DOCUMENTO 1
MEMORIA DESCRIPTIVA

Proyecto de **PROTOTIPO DE RECONOCIMIENTO DE HUELLA DACTILAR MEDIANTE DSP**.

El presente proyecto de prototipo se realiza por encargo de:

D./Dña. : _____
con N.I.F : _____ en representación de _____
_____ con C.I.F.:_____
domicilio en _____
de_____C.P.:_____
provincia de _____ Tlf: _____ Fax: _____
E-mail: _____ Web: _____

Ha sido realizado por D. Juan Manuel Fernández Muñoz, con N.I.F.: 15455297-X , con domicilio en C/ Pozo Nuevo, 31 de El Rubio, C.P. 41568 de SEVILLA; como Proyecto Fin de Carrera de Ingeniería Técnica Industrial especialidad en Electrónica Industrial, de la Escuela Politécnica Superior de Sevilla.

Sevilla a 22 de Septiembre de 2012

Fdo. D./Dña.
Firma Promotor

Fdo. D. Juan Manuel Fernández Muñoz
Firma Proyectista

Índice de la memoria

Capítulo 1. Introducción

El presente proyecto trata sobre el desarrollo de un prototipo para reconocimiento de huellas dactilares usando para ello un Procesador Digital de Señal (DSP). Se comentarán las técnicas existentes para la Comparación de Huellas Dactilares y el potencial que presentan los DSPs en el área del tratamiento intensivo de datos.

Para poder llevar a cabo la comparación de dos huellas, lo primero es obtener una imagen representativa de cada una de ellas, y una vez hecho esto se realizan todas las operaciones pertinentes sobre ellas.

Las huellas son tratadas como imágenes y la mayoría de algoritmos en el procesamiento de imágenes se caracterizan por realizar repetitivamente la misma operación sobre un grupo de píxeles. Las operaciones más comunes son las siguientes:

- Una de las operaciones más comunes implica el uso de una sola imagen y un escalar, como es el caso de una multiplicación escalar.

$$g(x,y) = \alpha \cdot f(x,y)$$

Fórmula 1.1. Multiplicación escalar de una imagen.

  Donde $g(x,y)$ y $f(x,y)$ son imágenes, x es la fila e y es la columna que indican la posición del píxel y α es un escalar. Lo que hace la expresión anterior es multiplicar cada píxel de la imagen por un escalar.

- Operaciones píxel a píxel de imágenes del mismo tamaño, como son:

$$g(x,y) = f_1(x,y) + f_2(x,y)$$

Fórmula 1.2. Suma de píxeles de dos imágenes.

$$g(x,y) = f_1(x,y) - f_2(x,y)$$

Fórmula 1.3. Resta de píxeles de dos imágenes.

$$g(x,y) = f_1(x,y) \cdot f_2(x,y)$$

Fórmula 1.4. Multiplicación de píxeles de dos imágenes.

$$g(x,y) = f_1(x,y) \,/\, f_2(x,y)$$

Fórmula 1.5. División de píxeles de dos imágenes.

  Las cuales realizan la operación indicada sobre el píxel (x,y) de ambas imágenes.

- Otro tipo de operación muy común en el tratamiento de imágenes implica multiplicar y acumular el resultado repetidas veces. Dicha operación por ser tan

común ha recibido nombre propio, el cual es operación MAC (multiply-accumulate operation, del inglés).

$$MAC = (a \cdot b) + c$$

Fórmula 1.6. Operación MAC.

Todas las operaciones anteriores tienen una característica primordial que destaca de entre todas las demás, la cual es que implican repetitivos cálculos numéricos que requieren un ancho de banda de memoria alto. En resumen, el procesamiento de imágenes es a la vez mucha computación y una generación intensiva de datos. Además, las aplicaciones de procesamiento de imágenes se están encontrando cada vez más en sistemas empotrados, a menudo en lugares donde los plazos de tiempo real deben cumplirse. Con respecto a los sistemas empotrados, pongamos por ejemplo una cámara digital de un teléfono celular. Tal dispositivo requiere un cerebro informático que realiza las tareas numéricas descritas anteriormente con una alta eficiencia, mientras que al mismo tiempo se minimiza el uso de energía, de memoria, y en el caso de productos de alto volumen, el coste de los mismos. Con estas limitaciones de tiempo real nos encontramos en un entorno en el que es muy probable que un Procesador de Propósito General (GPP) no es la opción adecuada. Éstos están diseñados para realizar una amplia gama de tareas de computación (muchos de ellos sin una orientación al cálculo numérico) y normalmente ejecutan sistemas operativos muy pesados computacionalmente, por lo que definitivamente no son aptos para sistemas empotrados y sobre todo de tiempo real.

Los Procesadores Digitales de Señal (DSP) llegaron a principios de 1980 para hacer frente a la necesidad de procesar flujos de datos continuos en tiempo real. Inicialmente fueron utilizados principalmente en aplicaciones de procesamiento de señales en campos como las telecomunicaciones o el audio, hasta hoy en día el mayor uso se sigue dando en los mismos campos. Sin embargo, el aumento de la multimedia en la década de 1990 coincidió con una creciente necesidad de procesar imágenes y flujos de datos de vídeo, muy a menudo en entornos en los que un GPP no es viable. Ha habido una clara divergencia en la evolución de los DSPs y GPPs, a pesar de que todos los fabricantes de GPPs, como Intel, AMD o SUN, han introducido extensiones DSP para sus procesadores. Pero el hecho es que, en general, las aplicaciones DSP difieren de sus homólogos GPP en que están casi siempre caracterizadas por programas relativamente pequeños (especialmente en comparación con programas gigantes como bases de datos, un navegador web normal, o procesadores de textos) que implican un procesamiento aritmético intensivo, en particular con el uso de la operación MAC que forma el bloque de construcción de muchos algoritmos para DSP. Normalmente hay menos lógica utilizada en los programas de un DSP, donde la lógica se refiere a las ramas y las instrucciones de control. Más bien, lo que encontramos es

que las aplicaciones de DSP están dominadas por bucles críticos bien codificados. Los DSPs tienen una arquitectura tal que maximizan el rendimiento de estos bucles críticos, a veces en detrimento de otras tareas de computación más orientados a la lógica. Un DSP es por tanto la mejor opción para el alto rendimiento del procesamiento de imágenes, donde los algoritmos consisten en gran medida en repetitivos cálculos numéricos que operan píxeles o grupos de píxeles, y donde dicho tratamiento debe realizarse a un bajo costo, baja potencia, en un sistema empotrado, y posiblemente en tiempo real.

Los DSPs poseen características estructurales únicas que les dan una gran ventaja en algoritmos de procesamiento de señales e imágenes, incluyendo cero bucles generales, un soporte especializado de E/S, estructuras únicas de memoria caracterizadas por múltiples bancos de memoria y buses, aritmética saturada, etc.

En definitiva, un sistema empotrado gobernado por un DSP parece viable para el desarrollo de nuestro proyecto, en el que implementaremos un algoritmo para el reconocimiento de huellas dactilares.

Capítulo 2. Antecedentes del reconocimiento de huellas dactilares

En este capítulo se realizará un resumen del estado de la técnica y de los conceptos esenciales en el reconocimiento biométrico mediante huella dactilar.

La huella dactilar es un identificador biométrico ampliamente utilizado y estudiado. La investigación de la comparación de huellas dactilares experimentó un auge durante el siglo pasado debido al desarrollo de sistemas automáticos para la comparación de huellas. Estos sistemas han sido profusamente utilizados tanto en el ámbito policial y forense como en el ámbito civil. La investigación en este campo ha proporcionado numerosas técnicas de comparación de huellas dactilares así como un gran abanico de sensores que proporcionan imágenes de resolución y calidad crecientes.

Se prestará especial atención a la etapa de comparación y a las diferentes técnicas utilizadas, subdividiéndose en: técnicas basadas en minucias, técnicas basadas en crestas y técnicas de correlación. Las técnicas basadas en minucias son las más estudiadas y utilizadas. Estas técnicas realizan la comparación de las huellas dactilares mediante la identificación y localización de los puntos singulares. El resto de las técnicas de comparación han surgido como alternativa a las técnicas basadas en minucias, tratando de solventar los problemas que presentan estas técnicas, que se detallarán a lo largo de este capítulo.

2.1. Antecedentes generales

2.1.1. Biometría

La biometría consiste en la identificación de un individuo a través del análisis de sus características físicas o rasgos conductuales. El término se deriva de las palabras griegas "bios" de vida y "métron" de medida.

2.1.2. Sistemas biométricos

Los sistemas biométricos son sistemas diseñados para conceder acceso a usuarios basándose en características únicas e inalterables de cada individuo que difícilmente pueden ser duplicadas, perdidas, olvidadas o robadas, pretendiendo reconocer quién es el individuo, contrariamente a los sistemas de acceso más comunes basados en objetos (llaves, smartcards…) o códigos (passwords, PIN…) que pueden ser perdidos, olvidados, robados o descifrados.

Son precisamente estas características físicas o conductivas de cada individuo lo que ha impulsado el desarrollo de distintos tipos de sistemas para su identificación. Desde el punto de vista conductivo podemos analizar la caligrafía o la voz de un individuo,

siendo esta última donde más esfuerzos se han aplicado desde una perspectiva tecnológica. Pero sin duda alguna el desarrollo de los sistemas biométricos centra sus esfuerzos en la identificación de las características físicas como son, por ejemplo, la retina, el iris, la geometría del la mano, la cara o la huella dactilar.

2.1.3. Procesos de autenticación e identificación biométrica

En el control de accesos mediante biometría existe una fase de verificación, durante la cual, el dato biométrico o plantilla, ya adquirido por el sistema, es cotejado con el que está siendo sometido a análisis, intentando confirmar la identidad declarada de un individuo, al confrontar la muestra suministrada con una o más plantillas registradas con anterioridad siguiendo uno de los dos procesos fundamentales, el de Autentificación o el de Identificación.

El Proceso de Autentificación (o Verificación), también conocido como Uno-Para-Uno (1:1), se basa en la comparación de los rasgos biométricos con los de un patrón ya guardado. El objetivo es confirmar la identidad del individuo.

Los sistemas automáticos para verificación se denominan AFAS (Automatic Fingerprint Autentication System).

El proceso de Identificación o Uno-Para-Muchos (1:N), compara los rasgos biométricos con los de un conjunto de plantillas residentes en una base de datos. Pretende identificar al individuo.

Cuando se conoce que el individuo a identificar está dentro de la base de datos, se dice que la identificación es “de grupo cerrado”.

En identificación "de grupo abierto”, también llamada “lista de vigilancia”, no existe garantía de que el individuo sea parte de la base de datos. El sistema debe determinar si el individuo es parte de la base de datos y en caso afirmativo proporcionar su identidad.

Los sistemas automáticos para identificación se denominan AFIS (Automatic Fingerprint Identification System).

Debido al número de comparaciones y procesamiento que debe llevarse a cabo en el proceso de Identificación, resulta más rápido el proceso de Autentificación, especialmente conforme va aumentando N en la Identificación.

2.1.4. Funcionamiento y rendimiento

La pauta habitual en un Sistema Biométrico, consta de un proceso previo de registro, donde se adquieren una o varias señales representantes de características biométricas

del usuario, se procesan mediante algún algoritmo numérico para finalmente ser almacenadas en una base de datos con la información única para cada individuo.

Idealmente, en la autentificación o identificación de un individuo, el sistema adquiere y procesa las características biométricas, las compara con las almacenadas en la base de datos y si concuerda permite el acceso.

Las tecnologías actuales tienen tasas de error que varían ampliamente, desde valores de seguridad bajos como el 60% hasta altos como el 99,9% .

Los parámetros más usados para medir el rendimiento de un sistema biométrico son los siguientes:

- Tasa de Falso Positivo (False Acceptance Rate o FAR): es la probabilidad de que un usuario no registrado sea aceptado en cualquier caso. Mide la frecuencia con que un usuario no registrado es admitido por equívoco. Actualmente el margen está entre el 0.0001% y el 0.1%.
  A partir del FAR se puede obtener una medida de la seguridad del sistema de verificación biométrica, según la ecuación, seguridad = (1 – FAR).

- Tasa de Falso Rechazo (False Rejection Rate o FRR): es la probabilidad de que un usuario esté registrado pero sea rechazado. Mide la frecuencia con que un usuario registrado es rehusado. Comercialmente su valor varía entre el 0.00066% y el 1%.
  Conociendo FRR, se puede precisar la conveniencia de un sistema biométrico, de acuerdo con la ecuación, conveniencia = (1 – FRR). Conforme aumenta FRR menos conveniente resulta el sistema por el alto número de accesos denegados de forma errónea.

- Tasa de Error Igual (Equal Error Rate, EER): es la intersección entre FAR y FRR siendo una de las medidas más usada para determinar el umbral de calidad del sistema biométrico, cuando los errores de aceptación y rechazo son iguales, el umbral se establece en el 50%. Cuanto más bajo es EER el sistema es más fiable. Estos umbrales deben ser ajustados de acuerdo con la garantía requerida.

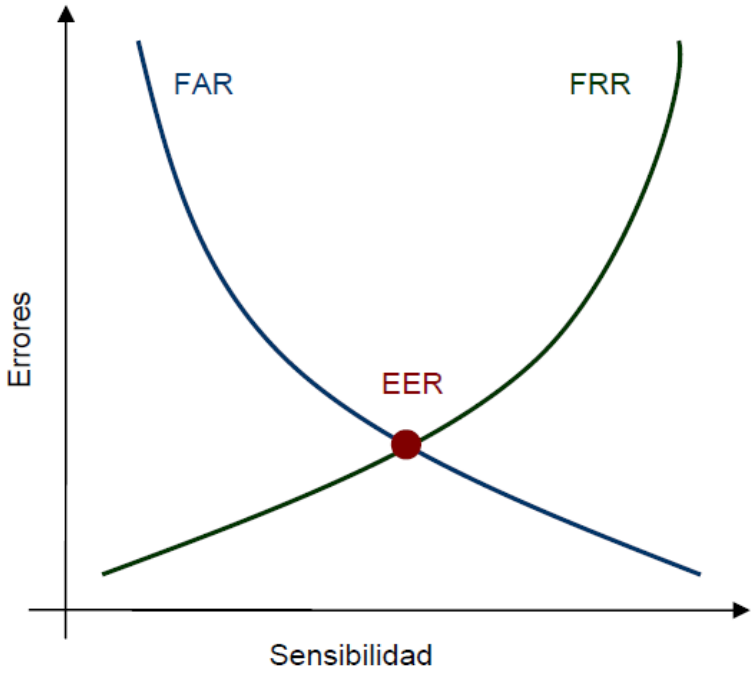

Figura 2.1. Tasa de error de un sistema biométrico.

2.2. Antecedentes particulares

2.2.1. Características de las huellas dactilares

Las huellas dactilares son la reproducción de la epidermis de la parte posterior de los dedos de la mano. Una huella dactilar está formada por un conjunto de líneas que se denominan crestas (líneas oscuras) y valles (líneas claras). Este conjunto de líneas que forman las huellas dactilares pueden asemejarse a patrones o texturas que se pueden analizar de diferentes maneras dependiendo del grado de detalle.

A nivel macroscópico podemos observar el flujo de las líneas descritas por las crestas. En éste nivel cobra especial importancia el núcleo (core) de las huellas, con el que se podría clasificar la huella según el tipo de núcleo.

A nivel microscópico podemos encontrar las minucias, las cuales son puntos singulares de la huella como terminaciones o bifurcaciones de las crestas. Las minucias son fáciles de extraer como características de una huella particular.

A nivel global podemos incluir todos los atributos dimensionales de las crestas excluidos en los niveles anteriores, como por ejemplo la anchura de las crestas, su forma, los poros contenidos en ellas, las crestas incipientes, los cortes, las cicatrices y otros muchos detalles permanentes presentes en las crestas. Las técnicas basadas en correlación de imágenes de huellas están dentro de este grupo.

Las principales características por las que la huella dactilar es un buen identificador biométrico son las siguientes:

- No existen dos huellas de individuos diferentes con un patrón de crestas coincidentes.
- El patrón de cada individuo es invariable en el tiempo durante toda la vida de éste.

Figura 2.2. Huella dactilar.

2.2.2. La huella dactilar como identificador biométrico

Hoy en día las huellas dactilares se utilizan extensamente tanto en aplicaciones civiles como forenses/policiales. Esta amplia utilización y aceptación de la huella dactilar como identificador biométrico es debida en gran medida a su temprano estudio científico como medio de identificación unívoco de las personas. A finales del siglo XIX aparecieron dos estudios científicos relevantes en este ámbito: Galton, con el análisis de las huellas dactilares y la aparición de las minucias para su comparación, y Henry, con la clasificación de las huellas dactilares en función de la forma macroscópica formada por sus crestas.

A principios del siglo XX las huellas comienzan a utilizarse profusamente en la ciencia forense, facilitando a su vez a la policía la identificación de criminales. Esto conlleva la creación de bases de datos conteniendo huellas en todos los países, que experimentan un aumento considerable en el número de huellas y por tanto requieren un número creciente de expertos para su evaluación y comparación.

Varios países y gobiernos ven la imperiosa necesidad de crear sistemas automáticos de identificación de huellas dactilares (AFIS) y empiezan las investigaciones en este ámbito a mediados del siglo XX. Este temprano conocimiento científico, unido a la

amplia utilización de la huella dactilar en el ámbito forense y policial, proporcionó un importante estudio y desarrollo de los AFIS. Este avance es palpable por ejemplo en los sensores de huella dactilar existentes en el mercado, ya que existe una variada gama de sensores con respecto a calidad de imagen, técnica de captura de la imagen e incluso precio. Este avance técnico y su amplia aceptación provocaron el paso de la huella dactilar del ámbito forense/policial al ámbito de aplicaciones civiles, entre las que destaca el control de accesos mediante sistemas automáticos de verificación (AFAS).

