas de cada firma al origen de coordenadas (X=0, Y=0). No se contempla normalización frente a rotación, ya que los hábitos al colocar el móvil y realizar el patrón son característicos de cada usuario y pueden ayudar a su reconocimiento. En cambio sí se realiza normalización frente a tamaño, pero en vez de realizarla sobre el patrón directamente, se realiza en la siguiente etapa, aplicando ZNorm sobre cada una de las características extraídas de la muestra.

2.2.2. Extracción de características

A la hora de extraer características (que serán las que caractericen cada muestra) existen dos tipos:

- **Estáticas :** Se extraen características globales del patrón como duración, tamaño en cada coordenada, máximo X, máximo Y, etc.
- **Dinámicas :** Basadas en la secuencia de puntos

En este proyecto se vana usar las segundas. Dentro de las muchas que se pueden extraer, aquí sólo se han probado algunas muy simples, pero que han mostrado un alto rendimiento en firma [3] [4] [5]:

- **Delta:** Diferencia de posicion entre dos muestras consecutivas.
- **Velocidad:** : El cálculo de esta característica se obtiene de la ecuación 2.1

$$\frac{\Delta}{t_n - t_{n-1}} \tag{2.1}$$

En el capitulo 5 se mostrarán los resultados obtenidos con cada una de ellas.

2.2.3. Clasificación

Este paso consiste en crear un modelo del patrón realizado por el usuario. Es la parte crítica con respecto a la respuesta en tiempo real del sistema. De todos los planteados en el reconocimiento de firma (sistema del que hereda el usado para reconocer un patrón libre [4] [6]), nos hemos decantado por los basados en Dynamic Time Warping (DTW) debido a su simplicidad y alto rendimiento [4] [7]. Esta técnica permite calcular una distancia entre dos muestras temporales de distinto tamaño.

Independientemente del método de clasificación a usar, antes de poderlo usar para el reconocimiento, necesitamos "entrenarlo" con muestras del patrón a reconocer; es lo que se denomina fase de inscripción. Uno de los problemas tanto del reconocimiento de firma como de patrón es la variabilidad del propio usuario con respecto a sí mismo (variabilidad intrausuario). Cuantas más muestras de inscripción se tomen, mejor se capturará esta variabilidad, pero más incómodo será para el usuario, pudiendo suponer un motivo de rechazo de sistema. Por ello, hay que llegar a un compromiso.

Dada la experiencia del grupo de investigación y de los resultados obtenidos se proponen dos opciones de inscripción al usuario: dar sólo tres ejemplos del patrón que se realizarán de manera consecutiva (sesión 1 de inscripción) o añadir otros dos más que ser realizarán también de manera consecutiva pero días después (sesión 2). Los resultados muestran la ventaja de esta segunda opción, pero es el usuario el que elige realizar o no esta segunda sesión, ya que el sistema permite reconocer patrones desde la sesión 1 de inscripción.

DTW y FastDTW

Como se ha comentado es un punto crítico a la hora de lograr una respuesta en tiempo real del sistema. En este punto, el uso de sistemas móviles hace que nos tengamos que enfrentar al problema de la capacidad de computo limitada de los procesadores. Aunque en continua mejora, todavía están lejos de las logradas en los ordenadores de sobremesa o servidores, donde hasta ahora se han probado este tipo de sistemas. DTW es una técnica de cáculo de distancias relativamente "pesada" en cuanto a procesamiento, sobre todo en su versión original.

Buscando implementaciones más eficientes, encontramos la FastDTW [9] A. Los autores de la propuesta proporcionan de manera libre una implementación en Java que es la implementación que hemos usado en el presente proyecto. Su uso no ha sido inmediato, y ha necesitado de un proceso, primero, de análisis y comprensión del código, y, segundo, de la realización de modificaciones sobre ese código para adaptarlo a las necesidades de nuestra aplicación.

Desde un punto de vista teórico, su uso aporta una mejora en el rendimiento temporal del sistema, sin disminuir la eficiencia en el reconocimiento. Ahora bien, antes de pasar a su uso, había que comprobar eso de manera práctica. Para ello se realizaron experimentos usando dos corpus de firmas, comparando el tiempo necesario para acabar los experimentos y los errores de reconocimiento. Como corpus se usaron los siguientes:

- **SVC**: SVC. Es un corpus con 40 firmas obtenidas mediante una tableta gráfica. Cada usuario tiene 20 firmas propias y 20 imitaciones. Usamos 5 firmas (las 5 primeras de cada usuario) para crear el modelo del firmante.
- **BioWam**: Corpus propio de firmas obtenidas mediante dispositivos móviles. Se compone de 39 firmantes cada uno con 18 firmas propias obtenidas en dos sesiones y 12 imitaciones. Para crear el modelo usamos 3 firmas del usuario de la sesión 1 y 2 de la sesión 2.

Como medición del error del sistema hemos usado la Tasa de Equierror (EER, Equal Error Rate), que es el error obtenido cuando los falsos positivos se igualan a los falsos negativos. Es una de las medidas de error del sistema más típicas en biometría. En la tablas 2.1 y 2.2.3 se muestran los resultados

Num experimento	Técnica	Radio	EER-SKILLED	EER-RANDOM	Duración (min:seg)
1	DTW	No aplicable	5,22	0,60	8:20
2	FastDTW	1	5,05	0,60	1:05
3	FastDTW	2	4,84	0,53	1:17
4	FastDTW	3	4,84	0,60	1:54
5	FastDTW	6	5,18	0,60	2:39
6	FastDTW	12	5,22	0,60	4:02

Tabla 2.1: Resultados del corpus BioWam

Num experimento	Técnica	Radio	EER-SKILLED	EER-RANDOM	Duración (min:seg)
1	DTW	No aplicable	9,08	0,17	9:08
2	FastDTW	1	11,54	0,23	1:14
3	FastDTW	2	9,00	0,17	1:44
4	FastDTW	3	9,13	0,17	2:01
5	FastDTW	6	8,96	0,17	2:46
6	FastDTW	12	9,08	0,17	4:09

Tabla 2.2: Resultados con el corpus SVC

El tiempo que se muestra es el necesario para realizar el experimento sobre todos los componentes de cada corpus. El "Radio" es uno de los parámetros variables del algoritmo FastDTW (A), por lo que se han probado varios. Se ha usado la misma máquina y el mismo entorno de desarrollo.

De los resultados mostrados en la tabla se puede ver claramente las ventajas en velocidad de procesamiento logradas al usar FastDTW, sin que ello suponga un empeoramiento en los resultados obtenidos; es más, en determinadas ocasiones los resultados con FastDTW son mejores incluso para valores del radio pequeños. Otras de las conclusiones del estudio experimental realizado es el valor del "Radio" a usar. Hemos escogido el valor 2, por ser el que mejor relación tiempo de procesamiento-rendimiento den reconocimiento da. Tras la obtención del corpus de patrones se probó este parámetro, llegando a la misma conclusión.

2.2.4. Decisión

El clasificador genera la distancia entre el modelo del patrón del usuario y el patrón introducido para su reconocimiento. Esta distancia es comparada con un umbral de decisión y si es mayor que éste, el patrón se rechaza, siendo aceptado en caso contrario.

El cálculo del umbral es otro de los puntos críticos del sistema en cuanto a respuesta temporal. Además, sería conveniente contar con distintos niveles de fiabilidad, de manera que el usuario pudiera elegir para cada caso el nivel que desea. Para esto se ha utilizado el método propuesto de predicción del umbral de decisión para firmas manuscritas y que se muestra en [8]. Se han escogido 3 niveles de confianza. Uno poco estricto, con muy poco (prácticamente 0) rechazo, pero poco fiable frente a imitaciones. El segundo nivel está pensado para tener también bajo rechazo, pero superior

al anterior, siendo más robusto frente a imitaciones pero no a un imitador muy bueno. Por último el nivel de mayor seguridad está calculado para un rechazo de, aproximadamente, el 25 % de la muestras propias teniendo, a su vez, una muy alta robustez frente a las imitaciones.

Capítulo 3

Análisis y Diseño

En esta sección se detallarán, todos aquellos requisitos que nos permitirán formar un dominio y poder llegar a hacernos una idea de cómo será la aplicación. Partiendo de estas premisas, se tomaran unas decisiones de diseño, las cuales se detallan también en este capítulo.

3.1. Requisitos de usuario

RU01 - El usuario podrá ingresar muestras al sistema para definir un patrón.

RU02 - El usuario podrá modificar un patrón definido anteriormente.

RU03 - El usuario podrá ingresar una muestra para compararla con un patrón definido.

Todos estos requisitos son acciones que podrán realizar el actor del sistema. Estos requisitos de usuario, definen los siguientes requesitos funcionales.

3.2. Requisitos funcionales

RF01 - El sistema permitirá al usuario crear un patrón.

RF02 - El sistema permitirá al usuario realizar muestras de inscripción.

RF03 - El sistema permitirá al usuario redefinir un patrón.

RF04 - El sistema permitirá borrar una muestra de la pantalla antes de confirmarla.

RF05 - El sistema permitirá realizar pruebas de ingreso del patrón en la aplicación.

RF06 - El sistema mostrará al usuario, cada nivel de seguridad en el acceso mostrando si ha sido superado o no, en una prueba de ingreso de patrón.

RF07 - El sistema permitirá al usuario elegir entre una o dos sesiones de inscripción.

RF08 - El sistema calculará los umbrales a superar a partir de las muestras de inscripción y unos ficheros de cohorte.

RF09 - El sistema calculará las características de cada muestra.

3.2.1. Requisitos no funcionales

RNF01 - La aplicación se ejecutará sobre un sistema operativo Android.

RNF02 - La aplicación móvil será compatible desde sistemas Android 2.3 (Gingerbread)

RNF03 - La funcionalidad de la aplicación se encontrará solamente en el dispositivo móvil. Es decir, su funcionamiento será local, tanto en almacenamiento de datos, como lógica de la aplicación.

RNF04 - Sólo existirá un patrón (como máximo) en la aplicación.

RNF05 - Cada muestra de un patrón, se almacenará en un fichero. La estructura de estos ficheros se detalla en el requisito de información 2.

RNF06 - En la aplicación existirán 3 niveles de seguridad en el acceso (bajo,medio y alto).

RNF07 - Esta solución, se realizará siguiendo el proceso marcado por OpenUP.

RNF08 - La aplicación no aceptará muestras de tamaño menor a 10 puntos.

RNF09 - La aplicación introducirá esperas entre toma de una muestra y otra. Esto es debido a que de esta forma, las muestras tengan cierta variabilidad entre ellas.

3.2.2. Usabilidad

RNF010 - Un usuario con conocimiento de uso de un smartphone podrá utilizar todas las funcionalidadeses de las que dispone el sistema en un breve período de tiempo.

RNF011 - Los mensajes de error presentados en la aplicación deben poder ser inequívocamente identificados por el usuario a través del uso de una descripción adecuada.

3.2.3. Rendimiento

RNF012 - El rendimiento de las funciones del sistema (recogida de muestras, reconocimiento de patrón...) será el más rápido posible.

3.2.4. Mantenibilidad

RNF013 - El código del sistema deberá estar completamente documentado.

RNF014 - El sistema deberá ser lo más simple posible y claro para su futuro mantenimiento.

RNF015 - El mantenimiento del sistema se realizará mediante el uso de los resources que proporcionan las actividades Android junto con la implementación de ciertas variables del código mediante constantes.

3.2.5. Seguridad

RNF016 - El sistema contará los siguientes tres ficheros log, sólo visibles mediante el uso del IDE Eclipse:

- Errores: Muestra los errores de la aplicación, como por ejemplo falta de permisos.
- Comparación de patrones: Guarda todos los intentos de entrada en el sistema.
- Escritura de ficheros: Guarda todos los cambios de estado realizados.

RNF017 - Los ficheros de muestra se guardarán en el espacio reservado para la aplicación en memoria interna, impidiendo a otras aplicaciones acceder a ellos.

3.3. Requisitos de información

RI01 - El modelo se compondrá de 3 muestras.

RI02 - El sistema guardará información sobre las muestras. Esta información, será la de cada punto de la misma. Se guardarán los parámetros detallados en 2.1 De esta forma, cada fichero tendrá los puntos de la muestra, siendo cada punto una linea en el fichero con el formato X,Y,T,P.

RI03 - El sistema guardará ficheros de inscripción para representar el patrón

RI04 - La distancia de una muestra respecto a un patrón, es la menor de las distancias de esa muestra respecto a cada una de las muestras del modelo.

3.4. Diagrama de casos de uso

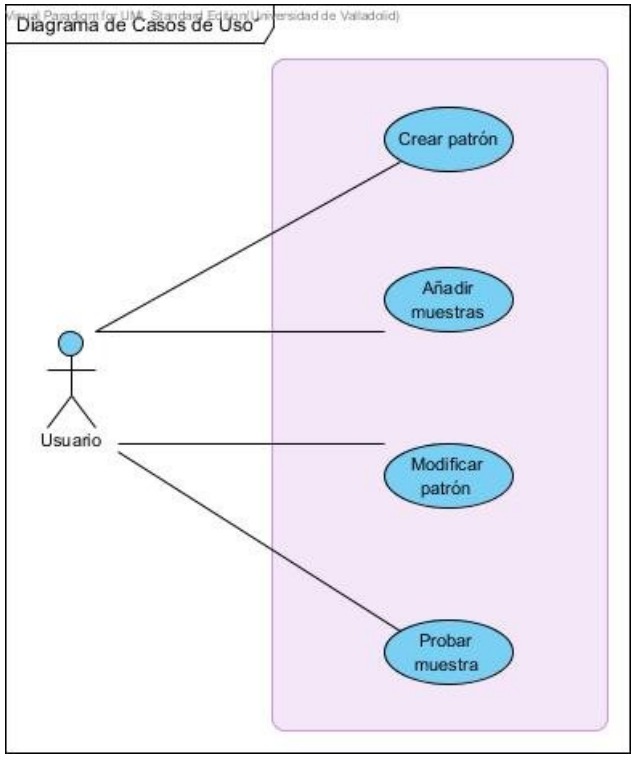


Figura 3.1: Diagrama de casos de uso

3.5. Casos de uso

3.5.1. CU01 - Creación de patrón

Descripción

Un usuario desea crear un patrón en la aplicación

Actores

- Usuario de la aplicación

Precondiciones
No hay precondiciones

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario abre la aplicación.
2. El sistema pide al usuario que introduzca una muestra para crear el modelo correspondiente al patrón.
3. El usuario dibuja un patrón libre y la confirma en el sistema.
4. El sistema guarda la muestra y vuelve al paso 2 hasta obtener 3 muestras.
5. El caso de uso termina.

Flujo alternativo

- **Sin dibujo**

  Si en el paso 3, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra.
  2. El caso de uso continua en el paso 2

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 3, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.

Escenarios clave
El usuario Carlos Eduardo, va a probar la aplicación. Lo primero que debe hacer, es crear un patrón para poder usar la aplicación. Para ello, introduce 3 veces un dibujo de una moneda de euro formado por 4 trazos. Tras hacerlo, Carlos llega a la pantalla de prueba de muestra, para si quiere, probar el patrón que acaba de definir.

Postcondiciones

- **Finalización satisfactoria:** El usuario habrá definido un patrón.

3.5.2. CU02 - Prueba de patrón

Descripción
Un usuario desea probar su patrón en la aplicación y ver que nivel de seguridad de acceso tiene.

Actores
- Usuario de la aplicación

Precondiciones
La aplicación debe tener un patrón definido.

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario llega a la pantalla de acceso.
2. El sistema pide al usuario que introduzca una muestra para probar el patrón.
3. El usuario realiza una muestra.
4. El sistema de compara el resultado del clasificador con cada umbral de decisión. Tras ello, muestra los resultados al usuario.
5. El caso de uso termina.

Flujo alternativo

- **Sin dibujo**

  Si en el paso 3, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra.
  2. El caso de uso continua en el paso 2

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 3, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.

Escenarios clave
El usuario Francisco López ya ha creado su patrón en la aplicación. Ahora quiere saber, qué nivel de acceso tiene al repetirle. Para ello, dibuja una muestra del patrón obteniendo como respuesta los niveles pasados.

Postcondiciones

- **Finalización satisfactoria** El usuario obtendrá un los resultados de nivel de acceso.

3.5.3. CU03 - Modificar patrón

Descripción
Un usuario desea probar a modificar su patrón en la aplicación.

Actores
- Usuario de la aplicación

Precondiciones
La aplicación debe tener un patrón definido.

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario desea cambiar su patrón.
2. El sistema elimina el patrón actual
3. El sistema pide al usuario que introduzca una muestra del patrón nuevo
4. El usuario realiza una muestra.
5. El sistema repite el paso 3 hasta obtener 3 muestras.
6. El caso de uso termina.

Flujos alternativos

- **Sin dibujo**

  Si en el paso 4, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra.
  2. El caso de uso continua en el paso 3

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 4, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.

Escenarios clave
El usuario Lope Rodríguez ha olvidado su patrón. Para poder hacer la prueba de patrón, decide modificar el patrón, introduciendo 3 muestras nuevas. Tras ello, se decide a realizar una prueba de este patrón recién definido.

Postcondiciones

- **Finalización satisfactoria** El usuario habrá definido un patrón.

3.5.4. CU04 - Añadir muestras

Descripción
Un usuario desea añadir otras dos muestras de inscripción para mejorar el reconocimiento

Actores
- Usuario de la aplicación

Precondiciones

La aplicación debe tener un patrón definido.

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario solicita añadir mas muestras.
2. El sistema pide al usuario que introduzca una muestra para añadir el modelo.
3. El usuario dibuja una muestra y la confirma en el sistema.
4. El sistema guarda la muestra y vuelve al paso 2 hasta obtener 2 muestras.
5. El caso de uso termina.

Flujo alternativo

- **Sin dibujo**

  Si en el paso 3, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra.
  2. El caso de uso continua en el paso 2

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 3, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.
- **Realizada segunda sesión** Si el usuario ya ha realizado la segunda sesión del patrón, no se le permitirá volver a hacerla.

Escenarios clave

El usuario Carlos Eduardo, ya había definido un patrón anteriormente. Tras probarle varias veces, es consciente de que sus niveles de reconocimiento soy bastante malos. Por ello, decide añadir otras dos muestras a su patrón, para que el sistema tenga un mayor abanico de posibilidades para realizar pruebas.

Postcondiciones

- **Finalización satisfactoria:** El usuario habrá definido un patrón.

3.6. Matriz de trazabilidad

En este apartado se va a mostrar la matriz de trazabilidad, en la cual se relacionan los requisitos funcionales con los casos de uso

- **Requisitos funcionales**

RF01 - El sistema permitirá al usuario crear un patrón.

RF02 - El sistema permitirá al usuario realizar muestras de inscripción.

RF03 - El sistema permitirá al usuario redefinir un patrón.

RF04 - El sistema permitirá borrar una muestra de la pantalla antes de confirmarla.

RF05 - El sistema permitirá realizar pruebas de ingreso del patrón en la aplicación.

RF06 - El sistema mostrará al usuario, cada nivel de seguridad en el acceso mostrando si ha sido superado o no, en una prueba de ingreso de patrón.

RF07 - El sistema permitirá al usuario elegir entre una o dos sesiones de inscripción.

RF08 - El sistema calculará los umbrales a superar a partir de las muestras de inscripción y unos ficheros de cohorte.

RF09 - El sistema calculará las características de cada muestra.

- **Casos de uso**.

CU01 - Creación de patrón.

CU02 - Prueba de muestra.

CU03 - Modificación de patrón.

CU04 - Añadir muestras.

	CU01	CU02	CU03	CU04
RF01	**X**			
RF02	**X**	**X**	**X**	**X**
RF03			**X**	
RF04	**X**	**X**	**X**	**X**
RF05		**X**		
RF06		**X**		
RF07				**X**
RF08	**X**		**X**	**X**
RF09	**X**	**X**	**X**	**X**

Figura 3.2: Matriz de trazabilidad

3.7. Diagramas de secuencia

En este apartado se presentan a continuación los diagramas de secuencia de los distintos casos de uso. Al ser el tamaño de página de la memoria pequeño, aqui se encuentran los diagramas para poder hacernos una idea de como se comporta el sistema. En el CD adjunto a esta memoria, se pueden encontrar estos mismos diagramas en alta resolución, para poder verles mejor.

3.7.1. CU01 - Creación de patrón

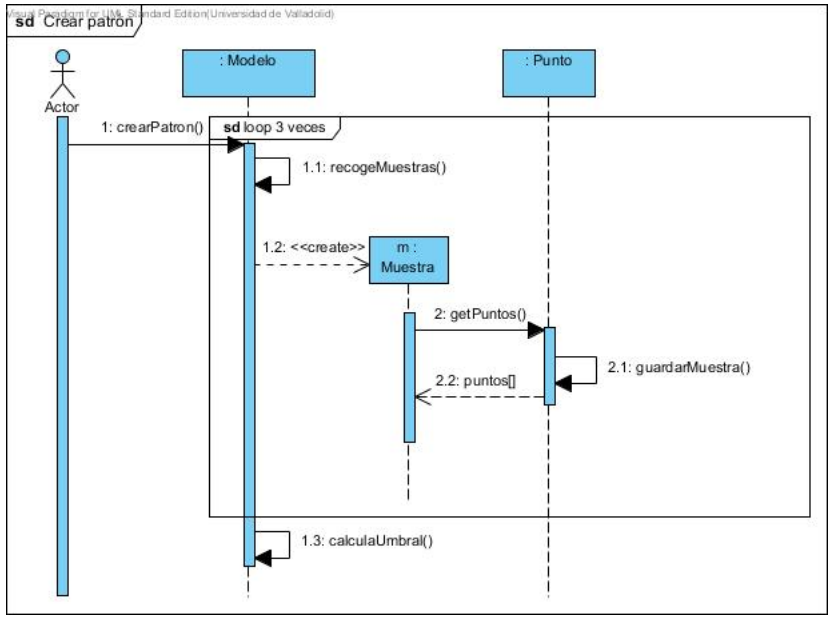


Figura 3.3: Diagrama de secuencia - Creación de patrón

3.7.2. CU02 - Prueba de muestra

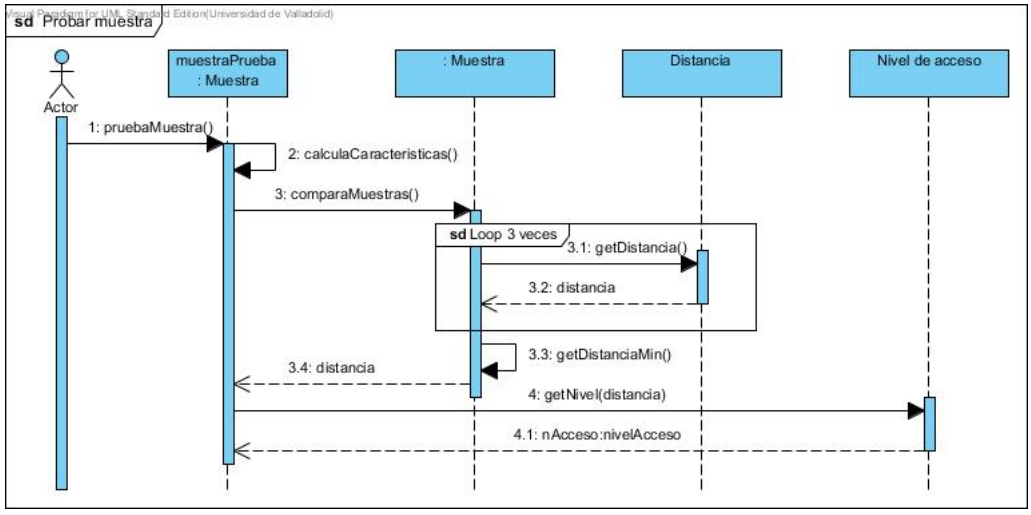


Figura 3.4: Diagrama de secuencia - Prueba muestra

3.7.3. CU03 - Modificación de patrón

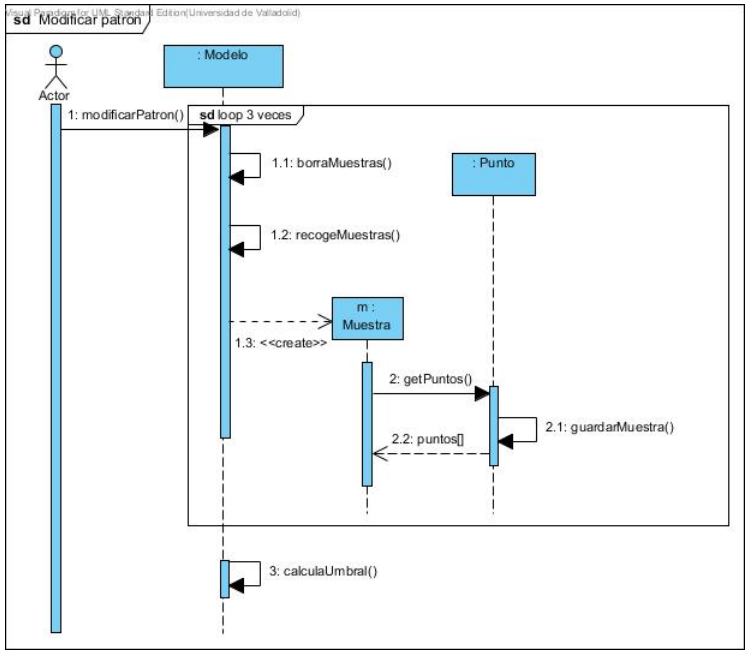


Figura 3.5: Diagrama de secuencia - Modificación de patrón

3.7.4. CU04 - Añadir muestras

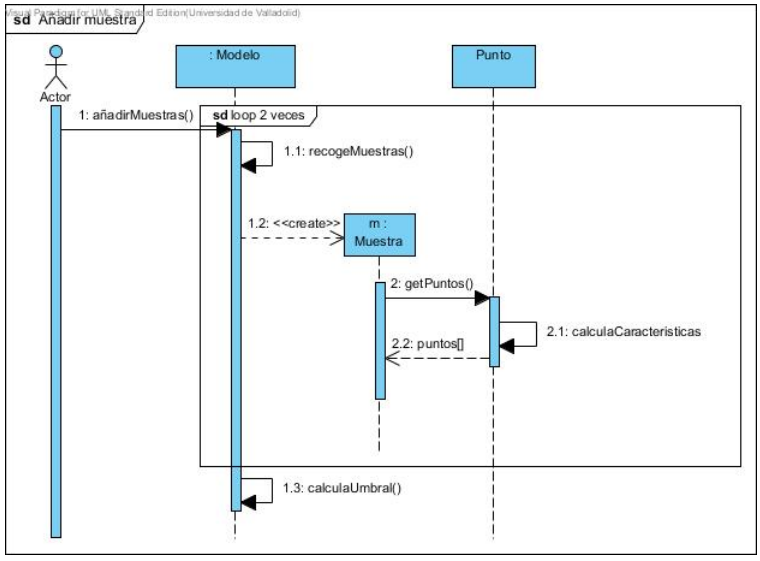


Figura 3.6: Diagrama de secuencia - Añadir muestras

3.8. Modelo de dominio

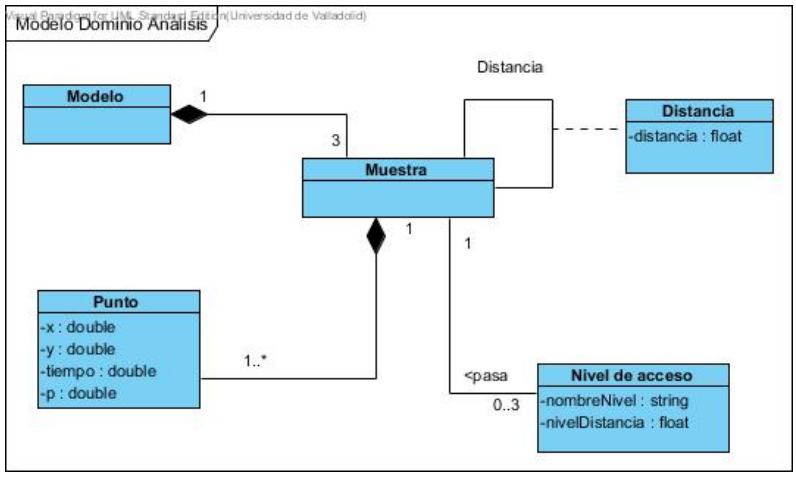


Figura 3.7: Modelo de dominio

3.8.1. Descripción de las clases del modelo

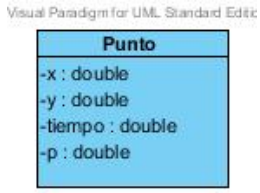


Figura 3.8: Clase punto

Esta clase representa los puntos de cada muestra. Sus atributos son:

- **x**: Atributo de la clase en la que se guarda en valor de la coordenada x del punto.
- **y**: Atributo de la clase en la que se guarda en valor de la coordenada y del punto.
- **tiempo**: Atributo de la clase que representa el tiempo transcurrido entre el punto y el punto anterior a este.
- **p**: Atributo booleano que indica, si el punto es el primero de un trazo de la muestra o no.

Figura 3.9: Clase muestra

Esta clase, es una agrupacion de al menos un punto. Esto significa que si la clase “Muestra” desapareciera del dominio, la clase “Punto” perdería su significado en el mismo.

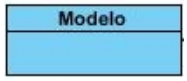


Figura 3.10: Clase modelo

Esta clase, es una agrupacion de muestras. Esto significa que si la clase “Patrón” desapareciera del dominio, la clase “Muestra” perdería su significado en el mismo.

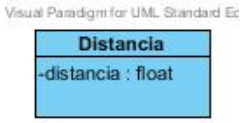


Figura 3.11: Clase distancia

Esta clase, representa la asociación entre dos muestras. Esta clase contiene un atributo **distancia** que representa la distancia entre ambas muestras.

Figura 3.12: Clase nivel de acceso

Esta clase, representa los distintos niveles que una muestra puede alcanzar. A partir de esta clase de análisis, se crean lo niveles de acceso, que una muestra supera respecto a otra.

3.9. Arquitectura software

Lo primero que se debe hacer, al diseñar una aplicación, ya sea de tipo web o una aplicación local (la cual es nuestro caso), es definir una arquitectura software sobre la cual va a funcionar. En el caso de nuestra solución, la arquitectura software nos viene dada por el propio entorno en la cual va a correr. En los sistemas Android, la arquitectura que se usa es la arquitectura de capas que se detalla en la imagen de la página siguiente. En concreto, la arquitectura utilizada en nuestro caso es la que se define en la figura 3.13:

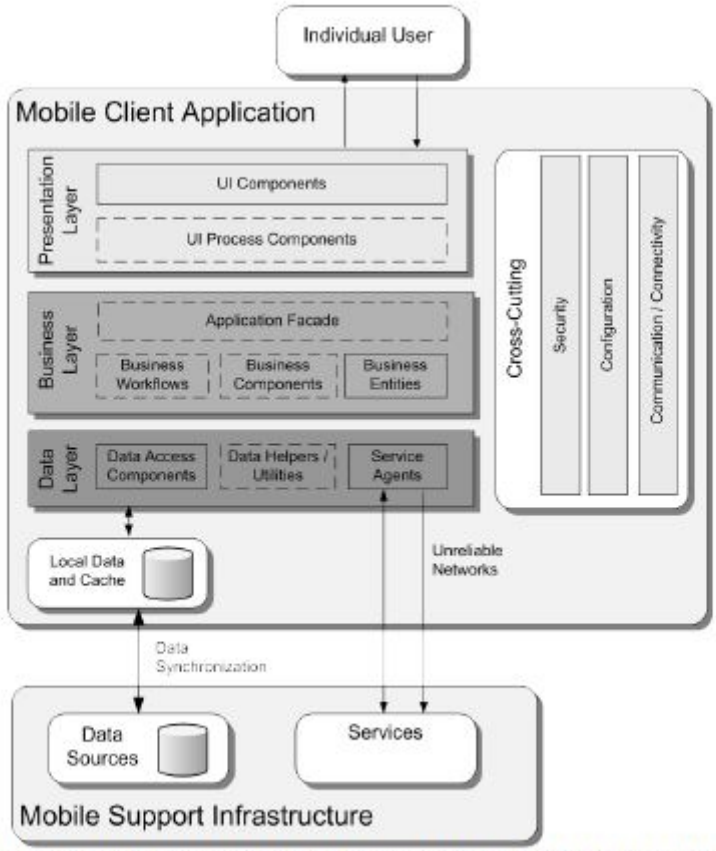


Figura 3.13: Arquitectura capas. Tomada del documento [12]

Como se ve en la figura, esta arquitectura está formada por 4 capas:

- *Capa de presentación (Presentation Layer).* En esta capa, se coloca todo aquello relacionado con la interacción del usuario, desde mostrar

la información hasta recoger y procesar datos del usuario para el funcionamiento de la aplicación.

- *Capa de negocio (Bussiness Layer).* En esta capa es en donde se encuentra la funcionalidad propiamente dicha de la aplicación. Es en esta capa en la que se encontrará, en nuestro caso, el reconocimiento de patrones.
- *Capa de datos (Data Layer).* Como toda aplicación ligeramente compleja, se necesita guardar información. Es en esta capa, la encargada de mantener una conexión entre la aplicación y sus datos.
- *Capa transversal (Cross-Cutting).* En esta capa, se localiza toda aquella funcionalidad transversal, como por ejemplo, la seguridad de la aplicación o los mecanismos de mantenibilidad.

Además de utilizar esta arquitectura software, también se incluirá el patrón arquitectónico MVC, el cual permite dividir la funcionalidad de cada capa de la arquitectura de móviles Android. El funcionamiento básico de este patrón se detalla en la siguiente imagen, donde se muestra la secuencia de mensajes que se realizan entre sus tres componentes:

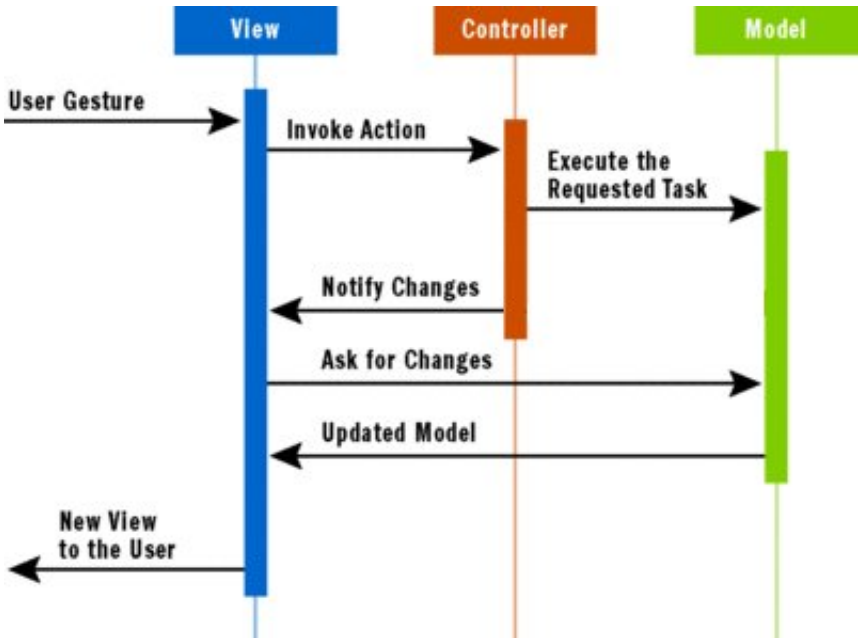


Figura 3.14: Comportamiento del patrón MVC

De esta forma, cada uno de los componentes del patrón, se localizarán en una de las capas de la arquitectura software, siendo estas localizaciones:

- Modelo: Capa de datos.

- Vista: Capa de presentación.
- Controlador: Capa de negocio.

Como ya se ha indicado anteriormente en el requisito 3, la aplicación que se va a realizar, va a correr en el propio móvil, sin necesidad de servicios externos, como APIs o servidores web. Por ello, se ha tomado al decisión de tener un sistema con un cliente rico, en el cual residirá toda la funcionalidad de la aplicación. Debido a esto, el despliegue de este software se realizará en dos niveles, los cuales se detallan en el siguiente diagrama de despliegue:

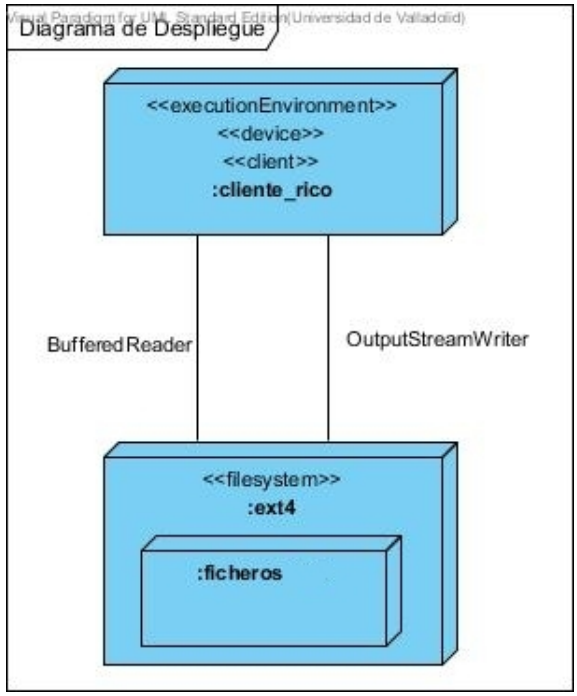


Figura 3.15: Diagrama de despliegue

3.10. Casos de uso de diseño

3.10.1. CU01 - Creación de patrón

Descripción

Un usuario desea crear un patrón en la aplicación

Actores

- Usuario de la aplicación

Precondiciones
No hay precondiciones

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario abre la aplicación, pulsando en el icono de la misma en el menú de aplicaciones.