Las minucias se utilizaban en el reconocimiento manual que realizaban los expertos antes de existir los AFIS y es por ello que al diseñar los primeros sistemas automáticos gran parte de los algoritmos se basaron en ellas. Esta tendencia ha seguido existiendo con el paso del tiempo y las técnicas basadas en minucias son las más estudiadas y sobre las que existe una mayor cantidad de algoritmos de identificación.

2.2.3. Algoritmos de comparación de huellas dactilares

Existen multitud de algoritmos de comparación de huellas dactilares. Las principales técnicas de comparación de huellas dactilares se pueden dividir en tres grandes subgrupos, los cuales se diferencian por los tipos de patrones comparados. Son las siguientes:

- Técnicas basadas en comparación de minucias. Son las más utilizadas y estudiadas. Básicamente consisten en conseguir el alineamiento óptimo de dos huellas para posteriormente realizar correspondencias entre el mayor número de pares de minucias.
- Técnicas basadas en comparación de características de las crestas. Estas técnicas realizan la comparación en base a otros atributos de las crestas, como pueden ser los poros, el grosor de las crestas, etc.
- Técnicas de correlación. Realizan la correlación de los píxeles de las imágenes de las huellas dactilares para determinar el grado de similitud de las imágenes.

2.2.3.1. Problemas a solventar en la identificación de huellas

La comparación de huellas dactilares presenta una serie de problemas comunes a todos los tipos de técnicas, ya que están asociados a las imágenes utilizadas y a las técnicas de captura. Estos problemas existentes en las muestras deben ser tratados para minimizar su efecto y se resumen a continuación:

- Desplazamiento relativo de las muestras.
- Rotación relativa de las muestras.

- Solapamiento parcial entre las muestras. Debido a los sensores de escasa superficie o a la mala captura de las muestras, pueden llegar a tomarse muestras de un mismo dedo con escaso solapamiento.
- Distorsión no lineal, debida a la elasticidad de la piel.
- Efectos en la imagen debidos a las características de la piel del dedo como pueden ser: presión, humedad, sequedad, etc.
- Ruido. Puede ser introducido por el sensor en el proceso de captura.

2.2.3.2. Técnicas basadas en comparación de minucias

Los algoritmos basados en minucias utilizan la comparación de minucias o puntos singulares de las crestas de las huellas dactilares. Son los más utilizados y estudiados, ya que son una versión automatizada del método que utilizan los expertos de la policía a nivel mundial desde hace décadas para el reconocimiento de criminales.

En este tipo de algoritmos cobra suma importancia la extracción de las minucias. En esta fase, a cada huella se le asocia un conjunto de minucias que estará formado por un vector de información para cada una de las mismas. El contenido del vector varía notablemente de un algoritmo a otro, pero es necesario que contenga la posición de la minucia acompañada por algún tipo de información relevante de la misma, como puede ser el tipo de minucia (bifurcación, terminación, etc.), orientación, parte de la cresta que lo contiene, la posición relativa respecto a otras minucias, etc. La cantidad de información contenida en el vector así como el número de minucias requeridas para cada huella determina el tamaño de la información que debe ser almacenada, lo que es un parámetro crítico para multitud de aplicaciones.

Una vez obtenidos los vectores, el algoritmo alineará los conjuntos de minucias para poder determinar el número de minucias coincidentes y con ello emitir un resultado o "score" sobre la similitud de dos muestras.

Debido a las características de la imagen de la huella dactilar, ni la extracción ni la comparación de minucias son triviales. Hay que tener en cuenta y contrarrestar numerosos efectos indeseados descritos anteriormente, como por ejemplo: desplazamiento, rotación, distorsión no lineal, ruido, presión y estado de la piel, etc.

Las técnicas de comparación basadas en minucias, aunque son las más utilizadas y estudiadas, presentan numerosos inconvenientes entre los que destacan:

- Carga computacional media, debida a la alta complejidad de los algoritmos necesarios para la extracción y comparación de minucias. El preprocesado necesario para este tipo de algoritmos es muy complejo y además debido a este

fuerte preprocesado se pueden generar crestas y minucias falsas que perjudican a la precisión de los algoritmos.

- Generación de falsas minucias en el proceso de comparación que será necesario evaluar para su posterior descarte.

- Sesgo de la información comparada. Al fin y al cabo se está extrayendo un conjunto de puntos de una imagen para caracterizarla, despreciando el resto de la información contenida en ella.

Todos estos inconvenientes han dado lugar al estudio y desarrollo de nuevas técnicas basadas en otras características para aumentar la precisión y la eficiencia de los AFIS.

2.2.3.3. Técnicas basadas en crestas

Los algoritmos de este grupo extraen características de las crestas para realizar la comparación de las huellas. En realidad la extracción de minucias es considerada por algunos autores como un subgrupo de estos algoritmos, ya que las minucias también se extraen de ellas, pero debido a su relevancia se han considerado aparte en esta clasificación de técnicas de comparación. En un principio estos algoritmos surgieron como una alternativa a los algoritmos basados en minucias y con ellos se buscaba reducir el coste computacional ahorrando la extracción de minucias para utilizar otras características que poseyeran las crestas. Existen numerosas aproximaciones en este grupo, de entre las que destacan:

- Poros. Los poros de sudor que se localizan en las crestas cumplen los requisitos para ser un identificador biométrico, pero es necesario recurrir a sensores de alta resolución y coste para poder detectarlos.

- Geometría y atributos de las crestas. Basados en la comparación geométrica de las crestas.

- Textura. Las texturas son la repetición espacial de elementos básicos y se pueden caracterizar por multitud de parámetros como son: frecuencia, escalado, orientación, simetría, etc. En una huella los cambios de textura se manifiestan de forma brusca en las regiones dónde existen discontinuidades, mientras que en el resto de la huella la frecuencia y la orientación tienen una variación suave. Estas discontinuidades en la textura manifiestan la situación de los puntos singulares de la huella.

En este tipo de algoritmos se engloban gran cantidad de diferentes técnicas que se caracterizan por la necesidad de imágenes de alta resolución y una alta complejidad computacional para extraer las características. En algunos casos las características extraídas proporcionan un conjunto de información más completa que las minucias,

pero su comparación y extracción no pueden considerarse en la mayoría de los casos ni más sencillas ni con menor complejidad computacional.

2.2.3.4. Técnicas basadas en correlación

Este tipo de algoritmos utiliza la correlación entre dos imágenes para medir su similitud. Básicamente, la correlación de dos imágenes acumula la comparación píxel a píxel del nivel de gris de las imágenes.

La correlación de las imágenes equivale a una convolución de ambas, proceso que es muy costoso computacionalmente.

Para reducir la carga computacional se puede calcular la correlación en el dominio de la frecuencia. El motivo de ello es que la una convolución en el dominio sin transformar equivale a un porducto en el dominio transformado. Ello se logra haciendo la transformada de Fourier en 2D a ambas imágenes, a una de ellas se le hace el complejo conjugado, luego se multiplican y por último se le hace la transformada inversa al resultado, dándonos éste información sobre la similitud entre ambas huellas.

La principal ventaja de éste método es que tiene en cuenta toda la información de la imagen, al contrario de los métodos explicados anteriormente que extraen solo parte de dicha información. Además la correlación es inmune a desplazamientos entre las imágenes.

El principal problema es su coste computacional, además de los problemas explicados anteriormente a excepción del desplazamiento.

Existe una diferencia en la forma de actuar en éste tipo de algoritmo respecto a los demás. Ésta es que no realiza un gran preprocesado de la imagen, sino que la etapa de mayor computación es la de verificación, mientras que en los otros tipos ocurre lo contrario.

Algunos algoritmos en vez de hacer la correlación a toda la imagen, la hacen a pequeñas partes de ella, con lo que se reduce el tiempo de computación, pero se desecha parte de la información de la huella.

El uso de este tipo de algoritmo tiene poca eficiencia en procesadores de propósito general debido a un alto coste computacional numérico y a una generación intensiva de datos, por lo que se tiende a implementarlos en un hardware más específico como FPGAs o DSPs.

Capítulo 3. Objeto del proyecto

El objetivo de este proyecto final de carrera es la realización de un prototipo de reconocimiento de huellas dactilares mediante DSP.

Para llevar a cabo dicha tarea se seleccionará un DSP comercial que satisfaga en la medida de lo posible nuestras necesidades, por lo que lo más razonable es la adquisición de un kit de desarrollo basado en DSP, el cual tenga opción de ampliar su funcionalidad añadiendo un dispositivo capaz de adquirir huellas dactilares.

A continuación mostraremos una lista con las principales etapas a desarrollar en el proyecto:

- Realización de un estudio general teórico del estado del arte en el reconocimiento de huellas dactilares.
- Desarrollo de un algoritmo capaz de comparar huellas dactilares. Se analizarán los datos obtenidos en el apartado anterior para la definición del algoritmo, y se intentará implementar alguna técnica nueva.
- Traducción del algoritmo a lenguaje C para poder ser implementado en un sistema real. Se creará un sistema con base de datos para mostrar un ejemplo práctico del uso del algoritmo de comparación. Además se estudiarán técnicas relacionadas con el procesado de señales e imágenes para reducir en la medida de lo posible el tiempo de ejecución del algoritmo y aprovechar las posibilidades que ofrecen los DSPs.
- Implementación del algoritmo en el DSP, con su consiguiente prueba de errores para determinar el rendimiento del algoritmo.

Este proyecto se ha realizado a petición de D. _____,
con una inversión estimada de _____ €, para la realización de 1 prototipo.

Capítulo 4. Marco normativo legal

4.1. Legislación de ámbito europeo

- **Directiva 2003/108/CE del Parlamento Europeo y del Consejo, de 8 de Diciembre de 2003**, por la que se modifica la Directiva 2002/96/CE del Parlamento Europeo y del consejo, de27 de Enero de 2003, sobre residuos de aparatos eléctricos o electrónicos (RAEE).

- **Directiva 2004/108/CE del Parlamento Europeo y del Consejo de 15 de Diciembre de 2004** relativa a la aproximación de las legislaciones de los estados miembros en materia de compatibilidad electromagnética y por la que se deroga la Directiva 89/336/CEE.

4.2. Legislación de ámbito nacional

- **Reglamento Electrotécnico de Baja Tensión,** aprobado por el Real Decreto 842/2002, de 2 de Agosto, así como sus instrucciones técnicas complementarias.

- **REAL DECRETO 208/2005, de 25 de Febrero,** sobre aparatos eléctricos y electrónicos y la gestión de sus residuos.

4.3. Normas

- **UNE-EN 60950 -** Equipos de Tecnologías de la Información. Seguridad.

- **UNE-EN 55024** y **UNE-EN 55022 -** Normativa de inmunidad y emisiones electromagnéticas.

- **UNE 157001:2002 -** Criterios generales para la elaboración de Proyectos.

- **Normativa de planos y documentación técnica.** Se atenderá a lo expuesto en las siguientes normas:

- **UNE 1027 -** Dibujos Técnicos. Plegado de planos.

- **UNE 1032 -** Dibujos Técnicos. Principios generales de representación.

- **UNE 1035 -** Dibujos Técnicos. Cuadro de rotulación.

- **UNE-EN ISO 3098 -** Documentación Técnica de Productos.

- **UNE-EN ISO 5455 -** Dibujos Técnicos. Escalas.

- **UNE-EN ISO 5457 -** Documentación Técnica de Productos. Formatos y presentación de los elementos gráficos de las hojas de dibujo.

- **UNE-EN ISO 6433 -** Dibujos Técnicos. Referencia de los elementos.

- **UNE-EN ISO 81714-1 -** Diseño de símbolos gráficos utilizables en la documentación técnica de productos. Parte 1: Reglas básicas.

Capítulo 5. Análisis de la solución adoptada

5.1. Análisis de las diferentes herramientas de desarrollo adquiridas

Buscando una solución al problema de encontrar un equipo que satisfaga las necesidades de nuestro proyecto en este capítulo presentaremos los elementos elegidos para llevarlo a cabo.

5.1.1. TMS320C6713 DSK

La tarjeta TMS320C6713 DSP Starter Kit (DSK), desarrollada por Spectrum Digital, es una plataforma de desarrollo de bajo coste para aplicaciones en tiempo real sobre señales digitales. El DSK consta de los siguientes elementos principales:

- Un procesador digital de señal (DSP) TMS320C6713, desarrollado por Texas Instruments, el cual soporta operaciones en coma flotante y opera a 225 MHz.
- Un circuito de interfaz analógica (códec) TLV320AIC23, desarrollado por Texas Instruments.
- Una memoria SDRAM con 16 MB de capacidad, para almacenamiento de datos.
- Una memoria flash no volátil de 512 kB de capacidad donde se alojará el código compilado del programa desarrollado.
- Conectores de expansión estándar para el uso de tarjetas hermanas (daughter cards).
- Conexión a PC mediante interfaz USB, el cual está conectado al puerto JTAG del DSK.
- Alimentación de la placa de +5 V.

El kit contiene el software necesario para escribir en código C o ensamblador y bien compilarlo o ensamblarlo, y luego enlazarlo y cargarlo en el DSK para su ejecución.

La arquitectura y el conjunto de instrucciones de un DSP están optimizados para el procesado digital de señales en tiempo real. También es importante el bajo consumo del mismo.

El DSP será el encargado de realizar el procesado de las imágenes y mostrará los resultados por el monitor de un PC. La comunicación entre el DSP y el PC se realizará vía USB. Esta conexión se utilizará tanto para cargar el código compilado por CCS en el DSP como para comunicar el dispositivo con el PC con el fin de darle órdenes.

En la figura 5.1 se muestra una vista del DSK con las entradas y salidas mostradas en él, además de la indicación de los principales elementos de éste. Los buses para expansión de memoria y para expansión de periféricos serán usados para la comunicación entre el DSP y la tarjeta hermana.

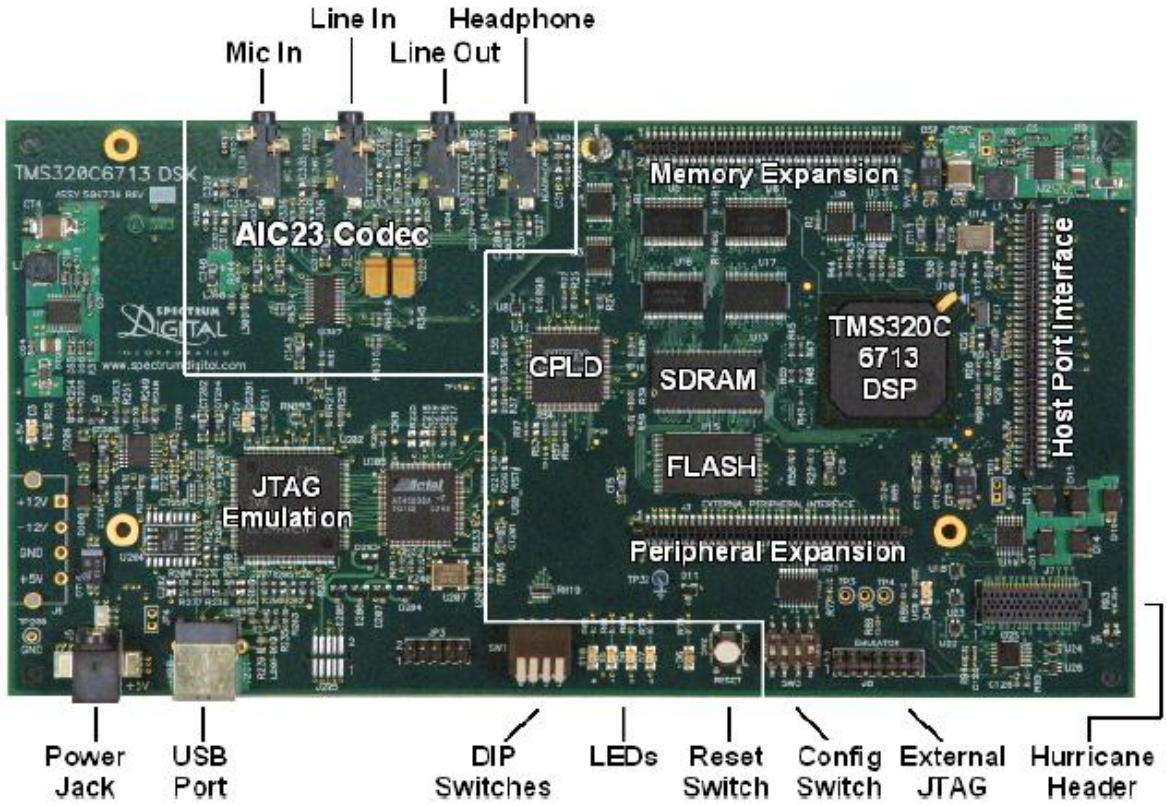

Figura 5.1. Vista del DSK.

En la figura 5.2 se muestra un diagrama de bloques con los elementos principales que componen la placa y las conexiones entre éstos.