2. El sistema pide al usuario que introduzca una muestra para crear el patrón, mediante un mensaje por pantalla.

3. El usuario dibuja una muestra y la confirma en el sistema pulsando el botón correspondiente.

4. El sistema guarda la muestra y vuelve al paso 2 hasta obtener 3 muestras.

5. El caso de uso termina.

Flujo alternativo

- **Sin dibujo**

  Si en el paso 3, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra mediante un mensaje.
  2. El caso de uso continua en el paso 2

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 3, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.

Escenarios clave
El usuario Carlos , va a probar la aplicación. Lo primero que debe hacer, es crear un patrón para poder usar la aplicación. Para ello, introduce 3 veces un dibujo de una moneda de euro formado por 4 trazos. Tras hacerlo, Carlos llega a la pantalla de prueba de muestra, para si quiere, probar el patrón que acaba de definir.

Postcondiciones

- **Finalización satisfactoria:** El usuario habrá definido un patrón.

3.10.2. CU02 - Prueba de patrón

Descripción

Un usuario desea probar a repetir su patrón en la aplicación y ver que nivel de acceso tiene.

Actores

- Usuario de la aplicación

Precondiciones

La aplicación debe tener un patrón definido.

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario llega a la pantalla de acceso.
2. El sistema pide al usuario que introduzca una muestra para crear el patrón, mostrandole la pantalla para realizar una muestra
3. El usuario dibuja realiza una muestra.
4. El sistema decide que niveles de acceso se pasan con esa muestra. Tras ello, muestra los resultados al usuario.
5. El caso de uso termina.

Flujo alternativo

- **Sin dibujo**

  Si en el paso 3, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra.
  2. El caso de uso continua en el paso 2

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 3, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.

Escenarios clave

El usuario Francisco López ya ha creado su patrón en la aplicación. Ahora quiere saber, qué nivel de acceso tiene al repetirle. Para ello, dibuja una muestra del patrón obteniendo como respuesta otra pantalla donde se le indica que niveles ha pasado y que niveles no.

Postcondiciones

- **Finalización satisfactoria** El usuario obtendrá un los resultados de nivel de acceso.

3.10.3. CU03 - Modificar patrón

Descripción
Un usuario desea probar a modificar su patrón en la aplicación.

Actores
- Usuario de la aplicación

Precondiciones
La aplicación debe tener un patrón definido.

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario desea cambiar su patrón, pulsando la opción correspondiente de la aplicación.
2. El sistema elimina el patrón actual.
3. El sistema pide al usuario que introduzca una muestra del patrón nuevo mediante un mensaje.
4. El usuario dibuja realiza una muestra.
5. El sistema repite el paso 3 hasta obtener 3 muestras.
6. El caso de uso termina.

Flujos alternativos

- **Sin dibujo**

  Si en el paso 4, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra.
  2. El caso de uso continua en el paso 3

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 4, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.

Escenarios clave
El usuario Lope Rodríguez ha olvidado su patrón. Para poder hacer la prueba de patrón, decide modificar el patrón, introduciendo 3 muestras nuevas. Tras ello, se decide a realizar una prueba de este patrón recién definido.

Postcondiciones

- **Finalización satisfactoria** El usuario habrá definido un patrón.

3.10.4. CU04 - Añadir muestras

Descripción

Un usuario desea añadir otras dos muestras para mejorar el reconocimiento

Actores

- Usuario de la aplicación

Precondiciones

El usuario debe tener un patrón definido.

Flujo normal de eventos

1. El caso de uso comienza cuando el usuario elige añadir dos muestras más, pulsando en la opción adecuada del menú contextual de la aplicación.
2. El sistema pide al usuario que introduzca una muestra para añadir al patrón, mediante un mensaje por pantalla.
3. El usuario dibuja una muestra y la confirma en el sistema pulsando el botón correspondiente.
4. El sistema guarda la muestra y vuelve al paso 2 hasta obtener 2 muestras.
5. El caso de uso termina.

Flujo alternativo

- **Sin dibujo**

  Si en el paso 3, el usuario no ha dibujado nada.

  1. El sistema avisará al usuario de que debe introducir una muestra mediante un mensaje.
  2. El caso de uso continua en el paso 2

- **Equivocación del usuario:** Si mientras el usuario realiza el paso 3, se ha equivocado al dibujar este podrá borrar la muestra realizada de la pantalla y volver al dibujarla.
- **Realizada segunda sesión** Si el usuario ya ha realizado la segunda sesión del patrón, no se le permitirá volver a hacerla.

Escenarios clave

El usuario Carlos Eduardo, quiere mejorar el reconocimiento de su patrón. Para ello, pulsa en la opción, añadir muestras del menu contextual de su dispositivo . En ella, él introduce dos veces un dibujo de una moneda de un euro como hizo en el caso de uso 3.10

Postcondiciones

- **Finalización satisfactoria:** El usuario habrá definido un patrón.

3.11. Diagramas de secuencia de diseño

En este apartado se presentan a continuación los diagramas de secuencia de los distintos casos de uso. Al ser el tamaño de página de la memoria pequeño, aqui se encuentran los diagramas para poder hacernos una idea de como se comporta el sistema. En el CD adjunto a esta memoria, se pueden encontrar estos mismos diagramas en alta resolución, para poder verles mejor.

3.11.1. CU01 - Creación de patrón

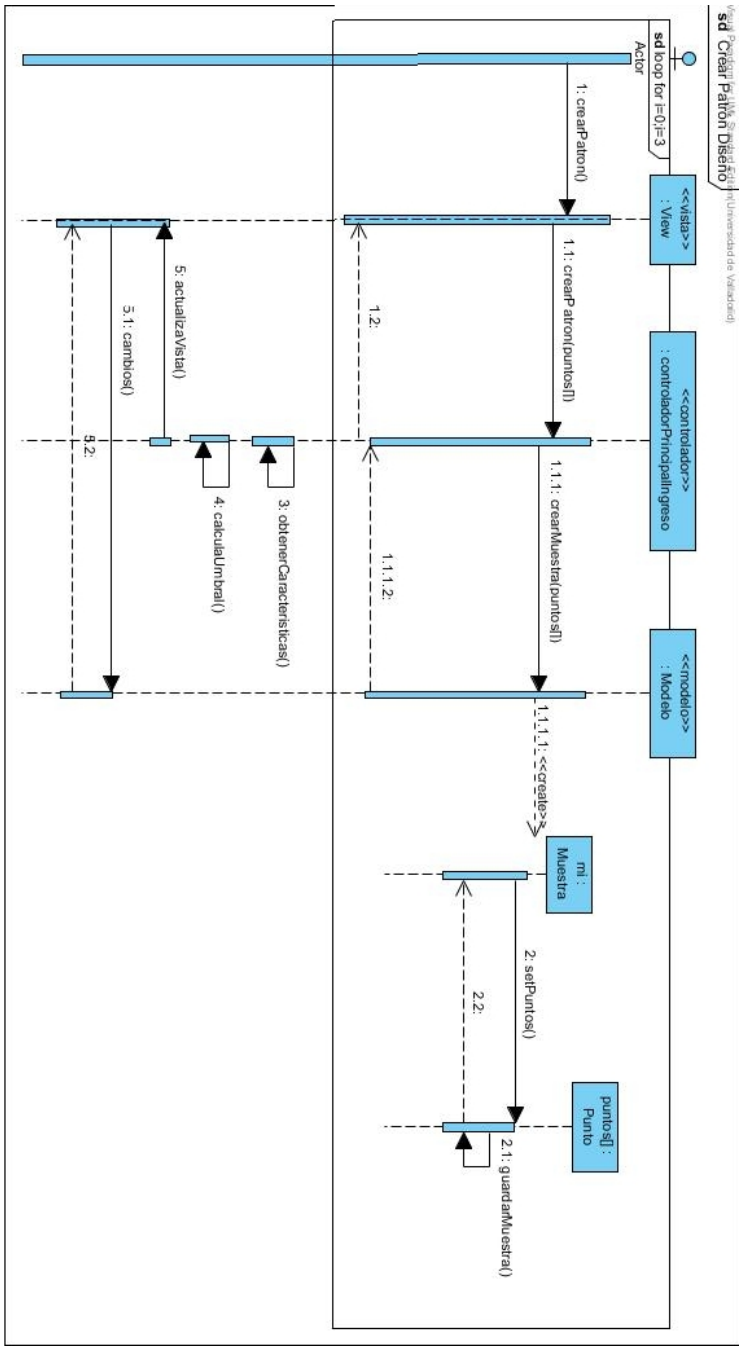


Figura 3.16: Diagrama de secuencia - Creación de patrón

3.11.2. CU02 - Prueba de muestra

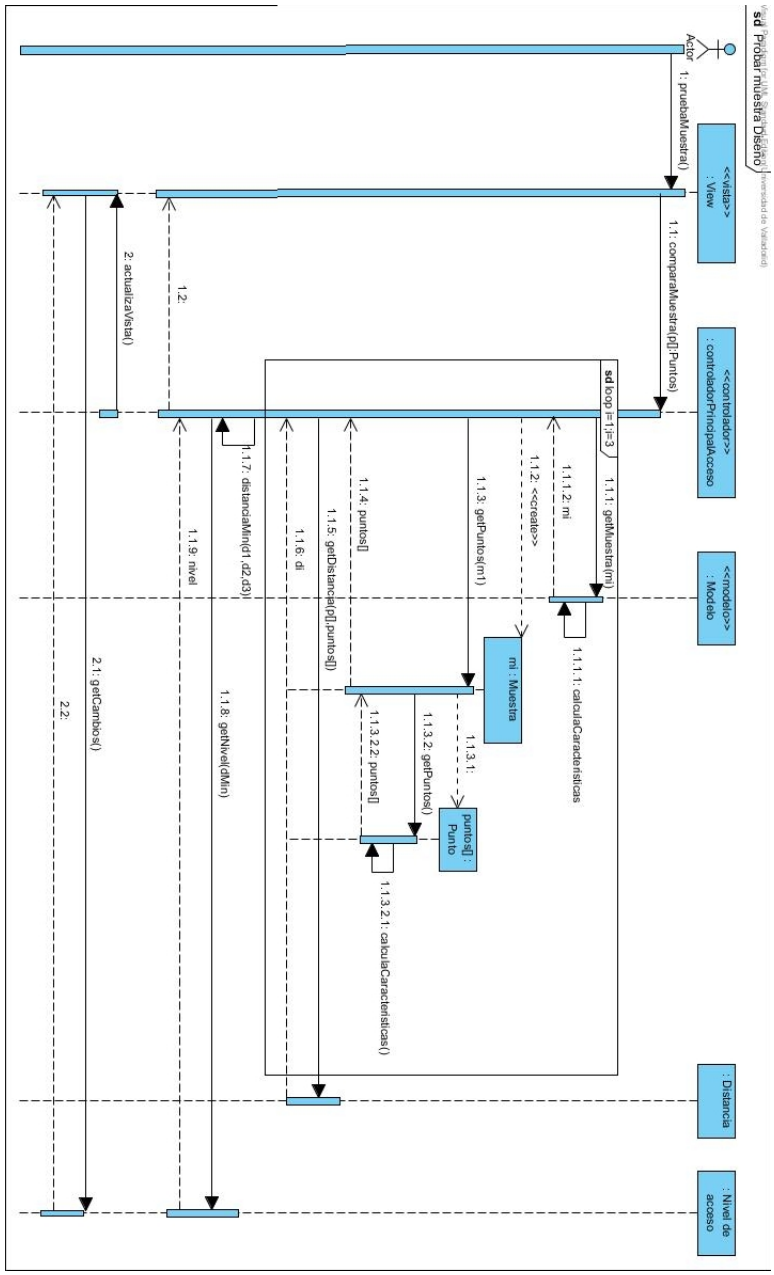


Figura 3.17: Diagrama de secuencia - Prueba muestra

3.11.3. CU03 - Modificación de patrón

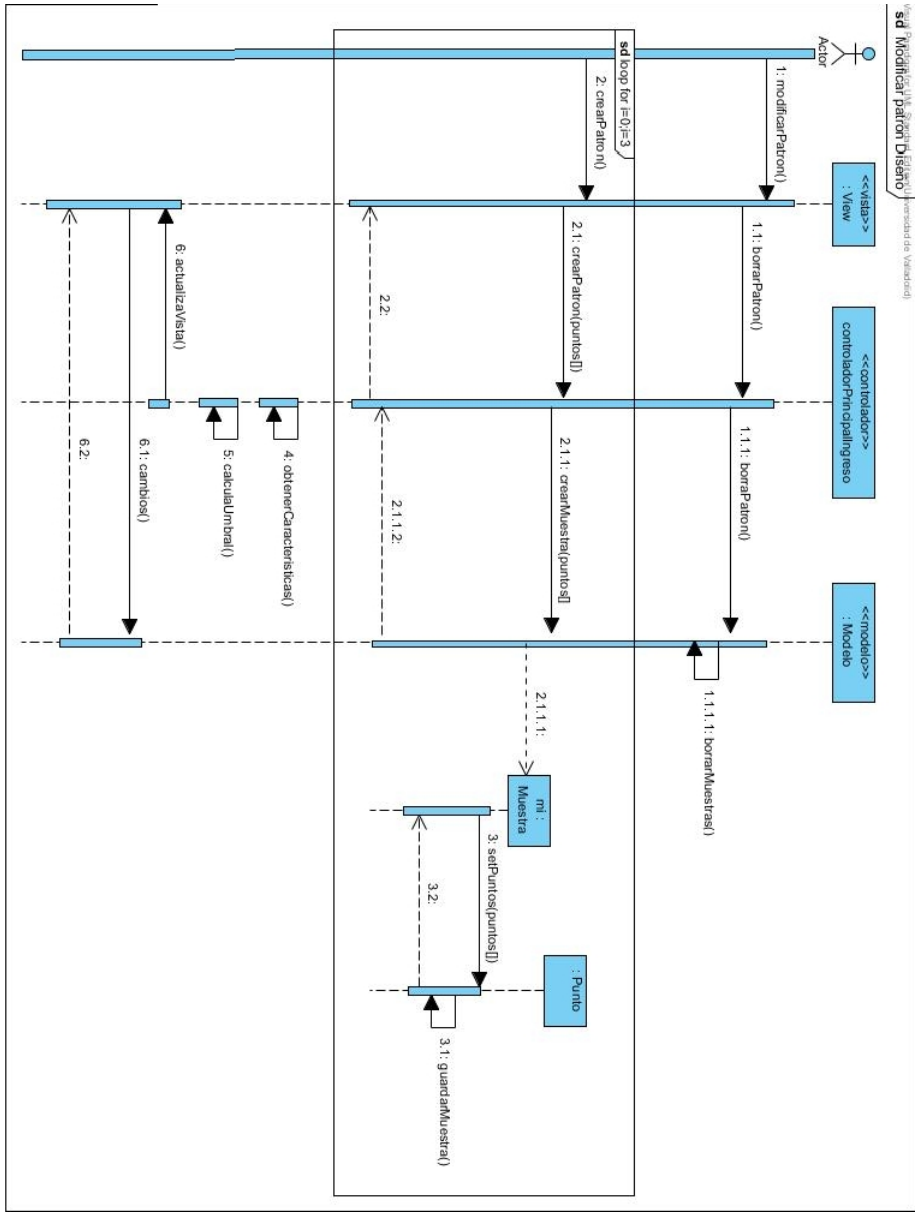


Figura 3.18: Diagrama de secuencia - Modificación de patrón

3.11.4. CU04 - Añadir muestras

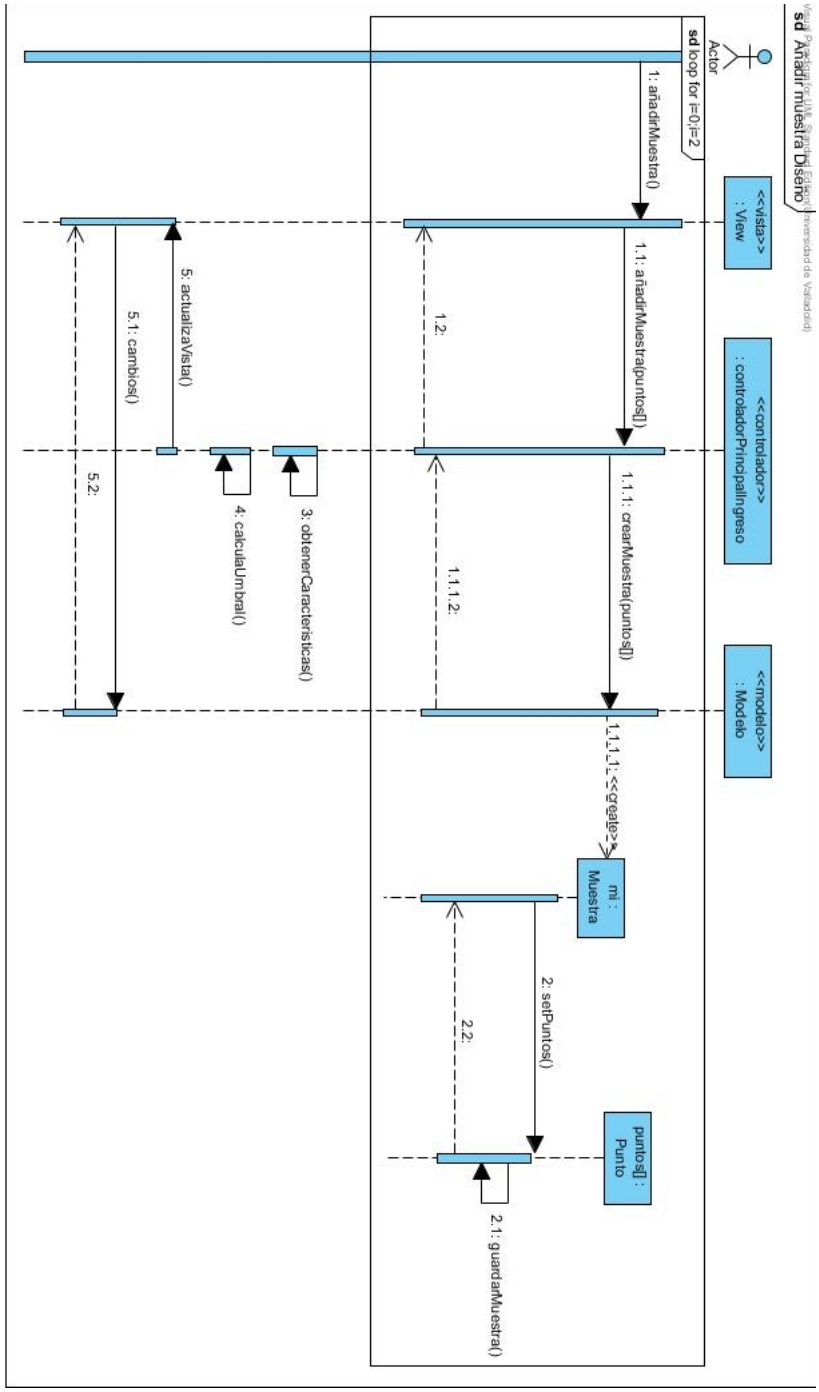


Figura 3.19: Diagrama de secuencia - Añadir muestras

3.12. Modelo de dominio de diseño

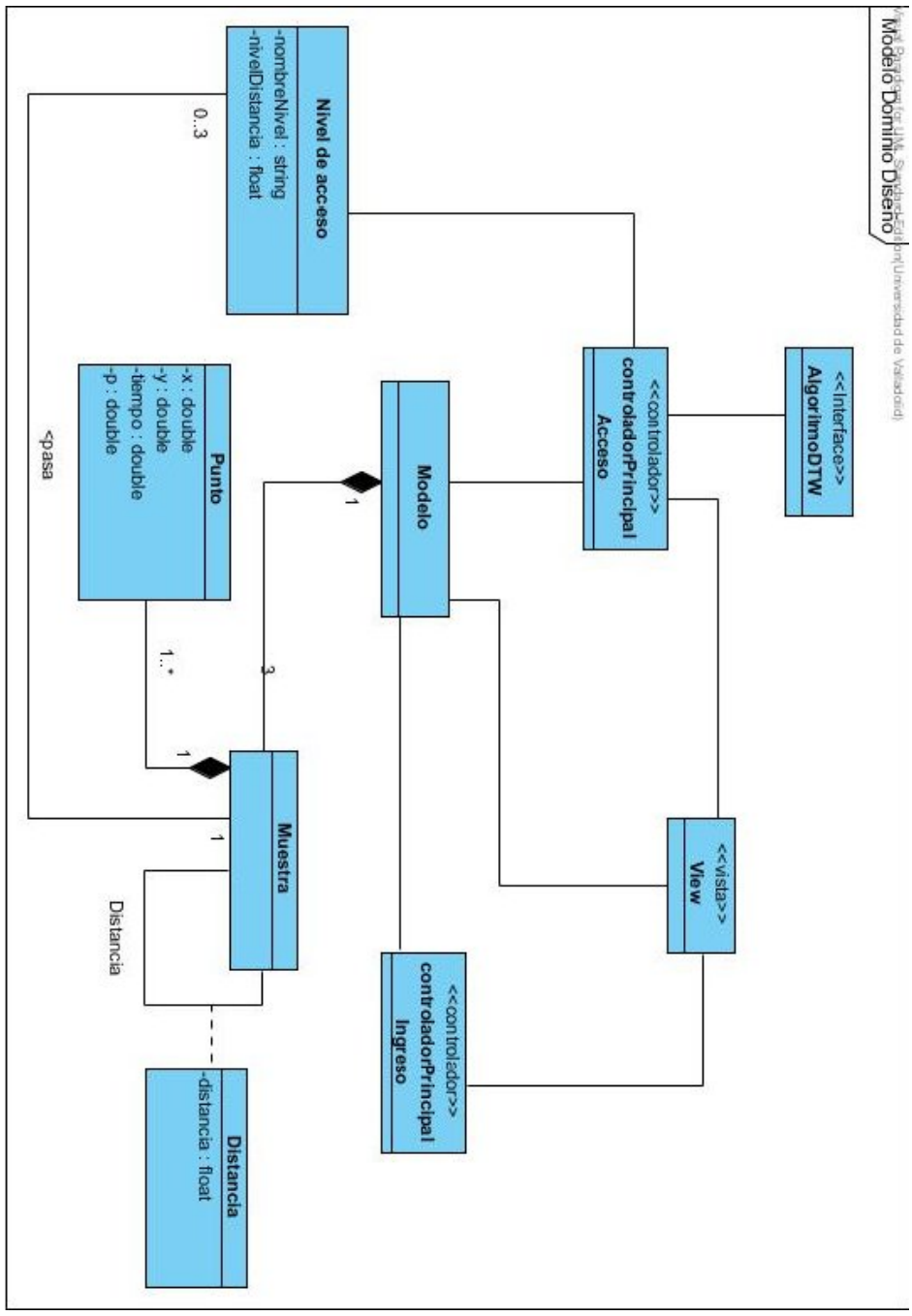


Figura 3.20: Modelo de dominio en la fase de diseño

3.12.1. Descripción de las nuevas clases del modelo

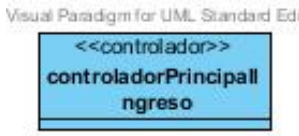


Figura 3.21: Controlador de ingreso

Esta clase, representa toda la lógica de negocio, relacionada con el ingreso de un patrón en la aplicación.

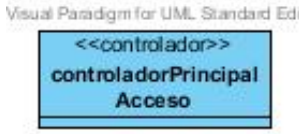


Figura 3.22: Clase Controlador de acceso

Esta clase, representa toda la lógica de negocio, relacionada con la prueba de un patrón en la aplicación.

Figura 3.23: Clase vista

Esta clase, representa las distintas pantallas que tiene la aplicación.

Capítulo 4

Implementación y Pruebas

En este capítulo se van a detallar las herramientas que se han usado para la implementación de la aplicación. También , se va a mostrar el diseño de las pruebas junto con los resultados arrojados en cada fase de pruebas.

4.1. Entorno de desarrollo

El objetivo de esta sección es mencionar aquellas herramientas que han sido relevantes para el desarrollo del proyecto, tanto hardware como software.

Debido a que, uno de los requisitos es que la aplicación funcione en un sistema operativo Android, el entorno de desarrollo elegido es el del uso de Android SDK. Este entorno nos proporciona un depurador de código especializado para sistemas Android llamado DDMS, todas las bibliotecas necesarias para realizar un desarrollo de aplicaciones Android y la documentación necesaria de las herramientas proporcionadas. Junto a ello, debido a que las aplicaciones Android están escritas, en Java y XML, se ha elegido utilizar el IDE Eclipse para su desarrollo. Este entorno es especialemente útil ya que proporciona un editor gráfico para diseñar la parte gráfica de la aplicación.

Para poder depurar la aplicación se ha elegido como hardware, un móvil personal de gama media. Esto es debido, a que, aunque el SDK nos proporciona un emulador , este es bastante lento e impreciso debido al retardo que provoca. Para poder correr la aplicación en el dispositivo, se debe instalar el driver USB del terminal en el ordenador y poner el dispositivo en modo depuración en la pantalla de ajustes.

4.2. Desarrollo de la interfaz

La recogida de las muestras, es una de las partes mas importantes de la aplicación. Por ello es, necesario que la respuesta de la pantalla, sea óptima. Para ello, Android proporciona una vista llamada "DrawView". El problema de ello es que no nos devuelve los datos como se necesitan. Por esta razón, para la recogida de las muestras, se ha personalizado este tipo de vista consiguiendo los puntos como necesitamos en nuestra aplicación.

En lo referente a la parte gráfica de la aplicación se ha realizado una parte de estudio de la interfaz. Para llevar a cabo, esta tarea se ha realizado un prototipo, el cual ha sido probado por un grupo de 10 personas. Gracias a este proceso, se han encontrado los siguientes problemas:

- Al presionar los botones inferiores de "ok" y "limpiar", el usuario no tenía realimentación, creyendo en algunas ocasiones que estos no habían sido pulsados.
- Al ser, tenerse que producir esperas entre recogida de muestra y recogida de muestra del patrón (ya sea por guardar el fichero de modelo o calcular umbrales), se ha introducido un mensaje de espera que dura varios segundos. De esta forma, el usuario obtiene realimentación de la aplicación.
- Al ser tan similares las vistas de recogida de muestras en la primera sesión, recogida de muestras en la segunda sesión y prueba de muestra, se ha introducido en la parte superior de la pantalla en qué pantalla se encuentra el usuario.

4.3. Pruebas

Es necesario seguir un plan de pruebas para comprobar que la aplicación funciona correctamente. Se presenta, por lo tanto, el plan de pruebas tanto de caja negra como de caja blanca que se ha seguido para la depuración del programa, con el fin de que funcione correctamente y de la manera más segura posible. Esta aplicación está pensada para realizar pruebas de reconocimiento. Esto implica una falta de condiciones para su funcionamiento (permisos distintos según usuarios, entradas variables con condiciones...). Por lo tanto, la batería de pruebas a realizar no será muy extensa.

4.3.1. Pruebas de caja negra

Para la realización de pruebas de caja negra se ha elegido el método de particiones en casos de equivalencia, en el cual se divide la correcta

funcionalidad de la aplicación en casos que se deben comprobar. De esta forma, el pasar las pruebas, es simplemente ir comprobando cada uno de todos los casos de prueba.

Particiones en casos de equivalencia

Condición	Casos válidos	Casos no válidos
Tamaño muestra	El tamaño de la muestra es de al menos 10 puntos (1)	El tamaño de la muestra es de al menos 10 puntos (3)
Falta de permisos	La aplicación tiene permisos para guardar en memoria interna (2)	La aplicación no tiene permisos para guardar en memoria interna (4)

Tabla 4.1: Particiones en casos de equivalencia

Casos de prueba

	Casos de prueba	Casos cubiertos	Salida esperada
✓	1.1 El usuario ha introducido una muestra de tamaño mayor a 10. Además de ello, la aplicación ha podido guardarla en memoria interna.	1,2	OK
✓	1.3 El usuario ha introducido una muestra de tamaño menor a 10.	3	Error
✓	1.4 Tras introducir una muestra, la aplicación no ha podido guardarla en memoria interna por falta de permisos	4	Error

Tabla 4.2: Casos de prueba

4.3.2. Pruebas de caja blanca

Las pruebas de caja blanca son aquellas pruebas necesarias para comprobar el correcto funcio- namiento de la aplicación en el ámbito interno. Para ello, es necesario tener una visión del flujo de la aplicación con el fin de comprobar que no falta ninguna conexión entre las distintas actividades Android, de la aplicación. El flujo de la aplicación es el siguiente:

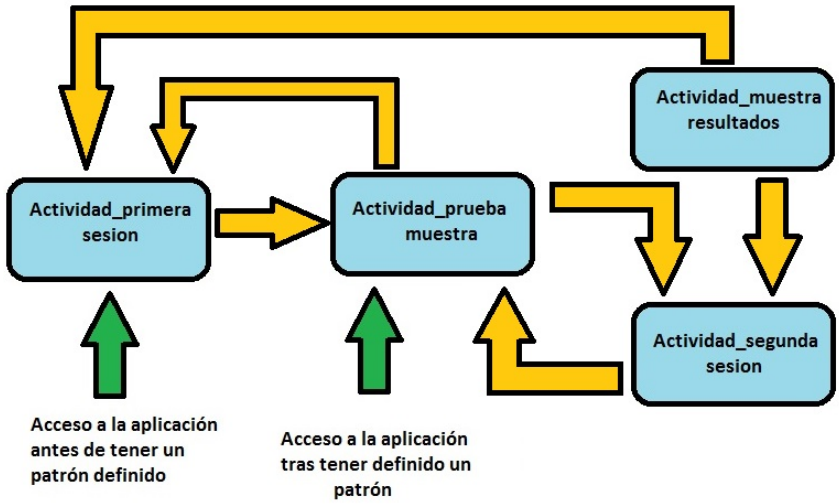


Figura 4.1: Flujo de la aplicación

A partir, de este flujo, se obtuvieron estos problemas:

- Redefinir el patrón o añadir muestras, debería ser accesible desde las pantallas de prueba y muestra de resultados.
- No era necesaria una segunda actividad para modificar un patrón, debido a que ya se tenía la de ingreso de primera sesión.

Capítulo 5

Pruebas del reconocedor

En el presente capítulo se muestra un resumen de las pruebas realizadas para fijar la configuración del clasificador. De las partes mostradas en el capítulo 2, las pruebas realizadas se centraron en la de extracción de características y obtención de la configuración óptima en inscripción. El resto de características del sistema son las mostradas en ese capítulo. También se ha realizado un estudio de la distribución del error por usuario. Para estas pruebas se adquirió, mediante una aplicación móvil de que dispone el grupo ECA-SIMM, un pequeño corpus. Se adquirieron patrones de 29 donantes en dos sesiones:

- **Sesión uno**: Cada donante realiza 9 repeticiones de su patrón, libremente escogido. Estas nueve repeticiones son realizadas en grupos de 3, intercalando entre el primer grupo y el segundo y entre este y el tercero, la imitación (3 veces) de patrones de los dos donantes anteriores. En esta sesión esta imitación se realiza viendo la forma del patrón, no su realización (dinámica) . El resultado de la primera sesión son, entonces, 9 muestras auténticas y la imitación, 3 veces cada uno, del patrón de 2 donantes del corpus.

- **Sesión dos**: Para realizar esta sesión, se han esperado varios días respecto a la sesión anterior. En esta sesión, se toma la muestra de patrón del usuario varias veces al igual que en la Sesión 1. La diferencia es que, las fases de imitación son realizadas sabiendo la dinámica de los patrones. de patrones ajenos sin conocer la dinámica del patrón ajeno.

Como medida del rendimiento del reconocedor se ha utilizado la Tasa de Equierror (EER), ya comentada en el apartado 2.2.3. Más concretamente, la EER del cada configuración será la media de las EER obtenidas para cada usuario del corpus.

A la hora de calcular estos errores, se distinguen dos tipos de atacantes o impostores:

- **Aleatorios (random)**: Son impostores que no conocen nuestro patrón, usando por lo tanto uno generado aleatoriamente.
- **Imitadores (skilled)**: Son impostores que conocen nuestro patrón, bien sólo su forma, o bien, además, cómo lo hacemos (caso más desfavorable).

5.1. Elección de características

Un paso del sistema de reconocimiento, es el de extraer las características de cada muestra del patrón libre (véase el apartado 2.2). Esto hace que unas características bien escogidas, proporcionen un mejor reconocimiento. Las características que se han usado para las pruebas son los valores recogidos de los patrones móviles (véase el apartado 2.1) junto con las características descritas en el apartado 2.2.2.

En la tabla 5.1 se muestran los resultados obtenidos.

Numero Experimento	Características	EER-skilled	EER-random
1	x	22,05	8,57
2	y	24,71	10,58
3	x,y	18,19	2,02
4	dx,dy	21,08	2,57
5	Vx,Vy	21,99	6,63
6	x,y,dx	18,04	2,19
7	x,y,dx,dy	18,32	1,69
8	x,y,ddx,ddy	21,42	2,23
9	x,y,dx,dy,ddx,ddy	20,36	2,01
10	**x,y,t**	**17,24**	**1,73**
11	x,y,dx,dy,t	18,81	1,68
12	x,y,p,t	17,17	1,73

Tabla 5.1: Prueba 1.Eleccion de características

Como se puede ver en los resultados, la mejor opción es utilizar las variables recogidas del dispositivo X,Y y T. Usando estas características, se obtiene la mejor protección frente a imitadores que ya conocen como es nuestro patrón. A pesar de que, estas características no ofrecen la mejor distinción frente a impostores aleatorios, la diferencia respecto a otra caractéristicas es lo suficientemente pequeña como para elegir la opción más robusta frente a imitadores.

5.2. Inscripción

Al igual que las características, el número de sesiones de inscripción y muestras recogidas en cada una, son variables importantes a la hora

de afinar en las pruebas el reconocimiento. Para la batería de pruebas, debemos ser conscientes que no podemos poner a prueba un número de inscripciones o muestras de patrones altas. Esto es debido a que la recogida de muestras debe ser cómoda para el usuario.

En la tabla 5.2, se muestran los resultados obtenidos.

Num Exp	Muestras Ent	EER-skilled	EER-random
1	**3 S1 + 2 S2**	**14,10**	**1,16**
2	3 S1 + 3 S2	14,05	1,27
3	5 S1	17,69	1,69

Tabla 5.2: Dependencia de los resultados con respecto al número de muestras y sesiones de inscripción

Como se puede ver en los resultados, el hecho de realizar un mismo número de muestras total en distintas sesiones (Experimentos 1 y 3) hace que la tasa de equierror sea menor. Por otra parte, el realizar un mismo número de sesiones pero aumentar las muestras totales tomadas, consigue mejorar los resultados, pero a costa de (posiblemente) hacer más tedioso el proceso de obtención. Por ello, se han elegido las sesiones y numero de muestras del experimento 1.

5.3. Resultados de imitación dependiendo de conocer la dinámica del patrón

Para que un patrón libre sea útil, este patrón debe ser rechazado (en nuestro caso, no debe pasar ningun nivel de acceso), a pesar de que el imitador sepa como es el patrón. Pero ¿realmente hay tanta diferencia entre conocer la dinámica del patrón y no? Gracias a las pruebas realizadas en el corpus, cuyos resultados se encuentran en la tabla 5.3, podremos saber si esta diferencia existe.

Num Experimento	Muestras introducidas	Prueba de habilidad	Error
1	3 S1	S1 (sin reproducir)	11,03
2	3 S1	S1 (reproduciendo)	19,49
3	3 S1 + 2 S2	S1 (sin reproducir)	9,08
4	3 S1 + 2 S2	S2 (reproduciendo)	14,66

Tabla 5.3: Dependencia de los resultados con respecto a ver o no, por parte del imitador, la dinámica del patrón.

A raíz de estos resultados, se puede ver claramente que el conocer la dinámica del patrón es más difícil diferenciar entre un imitador y el creador

del patrón. También gracias, a estas pruebas, se puede ver que el hecho de usar mas sesiones, hace que sea más fácil diferenciar a un imitador del creador del patrón.

5.4. Resultados por usuario

Por último, en la tabla 5.4, se pueden ver los resultados de tasas de equierror, para cada usuario del corpus en cada uno de los experimentos mostrados en el apartado anterior.

Para ver mejor visualmente, el número de usuarios con tasas de equierror similares, se han realizado los gráficos 5.1 y 5.2.