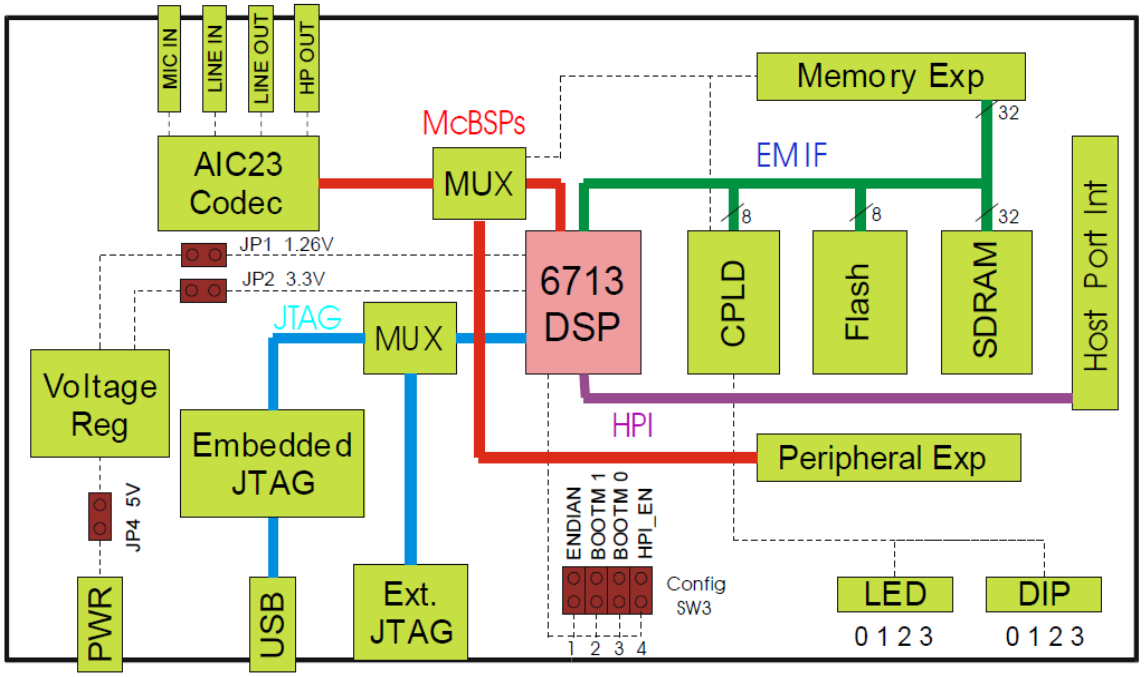

Figura 5.2. Diagrama de bloques del DSK.

5.1.2. FPC1010 Fingerprint Sensor Daughter Card

Es una tarjeta hermana del DSK mencionado anteriormente, desarrollada por Spectrum Digital, diseñada para uso en aplicaciones de reconocimiento de huellas dactilares.

La tarjeta consta de un sensor capacitivo para huellas de $200x152$ píxeles. Además incluye una serie de leds. Uno de ellos indica si la tarjeta hermana está siendo alimentada correctamente, otro si se ha reiniciado la placa, y los dos leds restantes pueden ser configurados por el usuario.

Esta tarjeta dispone de una interfaz SPI para la transferencia de la imagen capturada al DSK. Para ello va conectada en los puertos de expansión de memoria y de expansión de periféricos del DSK.

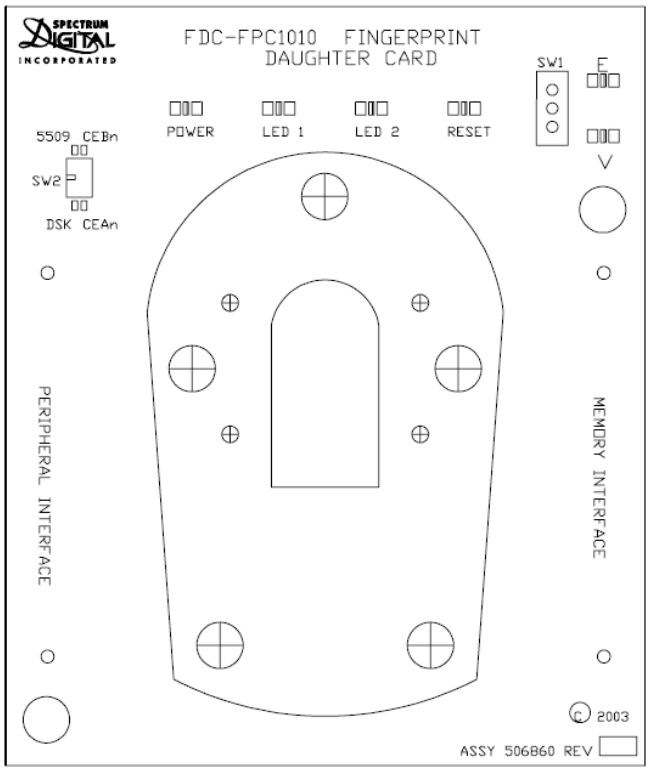

Figura 5.3. Vista de la tarjeta hermana.

5.1.3. Equipo PC

Debido a las especificaciones del DSP empleado, se requiere el uso de un PC con sistema operativo Windows XP o bien Windows 2000, pues por limitaciones del software desarrollado por TI, el sistema no es compatible con sistemas basados en Windows Vista o superiores. El PC empleado se utilizará para la programación del código necesario en la aplicación Code Composer Studio 3.0, incluida en el Starter Kit. Por tanto se podrá emplear cualquier tipo de PC que tenga capacidad para utilizar los sistemas operativos mencionados. En este caso que nos ocupa, utilizaremos un PC provisto de Windows XP.

5.1.4. Code Composer Studio 3.0

Code Composer Studio (CCS) provee un ambiente de desarrollo integral (IDE) para aplicaciones en tiempo real de señales basado en lenguaje de programación C y ensamblador. Este incorpora un compilador, ensamblador y un linker o enlazador. Tiene posibilidades gráficas y soporta depuración en tiempo real. Mediante la compilación del código .c se genera un código .asm que posteriormente es ensamblado en un objeto de lenguaje máquina .obj. Este luego es combinado por el "linker" tanto con librerías como con las entradas, para producir un fichero ejecutable de extensión .out.

Este fichero puede ser cargado y ejecutado directamente en el DSP. Un proyecto de CCS comprende todos los ficheros (o enlaces a todos los ficheros) requeridos para generar un archivo ejecutable por el DSP. Una variedad de opciones habilita que archivos de diferentes tipos sean añadidos o eliminados de un proyecto previamente proporcionado.

5.1.5. Interacción entre elementos del sistema

El IDE CCS 3.0 será instalado en el equipo PC, lo cual nos permitirá conectar el DSK con éste mediante su interfaz USB con el objetivo de poder cargar el código compilado de nuestro algoritmo en el DSK y poder darle instrucciones a través del PC en tiempo de ejecución.

Por otra parte el DSK se encargará de la gestión de la tarjeta hermana para la adquisición de huellas a partir de ella.

Para hacer más clara la forma en que interactúa cada elemento con los demás se representarán dichas relaciones en un diagrama, el cual se observamos en la figura 5.4.

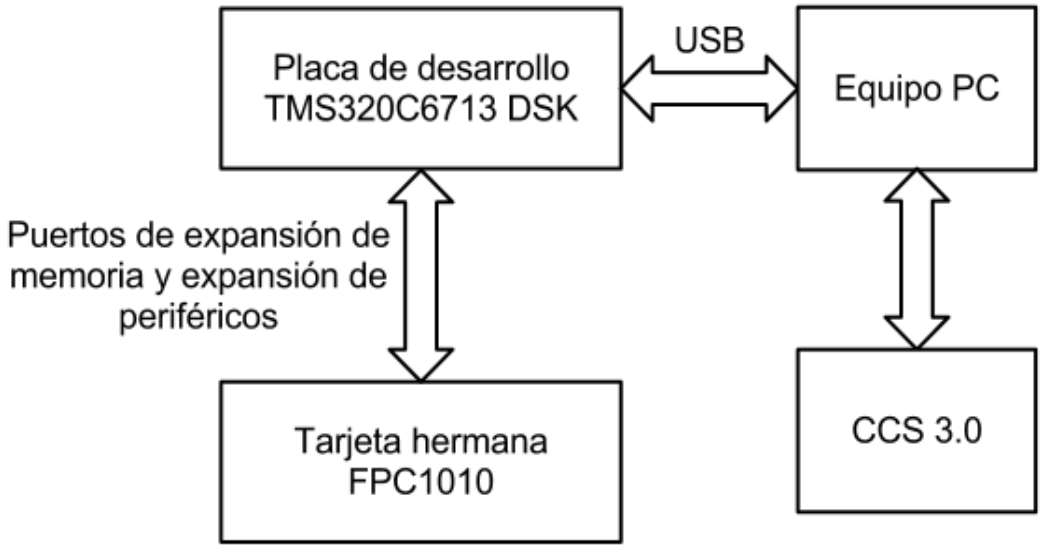

Figura 5.4. Diagrama de interacción entre elementos.

5.2. Especificaciones del algoritmo a desarrollar e implementación

Tal y como hemos visto en el capítulo 2, existen varios tipos de algoritmos capaces de realizar la comparación de dos huellas dactilares. La elección del tipo de algoritmo a desarrollar se ha visto influenciada por las capacidades que poseen los DSPs, el cual estará basado en correlación de imágenes. Este tipo de algoritmo hace un uso bastante significativo de la operación MAC, con la que los DSPs se desenvuelven con soltura debido al hardware específico del que disponen.

En cuanto a su implementación, se diseñará un sistema con base de datos para el almacenamiento de la huella de varios usuarios para posteriormente poder ser identificados por medio del algoritmo de comparación desarrollado. El sistema será capaz de realizar dos operaciones principales:

- Registro de un usuario en la base de datos.
- Verificación de la identidad de un usuario.

Debido al coste computacional que presenta este tipo de algoritmo nuestro sistema se centrará en la verificación de la identidad de un usuario y no en su identificación, por lo que nuestro sistema será del tipo AFAS.

Capítulo 6. Viabilidad del proyecto

6.1. Viabilidad tecnológica

La decantación por el uso de un DSP a la hora de implementar el algoritmo parece bastante acertada debido a que su tipo es el de correlación de imágenes, el cual tal y como comentábamos anteriormente conlleva una alta carga computacional.

La mayoría de operaciones realizadas en el algoritmo están basadas en multiplicación y suma, siendo el DSP capaz de procesar varias de estas operaciones al mismo tiempo, es decir, en paralelo. Esto es debido al hardware del que dispone el dispositivo en cuestión, especializado en este tipo de operaciones, lo que reduce enormemente el tiempo de ejecución del algoritmo.

Otro punto a favor es la existencia de tarjetas hermanas que permiten ampliar la versatilidad del DSP. En este caso el sensor capacitivo nos permite tener un sistema de reconocimiento compacto y de poco tamaño y peso.

Por otra parte el kit de desarrollo empleado junto con la tarjeta hermana se encuentran disponibles en el mercado nacional e internacional, pudiéndose adquirir los mismos directamente de sus fabricantes o a través de sus distribuidoras o representantes comerciales.

Unido a ello, el propio kit de desarrollo incluye las herramientas software necesarias para llevar a cabo el desarrollo del código y realizar la comunicación entre el PC y éste.

Finalmente podemos concluir que los DSPs son muy adecuados desde el punto de vista técnico para el desarrollo de la aplicación propuesta en el presente proyecto.

6.2. Viabilidad económica

El uso del DSK elegido se ha impuesto debido al gran número de operaciones necesarias a la hora de procesar imágenes, por lo que se ha optado por una de las familias de DSPs más potentes del mercado en la actualidad.

El precio del DSK resulta bastante elevado en comparación con otros sistemas basados en Procesadores de Propósito General (GPPs), pero debido a las características del algoritmo a desarrollar se hace inevitable el uso de un DSP en lugar de un GPP. Este DSK utiliza el DSP más potente de la empresa Texas Instruments, capaz de procesar operaciones en coma flotante, por lo que se optó por la adquisición del mismo en lugar de otros de una gama inferior, dado que el procesamiento de imágenes resulta muy costoso computacionalmente.

La tarjeta hermana que integra el sensor de huellas también tiene un precio relativamente alto, pero permite una fácil integración con el DSK e incluye los drivers necesarios para la comunicación con éste, por lo que se optó por la adquisición de la misma.

En cuanto al software empleado para el desarrollo del prototipo, viene incluido con el propio DSK, por lo que no supone un coste añadido.

Por todo ello, el punto de vista económico no es problema relevante para el desarrollo de nuestro prototipo.

6.3. Viabilidad legal

El proyectista posee las licencias de las herramientas software necesarias para el desarrollo de los programas del presente proyecto.
Legalmente no se necesita pagar por el uso de ninguna patente y no se incumple ninguna ley al usar o desarrollar este prototipo.

Capítulo 7. Algoritmo de comparación de huellas dactilares basado en correlación

7.1. Introducción

Las técnicas más utilizadas para comparar huellas dactilares se basan en la extracción y localización de minucias. Estas técnicas, que imitan a la comparación de huellas realizada manualmente, se basan en determinar los puntos singulares de las crestas de las huellas y comparar su localización. Las etapas de extracción y comparación de minucias son etapas complejas, ya que es necesario tener en cuenta, entre otros muchos efectos, la distorsión no lineal que provoca la elasticidad de la piel, el ruido introducido por el sensor, el estado de la piel, etc. Por otra parte, no es trivial determinar un punto de referencia en ambas huellas, lo cual es necesario para alinear las huellas, que pueden estar rotadas y desplazadas entre sí.

Sin embargo, las técnicas de comparación basadas en correlación no requieren extracción de características de las huellas dactilares a comparar. Estas técnicas utilizan el cálculo de la correlación para determinar la similitud entre dos imágenes. La principal desventaja que presentan frente a otros métodos es la elevada carga computacional que conlleva el cálculo de la correlación.

Este proyecto se ha orientado a la utilización de técnicas basadas en correlación de imágenes frente a las basadas en minucias, que son técnicas más maduras y sobre las que existen numerosos trabajos de investigación.

Utilizando las técnicas de correlación se reduce el preprocesado al que se somete la imagen. Esto implica dos consecuencias importantes: la imagen a comparar no incluye artefactos derivados del preprocesado y se reduce la carga computacional de esta etapa. Esto lleva a un planteamiento desde el punto de vista teórico diametralmente opuesto a las técnicas basadas en minucias.

En las técnicas basadas en minucias es necesario un fuerte preprocesado para modelar una huella real a imagen y semejanza de una “huella ideal”, pasando por numerosos filtrados que hacen aparecer artefactos y que en muchos casos generan crestas y minucias falsas incrementando con ello los errores. Con las técnicas basadas en correlación no se busca el filtrado respecto a una “huella ideal”, se elimina ese filtrado artificial para trasladar esa carga computacional a la etapa de comparación. En realidad el cálculo de la correlación se asemeja a un filtrado, pero sustituyendo los coeficientes de los filtros ideales por la imagen con la que se quiere comparar. Es por esto que se puede considerar como un “filtrado real” entre las dos huellas que se van a comparar. Este cambio conceptual además aporta una riqueza a la comparación en sí misma, ya que no se busca la correspondencia de una serie de puntos extraídos a partir de una huella modificada mediante el preprocesado, sino que se busca una correspondencia global de toda la información contenida en la huella original sin necesidad de preprocesar. Aplicando estas técnicas es posible comparar todas las características presentes en las imágenes sin necesidad de extraerlas ni compararlas una a una como hacen la mayoría de los algoritmos en este campo.

El algoritmo propuesto consta de cuatro etapas principales: preprocesado, alineamiento, transformación a coordenadas polares y comparación. En la etapa de prepocesado se elimina un poco de ruido de la imagen a tratar debido a su baja calidad. La etapa de alineamiento se realiza mediante la correlación de las imágenes y su objetivo es la extracción de la zona común de ambas huellas. Después de esto se realiza una transformación a coordenadas polares en ambas imágenes para suavizar el efecto de la rotación entre éstas a la hora de realizarles la correlación. Y por último se vuelve a hacer la correlación a lo obtenido en la etapa anterior para dar un veredicto final al comparar el resultado obtenido con un umbral preestablecido experimentalmente.

7.2. Nociones teóricas

En este apartado haremos mención a varios conceptos teóricos que serán utilizados para la elaboración del algoritmo de comparación de huellas dactilares.

7.2.1. Correlación de imágenes

Disponemos de una imagen patrón $p(n,m)$ y una imagen de entrada $e(n,m)$, ambas de tamaño NxM, donde n y m representan la posición de los píxeles y el valor característico de cada píxel es la intensidad luminosa del mismo.

La correlación de dos señales $p(n,m)$ y $e(n,m)$ se calcula mediante la fórmula 7.1.

$$c_{pe}(x,y) = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} p(n,m)\cdot e(x-n,y-m)$$

Fórmula 7.1. Correlación cruzada.

Otra forma de calcularla es usando el dominio frecuencial en vez del dominio espacial, que es lo que haremos en este proyecto. Para ello es necesario transformar al dominio de Fourier ambas imágenes.

A partir de dichas imágenes podemos obtener la transformada discreta de Fourier en 2D (DFT 2D, del inglés Discrete Fourier Transform) de cada una, a las que llamaremos $P(k,l)$ y $E(k,l)$ respectivamente. El cálculo de de la DFT 2D de una señal $x(n,m)$, de tamaño NxM, se realiza mediante la fórmula 7.2.

$$X(k,l) = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x(n,m)\cdot e^{-j\frac{2\pi}{N}kn}\cdot e^{-j\frac{2\pi}{M}lm} = A_x(k,l)\cdot e^{j\theta_x(k,l)}$$

Fórmula 7.2. Transformada discreta de Fourier en 2D.

Donde $X(k,l)$ será de tamaño NxM también y el resultado podrá estar compuesto por números complejos independientemente de que la señal de entrada esté compuesta solo por números reales. $A_x(k,l)$ es el módulo del par complejo de la posición (k,l) y $\theta_x(k,l)$ es la fase de dicha posición.