Num Firma	**Experimento1**	**Experimento2**	**Experimento3**	**Experimento4**
1	0	0	0	0
2	0	0	0	0
3	50	0	46,15	0
4	6,67	6,67	0	0
5	0	20	0	23,08
6	0	0	0	0
7	16,67	16,67	23,08	16,67
8	0	50	0	50
9	33,33	16,67	33,33	16,67
10	0	6,67	0	15,38
11	60	60	0	0
12	0	6,67	0	0
13	0	13,33	0	0
14	0	13,33	0	0
15	6,67	16,67	0	23,08
16	0	16,67	0	16,67
17	0	33,33	0	33,33
18	0	6,67	0	0
19	0	0	0	0
20	0	0	0	0
21	0	0	0	0
22	6,67	80	0	50
23	0	16,67	0	0
24	0	13,33	0	0
25	33,33	46,67	33,33	38,46
26	0	0	0	0
27	6,67	40	15,38	46,15
28	33,33	50	50	50
29	66,67	33,33	61,54	46,15

Tabla 5.4: Resultados por donante del corpus

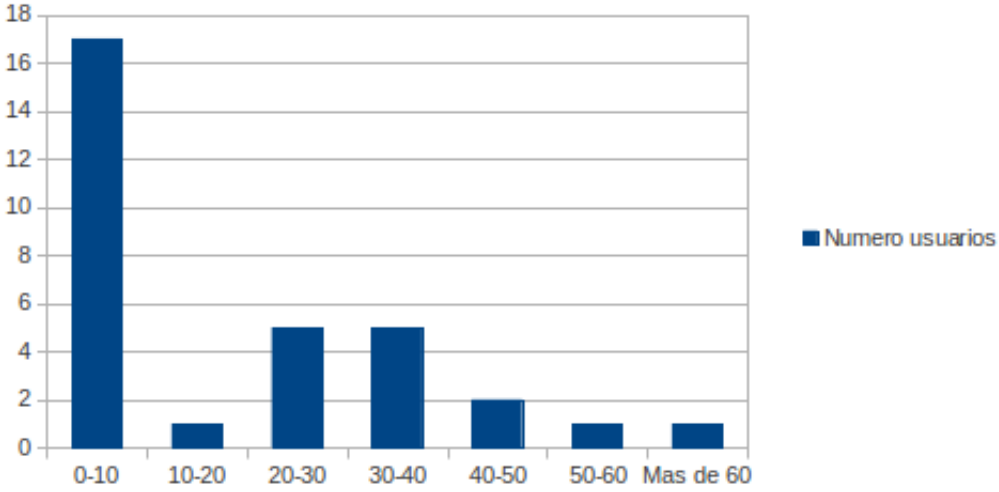


Figura 5.1: Numeros de usuarios con tasas de equierror iguales en el experimento 1

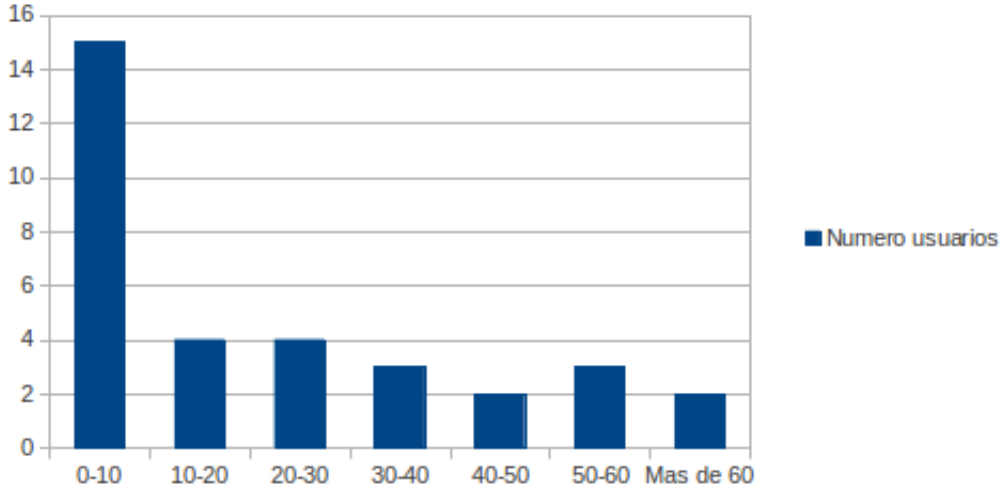


Figura 5.2: Numeros de usuarios con tasas de equierror iguales en el experimento 2

Con estas pruebas, se obtienen dos conclusiones:

- La forma que mejor rendimiento nos da en términos de equierror es la de recogida de 5 muestras en 2 sesiones.

- Como se ve en los gráficos, existen dos tipos de usuarios para el reconocimiento de patrones. Un tipo de usuario, es el que realiza sus patrones a prueba de cualquier tipo de imitadores. El otro tipo de usuario, es aquel cuyos patrones o son fácilmente repetibles o tiene una gran variabilidad en su repetición (variabilidad intrausuario). En algunos casos esta variabilidad se debió a que el usuario se olvidó de cómo era su patrón de una sesión de donación a otra.

5.5. Predicción de patrones con altas tasas de error

Es un tema de investigación totalmente y sin solución actualmente. Si ser una solución al problema, aquí vamos a implementar una solución sencilla que nos permite detectar patrones con alta variabilidad. Está basado en los resultados por usuario de la base de datos, mostrados en la tabla 5.5.

Como se pude ver en esa tabla de los 11 usuarios con errores de tipo de imitación superiores al 20 %, 7 de ellos (un 63 %) tienen también una tasa de error para ataques aleatorios mayores de 0. Si usáramos ese criterio para rechazar patrones, pasaríamos de un error en imitación del 14.1 % al 10 %, es decir, mejoraría el rendimiento del reconocedor en casi un 30 %.

Vemos como el error en impostores aleatorios permiten, de alguna manera, predecir la mayoría de los errores altos en imitación, mejorando el sistema. En el uso normal no contamos con patrones de imitación, pero sí con patrones para ataques aleatorios: los del corpus capturado. Vamos a usar estas muestras para calcular, de manera aproximada, los errores de la columna EER-Rnd de la tabla 5.5.

Se ha implementado esa solución en el sistema, de manera que tras cada sesión de inscripción se realiza el control de seguridad, avisando al usuario si se ha pasado o no, para en este caso, que pueda repetir el patrón.

Num Firma	EER_Ski	EER_Rnd
1	0	0
2	0	0
3	33,33	3,23
4	0	0
5	23,08	0
6	0	0
7	23,08	0
8	40	10
9	25	0
10	44,44	16,13
11	53,85	7,69
12	25	0
13	8,33	0
14	0	0
15	0	0
16	0	0
17	0	0
18	16,67	0
19	8,33	0
20	25	3,23
21	0	0
22	0	0
23	0	0
24	0	0
25	38,46	7,69
26	0	0
27	0	0
28	38,46	0
29	0	0
30	33,33	0
31	50	0
32	61,54	23,08

Tabla 5.5: Tasas de-equierror por firmante

Capítulo 6

Conclusiones y trabajo futuro

En primer lugar, en cuanto a funcionalidad cabe destacar que, tras el análisis presentado anteriormente, el proyecto se ha realizado de manera exitosa, ya que no solo cumple con la funcionalidad exigida sino que, además, se han cumplido cada uno de los objetivos específicos planteados.

En el aspecto del funcionamiento a tiempo real, se ha conseguido de manera satisfactoria gracias a la implantación de una vista Android personalizada que proporciona todos los datos necesarios para el reconocimiento, lo cual era otro de los objetivos que se pretendían alcanzar. Por otra parte, otro marcado objetivo fue que la aplicación sea fácil de modificar y/o ampliar. En este aspecto, el sistema creado es fácilmente modificable gracias a un fichero de constantes que nos permiten cambiar el formato de los ficheros, el numero de muestras por sesión... etc, simplemente cambiando las constantes de ese fichero.

Aunque no era uno de los objetivos del proyecto, también se ha conseguido un buen rendimiento tanto en el plano temporal como en el del reconocimiento. Una prueba de ello, es que las tasas de equierror para el caso de imitadores de un patrón sabiendo su dinámica, son favorables. Si el imitador se hubiese tenido que enfrentar a un sistema de seguridad de patrones dirigido por puntos, cada imitador habría pasado el reconocimiento con probabilidad del 100 %. Es cierto que en algunos casos, el creador del patrón libre es rechazado, pero por ello se proponen distintos niveles de seguridad para el acceso.

Como en todo proyecto siempre hay algo que mejorar. En el caso de este proyecto se proponen las siguientes líneas de trabajo futuro:

- Una opción de trabajo futuro, es la mejora del reconocimiento. Actualmente, el sistema de reconocimiento para dar un resultado lo más perfecto posible, está rechazando un 25 % de las veces al creador del patrón. Esto es un dato bueno, pero mejorable.

- Una buena idea sería integrar todo este sistema como un sistema de acceso al móvil al igual que el de los móviles actuales.

Capítulo 7

Glosario

Se definen a continuación algunos de los términos y abreviaturas que se emplearán a lo largo del documento.

- **Punto:** Cada uno de los puntos de coordenadas detectados por la pantalla del smartphone. En estos puntos también se incluyen los parámetros T (tiempo transcurrido entre puntos de una muestra) y P (si el punto es el primero del trazo)
- **Muestra:** Conjunto de puntos, que determina un dibujo en la pantalla.
- **Modelo:** Conjunto de tres muestras.
- **Nivel de acceso:** Cada uno de los niveles de seguridad definidos. Estos niveles están definidos por tres umbrales de decisión diferentes:
  - Nivel bajo: Pensado para rechazar patrones libres aleatorios.
  - Nivel medio: Pensado para evitar la entrada de imitadores. Un imitador muy bueno, podría superar este nivel de acceso.
  - Nivel alto: Pensado para un rechazo del 25 % de las pruebas propias. EN cambio ofrece, mucha robustez ante imitadores.
- **Inscripción:** Acción realizada en el sistema mediante la cual se define un patrón. Este proceso se compone de hasta dos sesiones (de 3 y 2 muestras respectivamente).
- **DTW:** Algoritmo que calcula la diferencia entre dos secuencias de datos de diferente tamaño. Este algoritmo tiene un orden de complejidad de O(n^2).
- **FastDTW:** Librería escrita en Java que implementa el algoritmo DTW con un orden de complejidad de O(n).

- **Tasa de Equierror** *EER, Equal Error Rate*: Error obtenido cuando los falsos positivos de la prueba de una muestra de un patrón libre, se igualan a los falsos negativos.
- **MVC:** Modelo-Vista-Controlador.
- **OpenUP**: Proceso de desarrollo de software publicado bajo licencia libre. Su ciclo de vida es el siguiente:

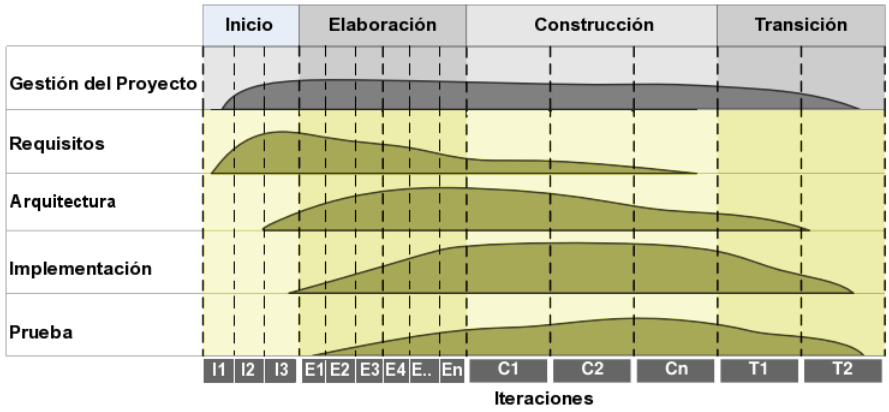


Figura 7.1: Ciclo de vida de un proyecto OpenUP

- **DDMS** (Dalvik Debug Monitor Server) Monitor de depuración de la máquina virtual Dalvik que es la máquina virtual de proceso de Android.
- **Actividad (Android):** Componente de una aplicación que proporciona una pantalla con la cual, el usuario puede interactuar para realizar alguna acción. Este componente se compone de una parte lógica (archivo .java) y una parte gráfica (fichero .xml).

Bibliografía

[1] Libro online de Latex `http://es.wikibooks.org/wiki/Manual_de_LaTeX`
Ultima visita: 04-07-2014

[2] Página de SigmaTechnologies `http://www.sigmatechnologies.es/productos/aplicaciones-moviles/`
Ultima visita: 03-04-2014

[3] *A.K. Jain, F.D. Griess, S.D. Connell* On-line signature verification, Pattern Recognition

Vol. 35 (12), pp. 2963-2972, 2002

[4] *J. Fierrez-Aguilar, L. Nanni, J. Lopez-Peñalba, J. Ortega-Garcia, D. Maltoni* An on-line signature verification system based on fusion of local and global information, Audio- and Video-Based Biometric Person Authentication, Lecture Notes in Computer Science

Vol. 3546/2005, Springer, Berlin, Heidelberg, pp. 523-532, 2005.

[5] *J. M. Pascual-Gaspar, V. Cardeñoso-Payo and C. E. Vivaracho-Pascual* Practical On-Line Signature Verification

LNCS/LNAI, Advances in Biometrics, 3th International Conference, ICB 2009, ISSN 0302-9743, n. 5558, pp. 1180-1189, 2009

[6] *M. Faundez-Zanuy* On-line signature recognition based on vq-dtw, Pattern Recognition

Vol. 40 (3), pp. 981-992, 2007.

[7] *N. Houmani, A. Mayoue, S. Garcia-Salicetti, B. Dorizzi, M.I. Khalil, M.N. Moustafa, H. Abbas, D. Muramatsu, B. Yanikoglu, A. Kholmatov, M. Martinez-Diaz, J. Fierrez, J. Ortega-Garcia, J. Roure Alcobé, J. Fabregas, M. Faundez-Zanuy, J.M. Pascual-Gaspar, V. Cardeñoso-Payo, C. Vivaracho-Pascual* BioSecure signature evaluation campaign (BSEC'2009): Evaluating online signature algorithms depending on the quality of signatures. Pattern Recognition

Volume 45 (3), pp. 993-1003, March 2012

[8] *Arancha Simon-Hurtado, Esperanza Manso-Martinez, Carlos Vivaracho-Pascual, Juan M. Pascual-Gaspar* A New Approach for a Priori Client Threshold Estimation in Biometric Signature Recognition Based on Multiple Linear Regression

LNCS, Part III, ICONIP 2012, ISSN 0302-9743, n. 7665, pp. 174-182, 2012

[9] Toward accurate dynamic time warping in linear time and space. Intelligent Data Analysi *Stan Salvador and Philip Chan* 2007. s, Vol. 11 (5), pp. 561-580, October 2007.

[10] Código de la implementación de fastDTW `http://code.google.com/p/fastdtw/`
Ultima visita: 01-05-2014

[11] VisualParadigm `http://www.visual-paradigm.com/`
Ultima visita: 24-05-2014

[12] Mobile Application Architecture Guide `http://robtiffany.com/wp-content/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf`
Ultima visita: 01-05-2014

[13] Documentación para desarrolladores de aplicaciones Android `https://developer.android.com/intl/es/guide/index.html`
Ultima visita: 01-07-2014

Apéndice A

Implementación de FastDTW

En este apéndice se detalla, todos los aspectos de implementación del algoritmo de distancias entre muestras que se ha utilizado en el proyecto, FastDTW

FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space

Stan Salvador and Philip Chan
Dept. of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{ssalvado, pkc}@cs.fit.edu

ABSTRACT
The dynamic time warping (DTW) algorithm is able to find the optimal alignment between two time series. It is often used to determine time series similarity, classification, and to find corresponding regions between two time series. DTW has a quadratic time and space complexity that limits its use to only small time series data sets. In this paper we introduce FastDTW, an approximation of DTW that has a linear time and space complexity. FastDTW uses a multilevel approach that recursively projects a solution from a coarse resolution and refines the projected solution. We prove the linear time and space complexity of FastDTW both theoretically and empirically. We also analyze the accuracy of FastDTW compared to two other existing approximate DTW algorithms: Sakoe-Chuba Bands and Data Abstraction. Our results show a large improvement in accuracy over the existing methods.

Keywords
dynamic time warping, time series

1. INTRODUCTION
Motivation. Dynamic time warping (DTW) is a technique that finds the optimal alignment between two time series if one time series may be "warped" non-linearly by stretching or shrinking it along its time axis. This warping between two time series can then be used to find corresponding regions between the two time series or to determine the similarity between the two time series. Dynamic time warping is often used in speech recognition to determine if two waveforms represent the same spoken phrase. In a speech waveform, the duration of each spoken sound and the interval between sounds are permitted to vary, but the overall speech waveforms must be similar. In addition to speech recognition, dynamic time warping has also been found useful in many other disciplines [8], including data mining, gesture recognition, robotics, manufacturing, and medicine. Dynamic time warping is commonly used in data mining as a distance measure between time series. An example of how one time series is "warped" to another is shown in Figure 1.

In Figure 1, each vertical line connects a point in one time series to its correspondingly similar point in the other time series. The lines actually have similar values on the *y*-axis but have been separated so the vertical lines between them can be viewed more easily. If both of the time series in Figure 1 were identical, all of the lines would be straight vertical lines because no warping would be necessary to 'line up' the two time series. The warp path distance is a measure of the difference between the two time series after they have been warped together, which is measured by the sum of the distances between each pair of points connected by the vertical lines in Figure 1. Thus, two time series that are identical except for localized stretching of the time axis will have DTW distances of zero.

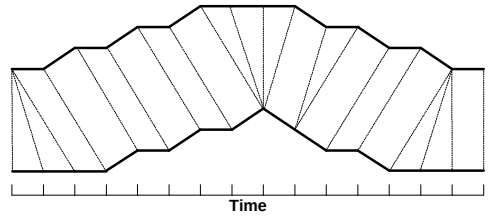


Figure 1. A warping between two time series.

Despite the effectiveness of the dynamic time warping algorithm, it has an $O(N^2)$ time and space complexity that limits its usefulness to small time series containing no more than a few thousand data points. More details of the dynamic time warping algorithm are contained in Section 2.1.

Problem. We desire to develop a dynamic time warping algorithm that is linear in both time and space complexity and can find a warp path between two time series that is nearly optimal.

Approach. In this paper we introduce the FastDTW algorithm, which is able to find an accurate approximation of the optimal warp path between two time series. The FastDTW algorithm avoids the brute-force dynamic programming approach of the standard DTW algorithm by using a multilevel approach. The time series are initially sampled down to a very low resolution. A warp path is found for the lowest resolution and "projected" onto an incrementally higher resolution time series. The projected warp path is refined and projected again to yet a higher resolution. The process of refining and projecting is continued until a warp path is found for the full resolution time series.