En el dominio de Fourier la correlación de dos imágenes se traduce en un simple producto, y se calcula mediante la fórmula 7.3.

$$C_{PE}(k,l) = P(k,l) \cdot \overline{E}(k,l) = A_P(k,l) \cdot A_E(k,l) \cdot e^{j\theta(k,l)}$$

Fórmula 7.3. Correlación en el dominio de Fourier.

Donde $\overline{E}(k,l)$ es el complejo conjugado de $E(k,l)$, y $\theta(k,l)$ es la diferencia de fase entre las imágenes.

Una vez obtenido $C_{PE}(k,l)$ se le aplica la transformada inversa (IDFT 2D, del inglés Inverse Discrete Fourier Transform) y obtenemos la correlación de las imágenes en el dominio espacial $c_{pe}(n,m)$. El cálculo de la IDFT 2D de una señal $X(k,l)$, de tamaño NxM, se realiza mediante la fórmula 7.4.

$$x(n,m) = \frac{1}{N \cdot M} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} X(k,l) \cdot e^{j\frac{2\pi}{N}kn} \cdot e^{j\frac{2\pi}{M}lm}$$

Fórmula 7.4. Transformada discreta inversa de Fourier en 2D.

Donde $x(n,m)$ será de tamaño NxM también y el resultado puede estar formado por números complejos.

En este proyecto usaremos una variante de la correlación. Se denomina función POC (del inglés Phase Only Correlation) y su cálculo para dos señales $P(k,l)$ y $E(k,l)$ en el dominio de Fourier se realiza mediante la fórmula 7.5.

$$POC_{PE}(k,l) = \frac{P(k,l) \cdot \overline{E}(k,l)}{\left\| P(k,l) \cdot \overline{E}(k,l) \right\|} = e^{j\theta(k,l)}$$

Fórmula 7.5. Función POC en el dominio de Fourier.

A partir de $POC_{PE}(k,l)$ podemos obtener su homólogo en el dominio espacial $poc_{pe}(n,m)$ aplicándole la IDFT 2D tal como se vio antes con la correlación ordinaria.

7.2.2. Propiedades de la función POC

Según lo anterior tenemos que aplicando la función POC a dos imágenes obtenemos el resultado mostrado en la fórmula 7.6.

$$poc_{pe}(n,m) = \frac{1}{N \cdot M} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} POC_{PE}(k,l) \cdot e^{j\frac{2\pi}{N}kn} \cdot e^{j\frac{2\pi}{M}lm}$$

Fórmula 7.6. Función POC en el dominio espacial.

Si en la ecuación anterior las dos imágenes resultan ser iguales obtenemos lo siguiente:

$$POC_{PP}(k,l) = \frac{A_P(k,l) \cdot A_P(k,l) \cdot e^{j0}}{A_P(k,l) \cdot A_P(k,l)} = 1$$

$$poc_{pp}(n,m) = \frac{1}{N \cdot M} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} 1 \cdot e^{j\frac{2\pi}{N}kn} \cdot e^{j\frac{2\pi}{M}lm} = \delta(n,m) = \begin{cases} 1 \; si \; n = m = 0 \\ 0 \; en \; otro \; caso \end{cases}$$

Ecuación 7.7. Función POC de dos imágenes iguales.

Donde $\delta(n,m)$ es la función delta en dos dimensiones.

Lo mostrado anteriormente deja ver que esta función hará una mejor discriminación a la hora de comparar dos señales. Veamos el siguiente ejemplo en el que comparamos dos imágenes de tamaño $128x128$, ambas mostradas en la figura 7.1.

Figura 7.1. Imágenes de tamaño $128x128$ en escala de grises.

En la figura 7.2 podemos observar el resultado de aplicar la correlación ordinaria y la función POC respectivamente a la primera imagen de la figura 7.2, mostrando la primera un pico alto en la posición $(0,0)$ y bastante información adicional que puede crear confusión, mientras que la segunda muestra un pico de valor 1 en la posición $(0,0)$ y todos las demás posiciones con valor 0.

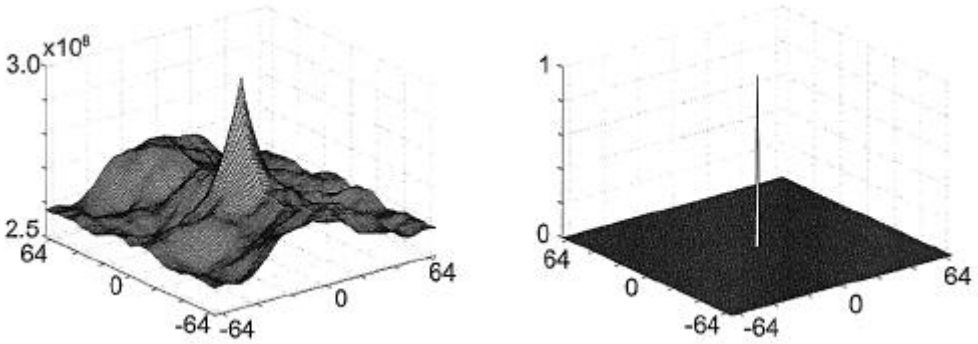

Figura 7.2. Correlación de una misma imagen.

En la figura 7.3 podemos ver el resultado de aplicar la correlación ordinaria y la función POC respectivamente a las dos imágenes de la figura 7.1, observando que la función POC ofrece una discriminación mucho mayor que la correlación ordinaria.

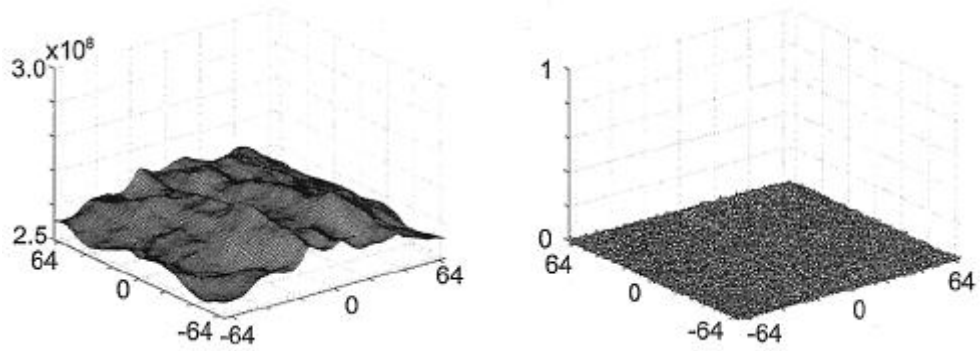

Figura 7.3. Correlación de las dos imágenes de la figura 7.1.

Una nota importante en el uso de esta función es que permite establecer un umbral fijo sin importar que las imágenes varíen de intensidad ya que normalizamos el valor.

A continuación se expondrán otras propiedades interesantes de la función POC.

- Propiedad de invarianza a desplazamientos:

Tenemos la imagen $p1(n,m)$ como la imagen $p(n,m)$ desplazada $d1$ filas y $d2$ columnas, cuya representación matemática se muestra en la fórmula 7.8.

$$p1(n,m) = p(n+d1, m+d2)$$

Fórmula 7.8. Imagen desplazada.

Si aplicamos la función POC a ambas imágenes obtenemos el mismo resultado que si lo hubiésemos hecho a dos imágenes idénticas con la única diferencia de que el resultado estará desplazado $d1$ filas y $d2$ columnas respecto al obtenido de imágenes idénticas. Atendiendo a esta propiedad podemos obtener el desplazamiento entre dos imágenes localizando la posición del pico más alto del resultado de la función POC.

- Propiedad de invarianza a los cambios de intensidad:

Tenemos la imagen $p2(n,m)$, la cual es similar a la imagen $p(n,m)$ con la diferencia de que tiene otro nivel de intensidad. Su representación matemática se muestra en la fórmula 7.9.

$$p2(n,m) = \alpha \cdot p(n,m)$$

Fórmula 7.9. Imagen con intensidad escalada.

Aplicando la función POC a ambas imágenes se obtiene lo siguiente:

$$POC_{P2P}(k,l) = \frac{\alpha \cdot A_P(k,l) \cdot A_P(k,l) \cdot e^{j0}}{\alpha \cdot A_P(k,l) \cdot A_P(k,l)} = 1 = POC_{PP}(k,l)$$

Fórmula 7.10. Resultado de la función POC de dos imágenes con variación en el nivel de intensidad.

Como observamos se obtiene el mismo resultado que si las imágenes son exactamente iguales. Con esta propiedad se elimina el problema de la presión ejercida por el dedo sobre el sensor y su consecuente cambio en el nivel de intensidad de la imagen.

- Propiedad de inmunidad al ruido:

Cuando aplicamos la función POC a dos imágenes idénticas obtenemos la función delta, la cual representa un pico en la posición $(0,0)$ de valor 1. Si cualquiera de las imágenes se ve afectada por una adición de ruido aleatorio, el resultado se verá afectado por un decremento en el valor del pico pero se podrá seguir distinguiendo una señal con la misma forma que en el caso sin ruido.

Esto hace que esta función sea bastante inmune a la adición de ruido aleatorio.

7.2.3. La transformada rápida de Fourier

Sin un algoritmo eficiente que calcule la DFT, no sería posible usar la transformada de Fourier en el tratamiento de la imagen. La aplicación directa de la expresión de la DFT resultaría tediosa. Cada punto en la imagen transformada (suponemos que es cuadrada) necesita N^2 multiplicaciones complejas y $N^2 - 1$ sumas complejas (sin contar el cálculo de las funciones base - senos y cosenos -). En total, necesitamos N^4 multiplicaciones complejas y $N^2 \cdot (N^2 - 1)$ sumas complejas.

Teniendo en cuenta únicamente las multiplicaciones, un PC que realizase 40000 multiplicaciones reales por segundo necesitaría aproximadamente dos meses para transformar una simple imagen de $512x512$ pixeles. Incluso un superordenador con una potencia operacional de 1000 MFLOPS (millones de operaciones en coma flotante por segundo) tardaría unos tres minutos. Estas cifras hacen urgente la minimización del número de operaciones que se necesitan en el tratamiento de la imagen mediante la elección de un algoritmo adecuado.

La transformada rápida de Fourier (FFT, del inglés Fast Fourier Transform) es un algoritmo que reduce el número de operaciones necesarias para obtener la DFT. Hay diversas variantes del algoritmo, pero el que usaremos será el algoritmo de la FFT con diezmado en el tiempo de base 2, el cual explicaremos a continuación.

Una DFT 2D puede separarse en dos pasos computacionales, cada uno de los cuales usa la DFT unidimensional. En el primero se le hace la DFT a cada fila y en el segundo se le aplica a cada columna. Además, una IDFT 2D puede considerarse como la DFT 2D del complejo conjugado de la señal. Por este motivo, el núcleo o la base de los cálculos de la DFT 2D y la IDFT 2D es la DFT, y en consecuencia la FFT.

La expresión de la DFT unidimensional es la de la fórmula 7.11.

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot {W_N}^{kn}$$

Fórmula 7.11. Transformada discreta de Fourier unidimensional.

Donde $W_N = e^{-j\frac{2\pi}{N}}$ se introduce para facilitar el análisis.

Para aplicar la DFT se necesitan N multiplicaciones complejas y N sumas complejas para cada cada punto, o lo que es lo mismo, un total de $2N^2$ operaciones aritméticas. Cuando N es grande, el número de operaciones implicadas en la transformada es enormemente grande. De ahí que la computación debe simplificarse para hacer práctica la técnica de la transformación. Esto fue posible desde que surgió la FFT en 1965. El principio fundamental del algoritmo FFT se basa en la descomposición del cálculo de la DFT de una secuencia de longitud N en DFT's cada vez más pequeñas. El mayor inconveniente de este método es que el número N debe ser una potencia de dos. Así que consideraremos $N = 2^p$ siendo p un número entero.

La fórmula 7.11 se puede dividir en muestras pares e impares como muestra la fórmula 7.12.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot {W_N}^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot {W_N}^{(2n+1)k}$$

Fórmula 7.12. Transformada discreta de Fourier dividida en muestras pares e impares.

O de esta otra forma:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot {W_{N/2}}^{kn} + {W_N}^{k} \cdot \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot {W_{N/2}}^{kn}$$

Fórmula 7.13. Transformada discreta de Fourier dividida en muestras pares e impares 2.

Donde ${W_N}^2 = (e^{-j\frac{2\pi}{N}})^2 = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$.

Así podemos decir que la DFT se puede dividir en 2 DFT's de $N/2$ puntos cada una. Lógicamente, esto reduce el tiempo de cálculo. De la misma manera, puede deducirse que una DFT de $N/2$ puntos puede obtenerse calculando dos DFT's de $N/4$ puntos, y así sucesivamente, para cualquier N que sea una potencia entera de dos, hasta que la DFT de N puntos se convierte en una DFT de 2 puntos.

Aplicando este algoritmo de sucesivas divisiones, el número total de operaciones complejas cambia de N^2, a $N \cdot \log_2 N$.

A continuación veremos la aplicación del algoritmo a una señal de 8 puntos. Para conservar la estructura de este algoritmo, las entradas al bloque DFT deben disponerse en el orden requerido para las sucesivas etapas. Para el cálculo de la FFT de una función de 8 puntos {*f*(0), *f*(1), ... , *f*(7)}, las entradas con argumentos pares [*f*(0), *f*(2), *f*(4), *f*(6)] se usan para la DFT de *N*/2 puntos superior (DFT de 4 puntos en este caso), mientras que las entradas con argumentos impares [*f*(1), *f*(3), *f*(5), *f*(7)] se usan para la DFT de 4 puntos inferior.

Cada DFT de 4 puntos se calcula como DFT's de 2 puntos. Hemos de dividir el primer conjunto de entradas en su parte par {*f*(0), *f*(4)} e impar {*f*(2), *f*(6)}. Del mismo modo, dividimos el segundo conjunto de entradas en su parte par {*f*(1), *f*(5)} e impar {*f*(3), *f*(7)}. Es decir, las entradas han de disponerse según este orden: {*f*(0), *f*(4), *f*(2), *f*(6), *f*(1), *f*(5), *f*(3), *f*(7)}. En la figura 7.4 se observan las agrupaciones de estas entradas.

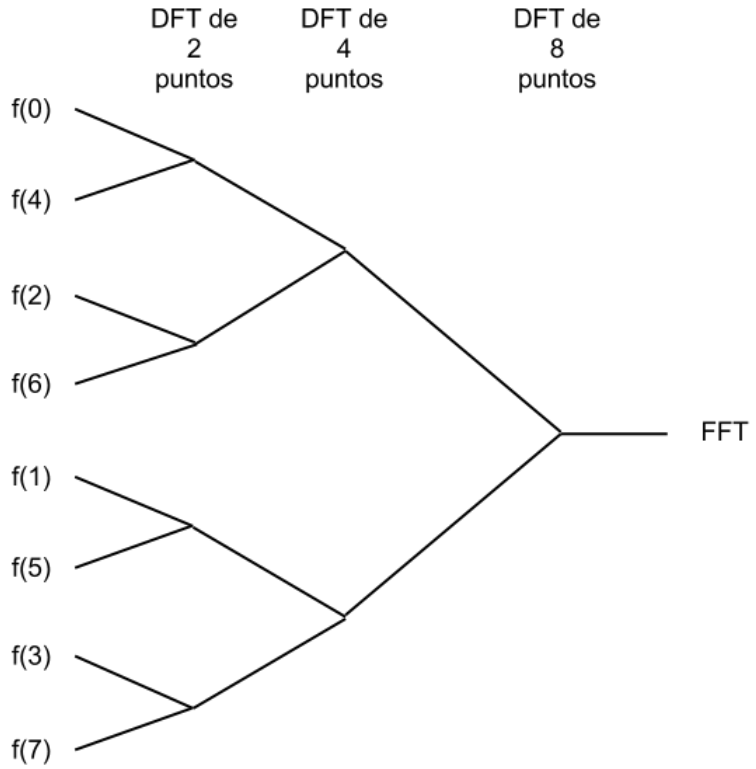

Figura 7.4. Reordenación de las entradas.

Se puede observar que la entrada y salida están relacionadas con una inversión de bits.

Se puede comprobar que ${W_N}^{r+\frac{N}{2}} = -{W_N}^{r}$ por lo que el número de multiplicaciones complejas necesarias para el algoritmo FFT se reduciría en un factor 2. De ahí que el número total de operaciones complejas necesarias para la DFT unidimensional será $(\frac{N}{2}) \cdot \log_2 N$. En la figura 7.5 se muestran la simetría y periodicidad de estos coeficientes, a los cuales se les denomina coeficientes twiddle.

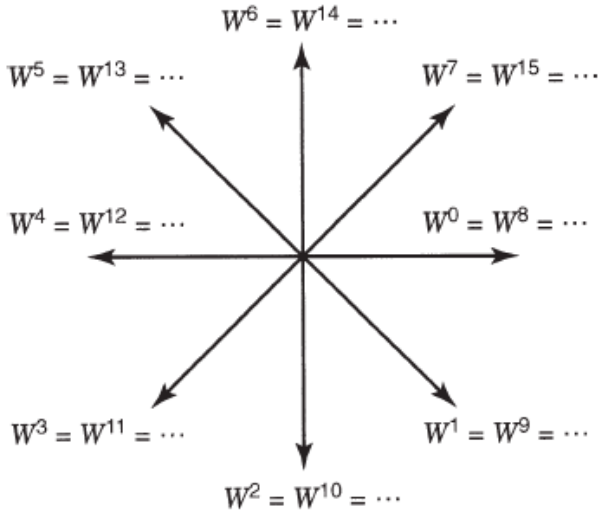

Figura 7.5. Periodicidad y simetría de los coeficientes twiddle.