Contributions. Our main contribution is the introduction of the FastDTW algorithm, which is an accurate approximation of DTW that runs in linear time and space. We prove the $O(N)$ time and space complexity both theoretically and empirically. We also empirically demonstrate that FastDTW produces an accurate minimum-distance warp path between two time series than is nearly optimal (standard DTW is optimal, but has a quadratic time and space complexity). In addition to the FastDTW algorithm, we evaluate other existing approximate DTW algorithms, and compare their accuracy on a large and diverse group of time series data sets.

Organization. The next section describes the standard dynamic time warping algorithm and existing approaches to speed it up. Section 3 provides a detailed explanation of our FastDTW algorithm. Section 4 discusses experimental evaluations of the FastDTW algorithm based on accuracy, and time/space complexity, and Section 5 summarizes our study.

2. RELATED WORK

2.1 Dynamic Time Warping (DTW)

A distance measurement between time series is needed to determine similarity between time series and for time series classification. Euclidean distance is an efficient distance measurement that can be used. The Euclidian distance between two time series is simply the sum of the squared distances from each *n*th point in one time series to the *n*th point in the other. The main disadvantage of using Euclidean distance for time series data is that its results are very unintuitive. If two time series are identical, but one is shifted slightly along the time axis, then Euclidean distance may consider them to be very different from each other. Dynamic time warping (DTW) was introduced [11] to overcome this limitation and give intuitive distance measurements between time series by ignoring both global and local shifts in the time dimension.

Problem Formulation. The dynamic time warping problem is stated as follows: Given two time series X, and Y, of lengths $|X|$ and $|Y|$,

$$X = x_1, x_2, \ldots, x_i, \ldots, x_{|X|}$$

$$Y = y_1, y_2, \ldots, y_j, \ldots, y_{|Y|}$$

construct a warp path W

$$W = w_1, w_2, \ldots, w_K \qquad \max(|X|, |Y|) \le K < |X| + |Y|$$

where K is the length of the warp path and the k^{th} element of the warp path is

$$w_k = (i, j)$$

where i is an index from time series X, and j is an index from time series Y. The warp path must start at the beginning of each time series at $w_1 = (1, 1)$ and finish at the end of both time series at $w_K = (|X|, |Y|)$. This ensures that every index of both time series is used in the warp path. There is also a constraint on the warp path that forces i and j to be monotonically increasing in the warp path, which is why the lines representing the warp path in Figure 1 do not overlap. Every index of each time series must be used. Stated more formally:

$$w_k = (i, j), w_{k+1} = (i', j') \qquad i \le i' \le i+1, \quad j \le j' \le j+1$$

The optimal warp path is the warp path is the minimum-distance warp path, where the distance of a warp path W is

$$Dist(W) = \sum_{k=1}^{k=K} Dist(w_{ki}, w_{kj})$$

Dist(W) is the distance (typically Euclidean distance) of warp path W, and *Dist*(w_{ki}, w_{kj}) is the distance between the two data point indexes (one from X and one from Y) in the k^{th} element of the warp path.

DTW Algorithm. A dynamic programming approach is used to find this minimum-distance warp path. Instead of attempting to solve the entire problem all at once, solutions to sub-problems (portions of the time series) are found, and used to repeatedly find solutions to a slightly larger problem until the solution is found for the entire time series. A two-dimensional $|X|$ by $|Y|$ cost matrix D, is constructed where the value at $D(i, j)$ is the minimum-distance warp path that can be constructed from the two time series $X'=x_1,...,x_i$ and $Y'=y_1,...,y_j$. The value at $D(|X|, |Y|)$ will contain the minimum-distance warp path between time series X and Y. Both axes of D represent time. The x-axis is the time of time series X, and the y-axis is the time of time series Y. Figure 2 D shows an example of a cost matrix and a minimum-distance warp path traced through it from $D(1, 1)$ to $D(|X|, |Y|)$.

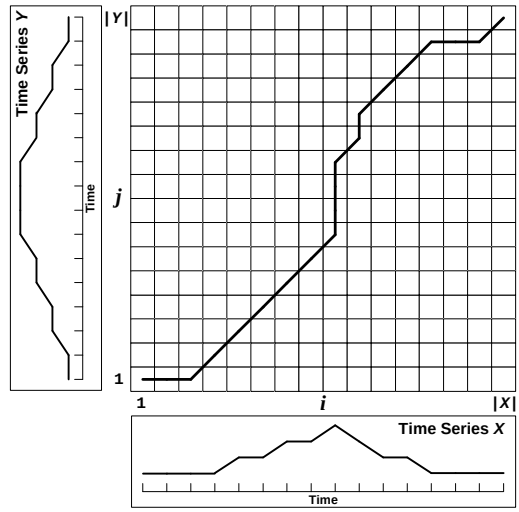


Figure 2. A cost matrix with the minimum-distance warp path traced through it.

The cost matrix and warp path in Figure 2 are for the same two time series shown in Figure 1. The warp path is W = {(1,1), (2,1), (3,1), (4,2), (5,3), (6,4), (7,5), (8,6), (9,7), (9,8), (9,9), (9,10), (10,11), (10,12), (11,13), (12,14), (13,15), (14,15), (15,15), (16,16)}. If the warp path passes through a cell $D(i, j)$ in the cost matrix, it means that the i^{th} point in time series X is warped to the j^{th} point in time series Y. Notice that where there are vertical sections of the warp path, a single point in time series X is warped to multiple points in time series Y, and the opposite is also true where the warp path is a horizontal line. Since a single point may map to multiple points in the other time series, the time series do not need to be of equal length. If X and Y were identical time series, the warp path through the matrix would be a straight diagonal line.

To find the minimum-distance warp path, every cell of the cost matrix must be filled. The rationale behind using a dynamic programming approach to this problem is that since the value at $D(i, j)$ is the minimum warp distance of two time series of lengths i and j, if the minimum warp distances are already known for all

slightly smaller portions of that time series that are a single data point away from lengths *i* and *j*, then the value at $D(i, j)$ is the minimum distance of all possible warp paths for time series that are one data point smaller than *i* and *j*, plus the distance between the two points x_i and y_j. Since the warp past must either be incremented by one or stay the same along the *i* and *j* axes, the distances of the optimal warp paths one data point smaller than lengths *i* and *j* are contained in the matrix at $D(i\text{-}1,\ j)$, $D(i,\ j\text{-}1)$, and $D(i\text{-}1,\ j\text{-}1)$. So the value of a cell in the cost matrix is:

$$D(i, j) = Dist(i, j) + \min[D(i-1, j), D(i, j-1), D(i-1, j-1)]$$

The warp path to $D(i, j)$ must pass through one of those three grid cells, and since the minimum possible warp path distance is already known for them, all that is needed is to simply add the distance of the current two points to the smallest one. Since this equation determines the value of a cell in the cost matrix by using the values in other cells, the order that they are evaluated in is very important. The cost matrix is filled one column at a time from the bottom up, from left to right as depicted in Figure 3.

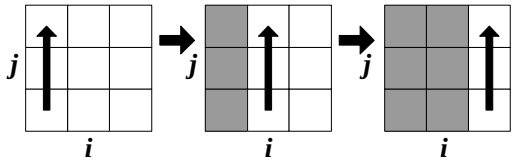


Figure 3. The order that the cost matrix is filled.

After the entire matrix is filled, a warp path must be found from $D(1, 1)$ to $D(|X|, |Y|)$. The warp path is actually calculated in reverse order starting at $D(|X|, |Y|)$. A greedy search is performed that evaluates cells to the left, down, and diagonally to the bottom-left. Whichever of these three adjacent cells has the smallest value is added to the beginning of the warp path found so far, and the search continues from that cell. The search stops when $D(1, 1)$ is reached.

Complexity of DTW. Time and Space complexity of the DTW is easy to determine. Each cell in the $|X|$ by $|Y|$ cost matrix is filled exactly once, and each cell is filled in constant time. This yields both a time and space complexity of $|X|$ by $|Y|$, which is $O(N^2)$ if $N=|X|=|Y|$. The quadratic space complexity is particularly prohibitive because memory requirements are in the *terabyte* range for time series containing only 177,000 measurements. A linear space-complexity implementation of the DTW algorithm is possible by only keeping the current and previous columns in memory as the cost matrix is filled from left to right (see Figure 3). By only retaining two columns at any one time, the optimal warp distance between the two time series can be determined. However it is not possible to reconstruct the warp path between these two time series because the information required to calculate the warp path is thrown away with the discarded columns. This is not a problem if only the distance between two time series is required, but applications that find corresponding regions between time series [14] or merge time series together [1][3] require the warp path to be found.

2.2 Speeding up Dynamic Time Warping

The quadratic time and space complexity of DTW creates the need for methods to speed up dynamic time warping. The methods used make DTW faster fall into three categories:

1) *Constraints* – Limit the number of cells that are evaluated in the cost matrix.
2) *Data Abstraction* – Perform DTW on a reduced representation of the data.
3) *Indexing* – Use lower bounding functions to reduce the number of times DTW must be run during time series classification or clustering.

Constraints are widely used to speed up DTW. Two of the most commonly used constraints are the Sakoe-Chuba Band [13] and the Itakura Parallelogram [4], which are shown in Figure 4.

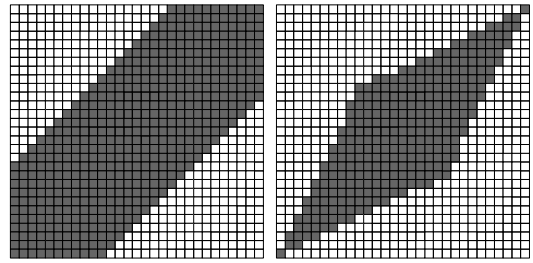


Figure 4. Two constraints: Sakoe-Chuba Band (left) and an Itakura Parallelogram (right), both have a width of 5.

The shaded areas in Figure 4 are the cells of the cost matrix that are filled in by the DTW algorithm for each constraint. The width of each shaded area, or window, is specified by a parameter. When constraints are used, the DTW algorithm finds the optimal warp path through the constraint window. However, the globally optimal warp path will not be found if it is not entirely inside the window. Using constraints speeds up DTW by a constant factor, but the DTW algorithm is still $O(N^2)$ if the size of the input window is a function of the length of the input time series. Constraints work well in domains where the optimal warp path is expected to be close to a linear warp and passes through the cost matrix diagonally in a relatively straight line. Constraints work poorly if time series are of events that start and stop at radically different times because the warp path can stray very far from a linear warp and nearly the entire cost matrix must be evaluated to find the optimal warp path.

Data abstraction speeds up the DTW algorithm by running DTW on a reduced representation of the data [2][9]. The left side of Figure 5 shows a full-resolution cost matrix for which a minimum-distance warp path must be found. Rather than running the DTW algorithm on the full resolution (1/1) cost matrix, the time series are reduced in size to make the number of cells in the cost matrix more manageable. A warp path is found for the lower-resolution time series and is mapped back to the full resolution cost matrix.

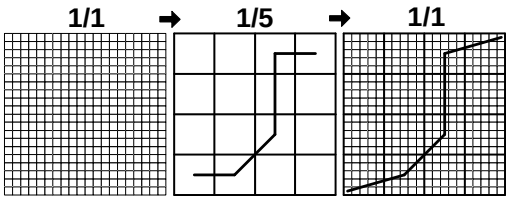


Figure 5. Speeding up DTW by data abstraction.

The result is that DTW is sped up by a large constant factor, but the algorithm still runs in $O(N^2)$ time and space. Obviously, the warp distance that is calculated between the two time series becomes increasingly inaccurate as the level of abstraction increases. Projecting the lower resolution warp path to the full resolution usually creates a warp path that is far from optimal because even *IF* the optimal warp path actually passes through the low-resolution cell, projecting the warp path to the higher resolution ignores local variations in the warp path that can be very significant.

Indexing uses lower-bounding functions to prune out the number of times DTW needs to be run for certain tasks such as clustering a set of time series or finding the time series that is most similar to a given time series [6][10]. Indexing significantly speeds up many DTW applications by reducing the number of times DTW is run, but does not speed up the actual DTW algorithm.

Our FastDTW algorithm uses ideas from both the constraints and data abstraction categories. Using a combination of both overcomes many limitations of using either method individually, and yields an algorithm that is $O(N)$ in both time and space.

3. APPROACH

The multilevel approach that FastDTW uses is inspired by the multilevel approach used for graph bisection [5]. Graph bisection is the task of splitting a graph into roughly equal portions, such that the sum of the edges that would be broken is as small as possible. Efficient and accurate algorithms exist for small graphs, but for large graphs, the solutions found are typically far from optimal. A multilevel approach can be used to find the optimal solution for a small graph, and then repeatedly expand the graph and "fix" the pre-existing solution for the slightly larger problem. A multilevel approach works well if a large problem is difficult to solve all at once, but partial solutions can effectively be refined at different levels of resolution. The dynamic time warping problem can also be solved with a multilevel approach. Our FastDTW algorithm uses the multilevel approach and is able to find an accurate warp path in linear time and space.

3.1 FastDTW Algorithm

The FastDTW algorithm uses a multilevel approach with three key operations:

1) ***Coarsening*** – Shrink a time series into a smaller time series that represents the same curve as accurately as possible with fewer data points.

2) ***Projection*** – Find a minimum-distance warp path at a lower resolution, and use that warp path as an initial guess for a higher resolution's minimum-distance warp path.

3) ***Refinement*** – Refine the warp path projected from a lower resolution through local adjustments of the warp path.

Coarsening reduces the size (or resolution) of a time series by averaging adjacent pairs of points. The resulting time series is a factor of two smaller than the original time series. Coarsening is run several times to produce many different resolutions of the time series. *Projection* takes a warp path calculated at a lower resolution and determines what cells in the next higher resolution time series the warp path passes through. Since the resolution is increasing by a factor of two, a single point in the low-resolution warp path will map to at least four points at the higher resolution (possibly >4 if $|X| \neq |Y|$). This projected path is then used as a heuristic during solution refinement to find a warp path at the higher resolution. *Refinement* finds the optimal warp path *in the neighborhood* of the projected path, where the size of the neighborhood is controlled by the *radius* parameter.

Standard dynamic time warping (DTW) is an $O(N^2)$ algorithm because every cell in the cost matrix must be filled to ensure an optimal answer is found, and the size of the matrix grows quadratically with the size of the time series. In the multilevel approach, the cost matrix is only filled in the neighborhood of the path projected from the previous resolution. Since the length of the warp path grows *linearly* with the size of the input time series, the multilevel approach is an $O(N)$ algorithm.

The FastDTW algorithm first uses coarsening to create all of the resolutions that will be evaluated. Figure 6 shows four resolutions that are created when running the FastDTW algorithm on the time series that were previously used in Figures 1 and 2. The standard DTW algorithm is run to find the optimal warp path for the lowest resolution time series. This lowest resolution warp path is shown in the left of Figure 6. After the warp path is found for the lowest resolution, it is projected to the next higher resolution. In Figure 6, the projection of the warp path from a resolution of 1/8 is shown as the heavily shaded cells at 1/4 resolution.

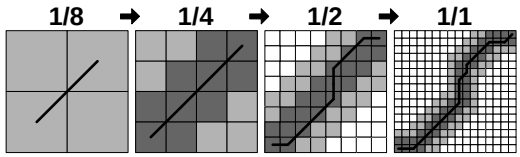


Figure 6. The four different resolutions evaluated during a complete run of the FastDTW algorithm.

To refine the projected path, a constrained DTW algorithm is run with the very specific constraint that only cells in the projected warp path are evaluated. This will find the optimal warp path *through the area of the warp path that was projected from the lower resolution*. However, the entire optimal warp path may not be contained within projected path. To increase the chances of finding the optimal solution, there is a *radius* parameter that controls the *additional* number of cells on each side of the projected path that will also be evaluated when refining the warp path. In Figure 6, the *radius* parameter is set to 1. The cells included during warp path refinement due to the *radius* are lightly

shaded. Once the warp path is refined at the 1/4 resolution, that warp path is projected to the 1/2 resolution, expanded by a *radius* of 1, and refined again. Finally, the warp path is projected to the full resolution (1/1) matrix in Figure 6. The projection is expanded by the *radius* and refined one last time. This refined warp path is the output of the algorithm.

Notice that the warp path found by the FastDTW algorithm in Figure 6 is the optimal warp path that was found by the standard DTW in Figure 2. However, FastDTW only evaluated the shaded cells, while DTW evaluates all of the cells in the cost matrix. FastDTW evaluated 4+16+44+100=164 cells at all resolutions, while DTW evaluates all 235 (16^2) cells. This increase in efficiency is not very significant for his small problem, especially considering the overhead of creating all four resolutions. However, the number of cells that FastDTW evaluates scales linearly with the length of the time series, while DTW always evaluates N^2 cells (if both time series are of length N). FastDTW scales linearly because the width of the path through the matrix that is being evaluated is constant at all resolutions.

The example in Figure 6 finds the optimal warp path, but the FastDTW algorithm is not guaranteed to always find a warp path that is optimal. However, the path found is usually very close to optimal. The larger the value of the *radius* parameter, the more accurate the warp path will be. If the *radius* parameter is set to be as large as one of the input time series, then FastDTW generalizes to the DTW algorithm (optimal but $O(N^2)$). The accuracy of FastDTW using different settings for the *radius* parameter will be demonstrated in Section 4.

The pseudocode for the FastDTW algorithm is shown Figure 7. The input to the algorithm is two time series, and the *radius* parameter. The output of FastDTW is a warp path and the distance between the two time series along that warp path. Line 2 determines the minimum length of a time series at the lowest resolution. This size is dependent on the *radius* parameter and determines the smallest possible resolution size for which decreasing the resolution further would be pointless because full dynamic time warping would need to be calculated at more than one resolution.

FastDTW has a straightforward recursive implementation. The base case is when one of the input time series has a length less than *minTSsize*. For the base case, the algorithm simply returns the result of the standard DTW algorithm. The recursive case has three main steps. First, two new lower-resolution time series are created that have half as many points as the input time series (***coarsening***). This is performed by lines 17-18 in Figure 7. Next, a low resolution path is found for the coarsened time series (lines 20-21) and ***projected*** to a higher resolution (lines 23-25). This projected path is also expanded by *radius* cells to create a search window that will be passed to a constrained version of the DTW algorithm that only evaluates the cells in the search window (line 27). The constrained DTW algorithm ***refines*** the warp path that was projected form the lower resolution. The result of this refinement is then returned.