En la figura 7.6 se muestra el diagrama de flujo resultante de aplicar el algoritmo a una señal con 8 muestras.

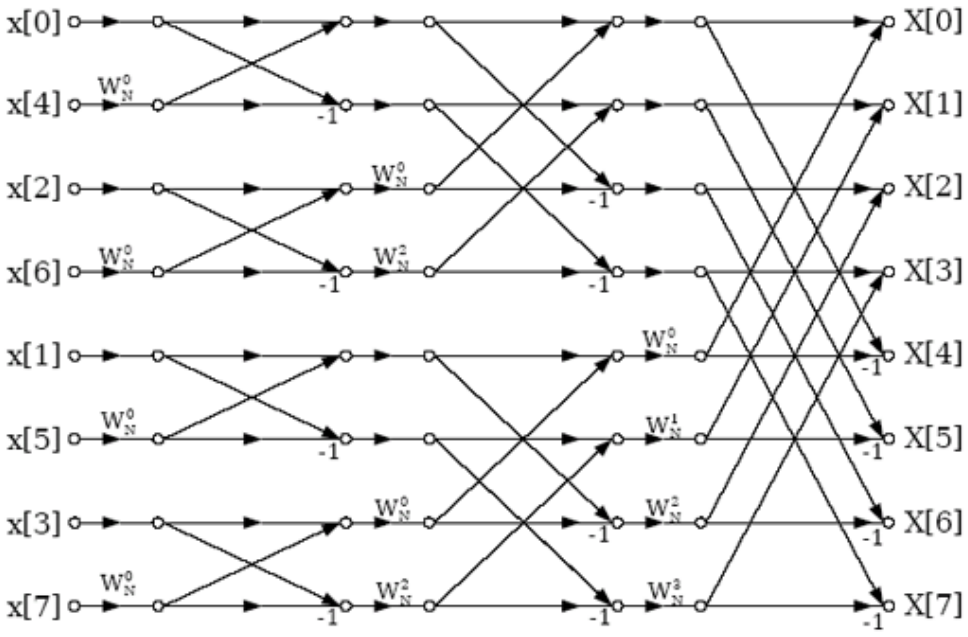

Figura 7.6. Aplicación de la FFT a una señal de 8 muestras.

Cada una de las estructuras elementales que se dan en la implementación del algoritmo FFT se denomina butterfly o mariposa.

En la figura 7.6 vemos que durante la primera etapa aparecen 4 mariposas, que están separadas. Durante la segunda etapa, hay 4 mariposas "solapadas" dos a dos. Y en la tercera y última etapa, las 4 mariposas aparecen solapadas.

Para una imagen (función discreta bidimensional) de dimensión NxN, se necesitan $(\frac{N}{2}) \cdot \log_2 N$ operaciones para un valor de k, y por tanto $(\frac{N}{2}) \cdot N \cdot \log_2 N$ operaciones para N valores de k. Por el mismo razonamiento, necesitamos $(\frac{N}{2}) \cdot N \cdot \log_2 N$ operaciones para N valores de l. De esta manera, el número total de operaciones complejas será $N^2 \cdot \log_2 N$, siendo mucho menor que si se evalúa directamente la DFT bidimensional, con la que se necesitan $N^2 \cdot N^2 = N^4$ operaciones complejas.

7.2.4. Simetría de la DFT 2D

Una propiedad de la que sacaremos provecho de esta transformada es la de simetría conjugada. Esto es, si tenemos una imagen $p(n,m)$ de tamaño NxM, aplicándole la DFT 2D obtenemos $P(k,l)$. Para la que se cumple lo siguiente:

$$P(k,l) = \bar{P}(N-k, M-l)$$

Fórmula 7.14. Simetría DFT 2D.

Usando esta propiedad en cualquier operación en el dominio de Fourier podemos reducir a la mitad el número de operaciones a realizar necesarias, ya que calculando la operación de un punto de la DFT 2D, tenemos automáticamente el de su punto simétrico.

7.3. Algoritmo desarrollado

El algoritmo de comparación propuesto consta de cuatro etapas principales: preprocesado, alineamiento, transformación a coordenadas polares y comparación.

Cada etapa del algoritmo realiza las siguientes funciones:

- Preprocesado: realiza una expansión del contraste de la imagen para diferenciar mejor las crestas de los valles mejorando la imagen para etapas posteriores. Además realiza una inversión en el valor de la intensidad para que las crestas contengan los valores más altos debido a que son las que aportan la información de la huella en la captura de ésta.
- Alineamiento: determina el desplazamiento entre huellas mediante el cálculo de la función POC (propiedad de invarianza al desplazamiento) y obtiene la zona común de ambas huellas.
- Transformación a coordenadas polares: una vez obtenida la zona común, ésta aun se ve influenciada por la rotación entre imágenes. Transformando a coordenadas polares esta zona hacemos la imagen inmune a pequeñas rotaciones.
- Comparación: realiza la función POC a lo obtenido en la etapa anterior, compara el resultado con un umbral preestablecido y da un veredicto final.

En la figura 7.7 se muestra el diagrama de flujo del algoritmo de comparación propuesto.

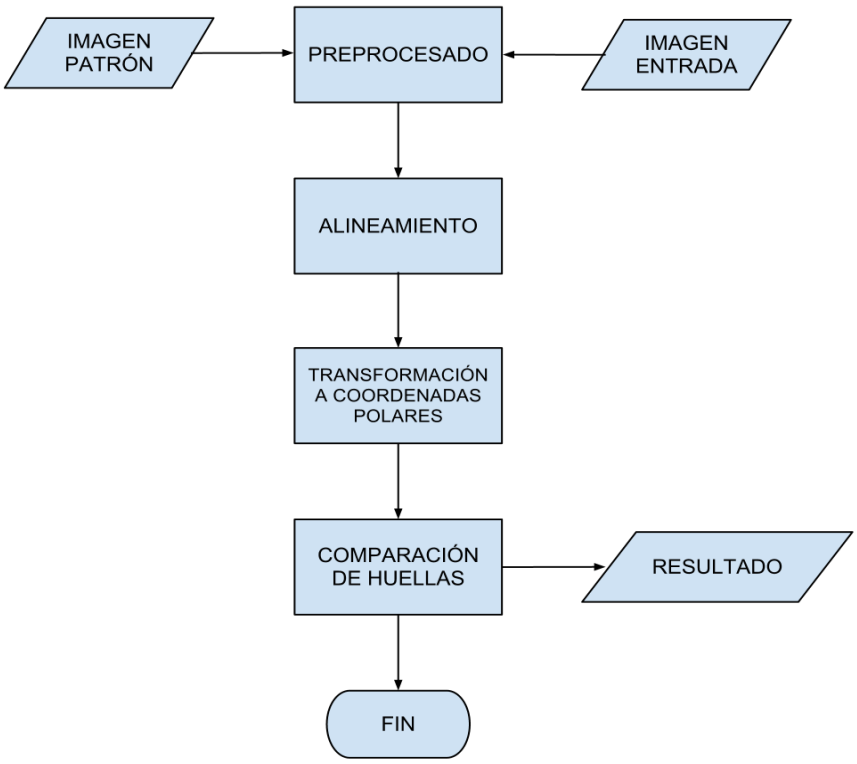

Figura 7.7. Diagrama de flujo del algoritmo de comparación de huellas.

7.3.1. Preprocesado

Los algoritmos de comparación basados en técnicas de correlación pueden utilizar las imágenes directamente sin realizar ningún paso previo de extracción de características. La extracción de características es un proceso costoso a nivel computacional y que lleva asociado un fuerte preprocesado. Por ejemplo para la extracción de minucias un preprocesado típico requiere las siguientes etapas: segmentación, binarización, adelgazamiento, cálculo del campo de orientación, etc.

Aunque en este caso se evite la extracción de características, es necesario garantizar que la calidad de la imagen es lo suficientemente buena para realizar el cálculo de la correlación. Esto se consigue gracias a un preprocesado ligero.

Las imágenes con las que trabajaremos tendrán un tamaño de $200x152$ píxeles y estarán en escala de grises. Cada píxel será almacenado en un bloque de memoria de 8 bits por lo que el valor de su intensidad tendrá un valor comprendido entre 0 y 255. Un valor de 0 corresponde al negro en la imagen y un valor de 255 corresponde al blanco en la imagen. En la figura 7.8 podemos observar una huella obtenida por el sensor capacitivo.

Figura 7.8. Huella proporcionada por el sensor.

En el preprocesado utilizado hemos extendido el contraste de la imagen usando para ello el histograma de la misma. Dicho histograma es una medición de las veces que se repite un cierto valor de intensidad en la imagen y se calcula mediante la fórmula 7.15.

$$h(i) = \frac{n_i}{N}$$

Fórmula 7.1. Histograma de una imagen.

Donde $h(i)$ es el valor obtenido para el valor de intensidad i, n_i es el número de píxeles en la imagen con valor de intensidad i y N es el número total de píxeles en la imagen. En la figura 7.9 podemos ver el histograma de la huella de la figura 7.8. Hay que tener en cuenta que el valor alcanzado por el histograma en la figura no ha sido dividido entre el número total de píxeles, sino que ha sido representado n_i directamente. Es algo irrelevante, ya que si se hubiese representado el histograma real la imagen sería igual, solo que los valores estarían comprendidos entre 0 y 1.

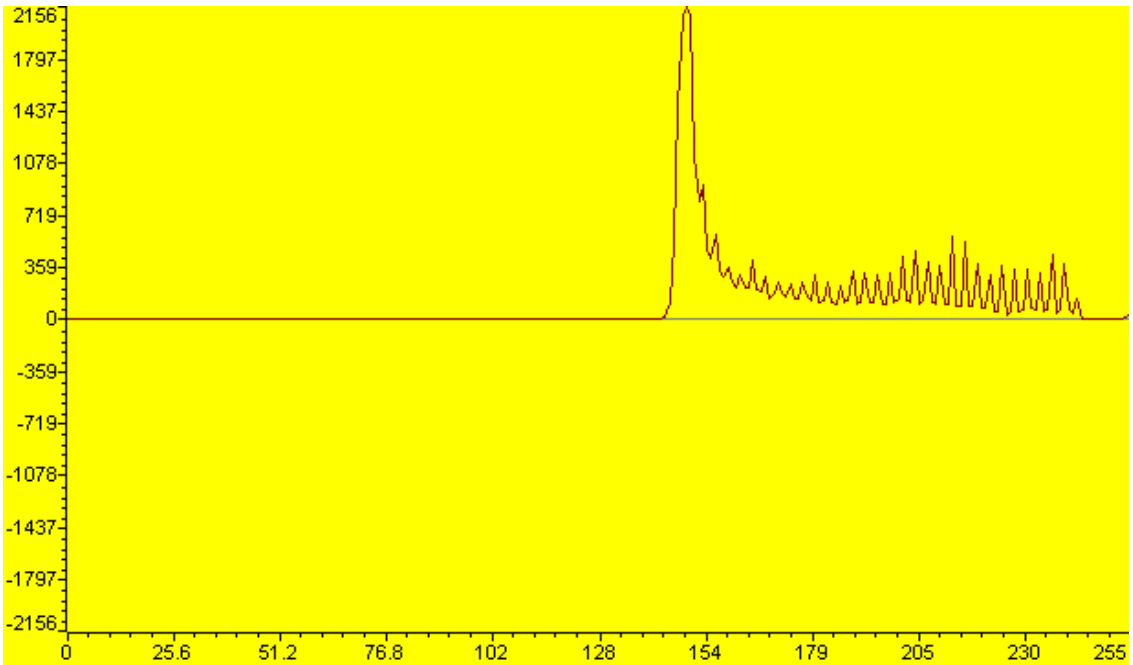

Figura 7.9. Histograma de la huella de la figura 7.15.

Observando el histograma nos percatamos de que la información de la huella se encuentra en un pequeño rango de valores de intensidad. Por lo tanto haremos una expansión del contraste de la imagen para que la información relevante del histograma ocupe todos los valores posibles de intensidad. Además invertiremos la escala de grises de la imagen para que los valores más altos de intensidad correspondan a las crestas, ya que éstas son las que aportan la información al sensor mediante la presión que ejercen sobre éste. En la figura 7.10 podemos ver la huella completamente preprocesada con las transformaciones antes mencionadas y en la figura 7.11 su histograma.

Figura 7.10. Huella preprocesada.

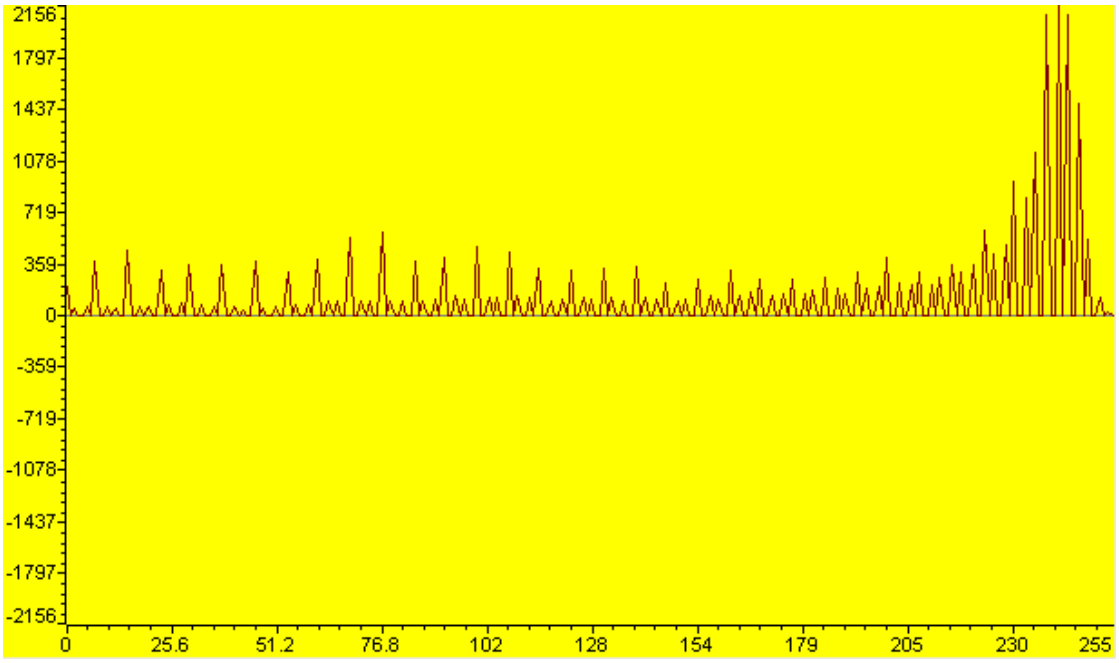

Figura 7.11. Histograma de huella preprocesada.

Podemos observar en la figura 7.11 una mejor distribución de la información en la huella preprocesada que en la original, que permite una mejor visualización de la misma debida la expansión del histograma, y una mejor diferenciación entre crestas y valles. Los valles no aportaban mucha información por si solos ya que se confunden con las zonas donde no hay presencia de la huella, por ello la inversión en la escala de

grises aporta una mejor representación de la información, ya que son las crestas las que la aportan en mayor medida.

7.3.2. Alineamiento

El objetivo de esta etapa es estimar el desplazamiento relativo entre el patrón y la imagen de entrada. Para ello haremos uso de la función POC y su propiedad de invarianza a traslaciones.

No trataremos el cálculo de la rotación entre huellas, debido a que no resulta necesario ya que el sensor utilizado tiene un perfil para apoyar el dedo que reduce dicha rotación, no obstante se debe procurar apoyar el dedo lo más afín al perfil del sensor. En la siguiente etapa a ésta usaremos un método para paliar el efecto de la pequeña rotación que pueda existir.

En nuestro algoritmo usaremos la FFT 2D para calcular la transformada de Fourier, por lo que nos vemos obligados a usar un tamaño de filas y columnas que sean potencias de base 2. Nuestra imagen es de $200x152$ píxeles, por lo que tenemos dos opciones: o bien pasamos esta imagen a una matriz de $256x256$ rellenando con ceros las posiciones en las que no se encuentra la imagen; o bien cogemos una ventana de la imagen de $128x128$ píxeles del centro de la imagen.

La solución adoptada es la segunda. Al realizar esta operación perdemos parte de la información de la huella, pero por otra parte se reduce bastante el tiempo empleado en el cálculo de las operaciones a realizar respecto a la primera solución.

Partiendo de la zona central de las imágenes preprocesadas $p(n,m)$ y $e(n,m)$ obtenemos sus respectivas transformadas $P(k,l)$ y $E(k,l)$. A dichas transformadas le aplicamos la función POC y a partir de ella obtenemos su homólogo en el dominio espacial $poc_{pe}(n,m)$ aplicandole la transformada inversa.

A partir de $poc_{pe}(n,m)$ obtenemos la situación del pico más alto, el cual indicará el desplazamiento entre huellas. En la figura 7.12 se muestra el desplazamiento entre dos huellas obtenido por este método.

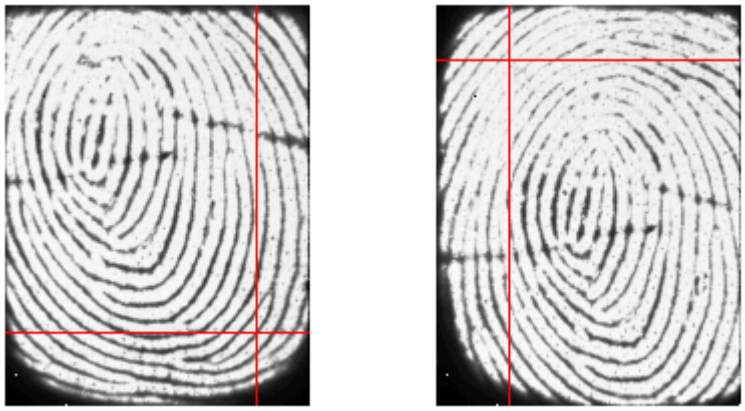

Figura 7.12. Desplazamiento entre huellas.

Con esta información obtenemos la zona común de ambas huellas y terminamos esta etapa.

En la figura 7.13 se muestra el diagrama de bloques de esta etapa.

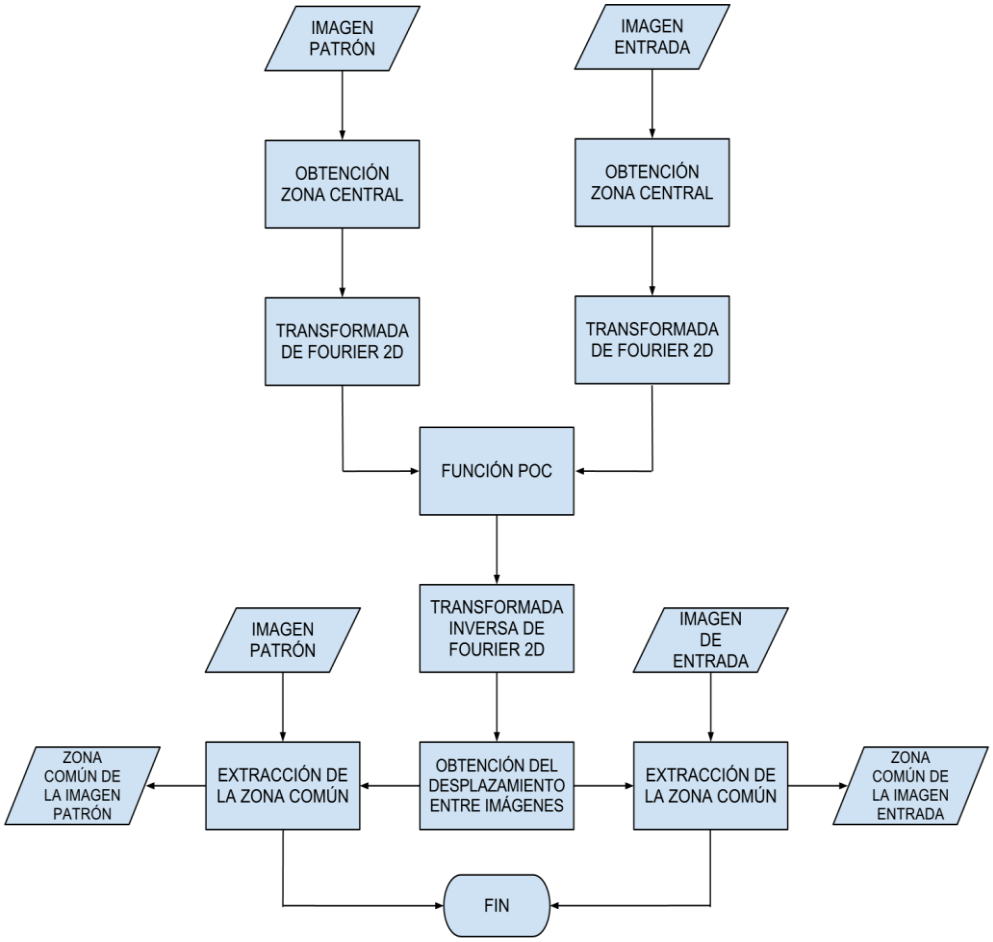

Figura 7.13. Diagrama de flujo de la etapa de alineamiento.

7.3.3. Transformación a coordenadas polares

En esta etapa haremos una transformación a coordenadas polares a las imágenes obtenidas en la etapa anterior. Con ello se consigue reducir un poco el efecto de la rotación entre ambas huellas.

En un principio disponemos de la imagen en coordenadas rectangulares o cartesianas, a partir de la cual obtendremos la imagen en coordenadas polares, la cual toma como ejes el radio y el ángulo que forma un punto respecto a un origen, mientras que las rectangulares toman como ejes la distancia a dos vectores ortonormales fijados en un punto origen.

Una vez tengamos extraída la zona común, cogeremos como origen de la transformación el extremo inferior derecho de la imagen, tomaremos como radio máximo el mínimo entre número de filas y columnas comunes y abarcaremos los

ángulos comprendidos entre 0^o y 90^o, cogiendo como sentido de rotación el de las agujas del reloj. En la figura 7.14 se muestra una imagen con la extracción de la zona común aplicada y la zona a transformar marcada, y en la figura 7.15 su transformación a coordenadas polares.

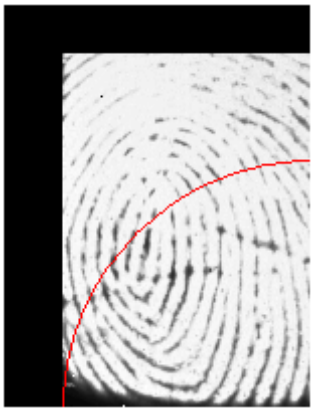

Figura 7.14. Imagen con zona común extraída y zona a transformar marcada.

La imagen en coordenadas rectangulares se representa de forma que la primera fila se encuentra en el extremo inferior y la última en el superior, mientras que la primera columna se encuentra en el extremo derecho y la última en el izquierdo.

En el caso de la imagen en coordenadas polares las filas representan la fase y las columnas el módulo. La primera fila corresponde con la fase 0º encontrándose en el extremo inferior y la última corresponde a la fase 90º encontrándose en el extremo superior. La primera columna corresponde con el módulo 0 encontrándose en el extremo derecho y la última corresponde con el módulo máximo, el cual depende del número de filas y columnas comunes de las dos imágenes a comparar, y se encuentra en el extremo izquierdo de la imagen.

La imagen obtenida en la transformación tendrá un tamaño de $128x128$ para poder aplicarle una FFT 2D de las mismas dimensiones que en la etapa anterior.

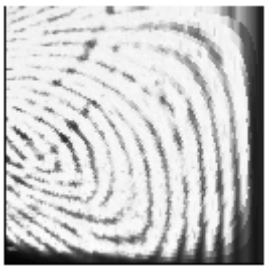

Figura 7.15. Transformación a coordenadas polares.

Este procedimiento es efectivo debido a que en la etapa anterior se elimina el efecto del desplazamiento entre imágenes, quedando solo el efecto de la rotación entre éstas. Realizando esta operación transformamos la rotación en un desplazamiento, y teniendo en cuenta que la función POC es inmune a desplazamientos el resultado final no se verá afectado por desplazamientos ni rotaciones, mejorando la efectividad del algoritmo.

Esta etapa es aportación original de este proyecto fin de carrera, y ha sido introducida debido al objetivo de seguir mejorando la eficiencia de los algoritmos basados en correlación de imágenes.

7.3.4. Comparación

Una vez acabada la etapa anterior se procede a comparar las imágenes transformadas obtenidas para dar un veredicto final.

Para ello calcularemos la función POC por el método visto anteriormente y teniendo en cuenta un umbral obtenido experimentalmente tomaremos una decisión.

Debido a que las imágenes no serán completamente iguales debido a efectos como la distorsión no lineal que aparece por la flexibilidad de la piel, el pico que muestra el nivel de similitud entre huellas quedará repartido, por lo que se tomará como medida de comparación con el umbral establecido la suma de los dos picos más altos obtenidos por la función POC de las imágenes.

En la figura 7.16 se muestra el diagrama de flujo de esta etapa.

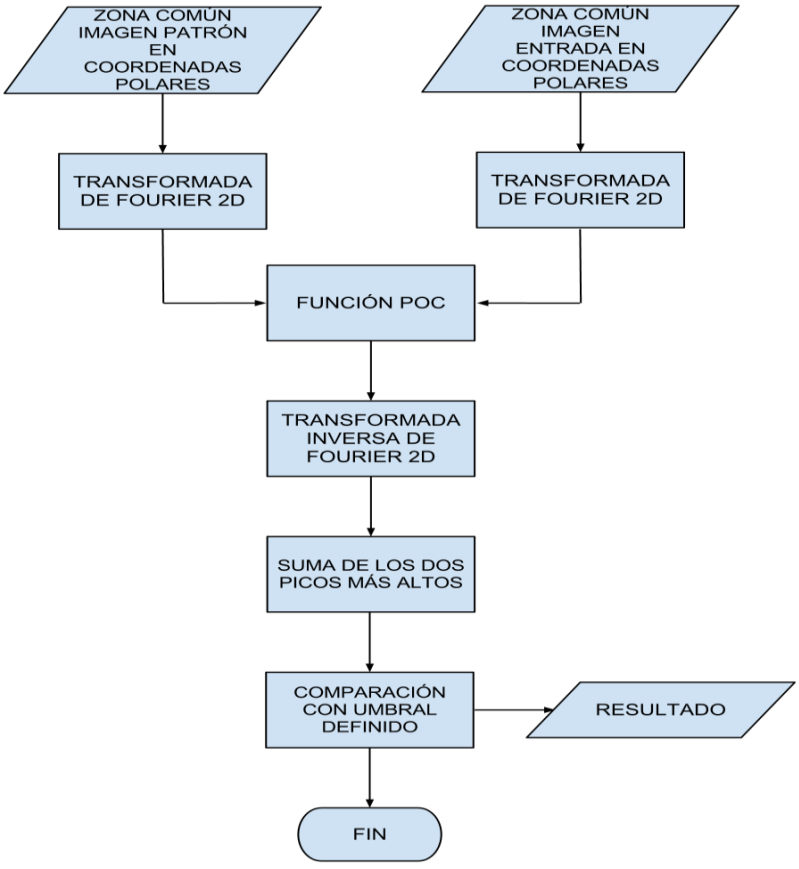

Figura 7.16. Diagrama de flujo de la etapa de comparación.

Capítulo 8. Software desarrollado para la implementación del algoritmo en el DSP

Para implementar el algoritmo descrito en el capítulo anterior en el DSP hemos introducido una base de datos en la que registraremos a varios usuarios para posteriormente poder identificarlos pidiéndoles la introducción de un número de usuario y comparando sus huellas. Por lo tanto nuestra implementación será un sistema AFAS.

El programa podrá realizar dos procesos principales:

- Registro de un nuevo usuario: la huella del individuo es capturada y a continuación se pide que se introduzca un número de usuario, el cual debe estar comprendido entre 0 y el número máximo de usuarios que puede almacenar la base de datos. Si el número está fuera del rango de valores posibles, se pide que introduzca de nuevo el número de usuario. Cuando éste sea válido se guarda la huella capturada en los datos de ese número de usuario y se indica que dicho número está siendo usado en un flag de estado.

- Verificación de la identidad del usuario: la huella del individuo es capturada y a continuación se pide que se introduzca un número de usuario, el cual debe estar comprendido entre 0 y el número máximo de usuarios que puede almacenar la base de datos. Si el número está fuera del rango de valores posibles o si no hay usuarios registrados con ese número, se pide que introduzca de nuevo el número de usuario. Cuando éste sea válido se compara la huella capturada con la huella registrada con dicho número de usuario, utilizando para ello el algoritmo desarrollado en el capítulo anterior, para finalmente dar un veredicto de la comparación. Si ésta es exitosa se encenderá un led verde en la placa del sensor, en caso contrario se encenderá un led rojo.

8.1. Organización del código desarrollado

El programa del DSP sigue un diseño modular, esto es, está organizado en distintos ficheros y con distintas funciones en los mismos, aunque lógicamente hay una función principal main en la que se ejecuta el código secuencialmente y se llaman a las distintas funciones.

Los ficheros son de dos tipos:

- Header Files, con extensión ".h", donde se incluyen definiciones de constantes, registros del DSP, definiciones de las líneas de entrada y salida para hacer más clara su programación, prototipos de funciones, definiciones de variables, estructuras de datos, etc.

- Source Files, con extensión ".c", donde se definen las variables, se declaran y llaman a las funciones a ejecutar.

8.2. Ficheros del proyecto

8.2.1. Ficheros de cabecera (Header files)

- csl.h: Contiene definiciones de la CSL (Chip Support Library, del inglés), la cual proporciona una interfaz de programación de aplicaciones (API) utilizada para la configuración y el control de los periféricos del DSP, permitiendo una mayor facilidad de uso y la compatibilidad de código entre los diversos dispositivos y hardware de la familia de DSP's C6000.
  Se compone de módulos discretos que están compilados y archivados en una librería. Cada módulo se refiere a un único periférico con la excepción de varios módulos que proporcionan programación general de apoyo, tales como la solicitud de interrupción (IRQ) del módulo que contiene una API para la gestión de interrupciones, y el módulo del chip que permite la configuración global de éste.

- csl_mcbsp.h: Contiene definiciones del módulo McBSP (multichannel buffered serial port, del inglés), el cual contiene un conjunto de funciones de una API para la configuración de los registros McBSP. Éste módulo es usado como interfaz SPI entre el DSK y la tarjeta hermana.

- dsk6713.h: Contiene definiciones de una API utilizada para la configuración y el control de los módulos del DSK que interactúan con el DSP.

- stdio.h: Es la biblioteca estándar del lenguaje de programación C, el archivo de cabecera que contiene las definiciones de macros, las constantes, las declaraciones de funciones y la definición de tipos usados por varias operaciones estándar de entrada y salida.

- math.h: es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C diseñado para operaciones matemáticas básicas.

- fpc1010.h: Contiene definiciones para el archivo fpc1010.c.

- spi.h: Contiene definiciones para el archivo spi.c.

- DSPF_sp_cfftr2_dit.h: Contiene el prototipo de la función que realiza la FTT con diezmado en el tiempo de base 2, la cual nos ha sido proporcionada por Texas Instruments y se encuentra en una librería.

- fingerprint.h: contiene definiciones de constantes, declaraciones de tipos, declaraciones de variables usadas y su localización en memoria, y los prototipos de las funciones usadas en fingerprint.c. Ha sido diseñada específicamente para este proyecto, por lo que explicaremos el objetivo de las constantes y variables declaradas en este fichero.

8.2.2. Ficheros fuente (Source files)

- fpc1010.c: Contiene las funciones necesarias para leer la imagen del sensor y guardarla en una matriz. Este código nos ha sido proporcionado por Texas Instruments, lo que ha ahorrado un importante tiempo de desarrollo.

- spi.c: Contiene las funciones necesarias para establecer la comunicación entre el DSP y el sensor, así como las necesarias para enviarle y recibir información mediante el protocolo SPI. Este código también nos ha sido proporcionado por Texas Instruments.

- fingerprint.c: Desarrollado específicamente para este proyecto, en él está el programa principal y todas las funciones necesarias específicas para esta aplicación, las cuales han sido desarrolladas por completo desde cero a excepción de la FFT unidimensional que ha sido proporcionada por Texas Instruments.

8.3. Ficheros relevantes del proyecto

En esta sección se hará una descripción del contenido de los ficheros más relevantes de este proyecto. Dos de ellos, fpc1010.c y spi.c, son aportación de Texas Instruments y se encargan de la comunicación entre el sensor y el DSP. Los dos restantes, fingerprint.h y fingerprint.c, son de desarrollo propio e implementan el algoritmo de comparación junto con la base de datos anteriormente comentada.

8.3.1. Fichero fingerprint.h

A continuación describiremos las principales variables usadas en este proyecto, cuyas declaraciones se encuentran en este fichero de cabecera.

- Contamos con las siguientes constantes globales dentro del programa:

```
#define PI 3.14159265358979323846        /* Constante que contiene un
                                            valor aproximado de la
                                            constante matemática pi.*/
#define NF 128          /* Constante que indica el número de filas que
                           tendrán las matrices complejas.*/
#define NC 128          /* Constante que indica el número de columnas
                           que tendrán las matrices complejas.*/
#define NFi 200         /* Constante que indica el número de filas que
                           tendrán las imágenes.*/
#define NCi 152         /* Constante que indica el número de columnas
                           que tendrán las imágenes.*/
#define NUMUSER 5       /* Constante que indica el número de usuarios
                           que puede haber registrados en la base de
                           datos.*/
```

- Para almacenar las imágenes se usará el siguiente tipo de variable:

```
unsigned char imagen[NFi][NCi];          /* Ejemplo de imagen. */
```

Como podemos observar serán almacenadas en vectores de tamaño $NFixNCi$. Las imágenes que usaremos estarán en escala de grises con valores comprendidos entre 0 (negro) y 255 (blanco), por lo que cada píxel de ésta ocupará 8 bits, de ahí que el tipo elegido sea el unsigned char.

A continuación veremos las variables de este tipo usadas en el proyecto:

```
unsigned char image[NFi][NCi];              /* Imagen recivida del
                                            sensor.*/
unsigned char imagecapture[NFi][NCi];       /* Copia de la imagen
                                            capturada con la que
                                            trabajaremos*/
unsigned char imagenroll[NFi][NCi];         /* Copia de la imagen de la
                                            base de datos con la que
                                            trabajaremos.*/
```

- Para realizar la FFT 2D a las imágenes se convertirán previamente a este tipo de variable:

```
float fft2dimage[NF][2*NC];                 /* Matriz compleja para la
                                            FFT 2D de una imagen.*/
```

El cálculo de la FFT 2D de una imagen se realizará a una ventana de la zona central de ésta de un tamaño $NFxNC$. Para ello es necesario almacenar la información en una matriz de números complejos y en coma flotante, por lo que será de tipo float. Los números complejos son implementados de la siguiente manera: la parte real del elemento complejo (0,0) irá en la posición (0,0), y la parte imaginaria en la (0,1); la parte real del elemento complejo (0,1) irá en la posición (0,2), y la parte imaginaria en la (0,3); y así sucesivamente para toda la fila. Respecto a las columnas: la parte real del elemento complejo (1,0) irá en la posición (1,0), y la parte imaginaria en la (1,1); y así sucesivamente. Por lo que podemos resumir que la parte real de los elementos irá en las posiciones pares y la imaginaria en las impares. Con todo esto el tamaño de la matriz deberá ser $NFx2NC$.