Function FastDTW()

Input: *X* – a TimeSeries of length |*X*|
Y – a TimeSeries of length |*Y*|
radius – distance to search outside of the projected warp path from the previous resolution when refining the warp path

Output: 1) A min. distance warp path between *X* and *Y*
2) The warped path distance between *X* and *Y*

```
 1| // The min size of the coarsest resolution.
 2| Integer minTSsize = radius+2
 3|
 4| IF (|X|≤minTSsize OR |Y|≤minTSsize)
 5| {
 6|   // Base Case: for a very small time series run
 7|   //   the full DTW algorithm.
 8|   RETURN DTW(X, Y)
 9| }
10| ELSE
11| {
12|   // Recursive Case: Project the warp path from
13|   //   a coarser resolution onto the current
14|   //   current resolution. Run DTW only along
15|   //   the projected path (and also 'radius' cells
16|   //   from the projected path).
17|   TimeSeries shrunkX = X.reduceByHalf()
18|   TimeSeries shrunkY = Y.reduceByHalf()
19|
20|   WarpPath lowResPath =
21|       FastDTW(shrunkX,shrunkY, radius)
22|
23|   SearchWindow window =
24|       ExpandedResWindow(lowResPath, X, Y,
25|                          radius)
26|
27|   RETURN DTW(X, Y, window)
28| }
```

Figure 7. The FastDTW algorithm.

The execution of the FastDTW algorithm repeatedly runs lines 17-18 in recursive calls to lower resolutions are made by line 21. This creates multiple resolutions until the base case is reached (line 8). The base case is executed only a single time, and afterwards lines 23-27 are executed for each recursive call (or resolution) on the stack.

Next, we will provide a theoretical analysis of FastDTW based on time and space complexity.

Time Complexity of FastDTW. To simplify the calculations we will assume that the two full-resolution time series *X* and *Y* are both of length *N*. All analysis will be performed on worst-case behavior.

The number of cells in the cost matrix that are filled by FastDTW in a single resolution is equal the number of cells in the projected warp path and any other cells within *radius* (denoted as *r* in the rest of this analysis to save space) cells away from the projected path. The worst case, a straight diagonal projected warp path is depicted in Figure 8.

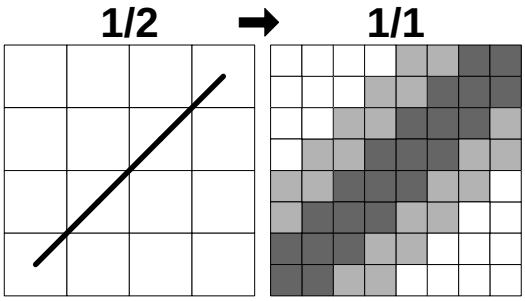


Figure 8. Maximum (worst-case) number of cells evaluated for a *radius* of 1.

The lightly shaded cells in Figure 8 are the 2*Nr* cells on each side of the projected path (heavily shaded cells), which itself has 3*N* cells. The projected path therefore has the following maximum number of cells at a resolution with two time series containing *N* points:

$$3N + 2(2Nr) = N(4r + 3) \quad [1]$$

The length of the time series at each resolution (*res*) follows the sequence (*N* points are contained in the original time series):

$$\left\{ \frac{N}{2^{res}} \right\}_{res=0}^{res=\infty} = N, \frac{N}{2}, \frac{N}{2^2}, \frac{N}{2^3}, \frac{N}{2^4}, \cdots \quad [2]$$

Therefore, the number of cells evaluated at all resolutions is (combine Equations 1 and 2)

$$\sum_{res=0}^{\infty} \frac{N}{2^{res}}(4r+3) = N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \cdots \quad [3]$$

The series in Equation 3 is very similar to the series

$$\sum_{res=0}^{\infty} \frac{1}{2^{res}} = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots = 2 \quad [4]$$

Multiplying Equation 4 by Equation 1 yields

$$N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \cdots = 2N(4r+3) \quad [5]$$

Since the sequence in Equation 5 is identical to the sequence in Equation 3, the number of cells evaluated at all resolutions is

$$\textit{Total number of cells filled} = 2N(4r+3) \quad [6]$$

In addition to the number of cells calculated there is also time complexity for creating the coarser resolutions and determining the warp path by tracing through the matrix.

The time complexity needed to create the resolutions is proportional to the number of points in all of the resolutions, which is the series in Equation 2. The solution of Equation 2 is obtained by multiplying Equation 4 by *N*, which yields 2*N*. Since multiple resolutions of both time series must be created, 2*N* is multiplied by two to get the final time complexity.

$$\textit{Time to create all resolutions} = 4N \quad [7]$$

The time complexity needed to trace the warp path back through a matrix is measured by the length of the warp path. A resolution containing *N* points has a length of 2*N* in the worst case (*N* is the best case for a diagonal line). Multiplying Equation 4 by 2*N* gives the worst-case length of all warp paths added together from every resolution:

$$\textit{Time to trace warp paths} = 4N \quad [8]$$

Adding Equations 6, 7, and 8 gives the total worst-cast time complexity of FastDTW

$$\textit{FastDTW time complexity} = N(8r+14) \quad [9]$$

which is O(*N*) if *r* (*radius*) is a small constant value.

Space Complexity of FastDTW. The space complexity of FastDTW consists of the space required to store the resolutions (other than the full-resolution input time series), the maximum amount of cells that are used at any one time in a cost matrix, and the size of the warp path stored In memory. The space complexity of storing all extra resolutions other than the full resolution for one input time series is Equation 2 without the first them, which is 2*N*-*N*=*N*. For both input time series the space complexity is

$$\textit{Space of resolutions (other that full resolution)} = 2N \quad [10]$$

The space complexity of the cost matrix is the maximum size cost matrix that is created for the full resolution matrix. The number of cells in the matrix is Equation 1

$$\textit{Space of cost matrix} = N(4r+3) \quad [11]$$

The space complexity of storing the warp path is equal to the longest warp path that can exist at full resolution. If the warp path traces the perimeter of the cost matrix, then the length of that path will be

$$\textit{Space complexity of storing the warp path} = 2N \quad [12]$$

And adding Equations 10, 11, and 12 gives the total worst-cast space complexity of

$$\textit{FastDTW space complexity} = N(4r+7) \quad [13]$$

which is also O(*N*) if *r* (*radius*) is a small (<*N*) constant value.

4. EMPIRICAL EVALUATION

The goal of this evaluation is demonstrate the efficiency and accuracy of the FastDTW algorithm on a wide range of time series data sets. To ensure reproducibility, all datasets and algorithms used in this evaluation can be found online at "http://cs.fit.edu/~pkc/FastDTW/". This evaluation will first demonstrate the accuracy of the FastDTW algorithm and will then empirically verify its linear time complexity.

4.1 Accuracy of FastDTW

4.1.1 Procedures and Criteria

The accuracy of an approximate DTW algorithm can be measured by determining how much the approximate warp path distance differs from the optimal warp path distance. The error of an

approximate DTW algorithm, such as our FastDTW algorithm, is calculated by the following equation:

$$\textbf{\textit{Error of a warp path}} = \frac{approxDist - optimalDist}{optimalDist} \times 100 \quad [14]$$

If the DTW algorithm finds a warp path with a distance equal to the optimal warp path distance, then there is zero error. The optimal warp path distance can be found by running the standard DTW algorithm. The error of a warp path will always be ≥0% (because *optimalDist* is never larger than *approxDist*) and can exceed 100% if the distance of the approximate warp path is more than double the optimal distance.

The FastDTW algorithm is evaluated against two other existing approximate DTW algorithms: Sakoe-Chuba bands and data abstraction. Sakoe-Chuba bands (see left side of Figure 4) constrain the DTW algorithm to only evaluate a specified *radius* away from a linear warp within the cost matrix. Itakura Parallelograms (see right side of Figure 4) are not evaluated because, for a given *radius*, a band will always find a warp path equal to or better than that of the parallelogram. This is because the parallelogram constraint is a subset of the band constraint. The data abstraction DTW algorithm used in this evaluation first samples the data, and then runs the standard DTW algorithm to find a warp path on the sampled data. This warp path is then projected to the full resolution as previously shown in Figure 5.

The *radius* parameter performs a similar function for all three algorithms. It expands the region of the cost matrix searched from an initial "guess". For bands, the initial guess is a linear warp. For data abstraction, it is the projected warp path from the sampled data, and for FastDTW it is the projected warp path from the previous resolution. Each algorithm will be run with multiple *radius* parameters on a wide range of data sets.

All three algorithms (FastDTW, bands, and data abstraction) are only being evaluated based on accuracy in this section. However, care has been taken to ensure that the time each algorithm requires to execute is similar for the same *radius*. The data abstraction algorithm is made O(*N*) by sampling the data down to $\sqrt{N}$ points before performing quadratic time warping (O($\sqrt{N}^2$) = O(*N*)). All three algorithms evaluate roughly the same number of cells in the cost matrix for any particular *radius*. FastDTW has some overhead for evaluating previous resolutions, and data abstraction has overhead for running standard DTW on the sampled time series. However, all three algorithms are linear with respect to the length of the input time series, and the number of cells evaluated for a given *radius* does not differ by more than a power of two of for any pair of algorithms.

The time series data sets used to evaluate the accuracy of the FastDTW algorithm include very similar data sets that are from the same domain, and dissimilar data sets that are from different domains. Both types of data are used to show that FastDTW works well on a wide range of data, regardless of the similarity or characteristics of the time series. Dynamic time warping is most frequently used to compare the similarity between time series, so it is likely that the majority of time series that are compared are similar and from the same domain. However, very dissimilar time series are also evaluated to ensure that the approximate FastDTW algorithm works well when warping two time series that do not share common features. The accuracy of each DTW algorithm is measured on three groups of data:

1) *Random* – 990 time warps between 45 time series from different domains (eeg, random walk, earthquake, speech, tide, etc.). The average length is 1128 points.

2) *Trace* - 10,900 time warps between 200 time series data sets. The Gun domain contains 4 classes that simulate instrumentation failure in a nuclear power plant. All time series have a length of 275 points.

3) *Gun* – 10,900 time warps between 200 time series data sets. The Gun domain contains 2 classes, with 100 time series of a gun being drawn from a holster and 100 time series of a gun being pointed. All time series have a length of 151 points.

All data sets used in this evaluation were obtained from the UCR Time Series Data Mining Archive and are publicly available [7]. Each algorithm and group of data is also run multiple times with the following settings for the *radius* parameter: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, and 30. For a given algorithm, group of data, and *radius*, the average error of all possible warp paths between time series in the group are recorded.

4.1.2 Results and Analysis

The FastDTW algorithm is very accurate for all three groups of data that it was tested on. FastDTW has an error of only 19.2% to 0.0%, depending on the value of the *radius* parameter. For all algorithms, the error decreases as the *radius* parameter increases. However, FastDTW converges to 0% error much faster than the other two algorithms. A summary of the results for several *radius* settings is contained in Table 1.

Table 1. Average error of three the algorithms at selected *radius* values (errors of the 3 groups of data are averaged).

	radius				
	0	*1*	*10*	*20*	*30*
FastDTW	19.2%	8.6%	1.5%	0.8%	0.6%
Abstraction	983.3%	547.9%	6.5%	2.8%	1.8%
Band	2749.2%	2385.7%	794.1%	136.8%	9.3%

Table 1 shows the average error for all three algorithms over all three test cases, when run with the *radius* set to 0, 1, 10, 20, and 30. FastDTW has a small amount of error for all *radius* settings, and begins to approach 0% error when *radius* is set at or above 10. Data abstraction is inaccurate for small *radius* values, but begins to be reasonably accurate when run with larger *radius* settings. The band algorithm is very inaccurate for all *radius* settings except for 30.

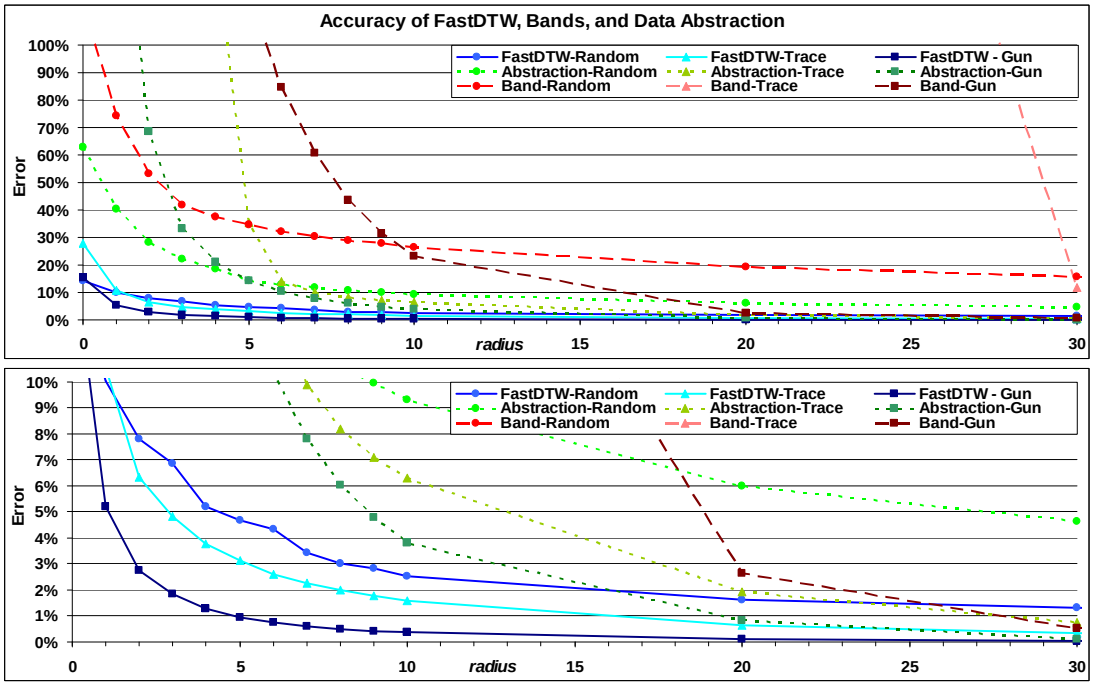


Figure 9. Accuracy of FastDTW compared to Bands and Data Abstraction. The top figure's *y*-axis is 0%-100% and the bottom figure's *y*-axis is 0%-10%.

Data abstraction is inaccurate (500-1000% error) for small *radius* settings because it blindly projects the warp path from a sampled time series onto a full resolution cost matrix. This projection may be "in the neighborhood" of a near-optimal warp path, but it fails to take into consideration any local variation in the warp path that is obscured by sampling. Local variations in the warp path can have a huge impact on the accuracy of a warp path. Increasing the *radius* setting (which is not part of the original data abstraction algorithm, it is introduced in this paper), can make it rather accurate because this begins to adjust the warp path to cover local variations. However, the accuracy is still worse than FastDTW for a given *radius* because FastDTW projects the "neighborhood" of the near-optimal warp path from the previous resolution in several small steps rather than a single large step.

Bands can only have good results if a near-optimal warp path is entirely contained within *radius* cells from a linear warp. When bands are used with a *radius* of 0, and the two time series are of equal length, it generalizes to Euclidean distance...which is a notoriously inaccurate similarity measure for time series [15]. A slight misalignment between the two time series being warped can cause a very large amount of error in the warp path.

The accuracy of each algorithm on the different groups of data is displayed in Figure 9.

In Figure 9, the *x*-axis is the radius parameter used, and the *y*-axis is the error of the tested algorithm. Each of the 9 lines is a combination between the three algorithms and the three groups of data sets. The FastDTW algorithm curves are solid lines, data abstraction curves are dotted lines, and band curves are dashed lines. The three groups of data can be identified by the shape of the markers on the curves. Round markers are used on curves using Random data, triangle markers are for the Trace data, and square markers are for the Gun data.

The three solid lines at the bottom of Figure 9 are the error curves for FastDTW on all three groups of data. The error is small for all three lines, meaning that the accuracy FastDTW is not effected very much by the characteristics or similarity of the input time series. FastDTW is significantly more accurate than the other two methods when the *radius* parameter is set to small values. When the *radius* parameter is larger, the abstraction method begins to approach the accuracy of FastDTW. However, FastDTW was always at least 2-3 times more accurate than abstraction in our experiments.

The three dotted Abstraction lines all have large errors for small *radius* values, but converge to less than 5% error on all data sets as the *radius* is increased to 30. This is due to the previously stated problem of the projected warp path being close to a near-optimal solution, but not taking local variations of the warp path into account. Abstraction does perform reasonably well if the *radius* is increased to at least 10. The ability of data abstraction to locally refine its projected path within the neighborhood of *radius* cells is not a part of the original algorithm, and is introduced in

this paper. The run-time of the original data abstraction algorithm is the same as our improved implementation when using a *radius* of 0, which has a very large average error of 983.3% over the three groups of data used in this evaluation.

The three dashed Band lines all have errors greater than 100% (as high as 7225%) for small *radius* values, and converge very slowly to 0% error as the *radius* increases. Band performs best on the random data because if two time series have almost nothing in common, an arbitrary warp path probably has a warp path distance that is not significantly much different from the minimum-distance or maximum-distance warp paths. The other two groups of data are data sets in a similar domain, which means that the optimal warp distance can be very small. Due to the way that error is calculated in Equation 14, if the optimal warp distance is very small, then the potential error can be very large because the optimal warp distance is the denominator of a fraction. The Band approach on the Trace data group has extremely poor accuracy because the time series contain events that are shifted in time, and bands only work well if a near-optimal warp path exists that is close to a linear warp. The Gun data group also does not work very well with the Band algorithm, which is surprising since the time series seem to be reasonably in phase with each other (near a linear warp).

4.2 Efficiency

4.2.1 Procedures and Criteria

The efficiency of the FastDTW algorithm will be measured in seconds, with respect to the length of the input time series, and compared to the standard DTW algorithm. The FastDTW algorithm will be run with the *radius* parameter set to: 0, 20, and 100 over a range of varying-length time series. The data sets used are synthetic data sets of a single period of a sine wave with Gaussian noise inserted. Only the lengths of the time series are significant because the shape of the time series has little significance on the run-time of either algorithm. The lengths of the time series evaluated vary from 10 to 150,000.

The standard DTW algorithm used in this evaluation is the linear-space implementation that only retains the last two columns of the cost matrix. If the standard DTW implementation is used, the test machine runs out of memory when the length of the time series exceeds ~3,000. The FastDTW algorithm is implemented as described in this paper except that the cost matrix is filled using secondary storage if the lengths of the time series grow so large that the number of cells in the search window is larger than can fit into main memory. Both algorithms are implemented in Java, and the runtime is measured using the system clock on a machine with minimal background processes running.

4.2.2 Results and Analysis

The FastDTW algorithm was significantly faster than the standard DTW algorithm for all but the smallest time series. FastDTW is 50 to 150 times faster than standard DTW (using *radius* values of 0 and 100 respectively) when the time series have lengths of 150,000 points. A sample of the results of the FastDTW algorithm can be seen in Table 2.

Table 2. Execution time (in seconds) of DTW and FastDTW on time series of four different lengths.