A continuación veremos las variables de este tipo usadas en el proyecto:

```
float fftenroll[NF][2*NC];          /* Matriz compleja para la FFT 2D
                                    de la imagen de la base de datos.*/
float fftcapture[NF][2*NC];         /* Matriz compleja para la FFT 2D
                                    de la imagen capturada.*/
```

- Para almacenar los valores twiddle necesarios para el cálculo de la FFT 2D y la IFFT 2D contamos con los siguientes vectores:

```
float tw[NC];           /* Tabla twiddle para la FFT 2D.*/
float twi[NC];          /* Tabla twiddle para la IFFT 2D.*/
```

Cada una almacenará $NC/2$ elementos complejos, por lo que necesitará NC posiciones de memoria.

- Para almacenar el resultado de aplicar el histograma a una imagen contamos con el siguiente vector:

```
unsigned int hist[256];         /* Vector que almacena el histograma de
                                una imagen.*/
```

Debido a que los píxeles de la imagen pueden tener 256 valores diferentes, el vector tendrá esos mismos elementos. Este vector no contendrá el histograma real de la

imagen debido a que el valor no está dividido entre el número de píxeles de la imagen, pero la información es la misma.

- El resultado final de la comparación de las huellas se almacenará en la siguiente variable:

```
float simil;        /* Varible que contendrá el valor de comparación de
                    las imágenes.*/
```

Este valor será comparado con el umbral establecido para dar el veredicto final de la comparación.

- El vector aquí descrito contendrá el desplazamiento entre imágenes:

```
int coord[2];       /*  Vector  para  almacenar  las  coordenadas  del
                    desplazamiento entre imágenes.*/
```

Su obtención es necesaria para la extracción de la zona común de las imágenes.

- El siguiente vector almacenará valores de senos y cosenos:

```
float phasepolar[2*NF];         /* Tabla con valores de cosenos y senos
                                necesarios  para  la  transformación  a
                                coordenadas polares.*/
```

Su uso permite la reducción del tiempo de ejecución del la función que transforma a coordenadas polares la imagen.

- Finalmente para la implementación de la base de datos hemos definido un nuevo tipo que contenga los datos de cada usuario, el cual se muestra a continuación:

```
typedef struct
{
      unsigned char imageuser[NFi][NCi];        // Imagen de la huella
                                                delusuario.*/
      int userenroll;           // Variable que indica si el número de
                                usario está registrado ya.*/
}user;
```

Con este tipo de dato crearemos un vector que contenga tantos usuarios como la variable NUMUSER indique. A continuación mostramos la declaración:

```
user database[NUMUSER];         /* Definición de una base de datos de
                                NUMUSER miembros.*/
```

Las demás declaraciones del fichero son los prototipos de las funciones usadas, los cuales no explicaremos debido a que lo haremos con las funciones. Con esto queda concluida la explicación de este fichero de cabecera.

8.3.2. Fichero fpc1010.c

En este fichero se encuentran las funciones encargadas de interactuar con el sensor. El objetivo de dichas funciones se explica a continuación:

- `void sensorInit(void)`

Configura el sensor para capturar una huella. Esta función debe ser llamada siempre antes de usar el sensor.

- `void readImage(unsigned char *image_ptr)`

Captura la huella del sensor y la almacena a partir de la dirección a la que apunta image_ptr, el cual es un puntero a la matriz que contendrá la imagen de la huella capturada.

8.3.3. Fichero spi.c

En este fichero se encuentran las funciones encargadas de establecer la comunicación SPI entre el sensor y el DSP. El objetivo de dichas funciones se explica a continuación:

- `int ConfigMcbsp(int spiSpeed, int swap)`

Configura los registros del McBSP para establecer a éste como la interfaz de comunicación SPI entre el sensor y el DSP.

- `int SPI_init(int spiSpeed,int swap)`

Se encarga de llamar a la función anterior para establecer la comunicación mencionada anteriormente. Tiene los mismos parámetros de entrada. El primero es la velocidad de la comunicación y el segundo es para indicar si deshabilitamos el sensor durante esta operación o no.

- `void SPI_write(int command)`

Es usada para mandar mensajes al sensor, tales como una petición de captura de huella.

- `unsigned char SPI_read(void)`

Es usada para leer los datos de la captura de huella del sensor.

8.3.4. Fichero fingerprint.c

Este fichero contiene la solución específica del proyecto, por lo que es el más importante de todos. A continuación pasaremos a explicar cada una de las funciones creadas, y en última instancia explicaremos la forma en que la función principal, main, controla el flujo del programa llamando a las demás funciones.

8.3.4.1. Funciones desarrollas

- `void dummyEnroll(void)`

Su objetivo es el control del botón virtual creado para registrar una nueva huella en la base de datos.

- `void dummyVerify(void)`

Su objetivo es el control del botón virtual creado para capturar una huella y compararla con otra registrada previamente en la base de datos.

- `void gen_phase_table(float* phase, int nf)`

Función que genera una tabla con valores de cosenos y senos en el vector phase. Esta tabla será usada por la función trans_polar_image y su objetivo es reducir el tiempo de ejecución de dicha función. Esto se logra evitando hacer las operaciones de seno y coseno en la función, las cuales son bastante costosas computacionalmente.
Parámetros de entrada:

phase: vector de tamaño 2*nf. En los elementos pares se almacenarán los valores de los cosenos, y en los impares los correspondientes a los senos.
nf: número de filas de la matriz que contendrá el resultado de la función trans_polar_image.

- `void copy_image(unsigned char imi[][NCi], unsigned char imo[][NCi], int nf, int nc)`

Copia el contenido de la imagen imi en la imagen imo, ambas de tamaño nfxnc.
Parámetros de entrada:

imi: imagen de entrada.
imo: imagen que contendrá la copia.
nf: número de filas de la imagen de entrada.
nc: número de filas de la imagen de entrada.

- `void histogram(unsigned char im[][NCi], unsigned int* h, int nf, int nc)`

Función que calcula el histograma de la imagen im, el cual es almacenado en h.
Parámetros de entrada:

im: imagen de entrada.
h: puntero al vector que contendrá el histograma.
nf: número de filas de la imagen.
nc: número de columnas de la imagen.

- `void preprocessing(unsigned char im[][NCi], unsigned int* h, int nf, int nc)`

Esta función realiza el el preprocesado a la imagen im. Dicho preprocesado es el que se explicó en el capítulo anterior. Para ello es necesario calcular previamente el histograma de la imagen, cuyo resultado es pasado a la función mediante el parámetro h.
Parámetros de entrada:

im: imagen de entrada.
h: puntero al vector que contiene el histograma de dicha imagen.
nf: número de filas de la imagen.
nc: número de columnas de la imagen.

- `void complex_image(unsigned char im[][NCi], float x[][2*NC], int nfi, int nci, int nf, int nc)`

Esta función coge una ventana de tamaño nfxnc del centro de la imagen im, la cual tiene un tamaño nfixnci, y la pasa a la matriz compleja x para posteriormente aplicarle la FFT 2D.
Parámetros de entrada:

im: imagen de entrada.
x: matriz que contendrá la información de la imagen en números complejos.
nfi: número de filas de la im.
nci: número de columnas de la im.
nf: número de filas de x.
nc: número de columnas de x.

- `void gen_twiddle(float* w, int n)`

Función que genera la tabla twiddle. Su uso disminuye considerablemente el tiempo de ejecución de la FFT, por lo que será usada posteriormente en dicha operación.
Parámetros de entrada:

w: tabla donde se guardarán los valores twiddle necesarios para la FFT.
n: número de elementos complejos a los que se les hará la FFT.

- `void copy_vector(float* x, float* y, int n)`

Función que copia un vector x en el vector y, ambos de tamaño n.
Parámetros de entrada:

x: vector que contiene los elementos a copiar.
y: vector que recibirá los valores de x.
n: número de elementos del vector x.

- `void complex_conj_vector(float* x, int n)`

Función que realiza el complejo conjugado a todos los elementos del vector complejo x, de n elementos complejos.
Parámetros de entrada:

x: vector complejo al que se le quiere hacer el complejo conjugado.
n: número de elementos complejos del vector.

- `void bit_rev(float* x, int n)`

Función que realiza un reordenamiento de los n elementos complejos del vector x mediante la operación del bit inverso.
Parámetros de entrada:

x: vector al que se le quiere hará dicha transformación.
n: número de elementos complejos del vector.

- `void trasp_matrix(float x[][2*NC], int nf, int nc)`

Dada la matriz compleja x, esta función realiza la traspuesta a la misma.
Parámetros de entrada:

x: matriz compleja a la que se le hará dicha transformación.
nf: número de filas.
nc: número de columnas.

- `void fft2d(float x[][2*NC], float* tw_f, float* tw_c, int nf, int nc)`

Función que realiza la FFT 2D a la matriz compleja x de tamaño nfxnc.
Parámetros de entrada:

x: matriz a la que se le hará la transformación.
tw_f: tabla twidle correspondiente a las filas.
tw_c: tabla twidle correspondiente a las columnas.
nf: número de filas.
nc: número de columnas.

Para llevar a cabo esta transformación hemos hecho uso de la siguiente función creada por Texas Instrumentes:

```
void DSPF_sp_cfftr2_dit(float* x, float* w, int n)
```

Función que realiza la FFT unidimensional al vector complejo x de tamaño n. Dicha FFT realiza diezmado en tiempo y es de base 2. Sus propiedades fueron explicadas en un capítulo anterior.
Parámetros de entrada:

x: vector al que se le hará la transformación. El resultado es dado en orden de bit inverso.
w: puntero a tabla twidle. Debe ser introducido en orden de bit inverso.
n: número de elementos complejos.
n: número de elementos complejos.

La estrategia elegida es la siguiente:

Para llevar a cabo la FFT 2D, descomponemos la transformación en la aplicación de FFT's unidimensionales. En primer lugar realizamos la FFT sobre todas las filas de la matriz. Una vez realizada esta etapa trasponemos la matriz y aplicamos la misma transformación. De esta manera se realiza la FFT a todas las columnas de la matriz. El último paso es ordenar nuestra matriz transformada, lo cual se hace trasponiendo otra vez dicha matriz.

El algoritmo que realiza la FFT necesita los factores twiddle en orden de bit inverso, por lo que a la hora de crearlos son almacenados en ese orden. Además por ser independientes la FFT aplicada a las filas de la de las columnas tenemos dos tablas twiddle como parámetros de entrada (una para cada FFT).

Cada vez que aplicamos una transformación FFT con este algoritmo el vector resultado es dado en orden de bit inverso, por lo que habrá que reordenarlo en cada aplicación de dicha función.

Esta función se puede usar para realizar la IFFT 2D a una matriz compleja también. Lo único que hay que hacer es aplicar el complejo conjugado a los factores twiddle suministrados como parámetros de entrada. El resultado final es la IFFT 2D sin dividir por el número total de elementos, por lo que dicha operación se realizará fuera de esta función con otra creada para dicho objetivo.

- `void divide_ifft2d(float x[][2*NC], int nf, int nc)`

Función que realiza la división por el producto nfxnc (número de elementos) a todos los elementos de la matriz compleja x. Es aplicada después de realizar una IFFT 2D.
Parámetros de entrada:

x: matriz compleja a la que se le hará dicha transformación.
nf: número de filas.
nc: número de columnas.

- `void poc_function(float x[][2*NC], float y[][2*NC], int f, int c)`

Función que realiza la función POC (Phase Only Correlation, del inglés) al par de elementos indicados por f (fila) y c (columna) a las matrices complejas x e y. El resultado se guardará en el elemento oportuno de la matriz x.
Parámetros de entrada:

x: matriz compleja 1.
y: matriz compleja 2.
f: fila del elemento al que se le aplica la función.
c: columna del elemento al que se le aplica la función.

- `void poc(float x[][2*NC], float y[][2*NC], int nf, int nc)`

Función que realiza la función POC (Phase Only Correlation, del inglés) a las matrices complejas x e y, ambas de tamaño nfxnc y previamente convertidas al dominio de Fourier, devolviendo el resultado en la matriz x. Para ello hace uso de la función poc_function y reduce el número de operaciones a realizar aprovechando la propiedad de simetría de la transformada de Fourier 2D.
Parámetros de entrada:

x: matriz compleja 1, en ésta matriz se guardará el resultado de la función.
y: matriz compleja 2.
nf: número de filas.
nc: número de columnas.

- `float add_max_poc(float x[][2*NC], int nf, int nc)`

Función que encuentra los dos valores más altos dados por la aplicación de la función POC a dos matrices complejas y devuelve su suma.
Parámetros de entrada:

x: matriz compleja con el resultado de la función POC a la que se le ha hecho la IFFT 2D previamente para convertirla al dominio espcial.
nf: número de filas.
nc: número de columnas.

- `void high_peak_poc(float x[][2*NC], int* coor, int nf, int nc)`

Función que encuentra la posición del pico más alto de la matriz compleja x, la cual continene el resultado de aplicar la función POC a dos matrices complejas, y a partir de esta posición determinar el despalzamiento entre las dos imágenes originales del proceso para almacenar el resultado en el vector coor.
Parámetros de entrada:

x: matriz compleja con el resultado de la función POC a la que se le ha hecho la IFFT 2D previamente.
nf: número de filas.
nc: número de columnas.

- `void desp_image(unsigned char im[][NCi], int* coor, int enr, int nf, int nc)`

Función que recorta la imagen im según el desplazamiento calculado previamente respecto a otra imagen indicado por el vector coor. Con esto se extrae la zona común de ambas imágenes.
Parámetros de entrada:

im: imagen que será recortada.
coor: desplazamineto entre huellas.
enr: = 0 => se refire a imagen capturada; = 1 => se refier a imagen de la base de datos.
nf: número de filas de la imagen.
nc: número de columnas de la imagen.

- `void trans_polar_image(unsigned char im[][NCi], float x[][2*NC], int* coor, int nfi, int nci, int nf, int nc)`

Función que transforma una imagen en coordenadas rectangulares a coordenadas polares usando el método visto en el capítulo anterior. El resultado es almacenado en una matriz compleja para su posterior transformación al dominio de Fourier. Para reducir su tiempo de ejecución se le pasará una tabla con los valores de senos y cosenos necesarios ya calculados anteriormente.
Parámetros de entrada:

im: imagen de entrada.
x: matriz compleja a la que se le pasará la información (filas => fase; columnas => radio).
coor: desplazaminento entre imágenes.
phase: dicho vector contiene valores de senos y cosenos usados en esta función.
nfi: número de filas de im.
nci: número de columnas de im.
nf: número de filas de x.
nc: número de columnas de x.

8.3.4.2. Función principal

- `void main(void)`

Esta función es la encargada de la gestión del flujo del programa. En la figura 8.1 se muestra el diagrama de bloques del flujo del programa.

A continuación veremos que operaciones se realizan en cada proceso del diagrama:

- Inicialización:

En esta etapa se configura la placa de acuerdo a los requerimientos de la aplicación. A groso modo se inicializa la placa en sí, el módulo de leds, la comunicación SPI entre la placa y el sensor, se establece el tamaño de la base de datos, y se generan las tablas de factores twiddle necesarias para el cálculo de la FFT 2D.

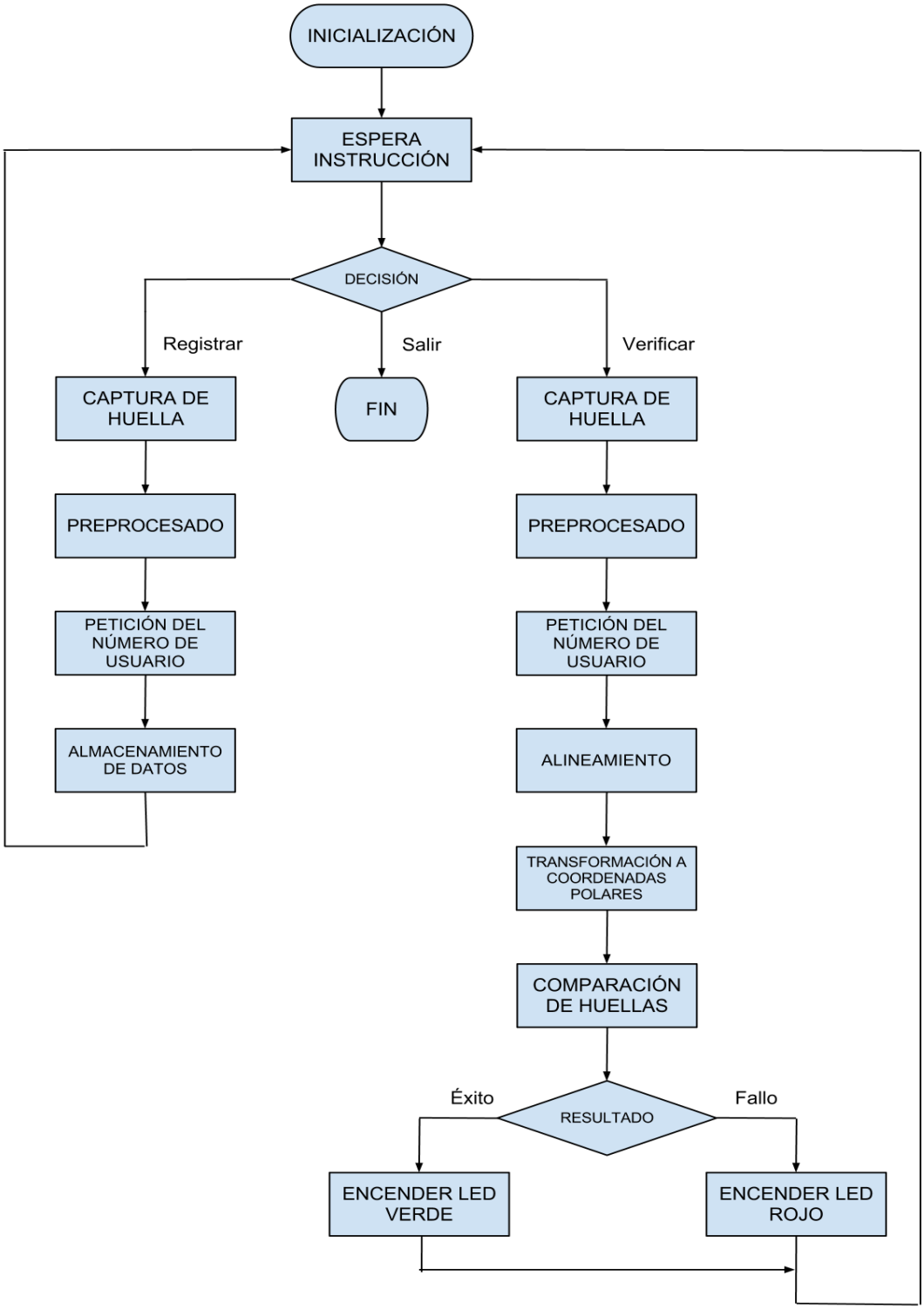

Figura 8.1. Diagrama de flujo del programa.