	Length of Time Series			
	100	*1,000*	*10,000*	*100,000*
DTW	0.02	0.92	57.45	7969.59
FastDTW (*radius*=0)	0.01	0.02	0.38	67.94
FastDTW (*radius*=100)	0.02	0.06	8.42	207.19

In Table 2, FastDTW and DTW have similar execution times for the 100 point time series. For the larger 10,000 and 100,000 point time series FastDTW runs much more quickly than DTW. But execution time for the 1,000 point time series is both faster and slower than DTW, depending on the *radius* parameter. The exact length at which FastDTW runs quicker than DTW depends on the *radius* parameter. Figure 10 shows the critical region where one algorithm is faster than the other depending on the *radius* parameter.

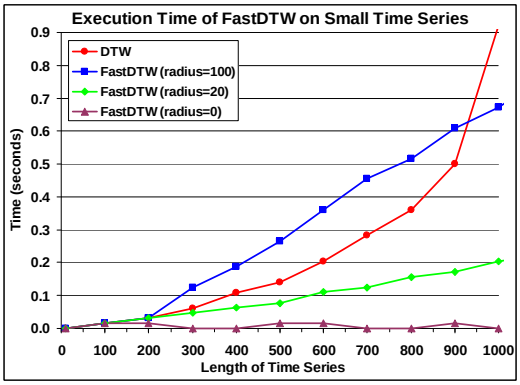


Figure 10. The efficiency of FastDTW and DTW on small time series.

The FastDTW algorithm, with a *radius* of 100, takes longer to run than DTW until the size of the time series exceeds approximately 900 points. However, with a *radius* of 0 or 20, the DTW algorithm is never faster than the FastDTW algorithm for small time series, and once the length of the time series exceed 200-300 points, FastDTW becomes the more efficient algorithm. For small time series it makes more sense to use the DTW algorithm rather than FastDTW. The FastDTW algorithm is not significantly faster (and possibly a little slower) than DTW for small time series, and DTW is guaranteed to always find the optimal warp path. However, for large time series, the quadratic time complexity becomes prohibitive.

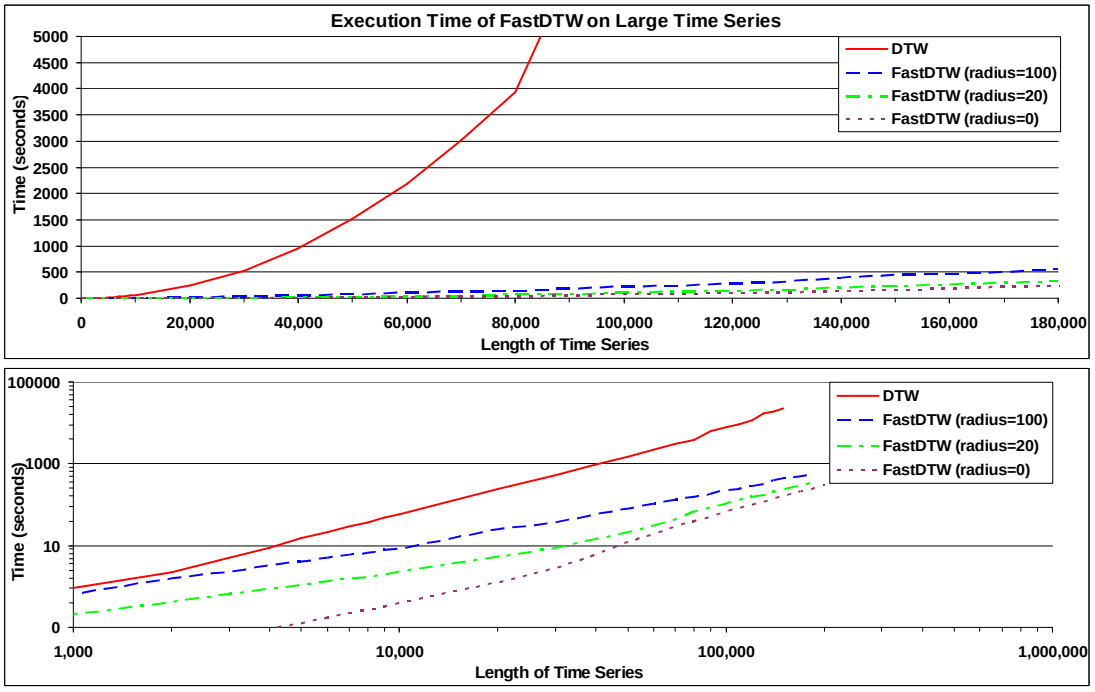


Figure 11. The eficiency of FastDTW and DTW on large time series. The top figure is scaled normally, and the bottom figure has log-log scaling.

The full results, using *radius* values of 0, 20, and 100 on time series ranging in length from 10 to 200,000 are shown in Figure 11. In Figure 11, the *y*-axis is the execution time and the *x*-axis is the length of the time series. The two graphs in Figure 11 are two views of the same data. The top graph is scaled normally, and the bottom graph has log-log scaling. Looking at the top graph, it is immediately obvious that the time complexity of DTW is much greater than that of FastDTW. DTW has an exponential curve, while all three FastDTW curves are approximately straight lines. In the log-scaled graph at the bottom of Figure 11, the three curves of FastDTW can be viewed more easily. The *radius* parameter increases the execution of FastDTW by a constant factor, which is why the three FastDTW lines seem to be converging on the log-scaled graph as length of the time series increases. The constant factor difference between them gets less significant as the length of the time series increases.

In Section 3.1, we proved theoretically that the FastDTW algorithm was O(*N*). Using the empirical data in Figure 11, the equation of the FastDTW curve with a radius of 100 is

$$y = 0.00000001x^2 + 0.001x - 0.7337$$

This coefficient of the squared term is very small, and it seems like the linear term is the most significant term in the equation...which would empirically prove that the FastDTW algorithm is O(*N*). However, since the values for *x* are so large, the squared term actually dominates the equation when *x*>100,000. The reason for this slight sub-linearity in the algorithm occurs when the number of cells being filled in the search window will not fit into main memory, and must be saved to the disk. Writing the cells to the disk can be performed in linear time. However, when reading the cells from the random-access file to construct a warp path, reading individual non-sequential cells from the disk cannot be performed in linear time. Larger time series create larger swap files, which require the disk head to move further to perform each random-access read operation. In other words, the number of cells in the cost matrix that must be filled/read is linear with respect to the length of the time series. So the *algorithm* is O(*N*), but the *implementation* is not quite O(*N*) for large time series when the entire search window will not fit into main memory.

5. CONCLUDING REMARKS

In this paper we introduced the FastDTW algorithm, a linear and accurate approximation of dynamic time warping (DTW). FastDTW uses a multilevel approach that recursively projects a warp path from a coarser resolution to the current resolution and refines it. While the quadratic time and space complexity of DTW has limited its use to only the smallest time series data sets, FastDTW can be run on much larger data sets. FastDTW is an order of magnitude faster than DTW, and it also compliments existing indexing methods that speed up time series similarity search and classification.

Our theoretical and empirical analysis showed that FastDTW has a linear time and space complexity. Expirical results have also

shown that FastDTW is accurate when warping both similar and dissimilar time series. With a *radius* of only 1, FastDTW had an average error of 8.6%, and increasing the *radius* to 20 lowers the error to under 1%. FastDTW's accuracy was compared to two existing methods, Data Abstraction and Sakoe-Chiba Bands, and was found to be far more accurate than either approach when using small *radius* values. FastDTW's solutions also always approached zero error (optimal warp path) with smaller *radius* values than the other two methods. An additional contribution of this paper is demonstrating how to apply the refinement portion of the FastDTW algorithm to the Data Abstraction approximate DTW algorithm. Doing so increased the accuracy of Data Abstraction by more than 100-fold in our evaluation with a *radius* of only 10.

The main limitation of the FastDTW algorithm is that it is an approximate algorithm and is not guaranteed to find the optimal solution (although it very often does). If for some reason a problem requires optimal warp paths to be found. Future work will look into increasing the accuracy of FastDTW. Possibilities to increase the accuracy of FastDTW include changing the step size (magnitute of the resolution change) between resolutions and evaluating search algorithms to guide search during the refinement step rather than simple expanding the search window in both directions.

6. REFERENCES.

[1] Abdulla, W., D. Chow, and G. Sin, Cross-words reference template for DTW-based speech recognition systems, in Proc. IEEE TENCON, Bangalore, India, 2003.

[2] Chu, S., E. Keogh, D. Hart & Michael Pazzani. Iterative Deepening Dynamic Time Warping for Time Series. In *Proc. of the Second SIAM Intl. Conf. on Data Mining*. Arlington, Virginia, 2002.

[3] Gupta, L., D. Molfese, R. Tammana & P. Simos. Nonlinear Alignment and Averaging for Estimating the Evoked Potential. In *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 4, pp. 346-356, 1996.

[4] Itakura, F. Minimum Prediction Residual Principle Applied to Speech Recognition. In *IEEE Trans. Acoustics, Speech, and Signal Proc*. vol. ASSP-23, pp 52-72, 1975.

[5] Karypis, G., R. Aggarwal, V. Kumar & S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Design Automation Conl*, pp. 526-530. Anaheim, California, 1997.

[6] Keogh, E. Exact Indexing of Dynamic Time Warping. In *VLDB*, pp. 406-417. Hong Kong, China, 2002.

[7] Keogh, E. and T. Folias. The UCR Time Series Data Mining Archive [http://www.cs.ucr.edu/~eamonn/TSDMA/ index.html], Riverside, CA, University of California – Computer Science and Engineering Department, 2002

[8] Keogh, E. & M. Pazzani. Derivative Dynamic Time Warping. In *Proc. of the First Intl. SIAM Intl. Conf. on Data Mining*, Chicago, Illinois, 2001.

[9] Keogh, E. & M. Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. In *Proc. of the Sixth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pp.285-289. Boston, Massachuseetts, 2000.

[10] Kim, S., S. Park & W. Chu. An Index-based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases. In *Proc. 17th Intl. Conf. on Data Engineering*, pp. 607-614. Heidelberg, Germany, 2001.

[11] Kruskall, J. & M. Liberman. The Symmetric Time Warping Problem: From Continuous to Discrete. In *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, pp. 125-161, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983

[12] Ratanamahatana, C. & E. Keogh. Making Time-series Classification More Accurate Using Learned Constraints. In *Proc of SIAM Intl. Conf. on Data Mining*, pp. 11-22. Lake Buena Vista, Florida, 2004.

[13] Sakoe, H. & S. Chiba. (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-26.

[14] Salvador, Stan. Learning States for Detecting Anomalies in Time Series. Master's Thesis, CS-2004-05, Dept. of Computer Sciences, Florida Institute of Technology, 2004.

[15] Vlachos, M., G. Kollios, & D. Gunopulos. Discovering Similar Multidimensional Trajectories. In *Proc. 18th Intl. Conf. on Data Engineering*, pp. 673-684, San Jose, California, 2002.

Apéndice B

Contenidos adicionales

En este apéndice se detallan los contenidos adicionales que se adjuntan a esta memoria.

- **Imágenes de diagramas de análisis y diseño**: Como ya se ha comentado, al poder no verse bien ciertos diagramas, se adjuntan las imagenes en tamaño más grande y en posición normal.
- **Código fuente**: También se adjunta, todo el proyecto Eclipse utilizado para el desarrollo de la aplicación.
- **Fichero de instalación**: Para poder instalar la aplicación en el dispositivo, se adjunta el fichero .apk que el cual al ejecutarse instalará la aplicación en el terminal.
- **Planificación del proyecto**: Como en todo buen proyecto, en este también se ha realizado una planificación. En el apendice C se detalla el seguimiento del mismo. Por si el lector quiere verlo, se añaden la planificación inicial y como ha sido el proyecto realmente. Estos ficheros están en formato para Microsoft Project.

Apéndice C

Seguimiento del proyecto

Para este proyecto, se realizó como paso previo una planificación solamente temporal, debido a que el proyecto ha sido realizado por una única persona, haciendo queremosla gestión de recursos no tuviera sentido.

La planificación se ha visto formada por distintas etapas (como se ve en la figura C.1):

1. **Trabajo previo**: En este paso, se realiza todo el trabajo anterior como la documentación del sistema Android. De esta forma, no se necesitará en pasos posteriores formación para, por ejmplo, implementar el sistema en Android.

2. **Análisis**: Fase de análisis proceso de ingeniería de software.

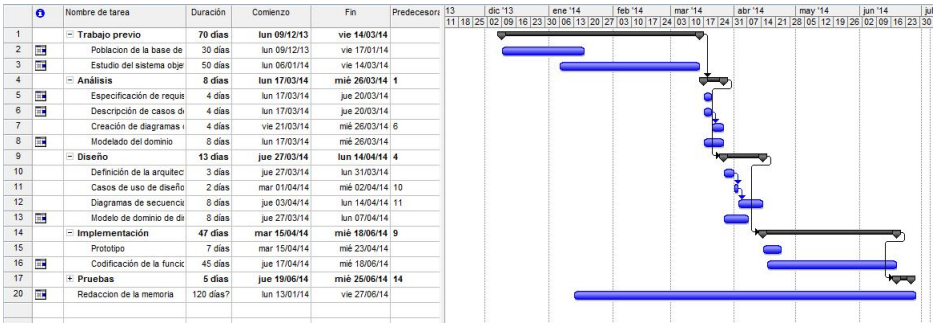
3. **Diseño**: Fase de diseño proceso de ingeniería de software.

4. **Implementación**: Fase de implementación proceso de ingeniería de software.

5. **Pruebas**: Fase de pruebas proceso de ingeniería de software.

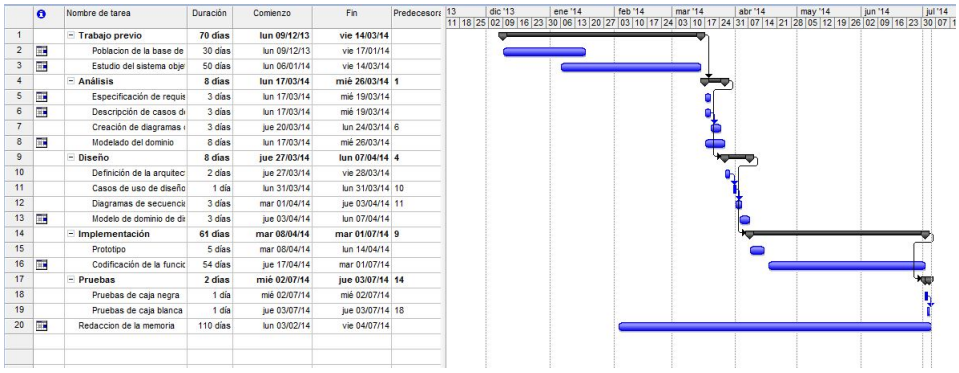
6. **Redacción**: Durante todo el proceso, se ha ido realizando la documentación. Esta etapa ha sido, por ello, constante a lo largo de ella.

Cómo se ve en la figura C.2, el desarrollo de toda la aplicación se ha visto aumentado en una semana, aunque debido a la sobreestimación de las fases de análisis y diseño, ha permitido que la entrega se haya realizado a tiempo. El principal problema sufrido es que la fase de implementación ha durado más de lo esperado, debido al retraso de los resultados sobre umbrales que tenía que proporcionar el grupo ECA-SIMM.



	Nombre de tarea	Duración	Comienzo	Fin	Predecesor
1	**Trabajo previo**	**70 días**	**lun 09/12/13**	**vie 14/03/14**	
2	Poblacion de la base de	30 días	lun 09/12/13	vie 17/01/14	
3	Estudio del sistema obje	50 días	lun 06/01/14	vie 14/03/14	
4	**Análisis**	**8 días**	**lun 17/03/14**	**mié 26/03/14**	**1**
5	Especificación de requis	4 días	lun 17/03/14	jue 20/03/14	
6	Descripción de casos d	4 días	lun 17/03/14	jue 20/03/14	
7	Creación de diagramas	4 días	vie 21/03/14	mié 26/03/14	6
8	Modelado del dominio	8 días	lun 17/03/14	mié 26/03/14	
9	**Diseño**	**13 días**	**jue 27/03/14**	**lun 14/04/14**	**4**
10	Definición de la arquitec	3 días	jue 27/03/14	lun 31/03/14	
11	Casos de uso de diseño	2 días	mar 01/04/14	mié 02/04/14	10
12	Diagramas de secuencia	8 días	jue 03/04/14	lun 14/04/14	11
13	Modelo de dominio de di	8 días	jue 27/03/14	lun 07/04/14	
14	**Implementación**	**47 días**	**mar 15/04/14**	**mié 18/06/14**	**9**
15	Prototipo	7 días	mar 15/04/14	mié 23/04/14	
16	Codificación de la funcio	45 días	jue 17/04/14	mié 18/06/14	
17	**Pruebas**	**5 días**	**jue 19/06/14**	**mié 25/06/14**	**14**
20	Redaccion de la memoria	120 días?	lun 13/01/14	vie 27/06/14	

Figura C.1: Planificación inicial



	Nombre de tarea	Duración	Comienzo	Fin	Predecesor
1	**Trabajo previo**	**70 días**	**lun 09/12/13**	**vie 14/03/14**	
2	Poblacion de la base de	30 días	lun 09/12/13	vie 17/01/14	
3	Estudio del sistema obje	50 días	lun 06/01/14	vie 14/03/14	
4	**Análisis**	**8 días**	**lun 17/03/14**	**mié 26/03/14**	**1**
5	Especificación de requis	3 días	lun 17/03/14	mié 19/03/14	
6	Descripción de casos d	3 días	lun 17/03/14	mié 19/03/14	
7	Creación de diagramas	3 días	jue 20/03/14	lun 24/03/14	6
8	Modelado del dominio	8 días	lun 17/03/14	mié 26/03/14	
9	**Diseño**	**8 días**	**jue 27/03/14**	**lun 07/04/14**	**4**
10	Definición de la arquitec	2 días	jue 27/03/14	vie 28/03/14	
11	Casos de uso de diseño	1 día	lun 31/03/14	lun 31/03/14	10
12	Diagramas de secuencia	3 días	mar 01/04/14	jue 03/04/14	11
13	Modelo de dominio de di	3 días	jue 03/04/14	lun 07/04/14	
14	**Implementación**	**61 días**	**mar 08/04/14**	**mar 01/07/14**	**9**
15	Prototipo	5 días	mar 08/04/14	lun 14/04/14	
16	Codificación de la funcio	54 días	jue 17/04/14	mar 01/07/14	
17	**Pruebas**	**2 días**	**mié 02/07/14**	**jue 03/07/14**	**14**
18	Pruebas de caja negra	1 día	mié 02/07/14	mié 02/07/14	
19	Pruebas de caja blanca	1 día	jue 03/07/14	jue 03/07/14	18
20	Redaccion de la memoria	110 días	lun 03/02/14	vie 04/07/14	

Figura C.2: Seguimiento real