- Espera instrucción:

En esta etapa se ejecuta un bucle en el que no se hace nada a parte de esperar a que el usuario haga uso de los botones virtuales creados en el fichero botones.gel para actuar en consecuencia. Según el botón usado el programa seguirá una secuencia de instrucciones diferente.

Mediante el botón Verificar ejecutaremos la rama de instrucciones necesarias para relaizar la verificación de la identidad de un usuario.

Mediante el botón Registrar ejecutaremos la rama de instrucciones necesarias para relaizar el regisro de un usuario en la base de datos.

Mediante el botón Salir ejecutaremos la rama de instrucciones necesarias para realizar la terminación del programa.

- Captura huella:

En esta etapa iniciamos la comunicación SPI con el sensor mediante la función sensorInit y leemos la imagen capturada mediante la función readImage, la cual guarda la información en una matriz.

- Preprocesado:

Realiza el preprocesado explicado en el capítulo anterior. Para ello se realiza el histograma a la matriz que contiene la imagen de la huella con la función histogram y posteriormente realiza una expansión del contraste y una inversión de la escala de grises mediante la función preprocessing.

- Petición del número de usuario:

En la secuencia de instrucciones para registrar a un usuario este proceso pide a dicho usuario que introduzca por teclado un número. Se comprueba que el número no supere al máximo de usuarios posibles de la base de datos, y si no cumple la especificación pide que vuelva a introducir el número por no ser aceptado el anterior.

En la secuencia de instrucciones para verificar la identidad de un usuario este proceso realiza la misma comprobación que en el caso anterior y además se comprueba que dicho número ya contiene una huella registrada de un usuario.

- Almacenamiento de datos:

En este proceso se almacena la huella preprocesada del usuario en la localización indicada por el número de usuario mediante la función copy_image.

- Alineamiento:

Se pasan las matrices que contienen la imagen capturada y la registrada ya preprocesadas a matrices complejas mediante la función complex_image. Una vez hecho esto le aplicamos la FFT 2D a cada matriz mediante la función fft_2d, y estas dos matrices son pasadas a la función poc para calcular la correlación entre ambas. Al

resultado le hacemos la IFFT 2D mediante la función fft_2d y la posterior aplicación de divide_n para obtener su homólogo en el dominio espacial. Mediante la función high_peak_poc obtenemos el desplazamiento entre ambas imágenes para posteriormente extraer la zona común de ambas imágenes mediante la función desp_image.

- Transformación a coordenadas polares:

En este proceso, a las imágenes obtenidas del porceso anterior se le aplica la función trans_polar_image, la cual relaiza la transformación a coordenadas polares de las imágenes tal y como explicamos en el capítulo anterior, guardando el resultado en una matriz compleja.

- Comparación de huellas:

A las matrices obtenidas en el proceso anterior se le realiza la FFT 2D mediante la función fft_2d, y estas dos matrices son pasadas a la función poc para calcular la correlación entre ambas. Al resultado le hacemos la IFFT 2D mediante la función fft_2d y la posterior aplicación de divide_n para obtener su homólogo en el dominio espacial. Al resultado se le aplica la función add_max_poc, la cual suma los dos picos más altos de éste.

Dicha suma será el resultado final del proceso, que será comparado con un umbral predefinido. Si resulta mayor, la comparación es exitosa. En caso contrario la comparación es fallida.

- Encender led verde:

Si la comparación resulta exitosa indicamos a la placa que encienda un led verde en la placa del sensor.

- Encender led rojo:

Si la comparación resulta fallida indicamos a la placa que encienda un led rojo en la placa del sensor.

- Fin:

Ejecutamos la función exit, la cual es aportada por la librería stdio y termina la ejecución del programa.

8.4. Botones virtuales

Para la inclusión de unos botones virtuales cuyo uso será la interacción del usuario con el DSP se ha creado un fichero con extensión ".gel".

Code Composer Studio puede usar este tipo de ficheros para crear elementos virtuales que comuniquen al usuario con el DSP.

El fichero creado se denomina botones.gel e implementa los siguientes botones:

- Registrar: interactua con el programa mandando la orden necesaria para realizar el registro de un nuevo usuario.
- Verificar: interactua con el programa mandando la orden necesaria para realizar la verificación de la identidad de un usuario.
- Salir: interactua con el programa mandando la orden necesaria para terminar la ejecución de éste.

Capítulo 9. Pruebas y resultados

9.1. Prueba de errores y determinación del umbral de decisión

La implementación en el DSP ha sido puesta a prueba de la siguiente manera:

- Para el cálculo de la tasa de falso rechazo o FRR se han registrado en la base de datos 5 huellas diferentes. Cada una de esas huellas a sido comparada con 6 capturas diferentes de la misma huella y se ha registrado el valor final obtenido en cada comparación. Ello hace un total de 30 muestras al final del proceso para ser estudiadas.

La siguiente tabla muestra los datos obtenidos:

Huella	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6
Nº 1	0,1361	0,2367	0,2229	0,0334	0,2446	0,4811
Nº 2	0,4359	0,2785	0,2493	0,1454	0,3407	0,2007
Nº 3	0,3841	0,1279	0,3548	0,4011	0,2730	0,1132
Nº4	0,5868	0,0703	0,4752	0,2103	0,2176	0,3767
Nº 5	0,3362	0,2470	0,1891	0,5171	0,3098	0,1677

Tabla 9.1. Datos FRR.

- Para el cálculo de la tasa de falsa aceptación o FAR se han registrado en la base de datos 6 huellas diferentes. Cada huella a sido comparada con capturas diferentes de las demás huellas y se ha registrado el vaor final obtenido en cada comparación. Ello hace un total de 30 muestras al final del proceso para ser estudiadas.

La siguiente tabla muestra los datos obtenidos:

Huella	Nº 1	Nº 2	Nº 3	Nº 4	Nº 5	Nº 6
Nº 1	X	0,1092	0,0630	0,0683	0,0971	0,0720
Nº 2	0,0702	X	0,0576	0,0616	0,0696	0,1082
Nº 3	0,0652	0,0764	X	0,1163	0,0849	0,0785
Nº 4	0,0705	0,0857	0,0641	X	0,0860	0,0728
Nº 5	0,0682	0,0690	0,0729	0,0684	X	0,0808
Nº 6	0,0606	0,0626	0,0632	0,0979	0,0651	X

Tabla 9.2. Datos FAR.

A partir de los datos obtenidos hemos obtenido la gráfica que representa la probabilidad de que ocurran FRR y FAR en función del umbral escogido para tomar la decisión de si la comparación ha sido exitosa o fallida.

La probabilidad de que ocurra FRR es el número de muestras que han obtenido un resultado menor que el umbral (sensibilidad) dividido entre el número total de muestras.

La probabilidad de que ocurra FAR es el número de muestras que han obtenido un resultado mayor que el umbral (sensibilidad) dividido entre el número total de muestras.

A continuación se muestra una gráfica con la representación de ambas tasas:

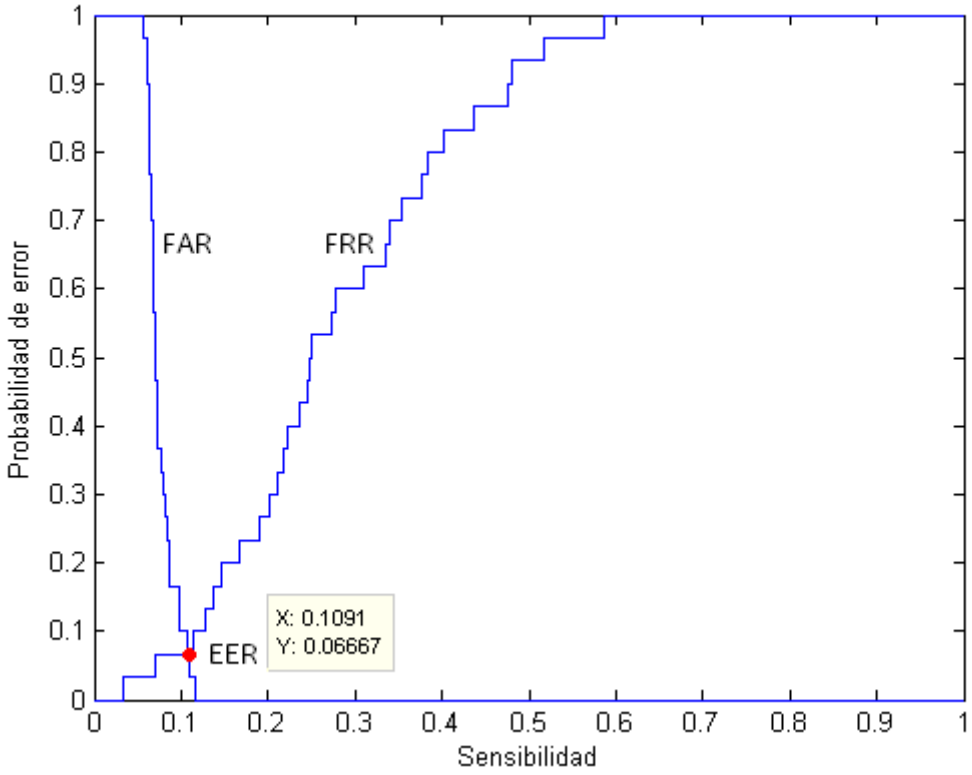

Figura 9.1. Probabildad de error en función del umbral de decisión.

Como podemos observar la tasa de error igual (EER) se da para un umbral de $0{,}1091$ con una probabilidad de que ocurran FRR o FAR del $6{,}67\%$.

La sensibilidad del sistema representa el umbral de comparación escogido. El resultado final del algoritmo de comparación puede tomar valores comprendidos entre 0 y 1, por lo que el umbral de decisión tendrá uno de esos valores.

Dados estos resultados vamos a escoger el umbral de decisión con un valor de $0{,}12$, es decir, cuando el resultado final se mayor que $0{,}12$ la identificación será satisfactoria; en caso contrario será fallida. Con ello aumentamos la tasa de falso rechazo, pero a cambio la tasa de falsa aceptación se hace nula. De esta manera nuestro sistema será mucho más seguro que con el umbral del punto EER.

9.2. Tiempo de ejecución

La última prueba realizada es la medición del tiempo de ejecución del algoritmo. Dicha prueba ha sido realizada a los procesos de registro y verificación de identidad. Se ha

excluido el tiempo que tarda el usuario en introducir el número de usuario, y los datos obtenidos son los siguientes:

Proceso	Tiempo
Registro de usuario	0,70 s
Verificación de identidad	7,36 s

Tabla 9.3. Tiempo de ejecución.

Capítulo 10. Conclusiones

10.1. Cumplimiento de los objetivos

En este proyecto se ha presentado un algoritmo de comparación de imágenes basado en técnicas de correlación, el cual consta de las siguientes etapas: preprocesado, alineamiento, transformación a coordenadas polares y comparación. En el preprocesado se mejora ligeramente la imagen de la huella. En el alineamiento se correlan las imágenes para estimar el desplazamiento y se extrae la zona común a ambas. Una vez hecho esto se transforman a coordenadas polares amabas imágenes para evitar el efecto de la rotación en la siguiente etapa. Finalmente en la etapa de comparación se correlan las imágenes obtenidas de la etapa anterior y se da el resultado del proceso.

El algoritmo se ha desarrollado en lenguaje C e implementado en un sistema DSP arrojando los resultados que comentaremos a continuación.

Los resultados experimentales demuestran que el algoritmo es preciso con EER del 6,67%. Además, este algoritmo presenta una nueva técnica para paliar el efecto de pequeñas rotaciones basada en la transformación de las coordenadas en polares. Dicha técnica es una aportación original de este proyecto.

Por otra parte el tiempo necesario para el registro de un usuario resulta ser de 0,70 segundos, y el necesario para la verificación de la identidad de un usuario es de 7,36 segundos.

Se ha trabajado bastante para que el tiempo de ejecución sea lo menor posible, sin embargo este es susceptible de una optimización mayor, desarrollando código en ensamblador que saque mayor provecho del hardware del DSP. El proceso de optimización del código es una tarea basatante difícil y requiere un largo tiempo de desarrollo, por lo que no hemos abordado este tema en nuestro proyecto.

10.2. Líneas futuras de investigación

A partir del trabajo realizado durante este proyecto, surgen futuras líneas de trabajo que se resumen a continuación.

En primer lugar, en cuanto a los algoritmos de comparación de huellas dactilares basados en técnicas de correlación, un campo interesante es la combinación de las técnicas propuestas con otros métodos, con el fin de mejorar las tasas de error.

En segundo lugar, se podría seguir optimizando el código de este proyecto para reducir su tiempo de ejecución y hacer uso de otras técnicas para mejorar la implementación del mismo algoritmo.

Referencias

[1] Chassaing, R. & Reay, D. (2008). *Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK.* New Jersey (USA): Wiley-Interscience. ISBN: 978-0-470-13866-3.

[2] Qureshi, Shehrzad. (2005). *Embedded Image Processing on the TMS320C6000 DSP: Examples in Code Composer Studio and MATLAB.* New York (USA): Springer Science + Business Media, Inc. ISBN: 978-0-307-25280-3.

[3] Nakajima, Hiroshi. (2004). *A Fingerprint Matching Algorithm Using Phase-Only Correlation.* En Q. Shi, Yun. *Transactions on Data Hiding and Multimedia Security V* (pp 682-691). New Jersey (USA): New Jersey Institute of Technology. ISBN: 978-3-642-14297-0.

[4] Ito, Koichi. (2005). *A Fingerprint Recognition Algorithm Using Phase-Based Image Matching for Low-Quality Fingerprints.* En S.P. Wang, Patrick. *Pattern Recognition, Machine Intelligence and Biometrics*. Boston (USA): Hungtinton Ave Northeastern University. ISBN: 978-3-642-22406-5.

[5] Ramírez Aragón, J. (Junio-2011). *Diseño e implementación de un sistema de ecualización de audio basado en DSP*. Proyecto Fin de Carrera. I.T.I. Esp. Electrónica Industrial. Sevilla: Departamento de Tecnología Electrónica. Escuela Politécnica Superior. Universidad de Sevilla.

[6] Lindoso Muñoz, Almudena. (2009). *Contribución al reconocimiento de huellas dactilares mediante técnicas de correlación y arquitecturas hardware para el aumento de prestaciones.* Tesis doctoral. Leganés: Departamento de Tecnología Electrónica. Universidad Carlos III de Madrid.

[7] Juste Meco, Rubén. (2010). *Módulo de Identificación Biométrica Mediante Huellas Dactilares para Sistemas Empotrados.* Proyecto fin de carrera. Leganés: Departamento de Tecnología Electrónica. Escuela Politécnica Superior. Universidad Carlos III de Madrid.

[8] *Code Composer Studio Getting Started Guide.* SPRU509. Texas Instruments. Dallas. TX, 2001.

[9] *TMS320C6000 Chip Support Library API Reference Guide*. SPRU401J. Texas Instruments. Dallas. TX, 2004.

[10] *TMS320C6000 Code Composer Studio Tutorial.* SPRU301C. Texas Instruments. Dallas. TX, 2000.

[11] *TMS320C6713 DSK Technical Reference*. Spectrum Digital, Stattfford. TX, 2003.

[12] *TMS320C6713 Floating-Point Digital Signal Processor.* Texas Instruments. Dallas. TX, 2001.

[13] *FPC1010 Finger Print Sensor Daughter Card Technical Reference.* Spectrum Digital, Stattfford. TX, 2003.

[14] *FPC1010 Sensor Drivers Quick Start Guide.* Texas Instruments. Dallas. TX, 2003.

[15] Web de Spectrum Digital. *Búsqueda de documentación sobre el TMS320C6713 DSK* [en línea]. http://c6000.spectrumdigital.com. [Consulta: 2012].

[16] Web de Texas Instruments. *Búsqueda de documentación sobre el TMS320C6713 DSP* [en línea]. http://www.ti.com. [Consulta: 2012].